



CHALMERS
UNIVERSITY OF TECHNOLOGY



UNIVERSITY OF GOTHENBURG

Genetic Algorithm and Simulated Annealing for Flexible Job Shop Scheduling Problem with Time Constraints

Master's thesis in Computer science and engineering

Zihao Zhou

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

MASTER'S THESIS 2025

**Genetic Algorithm and Simulated
Annealing for Flexible Job Shop
Scheduling Problem with Time Constraints**

Zihao Zhou



UNIVERSITY OF
GOTHENBURG



CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Genetic Algorithm and Simulated Annealing for Flexible Job Shop Scheduling Problem with Time Constraints

Zihao Zhou

© Zihao Zhou, 2025.

Supervisor: Gabriel Reder, Computer Science and Engineering
Examiner: Dag Wedelin, Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in L^AT_EX
Gothenburg, Sweden 2025

Genetic Algorithm and Simulated Annealing for Flexible Job Shop Scheduling Problem with Time Constraints

Zihao Zhou

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

In this work, we investigate the Scheduling for Laboratory Automation in Biology (S-LAB) problem, which arises in time-sensitive laboratory workflows. S-LAB is an extension of the Flexible Job Shop Scheduling Problem (FJSP) with Time Constraints by Mutual Boundaries (TCMB). We proposed a hybrid Genetic Algorithm-Simulated Annealing (GASA) approach that combines the global exploration capabilities of genetic algorithms with the local refinement strength of simulated annealing. The genetic algorithm component employed a novel triple encoding scheme incorporating Operation Sequence, Machine Selection, and Operation Delay to effectively represent scheduling solutions under strict time constraints. We conducted comprehensive experiments on multiple realistic laboratory protocol datasets to compare the performance of GASA with those of the Branch and Bound (B&B) and SAGAS algorithms. The results demonstrated that GASA achieved optimal solutions in simpler scenarios and near-optimal results in complex cases with significantly reduced computational time compared to B&B. On the most complex dataset (qPCR-RNAseq \times 5), GASA reduced execution time by more than 96% while incurring only a 10.1% increase in makespan relative to the optimum. The proposed approach provides an effective balance between solution quality and computational efficiency, making it particularly suitable for time-sensitive laboratory automation applications.

Keywords: Flexible Job Shop Scheduling, Genetic Algorithm, Simulated Annealing, Laboratory Automation, Time Constraints, S-LAB

Acknowledgements

Completing this thesis has been a difficult and often frustrating journey, which came with more setbacks than I had anticipated. There were moments when I seriously doubted whether I could see it through. Still, I feel incredibly grateful for the people who supported me during these times.

I would like to sincerely thank my supervisor, Gabriel, for his steady guidance and generous support throughout this process. His insights and detailed comments helped me refine my thinking and move forward whenever I felt stuck.

I'm also deeply grateful to my examiner, Dag, whose thoughtful and challenging questions pointed out some truly important aspects of the topic that I had completely missed. His feedback not only sharpened the structure of my argument, but also reminded me to be more critical and precise in my reasoning. Without his input, this thesis would not have reached its current level of clarity.

Last but not least, I want to thank my family. During the times I felt most overwhelmed and lost, it was their quiet support and unconditional encouragement that helped me keep going. I simply couldn't have done this without them.

Zihao Zhou, Gothenburg, 2025-08-04

Contents

List of Figures	xi
List of Tables	xiii
1 Introduction	1
1.1 Background	1
1.2 S-LAB problem	1
1.3 Genetic Algorithm for FJSP	2
1.4 Hybrid GA-SA Strategy	3
1.5 Proposed Methods	4
1.6 Paper Structure	4
2 Theory	5
2.1 Problem Formulation	5
2.1.1 Objective and Constraints	6
2.1.2 Assumptions	7
2.1.3 Scheduling Solution	7
2.2 Solution Approaches	7
2.3 Genetic Algorithm	8
2.3.1 Advantages of GA in Solving FJSP	8
2.3.2 Encoding Schemes	9
2.3.3 GA Components for FJSP	9
2.3.3.1 Initialization	9
2.3.3.2 Fitness Evaluation	9
2.3.3.3 Selection Strategies	10
2.3.3.4 Crossover	10
2.3.3.5 Mutation	10
2.3.3.6 Termination Criteria	11
2.4 Simulated Annealing	11
2.4.1 Initialization	11
2.4.2 Neighborhood Search	12
2.4.3 Cooling Schedule	13
2.4.4 Termination Criteria	13
3 Methods	15
3.1 Genetic Algorithm	15

3.1.1	Encoding and decoding	15
3.1.1.1	Encoding	15
3.1.1.2	Decoding	16
3.1.2	Initialization	16
3.1.3	Fitness Function	17
3.1.4	Genetic Operation	18
3.1.4.1	Selection	18
3.1.4.2	Crossover	18
3.1.4.3	Mutation	18
3.1.5	Local search for OD	21
3.1.6	Diversification Strategy	24
3.2	Simulated Annealing	25
3.2.1	Initial Solution	25
3.2.2	Objective Function with Penalization	25
3.2.3	Temperature Control	26
3.2.4	Neighborhood Generation	26
3.2.5	Solution Acceptance	27
3.2.6	Final modification	27
3.3	Proposed GASA	27
4	Results	31
4.1	Dataset Description	31
4.2	Implementation and Evaluation	32
4.3	Experimental Results	33
4.4	Performance Comparison and Analysis	37
4.4.1	Makespan Performance	37
4.4.1.1	Optimality	37
4.4.1.2	Stability	38
4.4.1.3	Impact of Problem Structure	38
4.4.2	Computational Efficiency	38
4.4.3	Practical Implications	38
5	Conclusion and Future Work	43
5.1	Conclusion	43
5.2	Future Work	43
	Bibliography	45
A	Appendix 1	I

List of Figures

3.1	Comparison of crossover methods for FJSP: (left) Precedence Operation Crossover (POX) preserves the relative order of operations within jobs; (right) Job-Based Crossover (JBX) maintains the position-based assignment integrity by directly copying operations from both parents to the corresponding positions in offspring.	20
3.2	MS Two-point Crossover: Exchanges machine assignment information while maintaining operation sequence	20
3.3	Operation Sequence mutation operators: (left) OS Swapping Mutation exchanges the positions of two operations within the operation sequence; (right) OS Insertion Mutation removes an operation from position p1 and inserts it at position p2.	22
3.4	Machine Selection mutation: MS Reassignment Mutation changes the machine assignments for selected operations while maintaining operation sequence integrity.	23
4.1	Workflow diagram of the Gu2016 protocol. The blue nodes (T) represent operations executed on the Transportation system, the yellow nodes represent operations executed on Reformatter, green node represents operations executed on Motoman, purple nodes represent operations executed on Biomek2000.A3, and red node represents operation executed on Biomek2000.A4. Arrows indicate execution order.	32
4.2	Workflow diagram of the qPCR protocol. The blue nodes (T) represent operations executed on the Transportation system, the green node represents operation executed on Tecan(Type1), the yellow nodes represent operations executed on Plate Centrifuge, the orange node represents operation executed on Tecan(Type2), the red node represents operation executed on Tecan(Type3), and the purple node represents operation executed on qRT-PCR. Arrows indicate execution order.	32
4.3	Workflow diagram of the RNAseq protocol. The blue nodes (T) represent the operations executed on the Transportation system, the green node represents the operation executed on Maholo, the purple nodes represent operations executed on Tecan and the red node represents the operation executed on PCR. The workflow includes 6 cycles of the Tecan-Transportation-PCR-Tecan sequence. Arrows indicate execution order.	33

4.4	Schedule results for Gu2016x1. (a) Branch and Bound algorithm with a makespan of 87 min. (b) GA-SA algorithm with a makespan of 87 min. (c) SAGAS algorithm with a makespan of 87 min.	33
4.5	Schedule results for Gu2016x5. Different colors represent operations belonging to different jobs. (a) Branch and Bound algorithm with a makespan of 297 min. (b) GA-SA algorithm with a makespan of 325 min. (c) SAGAS algorithm with a makespan of 386 min.	34
4.6	Schedule results for qPCRx5. Different colors represent operations belonging to different jobs. (a) Branch and Bound algorithm with a makespan of 148 min. (b) GA-SA algorithm with a makespan of 148 min. (c) SAGAS algorithm with a makespan of 152 min.	35
4.7	Schedule results for RNAseqx5. Different colors represent operations belonging to different jobs. (a) Branch and Bound algorithm with a makespan of 973 min. (b) GA-SA algorithm with a makespan of 1037 min. (c) SAGAS algorithm with a makespan of 1105 min.	36
4.8	Schedule results for qPCR-RNA x5. Different colors represent operations belonging to different jobs. (a) Branch and Bound algorithm with a makespan of 979 min. (b) GA-SA algorithm with a makespan of 1078 min. (c) SAGAS algorithm with a makespan of 1167 min. . .	37
4.9	Performance comparison of scheduling algorithms across different datasets. (a) Shows absolute makespan values in minutes, where Branch and Bound represents the optimal solution. (b) Shows the normalized performance relative to the optimal solution, with percentage values indicating how much each heuristic algorithm exceeds the optimal makespan.	40

List of Tables

4.1	Performance Comparison of Different Scheduling Algorithms	41
-----	---	----

1

Introduction

1.1 Background

In recent decades, automation has received increasing attention in manufacturing and laboratory environments due to its ability to reduce human errors, improve safety, lower operational costs, and improve throughput [1][2]. In complex automated systems, resource allocation and task scheduling are crucial for system efficiency. Research on the Flexible Job Shop Scheduling Problem (FJSP) [3] can effectively enhance the flexibility of resource allocation in automated systems, optimize resource utilization, and can be adjusted according to different objectives.

Traditional FJSP models, however, often face numerous challenges in practical applications due to various complex constraints in real production or experimental environments. Researchers have proposed multiple FJSP extensions to address these constraints: for instance, Wang and Zhu [4] considered setup times required for machines to execute operations and waiting times between job operations; Wei et al. [5] focused on resource consumption impacts; while Ren et al. [6] introduced transportation times and transportation tools required for job execution across different machines. These studies demonstrate that FJSP problems need to be customized according to specific application scenarios to accurately reflect the complex constraints in real-world environments.

1.2 S-LAB problem

In life science domains, experimental processes are subject to particularly stringent time constraints dictated by the biological characteristics of research materials. For example, experiments frequently require the processing of live cells [7], unstable biomolecules (such as RNA or enzymes) [8], and metabolomic samples [9] within strict time frames. Exceeding these time limits can lead to reduction in cellular activity, denaturation, and degradation of molecules, severely affecting the accuracy and consistency of experimental results. The scheduling requirements for these time-sensitive experiments far exceed the capabilities of traditional FJSP models.

To formally represent these complex time constraints, Schäfer [10] proposed the Time Constraints by Mutual Boundaries (TCMB) model. TCMB defines the maximum allowable time difference between the start or end time of one operation and

the boundary of another operation, providing a flexible and powerful framework for modeling strict time constraints. Focusing on the specific requirements in life science domains, Itoh et al. [11] defined the FJSP with TCMB constraints between operations as "Scheduling for Laboratory Automation in Biology" (S-LAB). The S-LAB problem retains the core characteristics of FJSP while incorporating TCMB constraints to meet the strict timing requirements of biological experiments, providing a more suitable mathematical model for laboratory automation.

For the S-LAB problem, Itoh et al. [11] initially proposed a solution based on Mixed Integer Programming (MIP). They modeled the FJSP problem with TCMB time constraints as an MIP model and solved it using a branch-and-bound algorithm. This method successfully found optimal scheduling solutions that minimized total execution time while satisfying TCMB constraints. However, due to the inherent high computational complexity of the MIP method, it is difficult to provide feasible solutions within reasonable time as the size of the problem increases, which limits its application in actual laboratory environments [12].

To address these computational efficiency issues, Arai et al. [12] proposed SAGAS, a hybrid scheduler that combines the Greedy algorithm and Simulated Annealing (SA) as an efficient solution for S-LAB problems. The Greedy algorithm in SAGAS determines feasible time windows and assigns the earliest start times for operations, performing backtracking adjustments when resource conflicts occur. The SA component utilizes metaheuristic techniques to generate initial solutions and implements a temperature-controlled neighborhood search strategy to iteratively identify superior solutions. Finally, SAGAS further optimizes the SA-generated solutions through a Final Modification (Mod) process to form the final scheduling plan. By comparing feasible solutions obtained from both Greedy and SA-Mod approaches and selecting the superior one, SAGAS can find high-quality feasible solutions for large-scale S-LAB problems involving hundreds of operations within reasonable computational time.

Despite its considerable effectiveness in practical applications, the SAGAS algorithm has notable limitations that require improvement. Its sequential job-based initialization strategy processes operations in a fixed order and considers only basic precedence constraints, making it inadequate for complex scenarios involving high resource contention. This constraint potentially hinders the algorithm's ability to discover superior solutions in more intricate scheduling environments.

1.3 Genetic Algorithm for FJSP

When exploring potential avenues for improvement, the Genetic Algorithm (GA) has attracted attention due to its successful applications in FJSP problems. Multiple studies [13][14][4][15] demonstrate that GA possesses excellent global search capabilities and flexibility to handle complex constraints, achieving competitive performance on various FJSP benchmarks. For example, the hybrid GA-TS (Genetic Algorithm-Tabu Search) algorithm proposed by Li and Gao [14] performed exceptionally in solving FJSP problems aimed at minimizing makespan; Wang and Zhu's [4] method

effectively solved the FJSP variant with sequence-dependent setup times and job lag times (FJSP-SDST-LT); while Xie et al.'s [15] hybrid genetic tabu search algorithm was successfully applied to distributed flexible job shop scheduling problems, further proving GA's applicability in solving complex FJSP variants.

However, these GA methods cannot be directly applied to S-LAB problems due to inherent limitations in their encoding and decoding mechanisms. Traditional GA encoding-decoding strategies typically aim to minimize the makespan and tend to assign the earliest feasible start times to operations, but this strategy may lead to violations of TCMB constraints in S-LAB problems. For example, when a TCMB constraint exists between operations A and B, executing operation A too early might cause the time interval between A and B to exceed the TCMB limit, thereby violating the constraint. Therefore, designing suitable GA encoding-decoding mechanisms for S-LAB problems remains an urgent challenge.

1.4 Hybrid GA-SA Strategy

A deeper analysis of GA and SA reveals their complementary nature in managing the fundamental trade-off between exploration and exploitation. In the context of optimization, exploration refers to the process of probing new and diverse regions of the search space to identify potentially promising areas, whereas exploitation focuses on intensifying the search within already discovered regions to refine solutions. Achieving an effective balance between these two opposing strategies is critical to the performance of any metaheuristic algorithm.

GA is effective in global exploration, generating a wide range of high-quality solutions in complex search spaces. However, its limited local search capability and lower exploitation efficiency often hinder the convergence to optimal solutions during the final optimization phase. In contrast, SA excels in exploitation, which is particularly suitable for fine-tuning high-quality solutions. However, SA heavily depends on the quality of initial solutions: poor starting points not only require more iterations but also tend to trap the algorithm in local optima.

Based on this complementarity, a hybrid strategy combining GA's global exploration capabilities with SA's local fine-tuning advantages could become an effective approach for solving S-LAB problems. In this hybrid strategy, GA maintains solution diversity through extensive exploration, effectively avoiding early convergence to local optima and providing quality starting points for SA; SA then efficiently performs local optimization on GA-generated solutions, significantly improving solution quality. This hybrid method effectively compensates for the shortcomings of each algorithm and its effectiveness has been empirically verified in various combinatorial optimization problems [16][17][18][19], offering a promising new direction to solve S-LAB problems.

The GA-SA hybrid algorithm has been successfully applied in multiple domains: Dalanezi et al. [16] applied it to reliability redundancy optimization problems; Jonasson and Norgren [17] used this hybrid strategy to solve university course timetabling problems; Yu et al. [18] employed it for regional location routing problems in waste

collection; while Suanpang et al. [19] demonstrated its advantages in tourism service scheduling in smart cities. These successful applications indicate that the GA-SA hybrid strategy has broad applicability in handling complex constraints and multi-objective optimization problems.

1.5 Proposed Methods

In this work, we propose a hybrid Genetic Algorithm-Simulated Annealing (GASA) approach to solve the S-LAB problem. Our method sequentially integrates GA and SA, where GA generates high-quality solutions, and the best solution is used as the starting point for the SA phase to perform fine-grained local optimization. To effectively represent scheduling solutions, we develop a novel triple encoding scheme that incorporates the operation sequence (OS), machine selection (MS), and operation delay (OD) components, allowing for the comprehensive handling of complex time constraints inherent in laboratory automation scheduling. Additionally, we implement specialized crossover and mutation operators tailored for the characteristics of the S-LAB problem, along with an adaptive local search mechanism. For the SA component, we adapt the approach based on [12], implementing a temperature-controlled neighborhood search strategy to progressively refine the quality of the solution. It concludes with a final modification phase that further optimizes the schedule by eliminating unnecessary idle time across machines and individual job sequences.

Contributions The main contributions of our work are summarized as follows:

1. We proposed a hybrid metaheuristic algorithm that integrates the complementary strengths of GA and SA to balance exploration and exploitation in the search process.
2. We developed a novel triple encoding scheme (OS-MS-OD) that effectively represents scheduling solutions under time constraints, extending traditional FJSP encoding approaches to accommodate TCMB requirements.
3. We carried out a comprehensive empirical evaluation using realistic laboratory protocols that demonstrates the practical applicability and computational advantages of our approach.

1.6 Paper Structure

The remainder of this paper is organized as follows. Section 2 presents the theoretical background about problem formulation, genetic algorithms, and simulated annealing. Section 3 presents our proposed methodology, which consists of three main parts: the GA component (encoding schemes, genetic operators, and local search), the simulated annealing component, and the integration of GA and SA. Section 4 presents the experimental setup including realistic laboratory protocol datasets, implementation details, and performance comparison. Finally, Section 5 concludes the paper with a summary of the findings.

2

Theory

2.1 Problem Formulation

The S-LAB problem can be formulated as a Flexible Job Shop Scheduling Problem with time constraints. [11]

Consider a set of N_j jobs $J = \{J_1, J_2, \dots, J_{N_j}\}$ to be executed on a set of N_m machines $M = \{M_1, M_2, \dots, M_{N_m}\}$. The machines are categorized into N_t different types, where each machine k is of exactly one type T_k , and multiple machines may belong to the same type.

Each job J_i consists of n_i operations, resulting in $N_{op} = \sum_{i=1}^{N_j} n_i$ total operations. The operation O_a is defined as the j -th operation of the job J_i where $a = \sum_{l=1}^{i-1} n_l + j$.

Operations within a job must follow specific dependency relationships. This relationship is formulated as a directed acyclic graph $G = (V, E)$, where each node represents an operation, and each directed edge signifies the dependency relationship. We define a binary variable $pre_{a,b}$ to represent the precedence relationships:

$$pre_{a,b} = \begin{cases} 1, & \text{if } (O_a, O_b) \in E \\ 0, & \text{otherwise} \end{cases} \quad (2.1)$$

Time constraints between specific pairs of operations are a critical aspect of S-LAB problems. These TCMB constraints require that the time gap between specific operation boundaries (start or end times) should not exceed specified limits. A TCMB is represented as a five-element tuple: $(O_a, \text{start/end}, O_b, \text{start/end}, t_{a,b})$. For example, $(O_4, \text{end}, O_7, \text{start}, 10)$ means that the time gap between the completion of Operation 4 and the beginning of Operation 7 cannot exceed 10 time units.

Each operation O_a has a compatible machine type Com_a , indicating that O_a can only be processed by machines of type Com_a . We denote the set of machines compatible with the operation O_a as $M^a \subset M$ to avoid ambiguity with machine indexing.

The processing time of operation O_a is denoted as p_a . This processing time is the same for all compatible machines that can process the operation. The starting time of operation O_a is denoted as s_a .

We use a binary variable $ass_{a,k}$ to represent the assignment of operation to machine:

$$ass_{a,k} = \begin{cases} 1, & \text{if } O_a \text{ is assigned to machine } M_k \text{ where } M_k \in M^a \\ 0, & \text{otherwise} \end{cases} \quad (2.2)$$

We introduce a ternary variable $y_{a,b,k}$ to represent the operation sequencing on machines:

$$y_{a,b,k} = \begin{cases} 1, & \text{if } O_a \text{ and } O_b \text{ are assigned to } M_k \text{ and } O_a \text{ is processed before } O_b \\ 0, & \text{if } O_a \text{ and } O_b \text{ are not assigned to the same machine} \\ -1, & \text{if } O_a \text{ and } O_b \text{ are assigned to } M_k \text{ and } O_b \text{ is processed before } O_a \end{cases} \quad (2.3)$$

2.1.1 Objective and Constraints

The objective is to minimize the total makespan C_{max} , which is the total completion time of all jobs:

$$\min C_{max} \quad (2.4)$$

Subject to the following constraints:

1. Uniqueness of operation assignment:

$$\sum_{k \in M^a} ass_{a,k} = 1, \quad \forall a \in \{1, 2, \dots, N_{op}\} \quad (2.5)$$

2. Operation precedence:

$$(s_b - p_a - s_a) \cdot pre_{a,b} \geq 0, \quad \forall a, b \in \{1, 2, \dots, N_{op}\} \quad (2.6)$$

3. Mutual exclusivity of machine usage:

$$s_a + p_a \leq s_b + H \cdot (1 - ass_{a,k} \cdot ass_{b,k}) + H \cdot (1 - y_{a,b,k}), \quad \forall a, b \in \{1, 2, \dots, N_{op}\}, \forall k \in \{1, 2, \dots, N_m\} \quad (2.7)$$

$$s_b + p_b \leq s_a + H \cdot (1 - ass_{a,k} \cdot ass_{b,k}) + H \cdot (1 + y_{a,b,k}), \quad \forall a, b \in \{1, 2, \dots, N_{op}\}, \forall k \in \{1, 2, \dots, N_m\} \quad (2.8)$$

4. Time constraints:

$$|s_a - s_b| \leq t_{a,b} \quad (\text{start-to-start}) \quad (2.9)$$

$$|s_a - (s_b + p_b)| \leq t_{a,b} \quad (\text{start-to-end}) \quad (2.10)$$

$$|(s_a + p_a) - s_b| \leq t_{a,b} \quad (\text{end-to-start}) \quad (2.11)$$

$$|(s_a + p_a) - (s_b + p_b)| \leq t_{a,b} \quad (\text{end-to-end}) \quad (2.12)$$

5. Makespan definition:

$$C_{max} \geq s_a + p_a \quad \forall a \in \{1, 2, \dots, N_{op}\} \quad (2.13)$$

Constraint 1: ensures that each operation is assigned to exactly one compatible machine.

Constraint 2: enforces precedence relationships between operations as defined by graph G , ensuring proper operation sequencing within jobs.

Constraint 3: prevents resource conflicts by ensuring operations on the same machine do not overlap. The sufficiently large constant H activates these constraints only when operations are assigned to the same machine and in the specific processing order being tested.

Constraint 4: implements the TCMB time constraints between the operation boundaries, covering all possible combinations of "start" and "end" time relationships.

Constraint 5: defines the makespan as the maximum completion time across all operations.

2.1.2 Assumptions

To simplify the problem, we make the following assumptions:

- Once an operation is processed on a compatible machine, it executes until completion without preemption.
- A machine has unlimited buffer capacity for operations that are scheduled on this machine.
- All processing times for the operations are positive.
- All time constraints are positive.
- The processing time of an operation is the same for all compatible machines that can process it.
- The transportation time and setup time between operations are negligible.
- The dependency relationship between operations within a job is acyclic.

2.1.3 Scheduling Solution

A feasible solution to the S-LAB problem consists of a complete assignment of operations to machines and a scheduling of start times for all operations with all constraints satisfied. The optimal solution is the feasible solution with the minimum makespan.

2.2 Solution Approaches

To solve complex optimization problems such as S-LAB, two primary algorithmic paradigms are commonly employed: exact algorithms and heuristic methods. Exact algorithms, such as the branch-and-bound approach used in this thesis for baseline

comparison, guarantee the optimal solution. However, their computational time often grows exponentially with problem size, rendering them impractical for large-scale instances in high-throughput automated laboratory environments, as the extensive execution time creates a planning bottleneck that directly undermines operational throughput.

Heuristic methods offer a practical alternative, aiming to find high-quality solutions within a reasonable time. Among these, local search serves as a fundamental strategy and forms the basis of many advanced techniques. A local search algorithm begins with an initial solution and iteratively explores its neighborhood—defined as the set of solutions reachable through a single, predefined modification, such as swapping two operations. The search proceeds by moving to a better neighbor until no further improvement is possible.

Despite its simplicity and efficiency, this greedy approach tends to get trapped in local optima. A local optimum is a solution that is superior to all of its neighbors but may be significantly inferior to the global optimum. Once the search converges to a local optimum, it is unable to proceed further, potentially leading to a premature termination with a suboptimal result.

To overcome this limitation, metaheuristics have been developed as high-level strategies that guide local search to explore the solution space more broadly. By incorporating mechanisms to escape local optima, metaheuristics enable a more global and diversified search process. This thesis therefore focuses on two well-established metaheuristics: Genetic Algorithm and Simulated Annealing, which are discussed in detail in the following sections.

2.3 Genetic Algorithm

The Genetic Algorithm, introduced by John Holland [20] in 1975, is a heuristic search method inspired by the principle of survival of the fittest and evolutionary processes. GA mimics biological evolution through selection, crossover, and mutation to iteratively evolve a population toward optimal solutions.

2.3.1 Advantages of GA in Solving FJSP

When solving the FJSP, traditional methods fall into three main categories: exact methods (such as the branch-and-bound algorithm), heuristic methods (such as dispatching rules) and other metaheuristic approaches (such as simulated annealing).

Compared to these approaches, GA presents several distinctive advantages in addressing the challenges of FJSP:

- **Global search capability:** By leveraging population-based search and maintaining diversity, GAs can effectively avoid local optima and explore a broader solution space. This makes them particularly well-suited for NP-hard problems like FJSP, which exhibit complex and rugged solution landscapes.

- **Flexible handling of complex constraints:** FJSP entails multiple constraints, including operation precedence and resource allocation. GA can accommodate such constraints through customized encoding schemes and repair mechanisms, often without the need for intricate mathematical formulations.
- **Multi-objective optimization:** In practical manufacturing settings, FJSP often requires simultaneous consideration of multiple objectives, such as minimizing makespan, balancing machine workloads, and reducing total processing time. GA is well-suited for such scenarios, as it can maintain Pareto-optimal solution sets and employ multiobjective evaluation strategies to produce a diverse set of trade-off solutions in a single run.

2.3.2 Encoding Schemes

When applying GA to solve the FJSP, the encoding scheme is a critical component of algorithm design, directly affecting solution representation, the efficiency of genetic operators and decoding complexity. Among various encoding strategies, operation-machine encoding is the most widely adopted for FJSP, as it clearly represents both the operation sequence and the machine assignment using a dual-layer structure. For an instance with N_j jobs and a total of N_m operations, the encoding consists of two components: an operation sequence vector of length N_{op} and a corresponding machine assignment vector of the same length. Alternative encoding schemes include priority-based encoding, decoupled operation-machine representations, and rule-based encoding. Each of these is suited to different algorithmic designs or specific application scenarios.

2.3.3 GA Components for FJSP

2.3.3.1 Initialization

Common population initialization strategies for the FJSP include random generation, rule-based methods such as the Shortest Processing Time (SPT) and Earliest Finish Time (EFT) and other heuristic-driven approaches. In practice, these strategies are often combined, generating part of the population randomly while constructing the remaining individuals based on heuristics. This hybrid strategy helps maintain population diversity while injecting promising individuals to accelerate convergence.

2.3.3.2 Fitness Evaluation

In FJSP, common fitness evaluation metrics include makespan, total tardiness, balance of machine workload, and weighted combinations of multiple objectives. Among these, makespan, total tardiness, and total machine workload are the most frequently used. [21] In multiobjective formulations of FJSP, fitness is typically evaluated using Pareto-based ranking or weighted aggregation methods.

2.3.3.3 Selection Strategies

Selection mechanisms in GA are essential for directing the search process toward high-quality solutions. They comprise two components: parent selection, which determines the individuals that reproduce, and survival selection, which determines those retained in the population.

For parent selection in FJSP, commonly adopted strategies include tournament selection, roulette wheel selection, and rank-based selection. Tournament selection chooses the best individual from a randomly sampled subset, providing controllable selection pressure and efficiency. Roulette wheel selection assigns probabilities proportional to fitness, enabling both exploitation of good solutions and limited exploration. Rank-based selection uses the fitness rankings of individuals to ensure robustness under large fitness variance, reducing the risk of premature convergence.

Survival selection determines the composition of the next generation. Generational replacement replaces the entire population with newly generated offspring, encouraging exploration but potentially discarding good solutions. In contrast, steady-state replacement updates only a few individuals per generation, which supports population stability. Elitism is a widely adopted approach that preserves a fraction of the best individuals, preserving high-quality solutions over generations [22].

In practice, these strategies are often combined—for example, using roulette wheel selection for parent choice and elitism for survival—to balance convergence speed and population diversity, adapted to the characteristics of the specific FJSP instance.

2.3.3.4 Crossover

Different crossover operators exhibit distinct characteristics and are suitable for various contexts in FJSP. Precedence Operation Crossover (POX) preserves the relative order of operations within each job, making it particularly effective in handling precedence constraints. It maintains solution feasibility while generating new operation combinations, and is especially useful in scenarios requiring strict adherence to operation sequences. Job-Based Crossover (JBX) exchanges entire jobs between individuals, thereby preserving job integrity. This reduces the reliance on repair mechanisms and is well-suited for problems characterized by strong inter-job relationships. Two-point crossover is a straightforward method typically applied to the machine assignment. It is easy to implement and works well in scenarios with flexible machine options. Uniform crossover promotes diversity by independently selecting the inheritance source for each gene. It is primarily applied to the machine assignment vector and is effective in contexts requiring high exploratory behavior.

2.3.3.5 Mutation

Mutation operators introduce search diversification in FJSP by perturbing existing solution structures. Swapping mutation randomly exchanges the positions of two operations in the sequence. Since operation precedence must be preserved, repair mechanisms are sometimes required to maintain the solution feasibility. Machine reassignment mutation randomly reassigns eligible machines to operations, directly

modifying the machine assignment vector. It is simple to implement and focuses on improving machine allocation efficiency. In practice, hybrid mutation strategies are often adopted, applying swapping mutation to optimize operation sequencing and machine reassignment to improve resource allocation. This combination enables joint optimization across multiple dimensions of the solution and enhances overall algorithm performance.

2.3.3.6 Termination Criteria

In FJSP optimization, defining appropriate termination criteria is essential to guide the solution process effectively.

Common termination criteria include:

- Reaching a predefined maximum number of iterations, which ensures the algorithm halts within finite steps;
- Hitting a maximum runtime limit, often used in real-time scheduling scenarios;
- Detecting no improvement over several consecutive generations, suggesting that the search has converged;
- Achieving a predefined solution quality threshold, applicable when clear performance targets exist.

In practice, these conditions are often combined to balance computational cost and solution quality.

2.4 Simulated Annealing

Simulated Annealing (SA) is inspired by the physical process of metal annealing. In metallurgy, annealing involves heating a metal and then cooling it slowly, which allows atoms at high temperatures to overcome energy barriers and migrate freely. This enables the system to explore a wider range of states. As the temperature gradually decreases, the system tends to stabilize, becoming more likely to settle into an ordered structure with minimal energy.

In 1983, Kirkpatrick et al. [23] introduced the Metropolis principle into combinatorial optimization, which became the foundation for the simulated annealing algorithm. SA builds upon the mechanism of physical annealing: At high temperatures, it exhibits strong global search capabilities by accepting inferior solutions with a certain probability to escape local optima; as the temperature decreases, the search behavior gradually shifts toward local regions, enabling more refined exploration of the solution space.

2.4.1 Initialization

The initialization phase defines the starting point and the initial search flexibility of the simulated annealing process. The choice of the initial solution has a substantial impact on the performance of the algorithm. Although random initialization is

simple to implement, it often yields solutions that are far from optimal. In contrast, heuristic-based strategies, such as dispatching rules such as Shortest Processing Time (SPT) for FJSP, can provide higher-quality starting points, which may accelerate convergence. However, these methods may also introduce structural bias towards specific regions of the solution space [24]. A hybrid approach that combines heuristic guidance with random perturbations, or the use of multi-start schemes, often strikes a better balance between solution quality and search diversity.

Equally important is the selection of the initial temperature $Temp_{init}$, which controls the likelihood of accepting inferior solutions during the early stages of the search. A sufficiently high $Temp_{init}$ facilitates the broad exploration and helps avoid premature convergence to poor local optima. A common and well-established technique is to determine $Temp_{init}$ based on a desired initial acceptance ratio χ_0 , typically ranging from 0.8 to 0.9. This involves sampling a set of neighbor transitions, identifying those that increase the objective value, and computing T_0 using the formula:

$$Temp_{init} = -\frac{\overline{\Delta f^+}}{\ln(\chi_0)} \quad (2.14)$$

where $\overline{\Delta f^+}$ is the average cost increase among the sampled worsening moves [25]. Adaptive strategies are also available to set $Temp_{init}$, such as incremental adjustment, bisection tuning, or preliminary calibration runs. Regardless of the method, it is essential to ensure that $Temp_{init}$ is high enough to allow exploration, yet not high enough that the search behaves like a random walk.

2.4.2 Neighborhood Search

The neighborhood search mechanism defines how new candidate solutions are derived from the current one and plays a central role in shaping the search trajectory. In scheduling problems such as FJSP, typical neighborhood operations include:

- Resequencing: Changing the order of operations either by swapping two operations or inserting one into a new position.
- Reassignment: Modifying the assignment of an operation to the machine, subject to compatibility constraints.

These operations explore variations in sequencing and resource allocation [26].

Combining multiple neighborhood structures into an integrated strategy can enhance solution diversity and help avoid premature convergence. In addition, the neighborhood move design can evolve during the annealing process, shifting from broad exploratory moves to more localized refinements as the temperature decreases. At higher temperatures, larger and more disruptive moves are favored to encourage exploration, whereas at lower temperatures, the focus shifts toward smaller, fine-tuning modifications. Effective neighborhood design should ensure three properties: connectivity (any feasible solution can be reached), locality (individual changes are not excessively disruptive), and problem-specific alignment (the operations meaningfully reflect the structure of the problem). These principles help maintain a balanced trade-off between exploration and exploitation.

2.4.3 Cooling Schedule

The cooling schedule governs how the temperature parameter $Temp$ decreases over time, transforming the algorithm from a predominantly exploratory phase to one focused on refinement. Among various strategies, geometric cooling is the most widely used and is defined as $Temp_{k+1} = \alpha \cdot Temp_k$, where α is the cooling rate, typically between 0.80 and 0.99. A higher α slows the temperature decay, allowing more extensive exploration, whereas a lower α accelerates convergence [27].

Other cooling schemes include logarithmic cooling $Temp_k \propto 1/\log(k)$, which offers theoretical guarantees of global convergence, but is usually too slow for practical use. Linear cooling, where temperature declines linearly over time, is simple to implement but lacks the adaptability required in complex search landscapes.

To improve adaptability, adaptive cooling schedules dynamically adjust the cooling rate based on search feedback, for example, by decreasing the temperature more rapidly if too many poor solutions are being accepted or slowing the rate when the acceptance ratio drops sharply. In addition, reheat mechanisms or restart strategies can periodically raise the temperature if the algorithm appears stuck in a local optimum, promoting renewed exploration. The choice of cooling schedule significantly affects the trade-off between computational time and solution quality.

2.4.4 Termination Criteria

Termination criteria determine when the simulated annealing process should stop and are crucial to balancing computational effort with solution quality. Common stopping conditions include the following:

- Final temperature: The search terminates when the temperature falls below a predefined threshold $Temp_f$, indicating that exploration is nearly exhausted.
- Maximum iterations or evaluations: The process ends after a fixed number of iterations or performance threshold, providing direct control over runtime.
- Stagnation Detection: The process stops if the best solution shows no improvement for a certain number of iterations, or if the acceptance rate becomes too low (e.g., under 1%), suggesting that further progress is unlikely. [28]

Relying on a single termination criterion can be risky, as it may not reflect the true state of the search. Therefore, compound termination strategies, which trigger the stop condition if any of several criteria are met, are commonly used in practice. These hybrid approaches improve robustness and help balance exploration completeness with computational efficiency.

3

Methods

3.1 Genetic Algorithm

3.1.1 Encoding and decoding

3.1.1.1 Encoding

As traditional encoding schemes using only operation sequence (OS) and machine selection (MS), combined with earliest-start-time decoding strategies [4], primarily focus on minimizing makespan. However, this approach is often insufficient to effectively address the specific constraints of the S-LAB problem, as discussed in 1. To overcome these limitations, we introduce a triple encoding scheme consisting of OS, MS, and operation delay (OD).

The OS encoding represents an ordered sequence of operations, where each integer in the sequence uniquely identifies an operation's global ID. The length of this sequence equals the total number of operations in all jobs. Operations subject to job precedence constraints must adhere to a valid topological order within the OS sequence.

The MS encoding specifies the machine assignment sequence, corresponding directly to the OS encoding and matching its length. Each position in the MS encoding indicates the machine ID assigned to the respective operation. The assigned machine must be compatible with the operation.

The OD encoding maintains a mapping between each operation ID and its associated delay time. The delay denotes the additional postponement of an operation's earliest feasible start time, introducing extra flexibility into the scheduling solution.

Compared to direct encoding start times, the OD component decouples timing adjustments from the OS and MS encoding. This separation simplifies the application of genetic operators, as OD modifications do not require immediate conflict resolution with OS or MS during variation. Theoretically, this combined OS-MS-OD representation, coupled with appropriate decoding, can express the full range of feasible schedules, enabling a more effective search for solutions satisfying the TCMB constraints.

3.1.1.2 Decoding

The decoding process translates the encoded chromosome, comprising OS, MS, and OD, into a uniquely determined schedule solution. As defined in 2.1, a schedule solution consists of the assignment of each operation to a compatible machine and the specific start time for each operation. The detailed implementation of this decoding procedure is described in Algorithm 1.

Algorithm 1 Schedule Decoding with Triple Encoding

```

1: Input: Operation Sequence (OS), Machine Selection (MS), Operation Delay (OD)
2: Output: Schedule with start times  $s_a$  and machine assignments  $ass_{a,k}$ 
3: Initialize idle time periods  $I_k = \{[0, \infty]\}$  for each machine  $M_k \in M$ 
4: Initialize earliest start time  $ES_a = 0$  for each operation  $O_a, a = 1, 2, \dots, N_{op}$ 
5: Initialize start time mapping  $s_a$  and machine assignment mapping  $ass_{a,k}$ 
6: for  $t = 1$  to  $N_{op}$  do
7:   Get operation  $O_a$  from the  $t$ -th position in OS sequence
8:   Get machine  $M_k$  from the  $t$ -th position in MS sequence
9:   Get operation delay  $delay_a$  from the  $t$ -th position in OD sequence
10:  Update earliest start time:  $ES_a \leftarrow ES_a + delay_a$ 
11:  for each idle period  $[s_{idle}, e_{idle}] \in I_k$  do
12:    if  $\max(s_{idle}, ES_a) + p_a \leq e_{idle}$  then
13:      Remove  $[s_{idle}, e_{idle}]$  from  $I_k$ 
14:      Set start time:  $s_a \leftarrow \max(s_{idle}, ES_a)$ 
15:      Set assignment:  $ass_{a,k} \leftarrow 1$ 
16:      Calculate completion time:  $c_a \leftarrow s_a + p_a$ 
17:      if  $ES_a > s_{idle}$  then
18:        Add  $[s_{idle}, ES_a]$  to  $I_k$ 
19:      end if
20:      if  $c_a < e_{idle}$  then
21:        Add  $[c_a, e_{idle}]$  to  $I_k$ 
22:      end if
23:      break
24:    end if
25:  end for
26:  for each operation  $O_b$  where  $pre_{a,b} = 1$  do
27:    Update earliest start time:  $ES_b \leftarrow \max(ES_b, c_a)$ 
28:  end for
29: end for
30: return Schedule with start times  $s_a$  and machine assignments  $ass_{a,k}$ 

```

3.1.2 Initialization

The initial population consists of N_{pop} individuals, each randomly generated to represent a scheduling solution. The OS, MS, and OD components that comprise the individual's chromosome are initialized differently.

The OS component is generated by creating a random permutation of all operation IDs, which is subsequently processed via topological sort to satisfy job precedence constraints. Subsequently, the MS component is constructed by assigning to each operation (in the order specified by OS) a randomly selected machine from its set of compatible machines.

The initialization of the OD component, which maps operations to delays, focuses on the first operation in TCMB-constrained operation pairs. For each operation O_a in TCMB-constrained operation pairs (O_a, O_b) , a nonnegative delay $delay_a$ is assigned by sampling from a log-normal distribution $delay_a \sim \text{LogNormal}(\mu, \sigma^2)$. The parameters μ and σ allow control over the expected magnitude and variance of initial delays, respectively.

3.1.3 Fitness Function

The fitness function evaluates the quality of each chromosome, representing a candidate scheduling solution denoted S . In this study, the fitness function is formulated to minimize two primary objectives: the total completion time (makespan, C_{\max}) and the extent of TCMB constraint violations. It is defined as:

$$f_{GA}(S) = C_{\max}(S) + \sum_{(a,b) \in \mathcal{T}} P(s_b - (s_a + p_a), t_{a,b}) \quad (3.1)$$

Where:

- $C_{\max}(S)$ is the makepan of the schedule S , defined as the completion time of the final operation.
- \mathcal{T} is the set of all operations pairs (a, b) subject to the TCMB constraints.
- s_a and s_b denote the start times of operations a and b , respectively.
- p_a is the processing time of the operation a on its assigned machine.
- $t_{a,b}$ represents the maximum allowable interval between the completion of a and the start of b .
- $P(x, y)$ is the penalty function applied when the actual interval $x = s_b - (s_a + p_a)$ exceeds the upper bound $y = t_{a,b}$.

The penalty function $P(x, y)$ is a piecewise function that combines linear and quadratic terms:

$$P(x, y) = \begin{cases} w_1 \cdot (x - y) + w_2 \cdot (x - y)^2, & \text{if } x > y \\ 0, & \text{otherwise} \end{cases} \quad (3.2)$$

Here, w_1 and w_2 are nonnegative coefficients for the linear and quadratic terms, respectively. This function applies escalating penalties to discourage larger violations, effectively suppressing significant deviations from the TCMB constraints. By integrating makespan minimization with explicit penalization of constraint violations,

the fitness function effectively directs the search toward solutions that achieve a trade-off between scheduling performance and temporal constraint satisfaction.

3.1.4 Genetic Operation

3.1.4.1 Selection

The selection operator determines which individuals in the current population contribute to the next generation, guiding the evolutionary search for higher-quality solutions. We employ a hybrid strategy that combines elitism and roulette wheel selection, as detailed in Section 2.3.3.3. Elitism ensures the preservation of best-performing individuals, while roulette selection helps maintain population diversity, mitigate premature convergence, and achieve a balance between exploration and exploitation.

3.1.4.2 Crossover

The crossover operator generates offspring by combining genetic material from selected parent individuals, thereby facilitating exploration of the solution space. Following the approach in [4], and given the heterogeneous structure of the chromosome, separate crossover mechanisms are applied to the OS and MS components. The OD component, which maps operations to delay values, is excluded from the crossover process; instead, it is inherited or adjusted later by mutation.

For the OS part, two precedence-preserving crossover operators adapted from [4] are applied with equal probability: Precedence Operation Crossover (POX) and Job-Based Crossover (JBX). The procedural details are presented in Algorithm 2 and Algorithm 3, and their execution is illustrated in Figure 3.1.

For the MS part, a standard two-point crossover is used, consistent with [4], as shown in Algorithm 4.

Because crossover is performed independently on OS and MS, a repair step is required to ensure machine compatibility. After crossover, each machine assignment in the offspring's MS string is validated against the corresponding operation in the OS string. Any incompatible assignment is rectified by randomly selecting a valid machine from the compatibility set of the operation, ensuring the feasibility of the offspring solutions.

3.1.4.3 Mutation

Mutation plays a key role in genetic algorithms by introducing diversity into the population through random modifications applied to individuals after crossover. This mechanism helps prevent premature convergence and enhances the exploratory capacity of the algorithm.

For the OS component, two mutation operators are employed with equal probability: the swapping mutation and the insertion mutation. Both operators are followed by a topological sorting procedure to restore precedence feasibility based on the acyclic

Algorithm 2 Precedence Operation Crossover (POX)

- 1: Randomly divide the job set J into two groups J_1 and J_2 such that $J_1 \cup J_2 = J$ and $J_1 \cap J_2 = \emptyset$

Create offspring O_1 :

- 2: Copy operations from P_1 that belong to jobs in J_1 to the same positions in O_1
- 3: Fill the remaining positions in O_1 with operations from P_2 that belong to jobs in J_2 , maintaining their relative order

Create offspring O_2 :

- 4: Copy operations from P_2 that belong to jobs in J_1 to the same positions in O_2
- 5: Fill the remaining positions in O_2 with operations from P_1 that belong to jobs in J_2 , maintaining their relative order

- 6: **return** O_1, O_2
-

Algorithm 3 Job-based Crossover (JBX)

- 1: Randomly divide the job set J into two groups J_1 and J_2 such that $J_1 \cup J_2 = J$ and $J_1 \cap J_2 = \emptyset$

Create offspring O_1 :

- 2: Copy operations from P_1 that belong to jobs in J_1 to the same positions in O_1
- 3: Fill the remaining positions in O_1 with operations from P_2 that belong to jobs in J_2 , maintaining their relative order

Create offspring O_2 :

- 4: Copy operations from P_2 that belong to jobs in J_1 to the same positions in O_2
- 5: Fill the remaining positions in O_2 with operations from P_1 that belong to jobs in J_2 , maintaining their relative order

- 6: **return** O_1, O_2
-

Algorithm 4 Two-Point Crossover for MS

- 1: Randomly select two distinct crossover points p_1, p_2 where $1 \leq p_1 < p_2 \leq L$

Create offspring O_1 and O_2 :

- 2: Initialize O_1 as a copy of P_1 and O_2 as a copy of P_2
- 3: Each position between p_1 and p_2 is swapped between O_1 and O_2

- 4: **return** O_1, O_2
-

graph G defined in Section 2.1. The swapping and insertion mutation operators are detailed in Algorithms 5 and 6, and illustrated in Figure 3.3.

Algorithm 5 OS Swapping Mutation

- 1: Randomly select two distinct positions p_1 and p_2 in the operation sequence
 - Create offspring O :**
 - 2: Initialize O as a copy of parent P
 - 3: Swap the operations at positions p_1 and p_2 in O
 - 4: Perform topological sorting based on acyclic graph G to ensure job precedence feasibility
 - 5: **return** O
-

Algorithm 6 OS Insertion Mutation

- 1: Randomly select two distinct positions p_1 and p_2 in the operation sequence
 - Create offspring O :**
 - 2: Initialize O as a copy of parent P
 - 3: **if** $p_1 < p_2$ **then**
 - 4: Store the operation at position p_1 in O
 - 5: Shift operations at positions $[p_1 + 1, p_2]$ one position to the left in O
 - 6: Insert the stored operation at position p_2 in O
 - 7: **else**
 - 8: Store the operation at position p_1 in O
 - 9: Shift operations at positions $[p_2, p_1 - 1]$ one position to the right in O
 - 10: Insert the stored operation at position p_2 in O
 - 11: **end if**
 - 12: Perform topological sorting based on acyclic graph G to ensure job precedence feasibility
 - 13: **return** O
-

The machine reassignment mutation operator is detailed in Algorithm 7 and illustrated in Figure 3.4. This operator modifies machine assignments for a small proportion of operations while maintaining the operation sequence, allowing the algorithm to explore alternative resource allocations that may lead to improved makespan.

The OD mutation operator examines each TCMB-constrained operation pair (O_a, O_b) . If the pair satisfies the prescribed time limit, the delay of the preceding operation O_a is either preserved or, with a certain probability, reset to zero. In contrast, if the pair violates the constraint, the delay of O_a increases based on the degree of violation. The detailed procedure is presented in Algorithm 8.

3.1.5 Local search for OD

Due to the mapping nature of OD, crossover operators are not applicable, and relying solely on mutation cannot sufficiently explore the OD solution space. To enhance the refinement of delay configurations, we incorporate a local search procedure. This

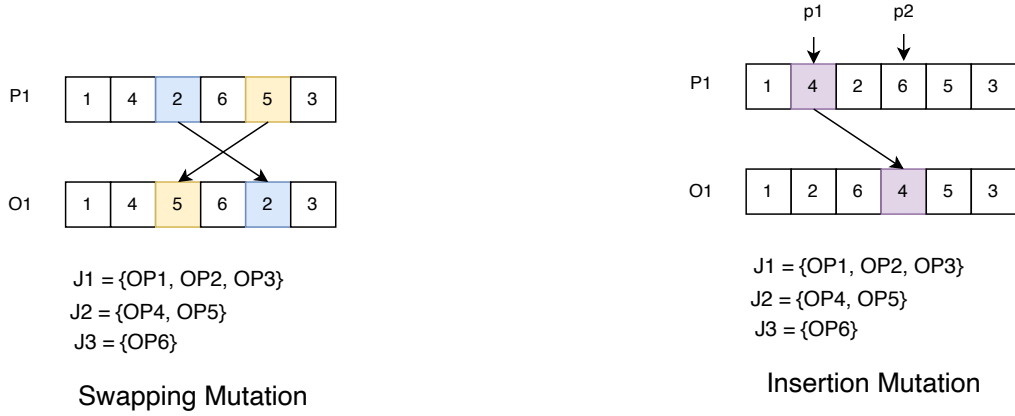


Figure 3.3: Operation Sequence mutation operators: (left) OS Swapping Mutation exchanges the positions of two operations within the operation sequence; (right) OS Insertion Mutation removes an operation from position $p1$ and inserts it at position $p2$.

Algorithm 7 MS Reassignment Mutation

- 1: Set $h = \lfloor 2/N_{op} \rfloor$, where N_{op} is the total number of operations
 - 2: Randomly select h positions in the machine selection chromosome
 - Create offspring O :**
 - 3: Initialize O as a copy of parent P
 - 4: **for** each selected position i in O **do**
 - 5: Identify the corresponding operation at position i
 - 6: Determine the compatible machine set for this operation
 - 7: Randomly select a machine from the compatible set that is different from the current assignment
 - 8: Update the machine assignment at position i in O
 - 9: **end for**
 - 10: **return** O
-

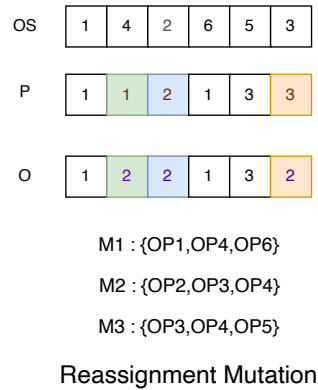


Figure 3.4: Machine Selection mutation: MS Reassignment Mutation changes the machine assignments for selected operations while maintaining operation sequence integrity.

Algorithm 8 OD Mutation

- 1: **for all** TCMB pair $(a, b) \in \mathcal{T}$ **do**
 - 2: Calculate interval $\Delta_{ab} \leftarrow s_b - (s_a + p_a)$.
 - 3: Let $t_{a,b}$ be the max allowed interval for (a, b) .
 - 4: **if** $\Delta_{ab} \leq t_{a,b}$ **then**
 - 5: With probability $P_{restore}$, set $delay_a \leftarrow 0$.
 - 6: **else**
 - 7: Calculate violation by $V_{ab} \leftarrow \Delta_{ab} - t_{a,b}$.
 - 8: Get delay increase by $\delta_a \sim U[0, V_{ab}]$.
 - 9: Update $delay_a \leftarrow delay_a + \delta_a$.
 - 10: **end if**
 - 11: **end for**
-

local search serves as an intensification component, exploring neighboring solutions by iteratively adjusting delay values to improve fitness. The procedure is detailed in Algorithm 9.

Algorithm 9 Local Search for OD

```

1: Input: Initial chromosome  $C_{init}$ , Max iterations  $Iter_{max}$ , Size of tabu list  $N_{tabu}$ 
   Tabu list  $L_{tabu}$ 
2: Output: Best chromosome found during this search  $C_{best\_local}$ 
3: Initialize current solution  $C_{current} \leftarrow C_{init}$ 
4: Initialize best-found solution  $C_{best\_local} \leftarrow C_{current}$  and its fitness  $f_{best\_local}$ 
5: for  $iter = 1$  to  $Iter_{max}$  do
6:    $N_{neighbors} \leftarrow \emptyset$ 
7:   for all TCMB-constrained operation pairs  $(O_a, O_b)$  with time constraint  $t_{a,b}$ 
   do
8:      $\Delta_{ab} \leftarrow s_b - (s_a + p_a)$ 
9:     if  $\Delta_{ab} \geq t_{a,b}$  then
10:      Calculate violation amount  $V_{ab} \leftarrow \Delta_{ab} - t_{a,b}$ 
11:      Determine a delay variation  $\delta_a \sim N(0, V_{ab}^2)$ 
12:      Generate  $C_{candidate}$  by  $delay_a \leftarrow \max(0, delay_a + \delta_a)$ 
13:      Generate a tabu key  $k_{candidate}$  based on the hash of  $C_{candidate}$ .
14:      if  $k_{candidate}$  is not in  $L_{tabu}$  then
15:        if  $|L_{tabu}| = N_{tabu}$  then
16:          remove the first key in  $L_{tabu}$ 
17:        end if
18:        Add  $C_{candidate}$  to  $N_{neighbors}$ 
19:      end if
20:    end if
21:  end for
22:  Select  $C_{best\_neighbor}$  with the best fitness from  $N_{neighbors}$ .
23:  if  $f_{best\_neighbor} < f_{best\_local}$  then
24:    Update  $C_{best\_local} \leftarrow C_{best\_neighbor}$  and  $f_{best\_local} \leftarrow f_{best\_neighbor}$ 
25:    Reset  $N_{no\_improve} \leftarrow 0$ 
26:  end if
27:   $C_{current} \leftarrow C_{best\_neighbor}$ 
28: end for
29: return  $C_{best\_local}$ 

```

3.1.6 Diversification Strategy

To prevent search stagnation and premature convergence, a diversification strategy is triggered when no improvement is detected over a predefined number of generations. This strategy involves replacing a portion of the population with randomly generated individuals. By introducing fresh genetic material, this partial restart enhances population diversity, thereby increasing the likelihood of escaping local optima.

3.2 Simulated Annealing

The SA component in GASA functions as a local refinement phase, initialized with the best solution produced from the preceding GA. The core mechanisms of this SA stage, including the generation of the neighborhood and the final modification, are directly adopted from the implementation in SAGAS [12]. Our contribution in this phase lies primarily in two aspects: (1) integrating the GA-derived solution as the initial state, and (2) applying minor adjustments to the objective function.

3.2.1 Initial Solution

Our SA algorithm accepts the best solution from the genetic algorithm phase as its starting point. This initialization approach differs significantly from SAGAS [12], which employs a sequential job-based strategy where operations are scheduled one after another without parallelization. By leveraging GA-derived solutions, our approach combines the extensive exploration capabilities of genetic algorithms with the intensified refinement offered by simulated annealing.

3.2.2 Objective Function with Penalization

To guide the search process while addressing the complex constraints of the S-LAB problem, we adopt a penalized objective function inspired by [12]:

$$f_{SA}(S) = C_{max}(S) + Time_p \times w_p \quad (3.3)$$

Here, $C_{max}(S)$ denotes the makespan of schedule solution S , and $Time_p$ represents the total duration of constraint violations, computed as:

$$Time_p = Time_{pD} + Time_{pT} + Time_{pM} \quad (3.4)$$

The three components of the violation correspond to the job precedence constraints ($Time_{pD}$), the TCMB constraints ($Time_{pT}$) and the machine capacity constraints ($Time_{pM}$), respectively.

Precedence violations are penalized by the following:

$$Time_{pD} = \sum_{a,b \in \{1, \dots, N_{op}\}} (\max(0, (s_a + p_a - s_b)))^2 \cdot pre_{a,b} \quad (3.5)$$

where $pre_{a,b}$ indicates whether the operation O_a is the predecessor of O_b in the same job, as defined in Equation (2.1).

Machine capacity violations are calculated as:

$$Time_{pM} = \sum_{k=1}^{N_m} \sum_{t \in T_k} \max(0, (n_{k,t} - cap_k)) \cdot \Delta t \quad (3.6)$$

This equation computes the total machine overuse, weighted by the duration of each time interval, across all types of machines. It sums the excess number of concurrent operations beyond capacity during periods of overuse. Here, N_m denotes the total number of machine types, and T_k represents the set of time intervals during which machine type k experiences changes in its usage (i.e., when operations start or end). For each time interval $t \in T_k$, $n_{k,t}$ is the number of simultaneous operations using machine type k , and cap_k is its available capacity. The Δt denotes the duration of the time interval t .

TCMB constraint violations are quantified by:

$$Time_{pT} = \sum_{(a,b) \in \mathcal{T}} \max(0, (s_b - (s_a + p_a) - t_{a,b}))^2 \quad (3.7)$$

This expression applies a penalty whenever the actual time interval between TCMB-constrained operation pairs (O_a, O_b) exceeds the maximum permitted duration $t_{a,b}$.

To prioritize feasible solutions, we assign a large penalty weight w_p . This allows the algorithm to explore infeasible regions during the early stages but strongly favors feasibility as the search progresses.

3.2.3 Temperature Control

Appropriate temperature setting is critical to the effectiveness of the SA algorithm. Following the methodology proposed in SAGAS [12], the initial temperature is set to the maximum start time among all operations in the initial solution. This design aligns the temperature scale with the overall time range of the schedule, ensuring compatibility with different dataset configurations. The final temperature $Temp_f$, serving as the termination criterion, is defined as:

$$Temp_f = \frac{\sum_{a=1}^{N_{op}} p_a}{N_m} \quad (3.8)$$

This value represents a theoretical lower bound for the makespan, assuming a perfectly balanced workload across all machines. To control the temperature reduction from $Temp_{init}$ to $Temp_f$, the geometric cooling schedule described in Section 2.4.3 is adopted. It is widely used for its simplicity and for providing a predictable exponential decay, which facilitates gradual stabilization of the search process.

3.2.4 Neighborhood Generation

Our SA implementation adopts the neighborhood generation approach from SAGAS [12], which is thoroughly described in their Algorithm 1.

For operations not subject to preceding job precedence or TCMB constraints, the algorithm randomly samples a time offset constrained by the current temperature. This offset is then applied to increase or decrease the operation start time with equal probability (50%).

For operations subject to preceding job precedence or TCMB constraints, the algorithm randomly selects a new start time within the feasible range $[Floor_a, Ceil_a]$ for operation O_a . The lower bound $Floor_a$ represents the earliest possible start time, determined as the maximum of two values: the completion times of all predecessor operations in the job sequence, and any lower limit imposed by relevant TCMB constraints. In contrast, the upper bound $Ceil_a$ denotes the latest feasible start time, calculated as the minimum of two values: (1) the upper limits imposed by the constraints of the TCMB, and (2) the start times of all successor operations in the job sequence minus the processing time of O_a .

3.2.5 Solution Acceptance

We use the classical Metropolis principle to decide whether a new solution is accepted:

$$p(S_{\text{current}}, S_{\text{next}}) = \min \left(1, \exp \left(\frac{\Delta f}{Temp} \right) \right) \quad (3.9)$$

where $\Delta f = f_{SA}(S_{\text{current}}) - f_{SA}(S_{\text{next}})$, and $Temp$ is the current temperature. This allows worse solutions to be accepted with diminishing probability, enabling escape from local optima.

3.2.6 Final modification

Following the refinement procedure proposed by Arai et al. [12], we apply a three-phase final modification to enhance the quality of the best solution. Full details of this procedure are presented in their Algorithm 2. This process comprises:

- Global time compression, which eliminates unnecessary idle time across all machines;
- Job-level compression, aimed at reducing idle time within individual job sequences;
- Fine-grained local adjustments to further improve scheduling efficiency at the operation level.

This final refinement step improves the makespan while preserving all feasibility constraints.

3.3 Proposed GASA

GA is effective in global exploration and maintaining population diversity, but it tends to be inefficient in refining solutions during the later stages of the search. In contrast, SA excels at local exploitation, making it particularly suitable for refining high-quality solutions; however, its performance is highly dependent on the quality of initial solutions. To address their respective limitations, the proposed hybrid approach, GASA, integrates GA and SA in a sequential manner, thereby leveraging

their complementary strengths. The overall framework of GASA is illustrated in Algorithm 10.

Algorithm 10 Framework of GASA

- 1: **Step 1: Initialization of GA:** Create an initial population of individuals randomly. Set the generation counter $Iter_{GA} := 1$. Set the counter for no improvement $No_Improve := 0$.
 - 2: **Step 2: Evaluation:** Compute the fitness $f(C)$ of each chromosome C and identify the best chromosome as C_{best} .
 - 3: **Step 3: Termination check:**
 - 4: **if** $Iter_{GA} > Max_Generation$ **then**
 - 5: go to Step 11
 - 6: **else**
 - 7: continue to Step 4
 - 8: **end if**
 - 9: **Step 4: Selection:** Select individuals by elitism and roulette wheel to form parents population for crossover and mutation.
 - 10: **Step 5: Crossover:** Perform POX and JBX on OS and two-point crossover on MS.
 - 11: **Step 6: Mutation:** Perform swapping mutation and insertion mutation on OS, machine reassignment mutation on MS and OD mutation.
 - 12: **Step 7: Local Search:** Select a subset of individuals by tournament selection and apply the local search procedure.
 - 13: **Step 8: Improvement check:** Find the best chromosome in the current population as $Chrom_current$.
 - 14: **if** $f_{GA}(Chrom_current) < f_{GA}(Chrom_best)$ **then**
 - 15: set $No_improve := 0$ and set $Chrom_best := Chrom_current$.
 - 16: **else**
 - 17: Increment $No_improve := No_improve + 1$.
 - 18: **end if**
 - 19: **Step 9: Diversification:**
 - 20: **if** $No_Improve > No_Improve_Limit$ **then**
 - 21: replace a fixed portion of individuals with randomly generated individuals.
 - 22: Set $No_improve := 0$.
 - 23: **end if**
 - 24: **Step 10: Iteration counter update:** Increment $Iter_{GA} := Iter_{GA} + 1$. Go to Step 3.
 - 25: **Step 11: GA Result:** Transform the best chromosome into the best scheduling solution of GA.
 - 26: **Step 12: Initialization of SA:** Take the best scheduling solution from GA to initialize current solution $S_{current}$. Compute and set current temperature to initial temperature $Temp := Temp_{init}$. Set iteration counter $Iter_{SA} := 1$. Initialize best solution $S_{best} := S_{current}$.
 - 27: **Step 13: Termination check:**
 - 28: **if** $Temp < Temp_f$ or $Iter_{SA} > Iter_{MAX}$ **then**
 - 29: go to Step 18
 - 30: **else**
 - 31: continue to Step 14.
 - 32: **end if**
 - 33: **Step 14: Neighborhood Generation:** Generate neighborhood solution $S_{neighbor}$ by adjusting the starting time of operations.
 - 34: **Step 15: Solution Acceptance:**
 - 35: **if** $S_{neighbor}$ is accepted by Metropolis principle **then**
 - 36: set $S_{current} := S_{neighbor}$
 - 37: **end if**
 - 38: **if** $f_{SA}(S_{current}) < f_S(S_{best})$ **then**
 - 39: set $S_{best} := S_{current}$.
 - 40: **end if**
 - 41: **Step 16: Cooling Schedule:** Decrease current temperature by geometric cooling $Temp := Temp * \alpha$
 - 42: **Step 17: Iteration counter update:** Set $Iter_{SA} := Iter_{SA} + 1$. Go to Step 13.
 - 43: **Step 18: Final modification:** Apply final modification to the best solution of SA.
 - 44: **Step 19: Output:** Return the best solution for GASA.
-

4

Results

4.1 Dataset Description

Our benchmarks use protocols developed by Arai et al. [12], which represent realistic laboratory workflows derived from actual experimental configurations. The datasets can be access via: https://github.com/bioinfo-tsukuba/SAGAS_paper. These include:

1. **Gu2016 Protocol**[29]: Based on laboratory automation systems, this protocol consists of 17 operations that utilize 5 types of instruments (Biomek2000.A3, Biomek2000, Reformatter, Motoman, and Transportation systems).The workflow of this protocol is illustrated in Figure 4.1. The dataset includes two variants: Gu2016 \times 1 (a single job) and Gu2016 \times 5 (five concurrent jobs). The TCMB time constraints are set to 10 min.
2. **qPCR Protocol**[30]: Used for the diagnosis of COVID-19 through quantitative reverse transcription-PCR. This protocol comprises 16 operations performed on six types of instruments (three variants of Tecan, plate centrifuge, RT-qPCR, and transportation systems). The workflow of this protocol is illustrated in Figure 4.2. The dataset includes qPCR \times 5 (five concurrent qPCR jobs). The time constraints of the TCMBs are set to 5 min.
3. **RNAseq Protocol** [31]: Designed for RNA sequencing with robotic assistance. This workflow contains 28 operations executed on 4 types of instruments (Maholo, Tecan, PCR, and Transportation systems). The workflow of this protocol is illustrated in Figure 4.3. The dataset includes RNAseq \times 5 configurations. The time constraints of the TCMBs are set to 5 min.
4. **Composite Protocol**: The dataset also includes a mixed instance combining qPCR \times 5 and RNAseq \times 5 jobs simultaneously, representing heterogeneous workloads in laboratory environments. The time constraints of the TCMBs are set to 5 min.

Each protocol defines the dependencies between operations and specifies compatible instruments, creating realistic scheduling challenges with varying complexity levels. Multiple job instances (e.g. \times 5, \times 10) represent scenarios where several identical experimental workflows must be scheduled concurrently, significantly increasing scheduling complexity while maintaining realistic laboratory constraints.

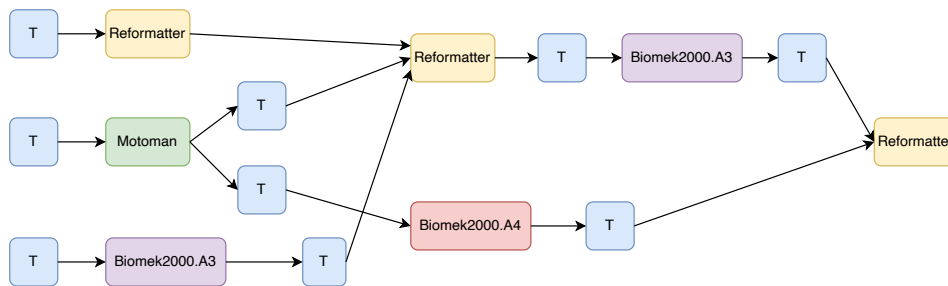


Figure 4.1: Workflow diagram of the Gu2016 protocol. The blue nodes (T) represent operations executed on the Transportation system, the yellow nodes represent operations executed on Reformatter, green node represents operations executed on Motoman, purple nodes represent operations executed on Biomek2000.A3, and red node represents operation executed on Biomek2000.A4. Arrows indicate execution order.

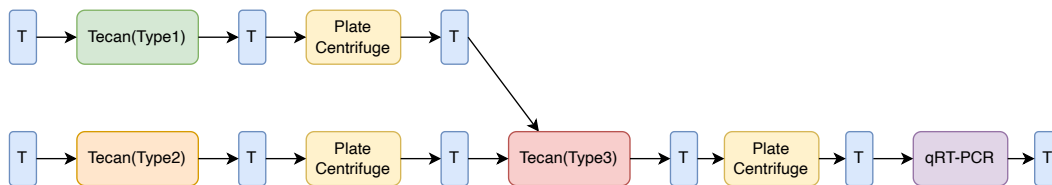


Figure 4.2: Workflow diagram of the qPCR protocol. The blue nodes (T) represent operations executed on the Transportation system, the green node represents operation executed on Tecan(Type1), the yellow nodes represent operations executed on Plate Centrifuge, the orange node represents operation executed on Tecan(Type2), the red node represents operation executed on Tecan(Type3), and the purple node represents operation executed on qRT-PCR. Arrows indicate execution order.

4.2 Implementation and Evaluation

To evaluate the performance of our proposed GASA algorithm, we compared it against two established baselines: Branch and Bound (B&B) [11] and SAGAS [12]. Experiments were carried out on the dataset described in Section 4.1, with performance evaluated in terms of makespan and execution time. Makespan, defined as the total time from the start of the first task to the completion of the last, reflects scheduling efficiency, and a lower value indicates a better schedule. Execution time denotes the computational time required to obtain the final solution.

The B&B algorithm was implemented based on the mathematical model proposed by Itoh et al. [11] for the S-LAB problem. The implementation was developed in Java 17.0.5 using the CBC (COIN-OR Branch and Cut) MIP solver from the OR-Tools library [32]. The SAGAS algorithm was implemented in C++17 following the approach described by Arai et al. [12], with the code publicly available at <https://github.com/bioinfo-tsukuba/SAGAS>. Our GASA algorithm was implemented in Java 17.0.5, following the methodology detailed in the Methods section. The SA component of our algorithm is based on the SAGAS framework.

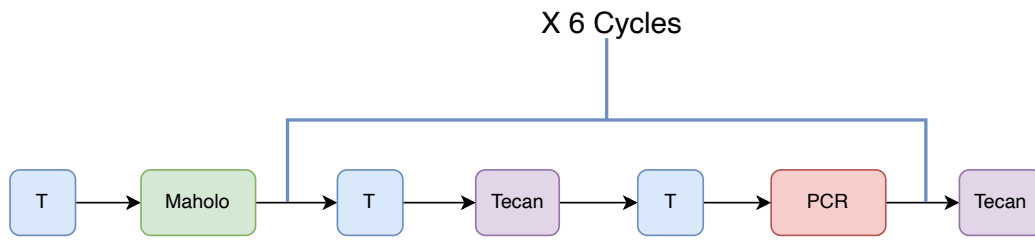


Figure 4.3: Workflow diagram of the RNAseq protocol. The blue nodes (T) represent the operations executed on the Transportation system, the green node represents the operation executed on Maholo, the purple nodes represent operations executed on Tecan and the red node represents the operation executed on PCR. The workflow includes 6 cycles of the Tecan-Transportation-PCR-Tecan sequence. Arrows indicate execution order.

All experiments were carried out on a MacBook Pro 18,3 equipped with an Apple M1 Pro (8-core CPU), 16 GB RAM, and running macOS 14.4.1.

4.3 Experimental Results

For the $\text{Gu2016} \times 1$ dataset, our proposed GASA algorithm achieved a makespan of 87 minutes, the same as the known optimum—as shown in Figure 4.4. The SAGAS algorithm also reached the optimal value. This result confirms that GASA is capable of identifying optimal solutions on this benchmark instance.

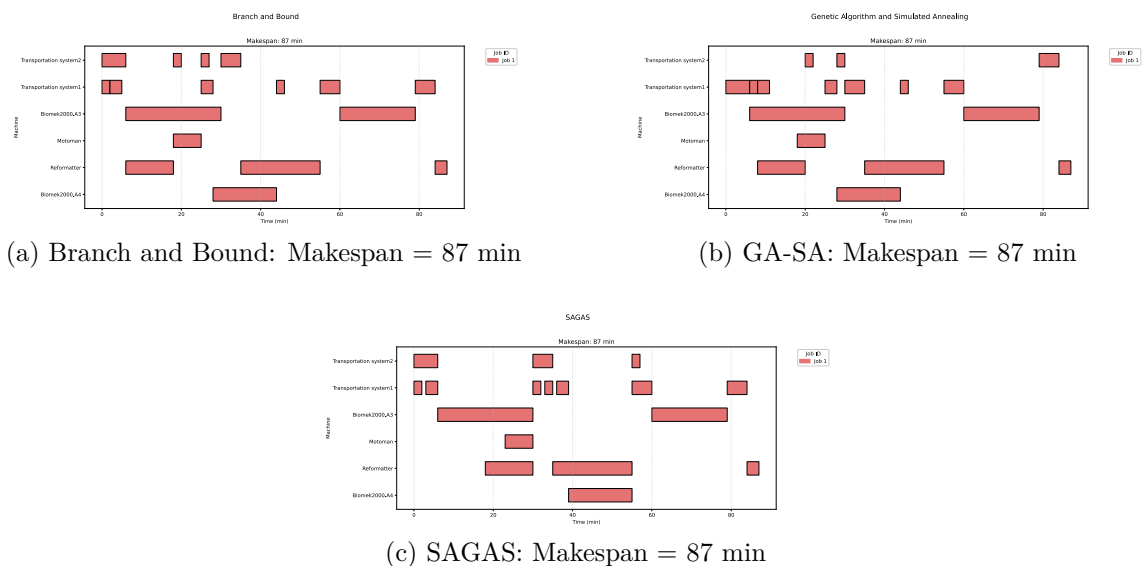


Figure 4.4: Schedule results for $\text{Gu2016} \times 1$. (a) Branch and Bound algorithm with a makespan of 87 min. (b) GA-SA algorithm with a makespan of 87 min. (c) SAGAS algorithm with a makespan of 87 min.

For the $\text{Gu2016} \times 5$ dataset, our experimental results (Figure 4.5) compare the per-

4. Results

formance of the three algorithms in terms of makespan in a scenario involving five concurrent jobs. The B&B algorithm achieved the optimal makespan of 297 minutes, while our proposed GASA algorithm resulted in a makespan of 325 minutes, representing a 9.4% increase over the optimal. The SAGAS algorithm yielded the longest makespan of 386 minutes. Despite not reaching the optimal, GASA’s close proximity to the B&B solution suggests reliable performance, particularly in this complex scheduling environment with high resource contention.

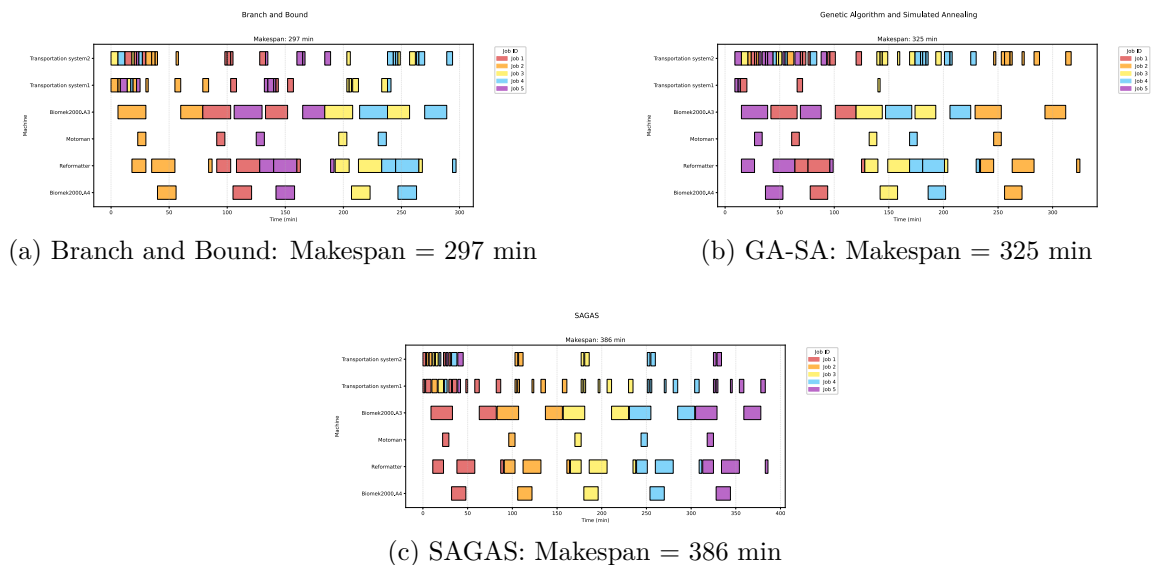
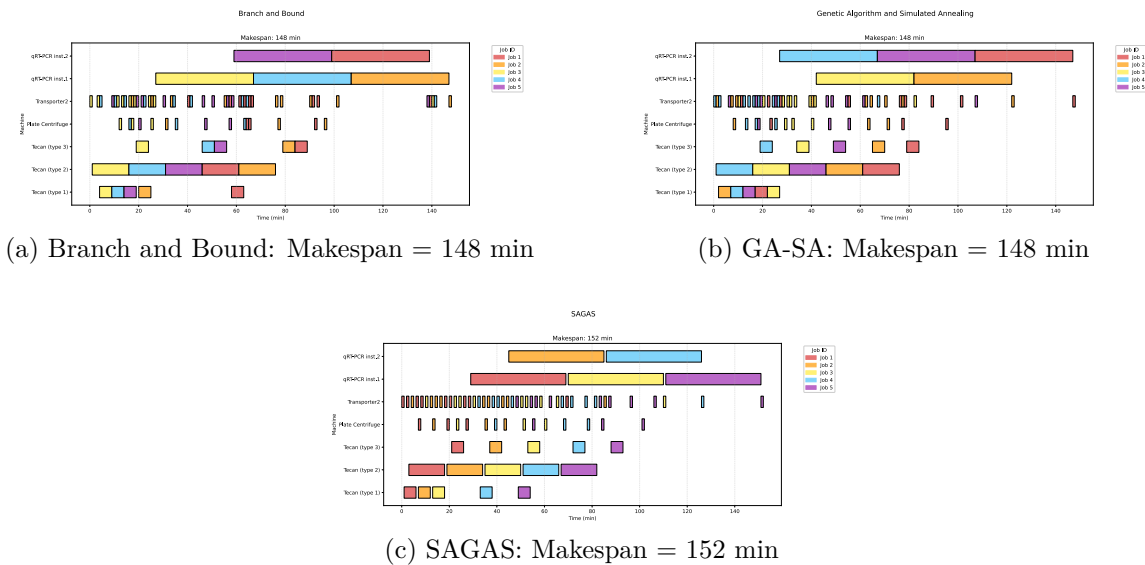


Figure 4.5: Schedule results for Gu2016x5. Different colors represent operations belonging to different jobs. (a) Branch and Bound algorithm with a makespan of 297 min. (b) GA-SA algorithm with a makespan of 325 min. (c) SAGAS algorithm with a makespan of 386 min.

For the qPCR \times 5 dataset, our experimental results (Figure 4.6) present the scheduling performance of the three algorithms in a scenario of the COVID-19 diagnostic protocol with five concurrent jobs. Both the B&B algorithm and our proposed GASA algorithm achieved the same optimal makespan of 148 minutes, while the SAGAS algorithm produced a slightly longer makespan of 152 minutes, representing only a 2.7% increase over the optimal. The identical performance between GASA and B&B on this dataset demonstrates that our algorithm can achieve optimal solutions for this specific qPCR protocol, despite its strict time constraints (5-minute TCMBs) and the resource competition among five concurrent diagnostic jobs. The small performance gap among the three algorithms suggests that the qPCR \times 5 dataset probably has limited scheduling flexibility due to its workflow structure. These inherent sequence dependencies in the diagnostic protocol appear to restrict possible scheduling variations, explaining why all three approaches converged to similar patterns.



(a) Branch and Bound: Makespan = 148 min

(b) GA-SA: Makespan = 148 min

(c) SAGAS: Makespan = 152 min

Figure 4.6: Schedule results for qPCR \times 5. Different colors represent operations belonging to different jobs. (a) Branch and Bound algorithm with a makespan of 148 min. (b) GA-SA algorithm with a makespan of 148 min. (c) SAGAS algorithm with a makespan of 152 min.

For the RNAseq \times 5 dataset, our experimental results (Figure 4.7) show that the B&B algorithm achieved a makespan of 973 minutes, while our proposed GASA algorithm resulted in a makespan of 1037 minutes (6.6% longer than B&B). In comparison, the SAGAS algorithm yielded the longest makespan of 1105 minutes.

The relatively small gap between GASA and B&B demonstrates the competitive performance of our algorithm for this complex RNAseq protocol, which involves 28 operations per job across multiple instrument types. Compared to previous protocols, this dataset presents a considerably more complex scheduling challenge, characterized by a larger number of operations and longer execution durations. Despite the increased complexity, GASA sustained competitive performance, with a makespan 6.6% longer than B&B’s but 6.2% shorter than SAGAS’s, demonstrating robust adaptability under challenging conditions.

Notably, the SAGAS Gantt chart shows that machines PCR_6 to PCR_10 were not utilized, suggesting inefficient resource allocation. This inefficient resource utilization appears to have contributed to SAGAS’s extended makespan. In contrast, both B&B and GASA made more effective use of all available PCR machines, enabling better parallelization and resulting in shorter makespan. These findings underscore the importance of comprehensive resource utilization in laboratory automation scheduling and illustrate GASA’s superior ability to identify efficient allocation patterns over SAGAS.

4. Results

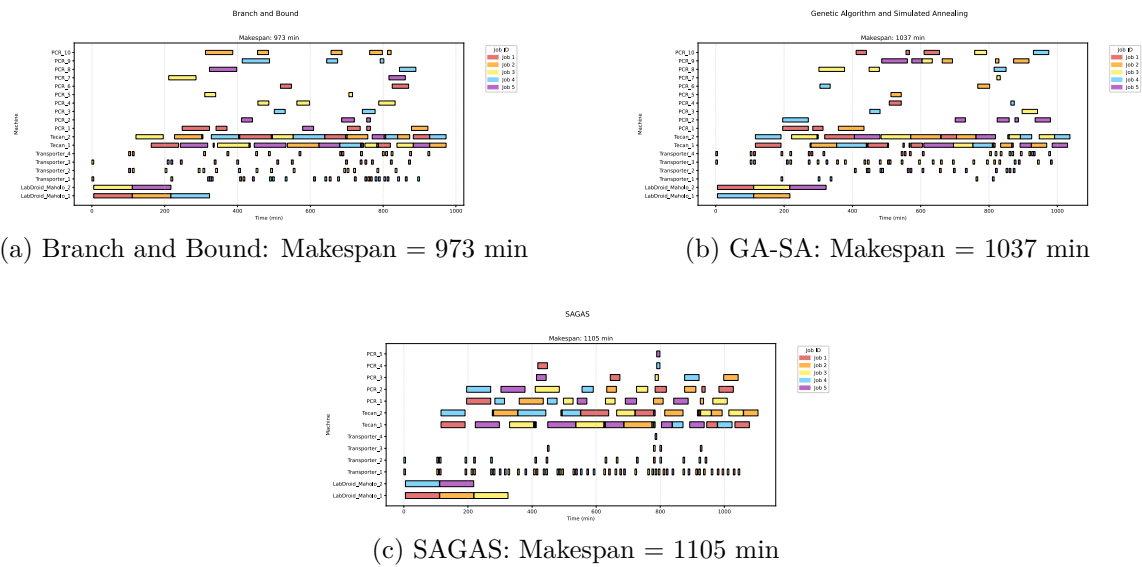


Figure 4.7: Schedule results for RNaseq5. Different colors represent operations belonging to different jobs. (a) Branch and Bound algorithm with a makespan of 973 min. (b) GA-SA algorithm with a makespan of 1037 min. (c) SAGAS algorithm with a makespan of 1105 min.

For the composite protocol dataset, which integrates five qPCR jobs and five RNAseq jobs, our experiments (Figure 4.8) reveal notable performance differences among the three algorithms. B&B achieved the best makespan of 979 minutes, followed by our GASA algorithm at 1078 minutes (10.1% longer), and SAGAS at 1167 minutes, with the performance gap between GASA and B&B widening slightly in this heterogeneous workload scenario.

This outcome is not surprising, as the composite dataset is the most challenging test case. It involves coordinating ten jobs, each comprising multiple sequential operations, all competing for shared laboratory resources. Despite the increased complexity, GASA still outperformed SAGAS by 7.6%, demonstrating its robustness and efficiency under intensive scheduling demands. These results further highlight the value of the hybrid approach of GASA in addressing complex and resource-constrained laboratory scheduling problems.

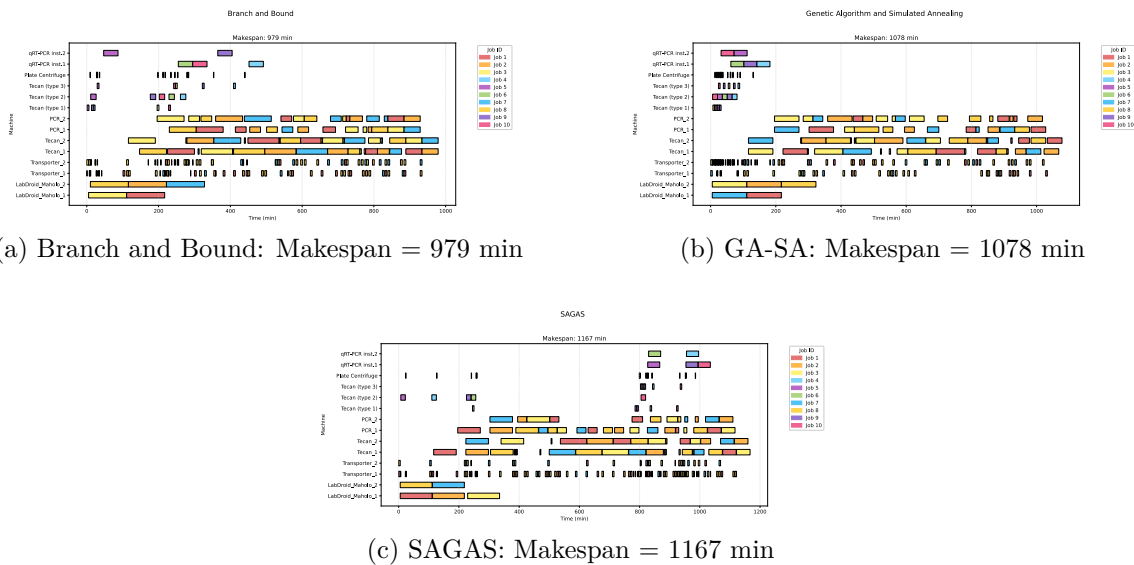


Figure 4.8: Schedule results for qPCR-RNA x5. Different colors represent operations belonging to different jobs. (a) Branch and Bound algorithm with a makespan of 979 min. (b) GA-SA algorithm with a makespan of 1078 min. (c) SAGAS algorithm with a makespan of 1167 min.

4.4 Performance Comparison and Analysis

The experimental results (Figure 4.9, Table 4.1) reveal notable performance differences among the three algorithms in various datasets. This section provides a detailed comparison in terms of makespan performance, computational efficiency, and practical applicability.

4.4.1 Makespan Performance

4.4.1.1 Optimality

The Branch and Bound (B&B) algorithm consistently achieved optimal solutions across all test cases, serving as the reference benchmark. Our proposed GASA algorithm also achieved the optimal makespan in simpler scenarios (e.g., Gu2016 \times 1, with a makespan of 87 minutes), while showing competitive performance in more complex settings. For example, on the composite qPCR-RNAseq \times 5 dataset, B&B achieved the optimal makespan of 979 min, compared to 1078 min for GASA (+10.1%) and 1167 min for SAGAS (+19.2%). This performance gap, as illustrated in Figure 4.9, may be partially explained by differences in initialization strategies. GASA benefits from high-quality initial solutions generated by genetic algorithms, which serve as effective starting points for simulated annealing. In contrast, SAGAS employs a sequential job-based initialization that processes operations in a fixed order with only basic precedence constraints, which can be insufficient in complex scenarios with high resource contention.

4.4.1.2 Stability

B&B, as a deterministic method, consistently produces identical results for each instance. In contrast, GASA shows moderate variability due to its stochastic nature. In Gu2016 \times 5, for example, the average makespan (341.4 minutes) was about 5% higher than the best value (325 minutes), indicating that there is room for further improvement. SAGAS exhibited greater variability; on qPCR-RNAseq \times 5, the average exceeded the best by 6.7%. In particular, on Gu2016 \times 5, both the best and average makespan were 386 minutes (Table 4.1), as SA-Mod consistently underperformed compared to Greedy initialization, leading SAGAS to always fall back to the Greedy result. Although fast, the Greedy strategy explores a narrow region of the solution space and lacks the ability to generate diverse scheduling combinations, resulting in up to 30% deviation from the optimum in complex scenarios.

4.4.1.3 Impact of Problem Structure

All algorithms performed similarly on the qPCR \times 5 dataset, probably due to its rigid workflow structure and strict temporal constraints, which significantly restricted scheduling flexibility and reduced the performance gap between methods. In contrast, RNAseq \times 5 and the composite datasets provided greater scheduling flexibility, making it easier to distinguish the relative performance of different algorithms. These observations suggest that the structural characteristics of the problem, rather than its size alone, have a major influence on the achievable performance and relative effectiveness of different approaches.

4.4.2 Computational Efficiency

As shown in Table 4.1, B&B's runtime increased exponentially with problem size—from 1.20 seconds on Gu2016 \times 1 to 2.95 hours on qPCR-RNAseq \times 5—which may be impractical for time-sensitive applications. In contrast, GASA and SAGAS completed the same task in 4.96 and 3.55 minutes, respectively.

From Gu2016 \times 1 to Gu2016 \times 5, B&B's runtime increased 606-fold (1.20 seconds to 12.13 minutes), while GASA increased only 19 times (1.93 seconds to 37.51 seconds), demonstrating much better scalability. In the most complex dataset (qPCR-RNAseq \times 5), GASA reduced the execution time by more than 96% at the cost of approximately 10% makespan degradation.

Furthermore, in qPCR \times 5, GASA achieved the same optimal makespan as B&B (148 minutes) while requiring only 5% of the runtime. On RNAseq \times 5, GASA's makespan was 6.6% longer than the optimum but was computed 95.8% faster. These results confirm the advantage of GASA in achieving a favorable balance between solution quality and execution efficiency.

4.4.3 Practical Implications

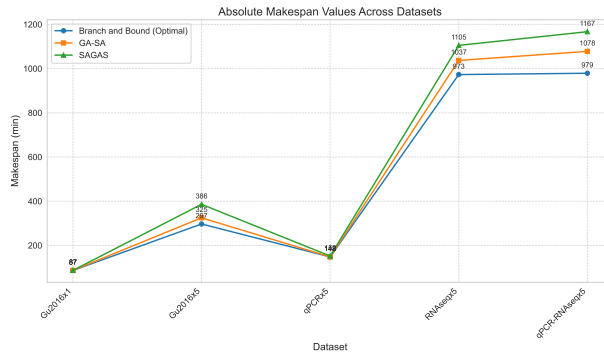
To provide a more integrated evaluation, this section introduces total time as a composite metric, which is defined as the sum of an algorithm's average execution time

and average makespan. This measure reflects the complete duration of an experimental workflow and can serve as a practical indicator of overall system throughput. Analysis of the data in Table 4.1 indicates that the relative performance of each algorithm is strongly influenced by the structural characteristics of the problem, rather than by problem size alone.

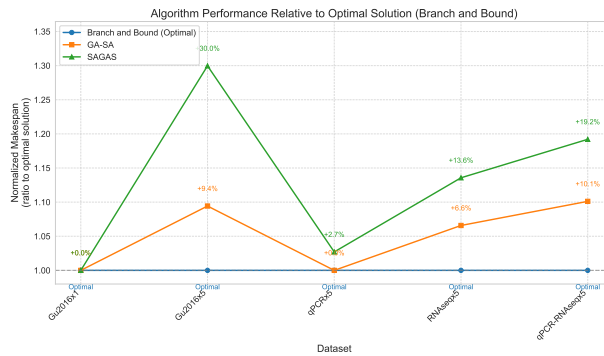
Compared to the heuristic method SAGAS, GASA consistently achieves higher overall throughput. Although GASA incurs a moderate increase in execution time, its substantial reductions in makespan more than offset this cost, particularly in complex problem instances. As shown in 4.1, GASA reduces total time by approximately 44, 50, and 119 minutes on the Gu2016 \times 5, RNAseq \times 5, and qPCR-RNAseq \times 5 instances, respectively. These results quantitatively demonstrate the effectiveness of GASAs search strategy in improving throughput in realistic laboratory workflows.

The comparison with the B&B method reveals a more complex relationship. On problem instances with high structural complexity but moderate scale, such as Gu2016 \times 5, the potential for makespan optimization is substantial. In this instance, the advantage GASA gains from its faster execution time is insufficient to offset the significantly better makespan achieved by B&B, resulting in a longer total time for GASA. However, GASA demonstrates its superiority in two other distinct types of scenarios. The first involves problems with limited makespan optimization potential, exemplified by qPCR \times 5. In this instance, the relatively narrow makespan gap among all methods means that shorter execution time of GASA becomes the decisive factor, resulting in a better total time than the computationally expensive B&B. The second scenario where GASA excels is on the larger-scale qPCR-RNAseq \times 5 instance. For this complex and large problem, B&B’s execution time extends into the range of hours, while GASA’s remains at the level of minutes. This vast difference in computational cost more than compensates for the makespan gap, ultimately enabling GASA to achieve the shortest total time and thereby the highest practical throughput.

In conclusion, this analysis provides a nuanced understanding of each algorithm’s strengths based on the tested scenarios. B&B proves to be the superior choice for maximizing throughput on problems that are structurally complex but moderate in scale, such as Gu2016 \times 5. The primary value of our proposed GASA algorithm lies in its robustness. This is reflected in its ability to reliably deliver a total time that ranks either first or very close to the best across all tested scenarios. It also proves most effective for two distinct types of challenges observed in our study: problems with limited optimization potential due to a rigid structure, as seen in qPCR \times 5, and problems where the computational cost of finding the true optimum becomes prohibitive due to large scale, as demonstrated by qPCR-RNAseq \times 5. This establishes GASA as a highly effective and scalable heuristic, and its consistent performance across the varied problem structures tested suggests its suitability for a broader range of complex scheduling tasks in modern laboratory automation.



(a) Absolute makespan values



(b) Normalized performance relative to optimal solution

Figure 4.9: Performance comparison of scheduling algorithms across different datasets. (a) Shows absolute makespan values in minutes, where Branch and Bound represents the optimal solution. (b) Shows the normalized performance relative to the optimal solution, with percentage values indicating how much each heuristic algorithm exceeds the optimal makespan.

Dataset	Algorithm	Minimum makespan (min)	Average makespan (min)	Average execution time	Total time (min)
Gu2016×1	B&B	87	87.0	1.20 s	87.02
	SAGAS	87	87.6	1.52 s	87.03
	GASA	87	87.0	1.93 s	87.03
Gu2016×5	B&B	297	297.0	12.13 min	309.13
	SAGAS	386	386.0	26.22 s	386.44
	GASA	325	341.4	37.51 s	325.63
qPCR×5	B&B	148	148.0	33.02 min	181.02
	SAGAS	152	152.0	60.44 s	153.01
	GASA	148	149.6	97.45 s	149.62
RNAseq×5	B&B	973	973.0	1.18 h	1043.80
	SAGAS	1105	1131.4	95.63 s	1106.59
	GASA	1037	1080.2	2.82 min	1039.82
qPCR-RNAseq×5	B&B	979	979.0	2.95 h	1156.00
	SAGAS	1167	1244.8	3.55 min	1170.55
	GASA	1078	1124.4	4.96 min	1082.96

Table 4.1: Performance Comparison of Different Scheduling Algorithms

5

Conclusion and Future Work

5.1 Conclusion

In this work, we addressed the S-LAB problem, a complex variant of the FJSP characterized by strict time constraints. Our primary contribution is the development of a hybrid metaheuristic algorithm, GASA, which sequentially integrates a Genetic Algorithm with Simulated Annealing to effectively balance global exploration and local exploitation. Central to our GASA approach is a novel triple-encoding scheme (OS-MS-OD), developed specifically to address the complex TCMB constraint. This representation enables the Genetic Algorithm to conduct an effective global search, producing high-quality initial solutions that are subsequently refined through Simulated Annealing's local exploitation phase, effectively combining the complementary strengths of both metaheuristics.

Our experimental evaluation on realistic laboratory datasets demonstrated GASA's effectiveness. The algorithm consistently produced optimal or near-optimal makespans, significantly outperforming the SAGAS heuristic while drastically reducing computational time compared to the exact B&B method. For instance, on the most complex dataset, GASA achieved a solution within 10.1% of the optimum while requiring over 96% less execution time. This balance between solution quality and speed is GASA's primary advantage. Analysis of total time, a practical measure of system throughput, revealed that GASA's computational efficiency is crucial, particularly on larger-scale problems where the cost of finding a true optimum becomes a bottleneck. Ultimately, GASA proves to be a robust, scalable, and highly practical approach for scheduling complex, resource-constrained workflows in modern automated laboratories.

5.2 Future Work

While this study provides valuable insights, it also opens several avenues for future research that could further enhance the practical applicability of scheduling solutions in laboratory automation.

First, our current model assumes that transportation times and machine setup times are negligible. In many real-world settings, these factors can be significant and, importantly, sequence-dependent, adding another layer of complexity. Future work

could extend the S-LAB model to incorporate these non-negligible time components, which would lead to more realistic and accurate scheduling solutions.

Second, this research focused on a single objective: minimizing makespan. However, laboratory operations often involve multiple, sometimes conflicting, goals. A valuable extension would be to develop a multi-objective optimization framework. This could include objectives such as minimizing total tardiness against deadlines, improving workload distribution across machines to avoid overuse of specific equipment, or minimizing total energy consumption for greener laboratory operations. A multi-objective approach would provide decision-makers with a set of Pareto-optimal solutions, offering greater flexibility to align schedules with diverse operational priorities.

Finally, our study was conducted under a static setting, assuming the state of all resources remains unchanged throughout the scheduling horizon. However, real-world laboratories can be dynamic environments and may be subject to disruptions, such as unexpected machine breakdowns. Future research could therefore focus on developing dynamic or real-time scheduling strategies. This would require algorithms capable of quickly adapting and rescheduling in response to these stochastic events, a critical step towards creating truly resilient and intelligent laboratory automation systems.

Bibliography

- [1] J. Zhang, G. Ding, Y. Zou, S. Qin, and J. Fu, “Review of job shop scheduling research and its new perspectives under Industry 4.0,” *Journal of Intelligent Manufacturing*, vol. 30, no. 4, pp. 1809–1830, Apr. 2019, ISSN: 0956-5515, 1572-8145. DOI: 10.1007/s10845-017-1350-2. (visited on 03/26/2025).
- [2] I. Holland and J. A. Davies, “Automation in the Life Science Research Laboratory,” *Frontiers in Bioengineering and Biotechnology*, vol. 8, p. 571777, Nov. 2020, ISSN: 2296-4185. DOI: 10.3389/fbioe.2020.571777. (visited on 03/28/2025).
- [3] I. A. Chaudhry and A. A. Khan, “A research survey: Review of flexible job shop scheduling techniques,” *International Transactions in Operational Research*, vol. 23, no. 3, pp. 551–591, May 2016, ISSN: 0969-6016, 1475-3995. DOI: 10.1111/itor.12199. (visited on 03/26/2025).
- [4] Y. Wang and Q. Zhu, “A Hybrid Genetic Algorithm for Flexible Job Shop Scheduling Problem With Sequence-Dependent Setup Times and Job Lag Times,” *IEEE Access*, vol. 9, pp. 104864–104873, 2021, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3096007. (visited on 03/31/2025).
- [5] F. F. Wei, C. Y. Cao, and H. P. Zhang, “An Improved Genetic Algorithm for Resource-Constrained Flexible Job-Shop Scheduling,” *International Journal of Simulation Modelling*, vol. 20, no. 1, pp. 201–211, Mar. 2021, ISSN: 17264529. DOI: 10.2507/IJSIMM20-1-C05. (visited on 03/28/2025).
- [6] W. Ren, Y. Yan, Y. Hu, and Y. Guan, “Joint optimisation for dynamic flexible job-shop scheduling problem with transportation time and resource constraints,” *International Journal of Production Research*, vol. 60, no. 18, pp. 5675–5696, Sep. 2022, ISSN: 0020-7543, 1366-588X. DOI: 10.1080/00207543.2021.1968526. (visited on 03/28/2025).
- [7] J. Michl, K. C. Park, and P. Swietach, “Evidence-based guidelines for controlling pH in mammalian live-cell culture systems,” *Communications Biology*, vol. 2, no. 1, p. 144, Apr. 2019, ISSN: 2399-3642. DOI: 10.1038/s42003-019-0393-7. (visited on 03/30/2025).
- [8] I. Gallego Romero, A. A. Pai, J. Tung, and Y. Gilad, “RNA-seq: Impact of RNA degradation on transcript quantification,” *BMC Biology*, vol. 12, no. 1, p. 42, Dec. 2014, ISSN: 1741-7007. DOI: 10.1186/1741-7007-12-42. (visited on 03/30/2025).
- [9] A. B. Canelas, A. Ten Pierick, C. Ras, *et al.*, “Quantitative Evaluation of Intracellular Metabolite Extraction Techniques for Yeast Metabolomics,” *An-*

- alytical Chemistry*, vol. 81, no. 17, pp. 7379–7389, Sep. 2009, ISSN: 0003-2700, 1520-6882. DOI: 10.1021/ac900999t. (visited on 03/30/2025).
- [10] R. Schäfer, “Concepts for Dynamic Scheduling in the Laboratory,” *JALA: Journal of the Association for Laboratory Automation*, vol. 9, no. 6, pp. 382–397, Dec. 2004, ISSN: 1535-5535. DOI: 10.1016/j.jala.2004.10.001. (visited on 04/10/2025).
- [11] T. D. Itoh, T. Horinouchi, H. Uchida, K. Takahashi, and H. Ozaki, “Optimal Scheduling for Laboratory Automation of Life Science Experiments with Time Constraints,” *SLAS Technology*, vol. 26, no. 6, pp. 650–659, Dec. 2021, ISSN: 24726303. DOI: 10.1177/24726303211021790. (visited on 03/26/2025).
- [12] Y. Arai, K. Takahashi, T. Horinouchi, K. Takahashi, and H. Ozaki, “SAGAS: Simulated annealing and greedy algorithm scheduler for laboratory automation,” *SLAS Technology*, vol. 28, no. 4, pp. 264–277, Aug. 2023, ISSN: 24726303. DOI: 10.1016/j.slast.2023.03.001. (visited on 03/26/2025).
- [13] F. Pezzella, G. Morganti, and G. Ciaschetti, “A genetic algorithm for the Flexible Job-shop Scheduling Problem,” *Computers & Operations Research*, vol. 35, no. 10, pp. 3202–3212, Oct. 2008, ISSN: 03050548. DOI: 10.1016/j.cor.2007.02.014. (visited on 03/31/2025).
- [14] X. Li and L. Gao, “An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem,” *International Journal of Production Economics*, vol. 174, pp. 93–110, Apr. 2016, ISSN: 09255273. DOI: 10.1016/j.ijpe.2016.01.016. (visited on 03/31/2025).
- [15] J. Xie, X. Li, L. Gao, and L. Gui, “A hybrid genetic tabu search algorithm for distributed flexible job shop scheduling problems,” *Journal of Manufacturing Systems*, vol. 71, pp. 82–94, Dec. 2023, ISSN: 02786125. DOI: 10.1016/j.jmsy.2023.09.002. (visited on 03/31/2025).
- [16] B. Dalanezi Mori, H. Fiori De Castro, and K. Lucchesi Cavalca, “Development of hybrid algorithm based on simulated annealing and genetic algorithm to reliability redundancy optimization,” *International Journal of Quality & Reliability Management*, vol. 24, no. 9, pp. 972–987, Oct. 2007, ISSN: 0265-671X. DOI: 10.1108/02656710710826225. (visited on 03/29/2025).
- [17] J. Jonasson and E. Norgren, “Investigating a Genetic Algorithm- Simulated Annealing Hybrid Applied to University Course Timetabling Problem,”
- [18] V. F. Yu, G. Aloina, H. Susanto, M. K. Effendi, and S.-W. Lin, “Regional Location Routing Problem for Waste Collection Using Hybrid Genetic Algorithm-Simulated Annealing,” *Mathematics*, vol. 10, no. 12, p. 2131, Jun. 2022, ISSN: 2227-7390. DOI: 10.3390/math10122131. (visited on 03/29/2025).
- [19] P. Suanpang, P. Jamjuntr, K. Jermstittiparsert, and P. Kaewyong, “Tourism Service Scheduling in Smart City Based on Hybrid Genetic Algorithm Simulated Annealing Algorithm,” *Sustainability*, vol. 14, no. 23, p. 16 293, Dec. 2022, ISSN: 2071-1050. DOI: 10.3390/su142316293. (visited on 03/29/2025).
- [20] “Genetic Algorithms,” *SCIENTIFIC AMERICAN*, 1992.
- [21] M. K. Amjad, S. I. Butt, R. Kousar, *et al.*, “Recent Research Trends in Genetic Algorithm Based Flexible Job Shop Scheduling Problems,” *Mathematical Problems in Engineering*, vol. 2018, pp. 1–32, 2018, ISSN: 1024-123X, 1563-5147. DOI: 10.1155/2018/9270802. (visited on 04/13/2025).

-
- [22] J. Gao, L. Sun, and M. Gen, “A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems,” *Computers & Operations Research*, vol. 35, no. 9, pp. 2892–2907, Sep. 2008, ISSN: 03050548. DOI: 10.1016/j.cor.2007.01.001. (visited on 04/13/2025).
- [23] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by Simulated Annealing,”
- [24] B. Suman and P. Kumar, “A survey of simulated annealing as a tool for single and multiobjective optimization,” *Journal of the Operational Research Society*, vol. 57, no. 10, pp. 1143–1160, Oct. 2006, ISSN: 0160-5682, 1476-9360. DOI: 10.1057/palgrave.jors.2602068. (visited on 04/17/2025).
- [25] L. Ingber, “Simulated annealing: Practice versus theory,” *Mathematical and Computer Modelling*, vol. 18, no. 11, pp. 29–57, Dec. 1993, ISSN: 08957177. DOI: 10.1016/0895-7177(93)90204-C. (visited on 04/17/2025).
- [26] P. J. Van Laarhoven, E. H. Aarts, P. J. van Laarhoven, and E. H. Aarts, *Simulated annealing*. Springer, 1987.
- [27] Y. Nourani and B. Andresen, “A comparison of simulated annealing cooling strategies,” *Journal of Physics A: Mathematical and General*, vol. 31, no. 41, pp. 8373–8385, Oct. 1998, ISSN: 0305-4470, 1361-6447. DOI: 10.1088/0305-4470/31/41/011. (visited on 04/17/2025).
- [28] H. Cohn and M. Fielding, “Simulated Annealing: Searching for an Optimal Temperature Schedule,” *SIAM Journal on Optimization*, vol. 9, no. 3, pp. 779–802, Jan. 1999, ISSN: 1052-6234, 1095-7189. DOI: 10.1137/S1052623497329683. (visited on 04/17/2025).
- [29] X. Gu, S. Neubert, N. Stoll, and K. Thurow, “Intelligent scheduling method for life science automation systems,” in *2016 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, 2016, pp. 156–161. DOI: 10.1109/MFI.2016.7849482.
- [30] M. Nagura-Ikeda, K. Imai, S. Tabata, *et al.*, “Clinical Evaluation of Self-Collected Saliva by Quantitative Reverse Transcription-PCR (RT-qPCR), Direct RT-qPCR, Reverse Transcription–Loop-Mediated Isothermal Amplification, and a Rapid Antigen Test To Diagnose COVID-19,” *Journal of Clinical Microbiology*, vol. 58, no. 9, M. B. Miller, Ed., e01438–20, Aug. 2020, ISSN: 0095-1137, 1098-660X. DOI: 10.1128/JCM.01438-20. (visited on 04/11/2025).
- [31] Robotic Biology Consortium, N. Yachie, and T. Natsume, “Robotic crowd biology with Maholo LabDroids,” *Nature Biotechnology*, vol. 35, no. 4, pp. 310–312, Apr. 2017, ISSN: 1087-0156, 1546-1696. DOI: 10.1038/nbt.3758. (visited on 04/11/2025).
- [32] Google, *Or-tools: Google optimization tools*, <https://developers.google.com/optimization>, Accessed: 2025-04-19, 2023.

A

Appendix 1

List of Acronyms

B&B: Branch and Bound

CBC: COIN-OR Branch and Cut

EFT: Earliest Finish Time

FJSP: Flexible Job Shop Scheduling Problem

FJSP-SDST-LT: Flexible Job Shop Scheduling Problem with Sequence-Dependent Setup Times and Job Lag Times

GA: Genetic Algorithm

GASA: Genetic Algorithm-Simulated Annealing

GA-TS: Genetic Algorithm-Tabu Search

JBX: Job-Based Crossover

MIP: Mixed Integer Programming

Mod: Final Modification (process in SAGAS)

MS: Machine Selection

OD: Operation Delay

OS: Operation Sequence

POX: Precedence Operation Crossover

qPCR: Quantitative Polymerase Chain Reaction

SA: Simulated Annealing

SAGAS: Simulated Annealing and Greedy Algorithm Scheduler

S-LAB: Scheduling for Laboratory Automation in Biology

SPT: Shortest Processing Time

TCMB: Time Constraints by Mutual Boundaries