Solving Industry 4.0 scheduling tasks

PhD Thesis

by

Krisztián Attila Bakon

Supervisors

Tibor Holczinger

Szilárd Jaskó

DOI:10.18136/PE.2026.986

Doctoral School of Information Science and Technology

University of Pannonia

Veszprém, Hungary

January 2026

SOLVING INDUSTRY 4.0 SCHEDULING TASKS

The thesis was prepared for the award of a doctoral degree (PhD) within the framework of the Doctoral School of Information Science and Technology at University of Pannonia

in the discipline of Computer Sciences

written by: Krisztián Attila Bakon

Supervisor(s): Tibor Holczinger, Dr. Szilárd Jaskó

I recommend the dissertation for acceptance: yes / no.

…….
Supervisor

I recommend the dissertation for acceptance: yes / no.

…….
Supervisor

I recommend the dissertation for peer review.

…….
Chair of the DDHC

The PhD-candidate has achieved ……. % at the public debate.

The composition of the Final Examination Committee:

Chair:.....

Reviewers:.....

Members:.....

Veszprém, ……

…….
Chair of the Committee

Qualification of degree: ….....

Veszprém, ……

…….
Chair of the UDHC

Abstract

This dissertation investigates advanced scheduling techniques for dynamic manufacturing environments, focusing on the integration of reactive adaptability, intermediate storage management, and multi-objective optimization within a unified framework. Motivated by the increasing complexity and real-time variability of production systems in the context of Industry 4.0 and 5.0, the research addresses two core scheduling paradigms: reactive scheduling under dynamic order arrivals, and deterministic scheduling with due date and storage constraints.

The first part of the research dissertation formulates a reactive branch-and-bound scheduling model based on the S-graph framework, capable of adapting to new job arrivals through policy-driven rescheduling strategies. A comparative analysis of different rescheduling policies demonstrates that full rescheduling consistently yields the best makespan performance, albeit with higher computational cost. Novel mechanisms such as zero-wait arcs, task sequence preservation, and partial solution reuse are introduced to enable efficient real-time adaptation.

The second part of the dissertation presents a deterministic scheduling model that integrates intermediate storage policies with due date adherence. A multi-objective cost function is proposed to minimize both earliness/tardiness penalties and storage time, incorporating dynamic schedule adjustments via local search. This approach significantly improves schedule quality in storage-constrained environments by ensuring that equipment blocking is minimized without violating precedence or timing constraints.

Finally, a comprehensive multi-objective scheduling framework is introduced, synthesizing the strengths of the reactive and deterministic approaches. It unifies policy-aware rescheduling, storage-aware placement, and due date optimization in a single S-graph-based model. The framework is supported by theoretical formulations, algorithmic procedures, and illustrative benchmark examples.

Tartalmi kivonat

A disszertáció korszerű ütemezési technikákat vizsgál dinamikus gyártási környezetben, különös tekintettel a reaktív alkalmazkodás, a köztes tárolás kezelése és a többcélú optimalizálás integrálására egy egységes keretrendszerben. A kutatás célja a növekvő gyártási komplexitás és a valós idejű változások kezelése, két fő ütemezési paradigmán keresztül: a reaktív ütemezésen (dinamikus megrendelések esetén), valamint a determinisztikus ütemezésen (határidők és tárolási korlátok figyelembevételével).

Az értekezés első része egy S-gráfon alapuló, reaktív korlátozás és szétválasztás ütemezési modellt dolgoz ki, amely képes új megrendelésekhez igazodni előre meghatározott újraütemezési irányelvek szerint. A különböző irányelvek összehasonlító elemzése kimutatja, hogy a teljes újraütemezés következetesen a legjobb átfutási időt eredményezi, magasabb számítási költség árán. Az algoritmus új mechanizmusokat vezet be, mint például a zéró-várakozás él, a feladat-sorrend megőrzése, valamint a részmegoldások újrahasznosítása, amelyek lehetővé teszik a hatékony valós idejű alkalmazkodást.

A dolgozat második része egy determinisztikus modellt mutat be, amely integrálja a köztes tárolási irányelveket a határidők betartásával. A dolgozat egy többcélú költségfüggvényt vezet be, amelynek célja a korai és késedelmes teljesítések büntetéseinek, valamint a köztes tárolási idő minimalizálása, az ütemezés helyi keresési műveleteken alapuló dinamikus finomhangolásával. A megközelítés jelentős előrelépést mutat a szűkös tárolási kapacitásokkal rendelkező gyártási környezetekben, mivel minimalizálja az eszközblokkolást a folyamatok megszakítása nélkül.

Végezetül a dolgozat egy olyan átfogó, többcélú ütemezési keretrendszert mutat be, amely egységesíti a reaktív és determinisztikus modellek előnyeit. A rendszer ötvözi az irányelvekre épülő újraütemezést, a tárolásra érzékeny feladatelhelyezést és a határidő-optimalizálást egy S-gráf alapú modellben. A keretrendszert elméleti megfogalmazás, algoritmikus eljárások és szemléletes példapéldák támasztják alá.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Tibor Holczinger, whose guidance, expertise, and continuous support were instrumental throughout the course of my doctoral research. His feedback and encouragement played a central role in shaping both the direction and the quality of this dissertation. I also thank my co-supervisor, Dr. Szilárd Jaskó, for his involvement in supervising my work.

I am grateful to the Doctoral School of Information Science and Technology at the University of Pannonia for providing a supportive academic environment and the opportunity to pursue this research.

Special thanks are due to my colleagues at the University of Pannonia - University Center for Circular Economy in Nagykanizsa. Their professional collaboration, helpful discussions, and friendly encouragement provided an inspiring environment that supported the development of this research.

Finally, I would like to thank my family for their patience, understanding, and continuous support throughout this journey. Their presence and encouragement were essential to the completion of this dissertation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In real-world manufacturing environments, production systems operate under conditions of inherent uncertainty and variability that render static scheduling approaches inadequate [1]. The dynamic nature of contemporary manufacturing necessitates scheduling methodologies that can adapt to unforeseen disruptions while maintaining operational efficiency and minimizing key performance metrics such as makespan [2].

This dissertation addresses the challenges of scheduling in uncertain manufacturing environments, focusing on two primary approaches: reactive scheduling, which aims to minimize makespan in dynamic conditions, and deterministic scheduling, which seeks to meet due dates while managing intermediate storage constraints. By exploring these two paradigms, this work contributes to the development of robust scheduling strategies that can effectively navigate the complexities of modern manufacturing systems.

1.1 Motivation

The motivation for this research stems from the increasing complexity and uncertainty in modern manufacturing environments, particularly in the context of Industry 4.0 and 5.0 [3]. As manufacturing systems evolve to incorporate advanced technologies such as IoT, big data analytics, and automation, they become more interconnected and responsive to real-time changes. However, this evolution also introduces significant challenges related to scheduling tasks effectively amidst uncertainties such as machine breakdowns, fluctuating demand, and supply chain disruptions. Effective scheduling is crucial for maintaining operational efficiency, minimizing production delays, and ensuring timely delivery of products. In uncertain environments, traditional scheduling methods often fall short, as they typically rely on static assumptions that do not account for the dynamic nature of modern manufacturing. Therefore, there is a pressing need for innovative scheduling

approaches that can adapt to real-time changes and uncertainties while optimizing key performance metrics.

1.2 Research Context

The context of this research lies within the broader field of scheduling in manufacturing, particularly as it pertains to the challenges posed by Industry 4.0 and 5.0. These paradigms represent a significant shift in manufacturing practices, emphasizing the integration of digital technologies, automation, and data-driven decision-making. As manufacturing systems become more complex and interconnected, the need for advanced scheduling techniques that can effectively manage uncertainty and variability becomes paramount [4]. The significance of Industry 4.0 and 5.0 in shaping the future of manufacturing cannot be overstated [5]. These paradigms not only enhance operational efficiency but also enable greater flexibility and responsiveness to market demands. However, they also introduce new challenges, particularly in the realm of scheduling, where traditional methods may no longer suffice. The ability to adapt to real-time changes, manage disruptions, and optimize production schedules is critical for maintaining competitiveness in this evolving landscape.

Modern manufacturing environments are characterized by multiple sources of uncertainty that continuously challenge the validity of predetermined schedules [6]. These uncertainties manifest through disruption categories including machine-related disruptions (breakdowns, processing time variability, resource availability fluctuations), demand uncertainties (order modifications, demand variability, dynamic job arrivals), and supply chain disruptions (material delays, quality issues). These dynamic conditions create a fundamental tension between schedule stability and operational adaptability. Traditional approaches face significant limitations: static scheduling becomes obsolete when disruptions occur, pure reactive methods may sacrifice global optimality for local responsiveness, and single-objective optimization fails to address competing operational priorities.

The core research problem addressed in this dissertation is the development of scheduling frameworks that simultaneously maintain high performance under disruptions, balance multiple objectives (makespan, due dates, storage constraints), and provide computational efficiency for real-world implementation. This requires addressing the inherent trade-offs between operational efficiency, customer service requirements, and resource utilization constraints in dynamic production environments [7].

1.3 Structure of the Dissertation

This dissertation is structured as follows:

- Chapter 1 introduces the research motivation, objectives, and context of the study, and outlines the overall methodological approach and structure of the dissertation.
- Chapter 2 presents a comprehensive literature review covering manufacturing scheduling problem types, sources of uncertainty, scheduling paradigms, exact and heuristic solution methods, multi-objective optimization techniques, and reactive scheduling approaches, with particular emphasis on S-graph-based models.
- Chapter 3 formulates the considered scheduling problems, including reactive scheduling with dynamic order arrivals, deterministic scheduling with due-date and intermediate storage objectives, and their integrated treatment within a unified problem framework.
- Chapter 4 investigates reactive scheduling strategies with makespan minimization. It introduces the proposed methodology, graph-based policy variants, and solution strategies, and evaluates their performance through computational experiments and literature-based comparisons.
- Chapter 5 addresses deterministic scheduling problems involving due-date compliance and intermediate storage minimization. The chapter extends the S-graph framework to earliness and tardiness objectives, integrates linear programming techniques, and presents numerical results and discussion.
- Chapter 6 develops an integrated reactive scheduling framework that jointly considers due dates and intermediate storage constraints. It introduces storage-aware reactive policies, algorithmic procedures, bounding techniques, and analyzes trade-offs between competing objectives.
- Chapter 7 concludes the dissertation by summarizing the main scientific contributions, discussing practical and industrial implications, outlining limitations, and proposing directions for future research.

Chapter 2

Literature Review

2.1 Overview of Scheduling in Manufacturing Systems

Manufacturing scheduling is a critical optimization problem that involves allocating limited resources to various jobs or tasks over time to achieve specific performance objectives [8, 9]. The field encompasses several distinct problem types, each with unique characteristics and constraints that reflect different manufacturing environments and operational requirements [10, 11].

2.1.1 Types of Scheduling Problems

Manufacturing scheduling problems are commonly classified into job shop [8, 10], flow shop [12], flexible job shop [13], and open shop environments [14], differing primarily in routing flexibility and machine assignment constraints. Among these, the flexible job shop scheduling problem (FJSP) has gained particular attention due to its relevance in modern manufacturing systems with multipurpose equipment [15]. Since the proposed approach addresses flexible task–machine assignments under dynamic conditions [16], the remainder of this work focuses on FJSP-type environments.

2.1.2 General Objectives of Manufacturing Scheduling

Manufacturing scheduling aims to optimize performance measures that reflect different operational priorities, such as efficiency, customer service, and resource utilization [17]. Among the wide range of objectives studied in the literature, due-date–related measures and time-based efficiency criteria are particularly relevant for dynamic and reactive scheduling environments.

One of the most fundamental objectives is *makespan minimization*, where the goal is to minimize the total completion time of all jobs [18]. Makespan reflects overall production efficiency and resource utilization and is commonly used as a baseline objective in deterministic scheduling problems. However, makespan alone does not capture product-level delivery performance, which is critical in customer-oriented manufacturing systems.

To address delivery performance, *earliness and tardiness* objectives are widely applied [19]. For a product p with completion time C_p and due date d_p, tardiness and earliness are defined as

$$T_p = \max(0, C_p - d_p), \qquad E_p = \max(0, d_p - C_p).$$

Minimizing earliness and tardiness penalties supports timely deliveries while avoiding excessive early completion that leads to increased inventory and work-in-process costs.

In practical manufacturing environments, scheduling decisions often involve *multiple conflicting objectives*. For example, minimizing makespan may increase tardiness for jobs with tight due dates, while strict adherence to due dates may reduce overall resource efficiency [20]. As a result, multi-objective formulations—typically combining time-based efficiency and due-date performance—have become increasingly important.

A common approach to handling multiple objectives is the use of weighted composite objective functions, where individual performance measures are aggregated using predefined weights [21]. This allows decision-makers to balance competing goals according to operational priorities. Such formulations are especially suitable for branch-and-bound–based optimization frameworks, where objective evaluation and bounding play a central role.

In addition to classical performance measures, modern scheduling problems increasingly consider constraints and objectives related to intermediate storage and material flow. Limiting intermediate storage can significantly affect task sequencing, equipment utilization, and delivery performance, making it an important factor in integrated scheduling models.

2.1.3 Performance Measures and Trade-offs

Manufacturing scheduling objectives are often inherently conflicting, requiring explicit consideration of trade-offs between competing performance measures [22]. Improvements in one objective frequently come at the expense of others, making single-objective optimization insufficient for realistic production environments.

A well-known trade-off exists between makespan minimization and due-date performance. Schedules that minimize makespan tend to prioritize overall throughput and

resource utilization, which may increase tardiness for jobs with tight due dates [23]. Conversely, schedules optimized for due-date adherence may introduce idle times or inefficient sequencing, leading to longer overall completion times.

Similarly, aggressive utilization of equipment can negatively affect flow-related objectives. High utilization levels often increase queueing effects, which may prolong completion times and exacerbate delivery delays [24]. These effects become more pronounced in dynamic environments where disturbances such as new job arrivals or machine unavailability occur.

The presence of intermediate storage constraints introduces additional trade-offs. Restricting or eliminating intermediate storage can increase blocking and reduce scheduling flexibility, potentially worsening due-date performance. On the other hand, allowing intermediate storage improves temporal flexibility but may increase work-in-process levels and waiting times between tasks. Consequently, storage-related decisions are closely intertwined with both time-based efficiency and delivery performance.

Understanding these trade-offs is essential for designing effective multi-objective scheduling frameworks. Rather than optimizing a single metric, modern scheduling approaches aim to balance competing objectives in a controlled manner, enabling decision-makers to select solutions that best align with operational priorities under dynamic conditions.

2.2 Sources and Types of Uncertainty in Scheduling

Manufacturing scheduling operates in inherently uncertain environments, where deviations from planned conditions frequently occur during schedule execution. Unlike deterministic models, real-world production systems must continuously adapt to dynamic changes that affect resource availability, task execution, and order fulfillment. These uncertainties motivate the need for reactive and adaptive scheduling approaches capable of revising decisions in response to unfolding events.

2.2.1 Operational Sources of Uncertainty

The most relevant sources of uncertainty in manufacturing scheduling arise at the operational level and directly impact feasibility and performance during execution.

- **Machine-related uncertainty.** Equipment breakdowns, maintenance interventions, and fluctuating machine availability represent a primary cause of schedule

disruption. Such events may force task interruption, reassignment, or resequencing, often propagating delays across multiple downstream operations [25, 26].

- **Processing time variability.** Actual processing times may deviate from planned values due to machine condition, operator skill, material quality, or environmental factors. Even small deviations can accumulate and significantly affect completion times and due-date performance, particularly in tightly coupled production systems [27].
- **Order-related uncertainty.** New job arrivals, order cancellations, priority changes, or due-date modifications frequently occur after schedule release. These events require rapid integration of new tasks without invalidating ongoing operations, posing a central challenge for online and reactive scheduling methods [28].

These uncertainty sources are especially critical in dynamic scheduling contexts, where decisions must be made without full knowledge of future system states.

2.2.2 Variability and Knowledge-Related Uncertainty

Uncertainty in scheduling can be broadly categorized into variability-based [29] and knowledge-based [30–32] components. Variability-based uncertainty stems from inherent randomness in system behavior, such as stochastic processing times or unpredictable machine failures [33]. This type of uncertainty cannot be eliminated but must be managed through adaptive decision-making mechanisms [34, 35].

Knowledge-related (epistemic) uncertainty arises from incomplete or imprecise information [36], including inaccurate processing time estimates, incomplete system state observation, or simplified modeling assumptions [30]. While such uncertainty may be reduced through improved sensing or data collection, it cannot be fully eliminated in practice.

Both forms of uncertainty coexist in manufacturing environments and jointly contribute to deviations between planned and realized schedules.

2.2.3 Implications for Reactive Scheduling

The presence of uncertainty fundamentally limits the effectiveness of static scheduling approaches. Schedules constructed under deterministic assumptions may quickly become infeasible or suboptimal once execution begins. Reactive scheduling addresses this challenge by enabling incremental schedule modification in response to observed disruptions, rather than full regeneration of plans.

In particular, uncertainty related to new order arrivals and processing delays necessitates scheduling frameworks that can insert or adjust tasks during execution while preserving feasibility with respect to precedence, resource, and storage constraints. This motivates the development of reactive, policy-aware scheduling algorithms that explicitly account for uncertainty-induced deviations and support real-time decision-making.

2.3 Scheduling Algorithms and Approaches

Manufacturing scheduling represents one of the most critical operational challenges in modern production environments, requiring sophisticated algorithmic approaches to optimize resource allocation, minimize costs, and meet delivery deadlines. The complexity of contemporary manufacturing systems, characterized by dynamic environments, uncertain demand patterns, and frequent disruptions, has driven the development of diverse scheduling methodologies ranging from exact optimization techniques to intelligent heuristic approaches [3, 37].

2.3.1 Static, Dynamic, and Reactive Scheduling Paradigms

Scheduling algorithms can be broadly categorized based on their responsiveness to changes in the production environment. This classification yields three fundamental paradigms: static, dynamic, and reactive scheduling. Each of these approaches differs in how it incorporates information, addresses uncertainty, and adapts to real-time events during execution. The selection of a suitable scheduling methodology is largely determined by the nature of the manufacturing system, the frequency and severity of disturbances, and the availability of computational resources for decision-making.

2.3.1.1 Static Scheduling Approaches

Static scheduling operates under the assumption of complete information availability and deterministic environments, generating predetermined schedules that remain fixed throughout execution [38]. These approaches are particularly suitable for stable manufacturing environments where disruptions are minimal and processing parameters are well-known [39]. The primary advantage of static scheduling lies in its ability to generate globally optimal or near-optimal solutions when environmental conditions match the planning assumptions [40, 41].

However, static scheduling approaches face significant limitations in real-world manufacturing environments where unexpected events such as machine breakdowns, urgent job

arrivals, and processing time variations are inevitable [42]. When disruptions occur, static schedules may become infeasible or severely suboptimal, requiring complete regeneration or manual intervention [43]. The computational burden of repeatedly solving static scheduling problems in response to disruptions often makes this approach impractical for dynamic manufacturing environments [44].

2.3.1.2 Dynamic Scheduling Methodologies

Dynamic scheduling addresses the limitations of static approaches by continuously adapting schedules in response to real-time information and changing conditions. This paradigm encompasses several distinct approaches, each with different levels of responsiveness and computational requirements. Dynamic scheduling systems utilize real-time data from various sources, including IoT sensors, enterprise resource planning systems, and production floor feedback, to make informed scheduling decisions [45].

The most prevalent form of dynamic scheduling is predictive-reactive scheduling, which generates baseline schedules and revises them in response to real-time events [43]. This approach balances the benefits of global optimization with the flexibility needed to handle disruptions. Predictive-reactive scheduling typically employs robustness measures that consider both schedule efficiency and stability, minimizing deviation from the original schedule while maintaining acceptable performance levels [46].

2.3.1.3 Reactive Scheduling Strategies

Completely reactive scheduling represents the most flexible approach, making scheduling decisions locally in real-time without generating firm advance schedules. This methodology primarily relies on dispatching rules and priority-based selection mechanisms to determine job sequencing at each decision point. While reactive scheduling offers maximum adaptability to changing conditions, it often sacrifices global optimality for local responsiveness [47].

The effectiveness of reactive scheduling depends heavily on the sophistication of the dispatching rules employed and the accuracy of real-time information available. Modern reactive scheduling systems increasingly incorporate artificial intelligence techniques to improve decision-making quality while maintaining the rapid response times essential for dynamic environments [48].

2.3.2 Exact Optimization Methods

2.3.2.1 Mixed Integer Programming Approaches

Mixed Integer Programming (MIP) represents the most rigorous approach to manufacturing scheduling, providing optimal solutions for problems that can be formulated within reasonable computational bounds [49]. MIP formulations excel in handling complex constraint structures, including resource limitations, precedence relationships, setup dependencies, and temporal constraints. The mathematical rigor of MIP approaches ensures solution optimality when computational resources permit complete enumeration or branch-and-bound exploration of the solution space [50].

2.3.2.2 Constraint Programming Techniques

Constraint Programming (CP) offers an alternative exact approach that excels in handling complex logical relationships and non-linear constraints common in manufacturing scheduling. CP techniques are particularly effective for problems involving alternative resources, complex precedence structures, and specialized constraints that are difficult to model in traditional MIP formulations. The declarative nature of constraint programming allows for more intuitive problem modeling while maintaining mathematical rigor [51].

2.3.2.3 S-Graph-Based Exact Methods

An important class of exact methods is the S-graph–based scheduling approach, which combines graph-theoretic representation with branch-and-bound search to efficiently solve various scheduling problems. Originally developed for multipurpose batch processes, the S-graph framework provides an exact solution methodology capable of handling precedence constraints, resource limitations, and intermediate storage policies.

The detailed structure and algorithmic procedures of the S-graph model are discussed in Section 2.6, highlighting its applicability to flexible job shop scheduling problems, particularly those involving makespan and storage-related objectives.

2.3.3 Heuristic Approaches

2.3.3.1 List Scheduling and Constructive Heuristics

List scheduling represents one of the most fundamental and widely applicable heuristic approaches in manufacturing scheduling. These methods construct schedules by iteratively

selecting jobs from priority-ordered lists and assigning them to available resources according to predefined rules [52]. The simplicity and computational efficiency of list scheduling make it particularly attractive for real-time applications and large-scale problems where exact methods are computationally prohibitive [53].

2.3.3.2 Dispatching Rules and Real-Time Decision Making

Dispatching rules form the cornerstone of reactive scheduling systems, providing simple yet effective mechanisms for making real-time scheduling decisions. These rules operate by assigning priorities to jobs awaiting processing and selecting the highest-priority job when resources become available. The computational simplicity of dispatching rules enables their application in real-time environments where decisions must be made within milliseconds [54].

2.3.4 Metaheuristic Optimization Techniques

Metaheuristic optimization techniques are widely used to address complex manufacturing scheduling problems that are difficult to solve with exact methods, with Genetic Algorithms (GA), Tabu Search, and Simulated Annealing being among the most effective approaches. Genetic Algorithms are well suited for large, multi-objective scheduling problems due to their population-based structure, which supports broad exploration, maintains solution diversity, and helps avoid premature convergence [55]; recent advances such as problem-specific operators, hybridization with local search, and adaptive parameter control have shown that properly configured GA methods can deliver near-optimal solutions with acceptable computational effort [56]. Tabu Search is particularly effective in escaping local optima through structured neighborhood exploration supported by memory mechanisms, and it has demonstrated strong performance in job shop, flexible manufacturing, and multi-product scheduling problems [57]. Its effectiveness depends on neighborhood and memory design, with recent developments including adaptive tabu tenure, intensification and diversification strategies, and hybrid variants, while its computational efficiency makes it suitable for real-time scheduling applications [58, 59]. Simulated Annealing offers another robust alternative, especially for problems with complex objective functions and constraints, as its probabilistic acceptance mechanism enables effective exploration while gradually focusing on high-quality solutions [60, 61]. Applications in production optimization and lean manufacturing have reported significant improvements over traditional methods [62], and advanced implementations increasingly rely on adaptive cooling schedules, informed neighborhood generation, and hybridization with other metaheuristics to enhance performance [63, 64].

2.3.5 Hybrid and Advanced Methodologies

2.3.5.1 Matheuristic Approaches

Matheuristic represent an emerging class of hybrid optimization approaches that combine the mathematical rigor of exact methods with the practical efficiency of heuristic techniques. These methods typically decompose complex scheduling problems into smaller subproblems that can be solved optimally using exact techniques, while employing heuristic approaches to coordinate overall solution construction. The matheuristic paradigm has proven particularly effective for large-scale scheduling problems where pure exact methods are computationally intractable [65].

2.3.5.2 Decomposition Techniques

Decomposition methods address the computational complexity of large-scale scheduling problems by breaking them into manageable subproblems that can be solved independently or with limited coordination [66]. These approaches are particularly valuable for integrated planning and scheduling problems where the full problem formulation would be computationally intractable. Decomposition strategies must carefully balance the trade-off between solution quality and computational efficiency [67].

2.3.6 Scheduling Under Uncertainty

2.3.6.1 Stochastic Programming Approaches

Stochastic programming provides a mathematically rigorous framework for addressing uncertainty in manufacturing scheduling by explicitly modeling uncertain parameters as random variables [68]. These approaches generate scheduling policies that optimize expected performance across multiple scenarios while maintaining feasibility under uncertainty. The computational complexity of stochastic programming approaches often requires approximation strategies for practical implementation [69].

2.3.6.2 Robust Optimization Methods

Robust optimization approaches address uncertainty by generating schedules that perform well across worst-case scenarios within defined uncertainty sets [70]. These methods are particularly attractive for manufacturing applications where the probability distributions of uncertain parameters are difficult to estimate accurately. Robust scheduling approaches

focus on minimizing the maximum deviation from optimal performance across all possible realizations of uncertain parameters [71].

2.3.7 Trade-offs and Computational Considerations

A fundamental trade-off in scheduling algorithm design lies between solution quality and computational effort [72]. Exact optimization methods provide optimality guarantees but typically suffer from exponential growth in computation time as problem size increases. In contrast, heuristic approaches offer rapid solutions with limited computational cost, at the expense of optimality guarantees, while metaheuristic methods aim to balance solution quality and runtime efficiency [73].

The choice of an appropriate scheduling approach therefore depends on the operational context, including problem scale, required response time, and the frequency and severity of disruptions. Real-time decision-making environments often favor fast heuristic or rule-based methods, whereas offline or strategic planning settings may justify the use of more computationally intensive exact techniques [74].

Scalability remains a critical challenge, particularly for exact methods applied to large-scale manufacturing systems involving numerous jobs and resources [75]. Consequently, there is continued interest in algorithmic frameworks that retain optimality guarantees while exploiting problem structure to limit computational complexity. Graph-based formulations and tailored branch-and-bound strategies represent one such direction, offering a compromise between modeling expressiveness and computational tractability.

2.4 Multi-Objective Scheduling

Manufacturing scheduling in contemporary industrial environments rarely involves the optimization of a single performance criterion. Instead, decision-makers must navigate complex trade-offs between multiple, often conflicting objectives that capture different aspects of production efficiency, customer satisfaction, and operational costs. This reality has driven the development of sophisticated multi-objective scheduling methodologies that can simultaneously consider various performance measures while providing decision-makers with flexible solution frameworks for making informed trade-offs [76, 77].

The importance of multi-objective scheduling becomes particularly evident when considering the diverse stakeholder interests in manufacturing systems. Production managers seek to minimize costs and maximize resource utilization, while sales departments prioritize meeting customer due dates, and quality control departments focus on maintaining

high product standards. These competing objectives require systematic approaches that can balance conflicting goals while maintaining overall system performance [78].

2.4.1 Common Objectives in Multi-Objective Scheduling

2.4.1.1 Makespan Minimization

Makespan represents the total completion time of all products in a production schedule and serves as one of the most fundamental objectives in manufacturing scheduling [79]. The makespan criterion reflects the system's throughput capacity and directly impacts facility utilization and production efficiency. Minimizing makespan typically leads to better resource utilization and reduced idle time, making it particularly valuable for capital-intensive manufacturing environments where equipment costs constitute a significant portion of total production costs [21, 80].

However, makespan optimization often conflicts with other objectives, particularly those related to customer service and product-specific requirements. A schedule that minimizes makespan may result in some products being completed well before their due dates while others experience significant delays, leading to inventory costs and customer dissatisfaction [81, 82]. This inherent trade-off has motivated the development of multi-objective approaches that balance makespan reduction with other performance measures.

2.4.1.2 Tardiness and Earliness Considerations

Tardiness and earliness represent fundamental measures of schedule performance relative to customer requirements and operational efficiency. Tardiness measures the extent to which products are completed after their due dates, directly impacting customer satisfaction and potentially incurring penalty costs. Mathematically, for product p with completion time C_p and due date d_p, tardiness is defined as $T_p = \max(0, C_p - d_p)$ [82].

Earliness, conversely, measures the completion of products before their scheduled due dates, which can lead to inventory holding costs and tied-up capital. The earliness for product p is defined as $E_p = \max(0, d_p - C_p)$. In manufacturing environments, both tardiness and earliness are undesirable, as they can lead to increased costs and reduced customer satisfaction. Consequently, modern scheduling approaches often aim to minimize both tardiness and earliness simultaneously, recognizing that achieving a balance between these two objectives is crucial for effective production management [83].

Common multi-objective formulations involving tardiness include minimizing both makespan and maximum tardiness, where the goal is to achieve good overall system

performance while controlling the worst-case customer service performance. Studies have shown that these bi-criteria problems are inherently NP-hard, requiring sophisticated optimization techniques to achieve near-optimal solutions [1].

2.4.1.3 Resource Utilization and Cost Objectives

Resource utilization objectives focus on maximizing the productive use of available manufacturing resources, including machines, labor, and materials [84]. These objectives are particularly important in capital-intensive industries where equipment costs represent a significant portion of total production expenses. Effective resource utilization not only reduces per-unit production costs but also improves overall system capacity and competitiveness [85].

Cost-based objectives encompass various economic considerations, including direct production costs, inventory holding costs, and penalty costs for missed deadlines. Multi-objective scheduling frameworks often incorporate multiple cost components simultaneously, requiring decision-makers to balance immediate production costs against long-term customer relationship costs and inventory management expenses [86].

Recent research has demonstrated the importance of considering energy consumption as a cost objective, particularly in the context of sustainable manufacturing [87]. Energy-aware scheduling approaches seek to minimize both production costs and environmental impact by optimizing equipment usage patterns and reducing energy waste during idle periods [80].

2.4.2 Pareto Optimality

Pareto optimality serves as a foundational concept in multi-objective scheduling, providing a framework for evaluating trade-offs between competing objectives [88]. A solution is considered Pareto optimal if no other solution exists that improves one objective without degrading another. This concept allows decision-makers to identify a set of optimal solutions, known as the Pareto front, which represents the best possible trade-offs among the considered objectives [89].

Pareto optimality is particularly valuable in scheduling applications because it provides decision-makers with a comprehensive view of available trade-offs without requiring a priori specification of objective weights or priorities. This approach acknowledges that different stakeholders may have different preferences regarding objective importance, allowing for post-optimization decision-making based on organizational priorities and current business conditions.

The Pareto front concept has been successfully applied to various manufacturing scheduling problems, including flexible job shop scheduling, parallel machine scheduling, and flow shop optimization. Research has shown that Pareto-based approaches often reveal non-intuitive trade-offs that can lead to improved overall system performance compared to single-objective optimization [90].

2.4.3 Trade-off Analysis Methodologies

Trade-off analysis in multi-objective scheduling involves systematic evaluation of the relationships between different objectives and the costs associated with improving one objective at the expense of others. Effective trade-off analysis requires both quantitative assessment of objective relationships and qualitative understanding of the business implications of different scheduling decisions [1].

Contemporary trade-off analysis approaches utilize various visualization and decision support techniques to help decision-makers understand the implications of different scheduling choices. These methods include Pareto front visualization, sensitivity analysis, and interactive decision support systems that allow real-time exploration of trade-offs [91].

The integration of trade-off analysis with real-time scheduling systems has become increasingly important in dynamic manufacturing environments where conditions change frequently. Advanced systems can provide decision-makers with updated trade-off information as new jobs arrive or equipment status changes, enabling more responsive and informed scheduling decisions [92, 93].

Multi-objective scheduling problems require specialized solution techniques that can effectively navigate the trade-offs between competing objectives while maintaining computational efficiency. Various methodologies have been developed to address these challenges, ranging from weighted sum approaches to advanced evolutionary algorithms.

2.4.3.1 Weighted Sum and Epsilon-Constraint Methods

The weighted sum approach represents one of the most straightforward techniques for solving multi-objective scheduling problems by converting multiple objectives into a single objective function through linear combination. For objectives $f_1, f_2, \ldots, f_k$, the weighted sum formulation creates a composite objective $F = w_1 f_1 + w_2 f_2 + \ldots + w_k f_k$, where w_i represents the weight assigned to objective i [94]. The primary advantage of weighted sum methods lies in their computational simplicity and compatibility with existing single-objective optimization techniques. This approach allows decision-makers to

leverage mature optimization algorithms and software tools without requiring specialized multi-objective implementations. However, the method's effectiveness depends critically on the appropriate selection of weights, which often requires extensive experimentation or domain expertise. Weighted sum methods face fundamental limitations when dealing with non-convex Pareto fronts, where certain Pareto optimal solutions cannot be obtained regardless of weight selection [95]. Additionally, the method's sensitivity to weight specification can make it difficult to achieve consistent results across different problem instances or changing business priorities [96].

The epsilon-constraint method addresses some limitations of weighted sum approaches by optimizing one objective while treating others as constraints. This technique transforms a multi-objective problem into a series of single-objective solutions by setting upper bounds (epsilon values) on all but one objective function. The method systematically varies these epsilon values to generate different points on the Pareto front [97]. The epsilon-constraint approach offers several advantages over weighted sum methods, including the ability to find Pareto optimal solutions on non-convex fronts and more intuitive parameter specification. Decision-makers can more easily understand and specify constraint bounds compared to abstract objective weights, making the method more practical for real-world applications [95]. However, epsilon-constraint methods require careful selection of epsilon values to ensure comprehensive coverage of the Pareto front. Inappropriate epsilon selection can result in infeasible problems or poor representation of available trade-offs. Recent research has focused on adaptive epsilon selection strategies that automatically adjust constraint bounds based on problem characteristics and solution quality [98].

2.4.4 Practical Importance of Balancing Conflicting Goals

The practical importance of balancing conflicting goals in manufacturing scheduling cannot be overstated [99]. Organizations must navigate the complex interplay between cost minimization, due date adherence, resource utilization, and quality requirements to achieve optimal performance across multiple dimensions. The ability to effectively balance these competing objectives is critical for maintaining competitiveness in today's dynamic manufacturing landscape. The trade-offs between cost and due date performance illustrate the inherent complexity of multi-objective scheduling [96]. Organizations must consider not only the direct costs associated with production but also the potential penalties for missed deadlines and the impact on customer relationships. Effective scheduling approaches must account for these trade-offs while providing decision-makers with actionable insights into the implications of different scheduling choices.

The fundamental tension between cost minimization and due date performance represents one of the most critical trade-offs in manufacturing scheduling [100]. Organizations

must balance the desire to minimize production costs through efficient resource utilization against the need to meet customer commitments and maintain service levels. This trade-off becomes particularly challenging in environments with tight margins and demanding customers.

Effective management of cost versus due date trade-offs requires sophisticated understanding of the relationships between different scheduling decisions and their impact on both cost and service performance. Research has shown that simple priority rules, such as earliest due date first, may not be optimal when production costs vary over time or when setup costs are significant [101]. Instead, organizations need adaptive scheduling approaches that can dynamically balance these competing objectives based on current conditions.

The development of cost-balancing heuristics has provided practical solutions for managing these trade-offs in real-time scheduling environments [102]. These approaches use mathematical models to predict the cost implications of different scheduling decisions while considering both direct production costs and penalty costs associated with missed deadlines.

Manufacturing organizations increasingly face pressure to balance production efficiency with quality requirements, particularly in industries where product defects can have significant cost and safety implications [1]. Multi-objective scheduling approaches that consider both throughput and quality metrics enable organizations to make informed trade-offs between these competing goals.

Quality-aware scheduling requires integration of quality prediction models with traditional scheduling optimization techniques. This integration allows schedulers to consider the impact of processing conditions, equipment selection, and sequence decisions on final product quality while maintaining reasonable production efficiency. Recent research has demonstrated that such integrated approaches can achieve better overall performance compared to sequential optimization of efficiency and quality objectives [103].

The incorporation of quality objectives into scheduling frameworks has particular relevance in industries adopting lean manufacturing and continuous improvement philosophies [104]. These approaches emphasize the importance of building quality into the production process rather than relying on post-production inspection and rework.

Modern manufacturing environments are characterized by high levels of uncertainty and variability, making flexibility and robustness increasingly important objectives in scheduling optimization. Multi-objective scheduling frameworks that explicitly consider schedule flexibility and robustness can provide significant advantages in dynamic operating environments [105].

Flexibility in scheduling refers to the ability to accommodate changes in job requirements, processing times, or resource availability without requiring complete schedule regeneration. Robust scheduling approaches focus on generating schedules that maintain good performance across a range of possible scenarios, even when actual conditions differ from planning assumptions.

The integration of flexibility and robustness objectives with traditional performance measures creates complex multi-objective optimization problems that require sophisticated solution techniques. Recent research has explored the use of stochastic optimization, scenario-based approaches, and adaptive scheduling techniques to address these challenges.

2.5 Reactive Scheduling Approaches

In real-world manufacturing environments, production systems operate under inherent uncertainty and variability, which often renders static scheduling approaches inadequate. Consequently, scheduling methodologies must be capable of adapting to unforeseen disruptions while maintaining operational efficiency and minimizing key performance metrics such as makespan.

2.5.1 Definitions and Fundamental Concepts

A fundamental distinction in dynamic scheduling theory exists between reactivity and robustness. Reactivity refers to the system's ability to respond to disruptions during schedule execution through real-time adjustments and rescheduling [43]. In contrast, robustness denotes the ability of a schedule to maintain acceptable performance levels despite disruptions without requiring significant modifications [106].

Reactive scheduling is defined as "the process of repairing the predictive schedule during online execution for internal disruptions (e.g., machine breakdowns) and external deviations (e.g., prepone or postpone of orders)" [107]. This perspective acknowledges the inevitability of disruptions in manufacturing systems and emphasizes effective response mechanisms rather than disruption avoidance.

Modern manufacturing systems are subject to numerous uncertainty sources that continuously challenge the validity of predetermined schedules [108]. These include machine breakdowns, processing time variations, urgent job arrivals, order cancellations, material shortages, and quality failures [109]. As a result, scheduling is increasingly viewed as an ongoing process requiring continuous revision in response to evolving conditions [110, 111]. The interconnected nature of manufacturing operations further

amplifies disruption effects, as local disturbances may propagate throughout the system, necessitating scheduling approaches that can address both immediate and cascading impacts [112].

2.5.2 Reactive Scheduling Policies and Methodologies

Reactive scheduling strategies often rely on repair-based mechanisms that minimally modify existing schedules to accommodate disruptions while preserving their original structure [113]. Common approaches include Right Shift Rescheduling (RSR) and Affected Operations Rescheduling (AOR) [111].

RSR represents the simplest repair strategy, postponing all subsequent operations by the disruption duration [114]. While computationally efficient, this approach often leads to suboptimal solutions due to its inability to exploit available slack or alternative resource assignments. AOR provides a more refined alternative by rescheduling only operations directly or indirectly affected by the disruption, thereby limiting schedule deviation [115]. Extensions of AOR have been proposed to address diverse disruption types, including urgent job arrivals and flexible routing scenarios.

More extensive approaches include complete rescheduling, which regenerates the entire schedule from the disruption point onward [116]. Although potentially optimal, this strategy is computationally demanding and may cause significant schedule instability, commonly referred to as "shop floor nervousness" [110]. Partial rescheduling offers a compromise by rescheduling only a selected subset of operations, balancing computational efficiency and solution quality [109, 116].

Predictive-reactive scheduling integrates predictive planning with reactive execution mechanisms [103]. An initial schedule is generated using predictive methods, often incorporating historical data and probabilistic models, while reactive procedures manage disruptions during execution [7]. Recent developments in robust predictive-reactive scheduling aim to jointly optimize efficiency and schedule stability, reducing the need for frequent reactive interventions [116].

2.5.3 Comparative Analysis: Proactive vs. Reactive vs. Hybrid Methods

Proactive scheduling incorporates uncertainty directly into the planning phase to generate disruption-resistant schedules [1]. Typical strategies include under-capacity scheduling based on historical performance rather than theoretical capacity [117]. While proactive approaches enhance schedule stability and reduce reactive intervention requirements

[103], they may lead to conservative resource utilization and limited adaptability to novel disruptions.

Pure reactive scheduling relies exclusively on real-time decision-making without anticipating disruptions [118]. These systems commonly employ dispatching rules and local optimization techniques [119]. Although highly flexible, reactive approaches may sacrifice global performance and introduce schedule instability and increased computational effort during execution [120].

Hybrid approaches combine proactive robustness with reactive adaptability to leverage the strengths of both paradigms [121]. Typically, a robust baseline schedule is generated and reactive mechanisms are applied only when disruptions exceed predefined thresholds [122]. Empirical studies indicate that hybrid strategies consistently outperform purely proactive or reactive approaches across multiple performance measures [123], offering improved resilience and adaptability [124].

2.5.4 Practical Examples and Applications

The arrival of new jobs during schedule execution is among the most common disruptions in manufacturing systems [125]. Reactive scheduling addresses this challenge through event-driven rescheduling policies that activate when buffer thresholds are exceeded [120]. Advanced methods employ parallel insertion algorithms with adjustment mechanisms to minimize makespan while preserving feasibility under sequence-dependent setup constraints [126].

Reactive scheduling is also critical in handling real-time disturbances such as machine breakdowns, processing time deviations, and changes in resource availability. Threshold-based adaptive control mechanisms monitor system performance and trigger rescheduling when deviations exceed acceptable limits [125]. Virtual queue–based control systems further enhance responsiveness by maintaining real-time system representations and evaluating schedule modifications before implementation [127].

2.6 S-Graph-Based Scheduling Models

The S-graph framework [128] represents a robust and efficient approach for tackling complex scheduling problems, particularly within the context of batch process scheduling. This framework integrates a mathematical model with a directed acyclic graph (DAG) structure, where nodes and arc weights are used to represent complete or partial schedules. It consists of two core components: the S-graph representation, which serves as a model

of the scheduling process, and the associated algorithmic procedures designed for solution generation.

The DAG structure used in the S-graph facilitates a compact and intuitive representation of scheduling problems, where nodes denote the start of tasks or the completion of products, and weighted arcs capture the temporal relationships among them. This modeling paradigm is augmented with advanced algorithmic strategies, including specialized branch-and-bound (B&B) algorithms and combinatorial techniques that enable efficient exploration of the solution space [129].

Initially developed for multipurpose batch process scheduling, the S-graph framework has demonstrated broad applicability across various production scheduling contexts [130]. It has been effectively extended to address resource-constrained project scheduling problems (RCPSP) in both single-mode and multi-mode configurations, and to accommodate scenarios with limited waiting time constraints, where intermediate products have constrained storage durations.

A key advantage of the S-graph approach lies in its efficient representation and flexibility. Its graph-based nature simplifies analysis and enhances the tractability of complex scheduling problems. Compared to generic optimization approaches, the S-graph's tailored algorithms facilitate more rapid and effective search within the solution space [131]. The framework excels in handling sophisticated constraints, including time-varying resource capacities and limited intermediate storage durations. Unlike many heuristic methods, it often yields optimal solutions for small- to medium-sized instances while maintaining the capability to deliver high-quality near-optimal solutions for larger instances. This balance between computational efficiency and solution quality underscores its utility for both academic research and industrial applications. Furthermore, recent extensions have enhanced the framework's adaptability by allowing it to incorporate dynamic resource availability, thereby broadening its applicability to a wider array of real-world scheduling environments.

In the S-graph model, let I denote the set of tasks and P denote the set of products. The complete node set N is partitioned into two disjoint subsets: task-nodes $N^t \subset N$ and product-nodes $N^p \subset N$, such that $N^t \cap N^p = \emptyset$ and $N^t \cup N^p = N$. Task-nodes represent the initiation or execution of tasks, whereas product-nodes denote the completion of products. Each task $i \in I$ is uniquely associated with a task-node in N^t, while each product $p \in P$ corresponds to a distinct product-node in N^p. This establishes the cardinalities $|I| = |N^t|$ and $|P| = |N^p|$.

The S-graph employs two categories of arcs to model scheduling relationships: *recipe-arcs* A_1 and *schedule-arcs* A_2. Recipe-arcs capture the precedence constraints intrinsic

to the production process. For example, if task i must precede task i', a recipe-arc $(i, i') \in A_1$ is drawn between the respective task-nodes.

Each task $i \in I$ must be assigned to one of the available equipment units from the set $J = \{1, 2, \ldots, |J|\}$. Let $J_i \subseteq J$ denote the set of equipment units capable of executing task i. Each equipment $j \in J$ may be able to perform only a subset of the tasks, and the processing time of task i on equipment j is denoted by $pt_{i,j}$. This processing time can vary significantly depending on the chosen equipment, adding complexity to the scheduling process.

The weight assigned to a recipe-arc reflects the minimal processing time required for task i across its feasible equipment set J_i, and is formally defined as:

$$c(i, i') = \min_{j \in J_i} pt_{i,j} \quad \text{for } (i, i') \in A_1. \tag{2.1}$$

This weight can be dynamically updated during optimization, contingent on the specific equipment allocation.

The resulting structure comprising all nodes and recipe-arcs is termed the *recipe-graph*, denoted by $G(N, A_1, \emptyset)$, which serves as the input for the scheduling algorithms. This graph encapsulates the sequence of tasks and their dependencies, alongside specifying the equipment capable of executing each task [128].

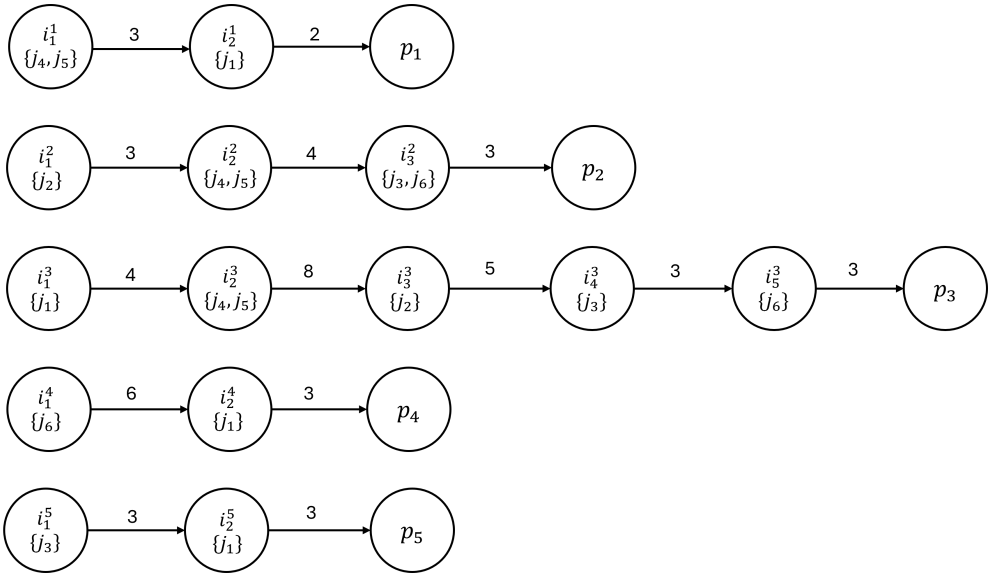
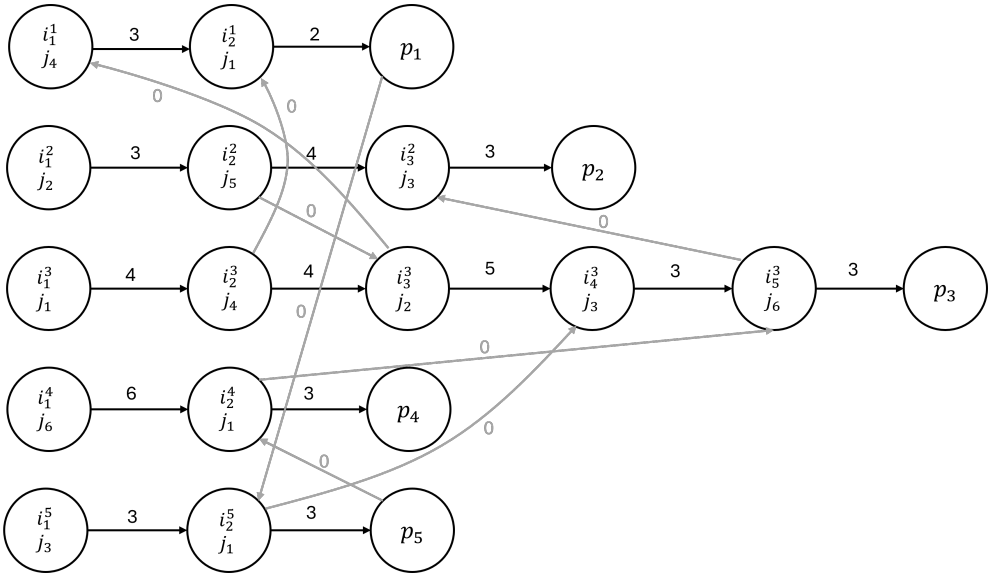


FIGURE 2.1: Recipe-graph for 5 products [128].

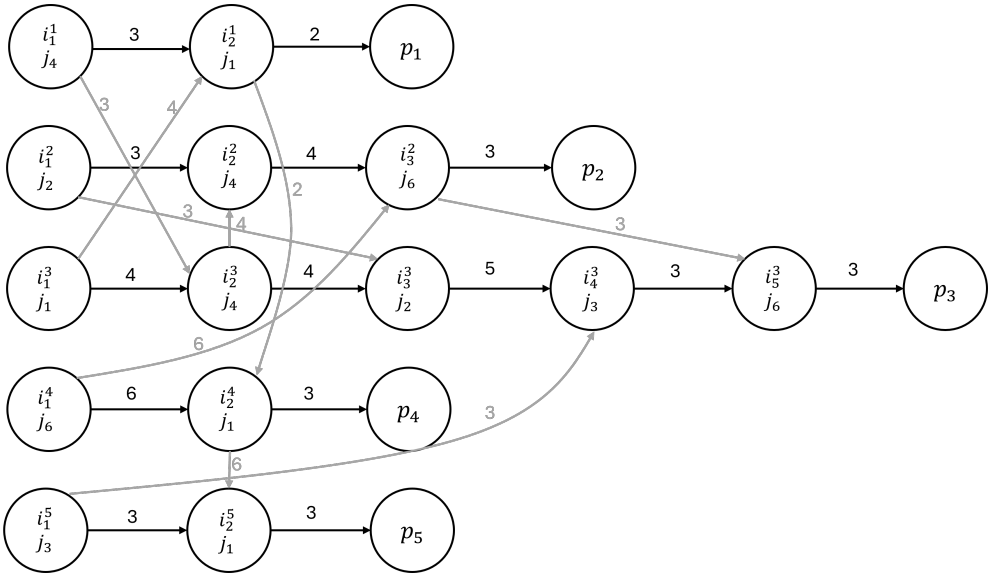
Figure 2.1 illustrates a sample recipe-graph constructed for five products. Task-nodes $(i_1^1, \ldots, i_2^5)$ depict the tasks, while product-nodes $(p_1, \ldots, p_5)$ signify products completions. Directed arcs between task-nodes reflect task precedence, and arcs from task-nodes to product-nodes represent the culmination of production. Each arc imposes a timing

constraint, ensuring correct sequencing. Within each task-node, the set of equipments capable of performing the task is specified.

A similar structure, the *schedule-graph*, models a fully scheduled solution derived from the recipe-graph. Unlike the recipe-graph, which retains general equipment eligibility, the schedule-graph specifies the exact equipment assigned to each task and includes additional arcs—referred to as schedule-arcs—that detail the sequence of tasks on each equipment.



(A): Solution in No Intermediate Storage (NIS) case.



(B): Solution in Unlimited Intermediate Storage (UIS) case.

FIGURE 2.2: Schedule-graphs for the problem illustrated in Figure 2.1 [132].

Figure 2.2 presents two schedule-graphs, based on the recipe-graph in Figure 2.1. The key distinctions include:

- Replacement of equipment sets with specific equipment selected by the scheduling algorithm.
- Introduction of schedule-arcs (depicted in grey) that denote the task sequence on individual equipments.

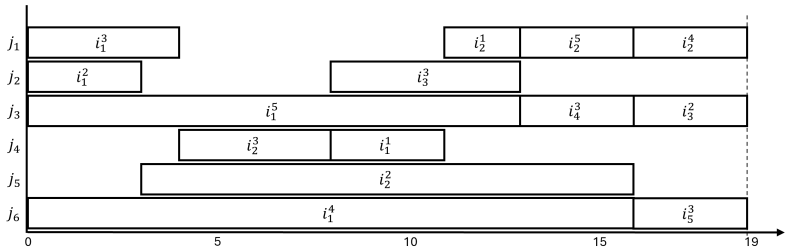
These schedule-arcs are essential for managing task sequencing on shared resources and are influenced by constraints such as the availability of storage for intermediate materials [132].

The availability of intermediate storage plays a critical role in determining the feasible scheduling strategies in production systems [132]. Two principal scenarios are considered in S-graph-based scheduling: no intermediate storage (NIS) and unlimited intermediate storage (UIS).

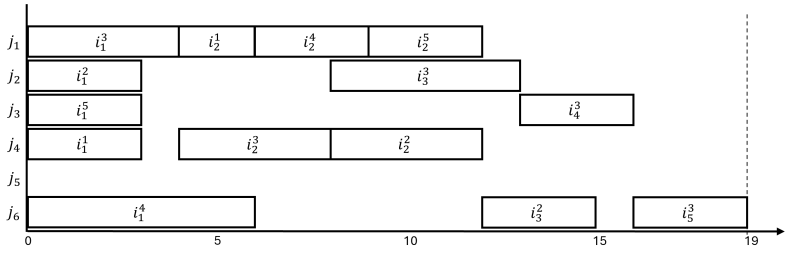
Under the NIS policy, equipment units are subject to strict material handling constraints. Specifically, an equipment cannot commence a new task until the material from its current task has been transferred to the equipment responsible for the subsequent task in the recipe. This imposes a tight coupling between task assignments and material flow. Such dependencies are represented through zero-weighted schedule-arcs in the schedule-graph to enforce sequential task execution without intermediate storage.

As illustrated in Figure 2.2(a), the sequence of tasks under NIS must follow directly from the source node of the recipe-graph. For example, equipment j_1 performs task i_1^3, transfers the output to equipment j_4, and only then begins task i_2^1, as shown by the zero-weighted schedule-arc between i_2^3 and i_2^1.

In contrast, the UIS case allows for material buffering between tasks, providing flexibility in sequencing tasks. Here, schedule-arcs are used to link tasks assigned to the same equipment, and the arc weights correspond to the processing times of tasks. Figure 2.2(b) depicts this configuration, where equipment j_1 performs task i_1^3 and subsequently i_2^1, with the arc weight between them equal to the processing time of i_1^3. In both storage scenarios, schedule-arcs may be adjusted to incorporate necessary changeover times.



(A): Solution in NIS case.



(B): Solution in UIS case.

FIGURE 2.3: Gantt charts for the problem illustrated in Figure 2.1 [132].

Figure 2.3 illustrates the Gantt charts corresponding to the schedule-graphs in Figure 2.2. The Gantt chart for the NIS case (Figure 2.3(a)) shows that tasks are executed sequentially on each equipment, with no intermediate storage allowed. In contrast, the UIS case (Figure 2.3(b)) allows for overlapping task execution on different equipments, reflecting the flexibility provided by intermediate storage. It is important to note that the processing time visualized in Gantt charts under NIS may differ from the timing information in the corresponding schedule-graph due to blocking. For instance, task i_1^5 assigned to equipment j_3 may appear to have a longer execution time because the subsequent task i_2^5 cannot start until equipment j_1 has completed task i_2^1, thereby occupying j_3 for material storage in the interim.

Every schedule-graph shares the same set of nodes and recipe-arcs as its originating recipe-graph $G(N, A_1, \emptyset)$. Once schedule-arcs A_2 are added, the full schedule-graph is represented as $G(N, A_1, A_2)$. Schedule-arcs impose precedence constraints just like recipe-arcs: if $(i, i') \in A_1 \cup A_2$, then

$$t_i^s + c(i, i') \leq t_{i'}^s \quad \text{for } (i, i') \in A_1 \cup A_2, \tag{2.2}$$

where t_i^s denotes the starting time of task i, and $c(i, i')$ represents the arc's weight (e.g., processing or transfer time).

A special class of arcs, called *zero-wait arcs*, enforces immediate succession between tasks. Mathematically, for all zero-wait arcs $(i, i') \in A_{\text{zw}}$, the condition becomes:

$$t_i^s + c(i, i') = t_{i'}^s \quad \text{for } (i, i') \in A_{\text{zw}}. \tag{2.3}$$

To visually differentiate these, diamond-headed arrows are used instead of standard arrowheads [133].

The set $A_{zw} \subseteq A_1 \cup A_2$ formally represents all arcs subject to zero-wait constraints, typically arising in scheduling policies such as NIS (No Intermediate Storage), where intermediate buffering is disallowed.

The recipe-graph $G(N, A_1, \emptyset)$ defines the structure of the scheduling problem, while a complete schedule-graph $G(N, A_1, A_2)$ represents a fully scheduled solution. A partially scheduled graph $G'(N, A_1, A'_2)$, where $A'_2 \subset A_2$, corresponds to an intermediate state within the solution space.

The scheduling problem is tackled using a branch-and-bound (B&B) procedure, which systematically explores the space of feasible schedules. Each node in the enumeration tree corresponds to a subproblem, represented by an S-graph and a partial assignment of tasks to equipment units. The root of the tree is the recipe-graph $G(N, A_1, \emptyset)$, which contains no schedule-arcs. As the tree is traversed, schedule-arcs are incrementally introduced, transforming the recipe-graph into fully scheduled S-graphs $G(N, A_1, A_2)$ at the leaves.

In each branching step, assignment and sequencing decisions are made by adding new schedule-arcs to the S-graph. This process is governed by two principal strategies, namely, the equipment-based and the task-based approaches [128, 134, 135].

The branching strategy determines how schedule-arcs are added to extend the partial schedule:

- **Equipment-Based Method:** At each branching node, a specific equipment is selected. Child nodes are generated by assigning this equipment to all compatible unscheduled tasks, appending each task to the end of the current sequence on that equipment. This approach maintains a fixed sequencing order for previously scheduled tasks on the selected equipment [128].
- **Task-Based Method:** Conversely, this method selects an unscheduled task and considers assigning it to various eligible equipment units. Each child node places the task in a different position within the unit's production sequence. This allows more flexibility in sequencing, as task insertions can occur at any point [134, 135].

Both methods are capable of finding optimal solutions. Their relative performance, however, may vary depending on the structure and complexity of the scheduling problem.

For each partial problem, the bounding procedure estimates whether it can lead to a feasible solution and, if so, computes a lower bound for the makespan. This is accomplished in two steps:

- **Feasibility Check:** The current S-graph is checked for cycles. A cycle indicates that the partial schedule contains contradictory constraints and is thus infeasible. In such cases, the corresponding branch is pruned.
- **Lower Bound Computation:** If the S-graph is acyclic, the longest path in the graph provides a lower bound on the makespan. This value represents the earliest possible completion time of the schedule, given the current set of decisions.

The branch is further explored only if its lower bound does not exceed the best known solution's makespan. Otherwise, it is eliminated from the search, improving computational efficiency.

The solution process continues until all feasible schedule-graphs are generated or the optimal solution is found. A solution is defined as a schedule where all tasks are assigned to equipments and all activity orders are determined. Feasible solutions must be acyclic and comply with the processing and precedence constraints.

This systematic exploration and pruning mechanism ensures that the algorithm can efficiently navigate the solution space, balancing exhaustive search with intelligent decision-making.

Chapter 3

Problem definition

The problems explored in this work belong to the FJSP category, incorporating routing flexibility and processing time variability. The formulations are further extended to accommodate real-time disruptions and optimize time-based penalties, making them highly relevant for modern manufacturing environments.

Let P denote the finite set of products to be scheduled for production. Each product $p \in P$ is defined by a set of tasks I_p, and the complete task set across all products is denoted by $I = \bigcup_{p \in P} I_p$. Each task $i \in I_p$ may have a set of prerequisite tasks $I_i^- \subseteq I_p$ that must be completed before it can begin, defining a directed acyclic graph (DAG) over the task set to preserve feasibility and logical progression. Let i^+ and i^- denote the immediate successor and predecessor of task i in the recipe graph, respectively, if they exist.

The system includes a set of available equipment units, J, and each task i can be processed on a subset of equipments $J_i \subseteq J$. The inverse relationship defines I_j as the set of tasks that can be executed on equipment j. The processing time of task i on equipment j is denoted by $pt_{i,j}$, and this value may vary across equipments, reflecting the flexibility inherent in the FJSP formulation.

A feasible schedule is represented as a set of tuples of the form $S = \{(i, j, t_i^s, t_i^f)\}$, where task i is assigned to equipment j in the time interval $[t_i^s, t_i^f]$. This set of assignments is denoted as S, and each tuple must satisfy fundamental feasibility constraints, such as $j \in J_i$ and $t_i^f - t_i^s \geq pt_{i,j}$. Additional constraints include precedence requirements and resource availability constraints that ensure no equipment processes multiple tasks simultaneously.

This foundational framework serves as the basis for the specialized problem variants addressed in the subsequent sections, each incorporating distinct operational considerations and objectives relevant to different manufacturing scenarios.

Two material flow policies are considered in this scheduling context:

- No Intermediate Storage (NIS): Intermediate materials must remain on the equipment that produced them until the next task in the recipe starts processing them. This effectively blocks the equipment, as it cannot begin a new task until the successor operation has started. Consequently, the schedule must ensure a tight alignment between consecutive tasks to avoid equipment idle time caused by material holdover.
- Unlimited Intermediate Storage (UIS): Intermediate buffering is permitted, allowing more flexible scheduling and reducing equipment idle times.

3.1 Reactive Scheduling with Dynamic Order Arrivals

In practice, production systems do not operate under static conditions. New customer orders may arrive at unpredictable times, requiring the system to adapt ongoing schedules without restarting from scratch. This reactive scheduling capability is essential for maintaining operational efficiency in dynamic manufacturing environments.

Let $O = o_1, o_2, \ldots, o_n$ be the ordered set of production orders. Each order o_k is characterized by the tuple (I_k, t_k), where I_k is the set of tasks associated with the order and t_k is its arrival time. We assume that all arrival times are unique, i.e., $t_{k_1} \neq t_{k_2}$ for $k_1 \neq k_2$, and without loss of generality, we set $t_1 = 0$ for the first order.

The union of tasks required by the first k orders is denoted as $I_k^* = \bigcup_{k' \leq k} I_{k'}$, and the corresponding schedule is S_k. When a new order o_{k+1} arrives at time t_{k+1}, the scheduler must generate a new schedule S_{k+1} that integrates the new order while preserving consistency with the portion of the schedule already in execution.

We define the subset of tasks already started or completed by time t_{k+1} as $\underline{I}_k^*$ and the tasks not yet started as $\overline{I}_k^*$, with $I_k^* = \underline{I}_k^* \cup \overline{I}_k^*$ and $\underline{I}_k^* \cap \overline{I}_k^* = \emptyset$. Accordingly, the current schedule S_k is partitioned into:

- $\underline{S}_k$: tasks that began before t_{k+1} and must remain unchanged,
- $\overline{S}_k$: tasks scheduled to start after t_{k+1}, which can be modified.

The objective is to generate a new feasible schedule S_{k+1} for all tasks up to and including the newly arrived order o_{k+1}, under the following constraints:

- The schedule must be feasible for all tasks in $I_{k+1}^* = \bigcup_{k' \leq k+1} I_{k'}$.

- The schedule must preserve all assignments in $\underline{S}_k$, i.e., $S_{k+1} \supseteq \underline{S}_k$.
- All newly scheduled tasks must start at or after t_{k+1}: $\forall (i, j, t_i^s, t_i^f) \in S_{k+1} \setminus \underline{S}_k$, it must hold that $t_i^s \geq t_{k+1}$.
- Among all feasible schedules satisfying the above constraints, S_{k+1} should minimize the makespan.

This formulation captures the essence of reactive scheduling under dynamic order arrivals, providing a foundation for developing algorithmic approaches capable of adapting to real-time changes in production environments.

3.2 Due Date and Intermediate Storage Minimization

The primary objective in this variant is to develop an efficient schedule that minimizes the total penalty incurred due to earliness and tardiness of product completion times. This involves optimizing task-to-equipment assignments and task sequencing while satisfying all technological and temporal constraints.

Each product $p \in P$ has a due date d_p, which may be externally defined or calculated based on processing requirements. A product's completion time is denoted by C_p. Earliness is calculated as $E_p = \max(0, d_p - C_p)$ and occurs when a product finishes ahead of schedule, while tardiness is $T_p = \max(0, C_p - d_p)$ and occurs when a product finishes late.

The problem is characterized in the classical three-field notation as $FJ \mid prec \mid \sum(E_p + T_p)$, where:

- FJ indicates the Flexible Job Shop environment with equipment-dependent processing times;
- $prec$ denotes the precedence constraints among tasks within each product;
- The objective $\sum(E_p + T_p)$ aims to minimize the total earliness and tardiness penalties across all products.

The overall objective function becomes:

$$\min \sum_{p \in P} (w_p^E E_p + w_p^T T_p), \tag{3.1}$$

where w_p^E and w_p^T are the penalty weights associated with the earliness and tardiness of product p, respectively. These weights can vary between products based on operational

priorities, such as delivery urgency or contractual obligations. Tardiness is typically penalized more heavily, as it directly impacts customer satisfaction and business reliability.

In cases where due dates are not explicitly provided, they are calculated using an estimated lower bound on processing time [136, 137], scaled by a factor f:

$$d_p = \left\lfloor f \sum_{i \in I_p} \min_{j \in J_i} pt_{i,j} \right\rfloor \quad \forall p \in P. \tag{3.2}$$

Here, f serves as a tightness factor that reflects the acceptable slack in the schedule. A higher f value allows more leniency and reduces the chance of infeasibility, while a lower f increases realism by incorporating system congestion. Following established literature [138, 139], a value of $f = 1.3$ is adopted in this study to strike a balance between tractability and practical realism.

The storage policies impose distinct temporal constraints on task execution:

- NIS: $\forall i \in I, \quad t^s_{i'} = t^f_i$ (if equipment available) and $\forall i \in I, i' \in I^-_i, \quad t^s_i = t^f_{i'}$
- UIS: $\forall i \in I, \quad t^s_{i^+} \geq t^f_i$ and $\forall i \in I, i' \in I^-_i, \quad t^s_i \geq t^f_{i'}$.

Minimizing earliness and tardiness penalties is a fundamental objective in due-date–driven scheduling. However, an exclusive focus on E/T optimization may lead to suboptimal resource utilization. In particular, a schedule may achieve low E/T penalties by aligning only the final operation of a product with its due date, while allowing large temporal gaps between successive operations. Although such solutions satisfy due-date requirements, they can introduce extended idle periods for machines and materials.

These idle gaps give rise to intermediate storage time (IST), during which partially completed products wait between consecutive operations. Excessive IST increases material handling and storage costs, prolongs work-in-process inventory, and contributes to inefficient machine utilization. Therefore, minimizing E/T penalties alone is insufficient to ensure efficient production flow.

To address this issue, the scheduling problem is extended to jointly minimize both due-date deviations and intermediate storage time. This integrated objective promotes tighter coordination between successive operations, ensuring that products are completed close to their due dates while avoiding unnecessary storage-induced delays.

Formally, the problem is classified as:

$$FJ \mid prec, \text{ IST-policy} \mid \sum (E_p + T_p) + IST,$$

where *IST-policy* specifies the applied intermediate storage policy, namely No Intermediate Storage (NIS) or Unlimited Intermediate Storage (UIS).

The combined objective function is defined as:

$$\min \left(\alpha \cdot IS_{\text{total}} + \beta \cdot \sum_{p \in P} (E_p + T_p) \right), \tag{3.3}$$

where $\alpha, \beta \geq 0$ are weighting coefficients controlling the trade-off between intermediate storage minimization and due-date performance. A common normalization is $\alpha + \beta = 1$, though alternative scalings may be adopted to reflect domain-specific priorities.

The total intermediate storage time is computed as:

$$IS_{\text{total}} = \sum_{i \in I} is_i, \tag{3.4}$$

where is_i denotes the storage time incurred between the completion of task i and the start of its immediate successor within the same product. The exact value of is_i depends on the relative timing of consecutive tasks and the applied storage policy.

3.3 Integrated Reactive Scheduling with Due Dates and Intermediate Storage

The integrated formulation combines the dynamic aspects of reactive scheduling with the time-penalty optimization of due date considerations, creating a comprehensive model for realistic manufacturing scenarios. In dynamic manufacturing systems, characterized by the unpredictable arrival of new orders, ensuring schedule feasibility while minimizing performance penalties, such as earliness, tardiness, and intermediate storage duration poses additional challenges. This section extends the due-date–driven scheduling methodology by incorporating the challenges of real-time order arrivals, while also minimizing IST under different storage policies.

The production system operates under a dynamic arrival process where orders $O = \{o_1, o_2, \ldots, o_n\}$ arrive over time. Each order $o_k = (I_k, t_k)$ contains a product-specific task set $I_k \subseteq I$ and has an associated arrival time t_k. We assume all arrival times are distinct and $t_1 = 0$ without loss of generality. Let $I_k^* = \bigcup_{k' \leq k} I_{k'}$ denote the cumulative set of tasks from the first k orders.

Each product $p \in P$ is associated with a due date d_p, either explicitly specified or computed based on the task processing time lower bound using Equation 3.2.

At time t_{k+1}, when a new order o_{k+1} arrives, the current schedule S_k is partitioned into:

- $\underline{S}_k$: the set of assignments that have started before t_{k+1}, and are preserved as fixed assignments in the revised schedule;
- $\overline{S}_k$: the set of assignments that have not yet started and may need to be rescheduled.

Let $\underline{I}_k^*$ and $\overline{I}_k^*$ be the corresponding task sets, such that $I_k^* = \underline{I}_k^* \cup \overline{I}_k^*$.

The reactive scheduling problem considered here involves determining an updated schedule S_{k+1} upon the arrival of a new order o_{k+1} at time t_{k+1}, such that E/T and IST across all jobs are minimized. The multi-objective function is expressed as:

$$\min\left(\alpha \cdot IS_{\text{total}} + \beta \cdot (E_{\text{total}} + T_{\text{total}})\right), \tag{3.5}$$

where $\alpha, \beta \geq 0$ are weighting coefficients controlling the trade-off between the two objectives. Often, these are normalized such that $\alpha + \beta = 1$ to facilitate balanced decision-making.

The E/T penalties are computed based on the completion time C_p of the final task $\hat{i}_p$ of each product $p \in P$:

$$E_{\text{total}} = \sum_{p \in P} \max(0, d_p - t^f_{\hat{i}_p}), \tag{3.6}$$

$$T_{\text{total}} = \sum_{p \in P} \max(0, t^f_{\hat{i}_p} - d_p), \tag{3.7}$$

while the total intermediate storage time is given by:

$$IS_{\text{total}} = \sum_{i \in I} is_i. \tag{3.8}$$

The objective is to construct a new feasible schedule S_{k+1} satisfying the following:

- All tasks from $I_{k+1} \cup \overline{I}_k^*$ are rescheduled, maintaining precedence and resource constraints;
- Assignments in $\underline{S}_k$ remain unchanged, i.e., $\underline{S}_k \subseteq S_{k+1}$;
- No task begins before its order's arrival time: $t^s \geq t_{k+1}$ for all new assignments $t^s_i \geq t_{k+1}, \forall i \in I_{k+1}$;
- The multi-objective function combining IST and E/T penalties is minimized.

The scheduling environment adheres to one of the two intermediate storage policies described in the previous section, each imposing the corresponding temporal constraints on task execution and inter-operation coordination.

This integrated problem formulation captures the complexity of modern production systems by combining dynamic order arrivals, task-equipment flexibility, precedence constraints, E/T penalties, and realistic storage behavior into a unified scheduling model. The aim is to develop robust, scalable, and efficient solution methods capable of addressing both the reactivity and optimization challenges inherent in contemporary manufacturing environments.

Chapter 4

Reactive Scheduling with Makespan Minimization

In this chapter, a class of scheduling problems is addressed that arise in flexible job shop environments, where each task can be processed on multiple equipments with varying processing times. This environment reflects real-world manufacturing systems, where flexibility in resource allocation must be balanced against timing constraints and production efficiency. The foundational problem formulation is established in Chapter 3, specifically the reactive scheduling framework described in Section 3.1. My focus is on reactive scheduling, particularly the scenario where new production orders arrive dynamically during the execution of an existing schedule [106].

4.1 Methodology

Reactive scheduling refers to the set of strategies used to adjust production schedules in response to unplanned disturbances—such as the arrival of new orders, equipment breakdowns, or changes in processing parameters. Its primary role is to maintain both feasibility (i.e., no task or equipment conflict) and schedule performance (such as low makespan or tardiness). Within this context, *rescheduling* specifically denotes the process of modifying an existing schedule when such events occur.

Reactive scheduling approaches vary in the degree to which the original schedule is preserved. In this work we examine three distinct policies:

- **Policy 1 (Append-only)**: The schedule for all existing tasks is frozen. Any tasks belonging to the newly arrived order o_{k+1} are appended to the end of the schedule without affecting previously planned operations.

- **Policy 2 (Partial Insertion)**: The planned order among tasks in the set $\overline{I}_k^*$ (those not yet started at the arrival time t_{k+1}) is preserved. However, insertion of new tasks from o_{k+1} is allowed between them. This policy splits into:
  - **Policy 2.1 (Strict Insertion)**: The timing of existing tasks remains strictly unchanged—i.e. $S_k \subseteq S_{k+1}$—so new tasks must fit into pre-existing idle gaps.
  - **Policy 2.2 (Flexible Delay)**: While the order and assigned equipments of $\overline{I}_k^*$ tasks remain fixed, their start times may shift later to accommodate the insertion of new tasks.
- **Policy 3 (Full Rescheduling)**: All tasks in $\overline{I}_k^*$ are fully reschedulable—both their sequence and timing are subject to change to optimally integrate the new order. In this policy, equipment assignments are also released and can be reallocated, meaning that the scheduling algorithm may select different machines for tasks compared to the original plan.

These three policies form a spectrum from conservative (Policy 1) to flexible (Policy 3), providing a foundation for different levels of reactiveness and computational complexity.

Between Policy 2.2 and Policy 3, a meaningful intermediate option exists that is often studied in the literature: preserve the equipment assignments of tasks in $\overline{I}_k^*$, while allowing their execution sequence to be modified. This policy fixes which equipment each task is assigned to but permits rescheduling their order and start times to better integrate new arrivals.

This corresponds to the well-known problem of job shop scheduling with fixed machine assignments, where the equipment is predefined and the optimization focuses solely on sequencing. Such decomposed approaches are common in flexible job shop scheduling: Brandimarte [140] first determined equipment allocations heuristically and then optimized sequencing with tabu search. Hurink et al. [57] applied integrated tabu search, but also noted that if machine assignments are fixed, the problem reduces to classical job shop scheduling (JSSP).

This intermediate policy can be naturally embedded into the S-graph framework by fixing equipment decisions at the root subproblem while keeping sequencing arcs mutable. It offers a balance between stability and optimization potential, especially in environments where equipment planning is finalized early but sequencing flexibility is still desired.

4.1.1 S-Graph Model of the Current Manufacturing State

To enable rescheduling under the reactive S-graph framework, we represent the system as follows:

Each order o_k corresponds to a recipe graph $G_k(N_k, A_{1k}, \emptyset)$, where nodes N_k are tasks and arcs A_{1k} are precedence constraints. To enforce order release times, we introduce two special types of nodes and arcs, which together ensure that no task from an order can start before its release time t_k:

- A *release node* $n_k \in N_k$ is added for each order o_k, with zero-weight arcs from n_k to every task $i \in N_k$ such that $I_i^- = \emptyset$. These arcs denote that these tasks cannot begin before n_k is activated.
- A global *source node* n_0 is introduced, with an arc of weight t_k from n_0 to each release node n_k. This arc enforces that n_k—and thus the initial tasks of order o_k—can only be scheduled at or after time t_k.

Together, these components enforce the release-time constraint: tasks belonging to an order o_k cannot start before the designated release time t_k, as they depend on n_k, which in turn depends on n_0. Figure 4.1 provides an illustrative scheme of this reactive scheduling S-graph.

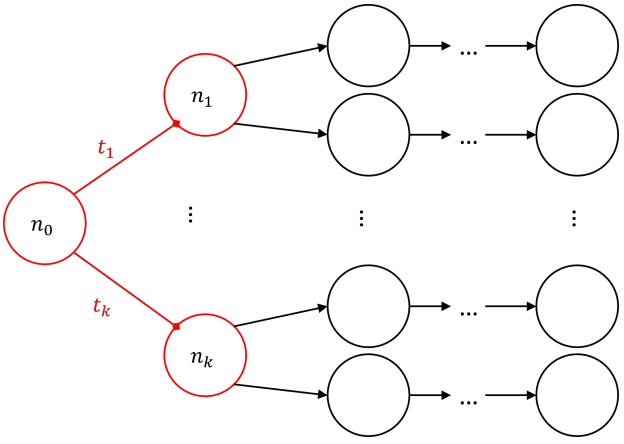


FIGURE 4.1: Reactive scheduling scheme using S-graph representation.

4.1.2 Incorporating New Order Arrivals

When a new order o_{k+1} arrives at time t_{k+1}, we assume that:

- Orders $o_1, \ldots, o_k$ are already scheduled within S_k and execution is underway.

- Tasks in $\underline{I}_k^*$ have already started execution and must remain fixed.
- The new order o_{k+1} must respect the release date constraint $t_s \geq t_{k+1}$ for all its tasks.

These constraints are modeled by:

- Adding zero-wait arcs from the corresponding order release node $n_1, \ldots, n_k$ to each task in $\underline{I}_k^*$, thereby fixing their start times exactly as in the previous schedule. Each arc enforces that tasks associated with earlier orders begin precisely at their previously assigned times.
- Ensuring all tasks belonging to the newly arriving order are connected with release arcs from their order node n_{k+1}, which is itself connected to the source node n_0. This structure enforces that these tasks cannot be scheduled before the release time t_{k+1}.

Figure 4.2 exemplifies this state immediately after the arrival of a new order.

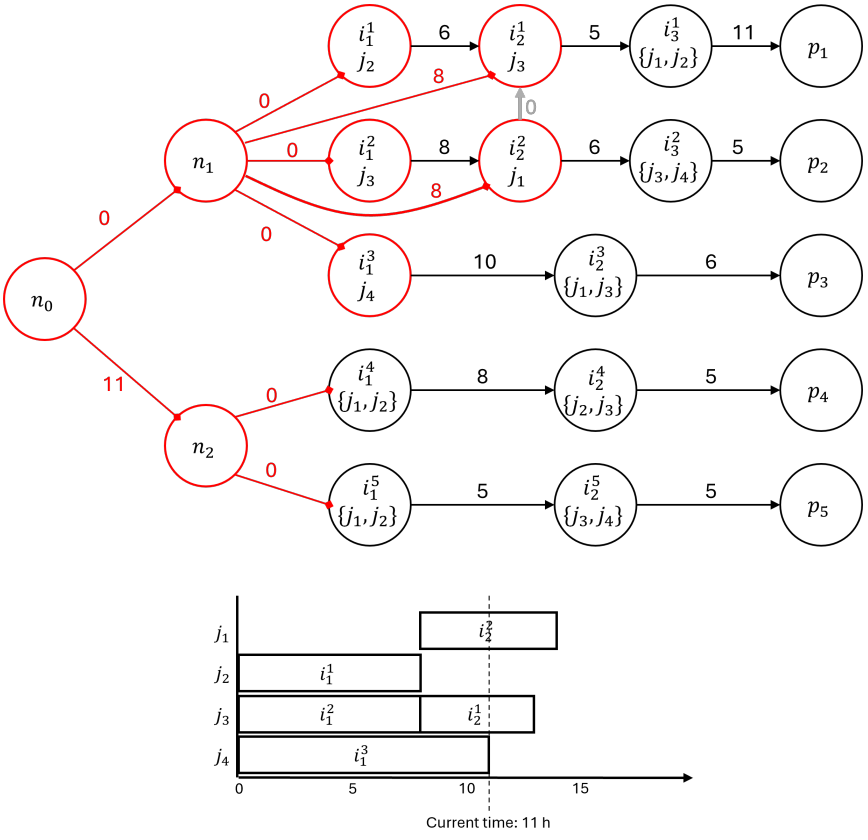
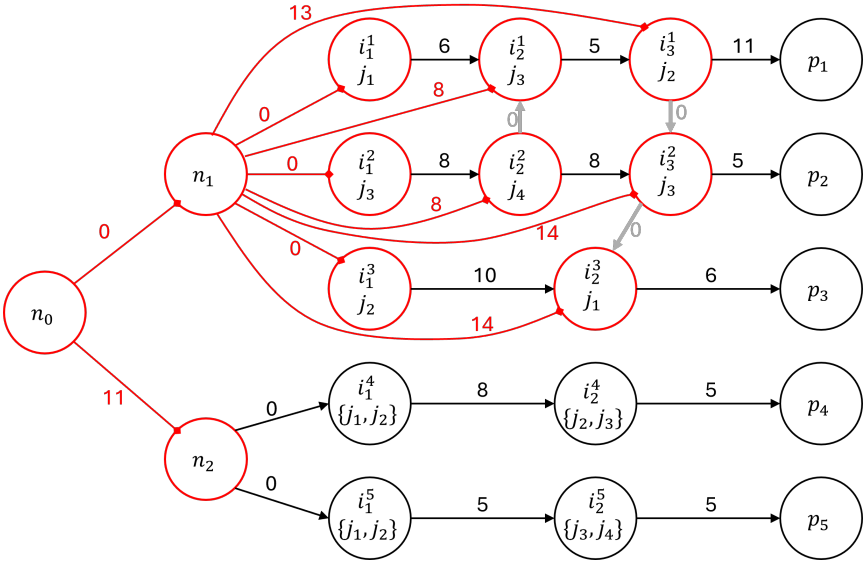


FIGURE 4.2: S-graph representation after arrival of new order o_2 at $t_2 = 11$.

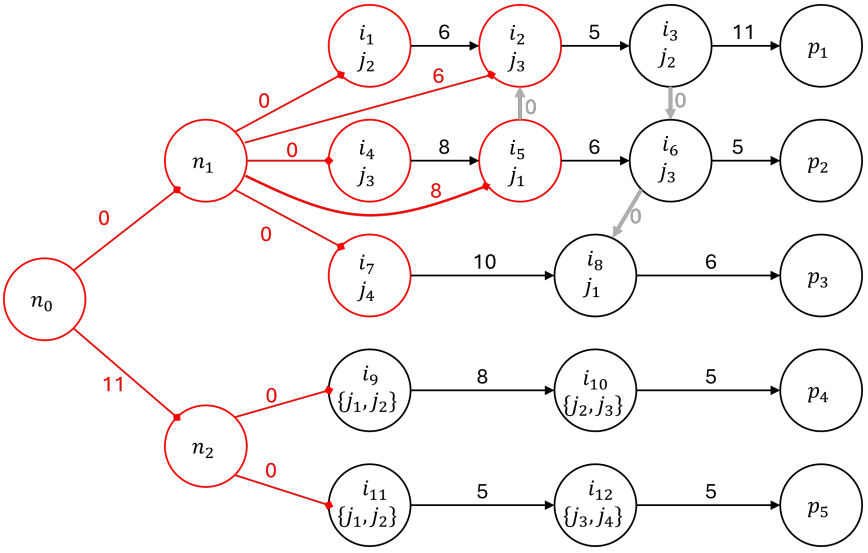
4.1.3 Graph-Based Policy Variants

Each of the scheduling policies 1–3 corresponds to specific S-graph modifications (see Figure 4.3):

- **Policy 1 & 2.1**: All future tasks are fully locked in both sequence and timing. They are depicted as red nodes with zero-wait arcs fixing their start times (see Figure 4.3(a)).
- **Policy 2.2**: Ordering is locked (grey arcs), and tasks begun before t_{k+1} are start-time fixed (red nodes), but remaining tasks may shift in time to permit insertion of new order tasks (see Figure 4.3(b)).
- **Policy 3**: Only started tasks are fixed; the rest of the graph is open for full rescheduling to integrate the new order optimally. The S-graph does not need modification (see Figure 4.2).



(A): Policy 1 and Policy 2.1.



(B): Policy 2.2.

FIGURE 4.3: S-graph structure based on scheduling policy.

4.1.4 Solution Strategies

The initial schedule S_1 is constructed traditionally via S-graph, using either:

- *Equipment-based branching*: Each equipment is considered, and tasks are assigned sequentially.
- *Task-based branching*: Tasks are scheduled one at a time, respecting precedences.

TABLE 4.1: Reactive S-graph approaches for order o_{k+1}.

Policy	Tasks with ZW-arc	Tasks to Schedule	EQ-Based Branching	Task-Based Branching
Policy 1	I_k^*	I_{k+1}	✓	Restricted
Policy 2.1	I_k^*	I_{k+1}	Extended	Restricted
Policy 2.2	$\underline{I}_k^*$	I_{k+1}	Extended	✓
Policy 3	$\underline{I}_k^*$	$I_{k+1} \cup \overline{I}_k^*$	✓	✓

Table 4.1 summarizes the configuration and algorithmic implications of each reactive scheduling policy. The first column identifies the policy type, ranging from the most restrictive (Policy 1) to the most flexible (Policy 3). The second column, **Tasks with ZW-arc**, indicates which tasks are constrained with zero-wait arcs to preserve their exact start times—these tasks cannot be rescheduled.

The third column, **Tasks to Schedule**, lists the subset of tasks that must be handled by the scheduling algorithm after a new order arrives. This includes either just the tasks of the new order or both new and yet-unstarted tasks from earlier orders.

The last two columns show which of the two S-graph scheduling strategies—equipment-based or task-based branching—are applicable under each policy. A ✓ indicates that the method can be applied directly without modification, while "Extended" implies that adaptations are necessary to accommodate insertion logic or preserve task sequence. "Restricted" means the strategy needs constraints to avoid violating policy rules (e.g., preserving fixed sequences).

This table acts as a guideline for selecting the appropriate algorithmic strategy based on the level of reactivity and flexibility permitted in the scheduling scenario.

4.1.4.1 Initialization

Once a reactive event occurs (new order o_{k+1}), the algorithm begins by initializing the branch-and-bound subproblem using the existing schedule S_k. This setup includes:

1. Zero–wait arc insertion

- Add zero–wait arcs from global node Z to each task in $\underline{I}_k^*$ to fix their start times.
- Add zero–wait arcs from node Z to the new order's release node n_{k+1}, weighted by t_{k+1}, ensuring new tasks cannot start before arrival.

2. Task classification

- $\underline{I}_k^*$: tasks already started (immutable).
- $\overline{I}_k^*$: tasks scheduled for after t_{k+1} (mutable under some policies).
- I_{k+1}: tasks of the newly arrived order (must be scheduled).

3. Data structures and root subproblem

Equipment-based branching: Maintain a mapping from each unit j to its last executed task in $\underline{I}_k^*$.

Task-based branching: Each unit maintains an activity list. For reactive initialization, these lists include all tasks in $\underline{I}_k^* \cup \overline{I}_k^*$ in their scheduled order.

Forming root node: The root subproblem includes:

- The current scheduling S-graph, modified according to the selected policy (see below).
- Lists of unscheduled tasks ($\overline{I}_k^*$ and/or I_{k+1}).
- Branching-specific data structures.

The root S-graph is constructed by modifying the original schedule S_k in accordance with the chosen reactive policy:

- Tasks in $\underline{I}_k^*$ (already started) remain fixed. Their equipment assignments and start/end times are preserved via zero-wait arcs.
- Tasks in $\overline{I}_k^*$ (not yet started) are handled as follows:
  - **Policy 1:** Both equipment assignments and scheduling arcs are retained from S_k.
  - **Policy 2.1 / 2.2:** Equipment assignments and task sequence are preserved, but scheduling arcs are modified or removed to permit limited shifting.
  - **Policy 3:** All scheduling and equipment assignments for $\overline{I}_k^*$ are discarded; these tasks are fully reschedulable.
- Tasks in I_{k+1} are added to the graph from their recipe definitions without any scheduling arcs. Their placement will be determined during the search process.

This setup ensures that the root node captures the current production state while allowing appropriate flexibility during branching.

4. Policy-specific initial states

- **Policy 1**: All tasks in $\underline{I}_k^* \cup \overline{I}_k^*$ are fixed. Only I_{k+1} remain unscheduled.
- **Policy 2.1 / 2.2**: Only $\underline{I}_k^*$ are fixed; $\overline{I}_k^*$ remain mutable. I_{k+1} also unscheduled.
- **Policy 3**: Both $\overline{I}_k^*$ and I_{k+1} are unscheduled—eligible for rescheduling.

By embedding this information into the root subproblem, the search is properly initialized for the chosen policy.

4.1.4.2 Algorithms

Both branching methods—equipment-based and task-based—are adapted to respect each policy:

Policy 1

- **Equipment-based:** Operates without changes—new tasks added sequentially after the last assignments.
- **Task-based:** Must be constrained to prevent insertion between scheduled tasks. During branching, any insertion attempts in $\underline{I}_k^* \cup \overline{I}_k^*$ are disallowed.

Policy 2 (2.1 & 2.2)

- **Equipment-based:** Extended to allow insertion of I_{k+1} into the activity lists of $\overline{I}_k^*$, while preserving their sequence. Insert positions must maintain relative ordering.
- **Task-based:**
  - *Policy 2.1:* New tasks may only insert into genuine idle gaps—start times of $\overline{I}_k^*$ cannot be shifted.
  - *Policy 2.2:* Sequence fixed, but $\overline{I}_k^*$ start times can be delayed. Insertion is not restricted by empty slots.

  No modifications are needed to the task-based algorithm itself—only constraints defined by zero-wait arcs and available idle windows applied.

Policy 3

- **Equipment-based:** Works as normal. Both $\overline{I}_k^*$ and I_{k+1} are seen as unscheduled and appended to equipment lists.
- **Task-based:** Also no changes needed—existing tasks are movable within activity lists, subject only to zero-wait constraints from started tasks.

Comparative Remarks

- **Policy 1** has the smallest search space—fastest solving time, but low flexibility.
- **Policy 3** introduces the largest search space—allows maximum optimization at the cost of computational demand.
- **Policy 2** is a compromise between the two: sequence constraints reduce space, but insertions yield better quality solutions.

Graph maintenance After each reaction, completed order fragments (o_k)—their nodes n_k, tasks, and arcs—can be pruned from the S–graph to minimize future search overhead.

4.2 Experimental Setup

To evaluate the effectiveness of the proposed reactive S-graph approach, a C++ implementation was developed and executed on a system equipped with a 2.9 GHz processor and 16 GB of RAM running Windows 11. As the specific class of problems addressed here is not frequently studied in its entirety, several benchmark examples from the literature were adapted—modified slightly to conform to our assumptions. Two such examples are detailed in this section, followed by a summary of additional benchmark comparisons.

4.2.1 Example 1

This example is based on the scheduling problem proposed by Rahmani and Heydari [141], involving a two-machine flow shop configuration. The system consists of two equipment units, j_1 and j_2, with five products to be processed. Each product has two sequential tasks: the first on j_1 and the second on j_2.

In the original study, task durations are uncertain, defined under three scenarios. We selected the second scenario for our experiments, as it was also assumed to have occurred

in practice according to the original authors. While the primary optimization goal in the original study was a weighted metric called MSR (incorporating makespan, stability, and robustness), our focus is on makespan minimization. Hence, although we use the original schedule as the baseline S_1, our evaluation may produce different outcomes.

The initial schedule (Figure 4.4) reflects the use of the UIS policy, where certain tasks on j_1 begin before corresponding materials are transferred to j_2. A new order, o_2, arrives during execution at time $t_2 = 8$, containing one product with two tasks.

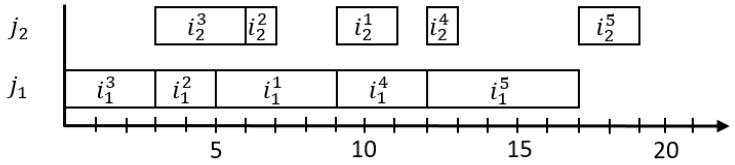


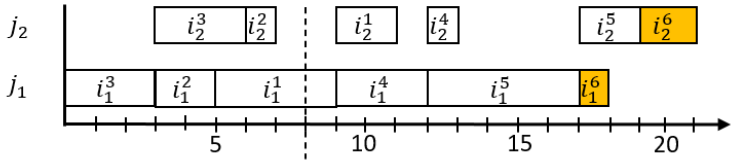
FIGURE 4.4: Initial schedule from literature [141].

At $t_2 = 8$, the ongoing schedule is defined by:

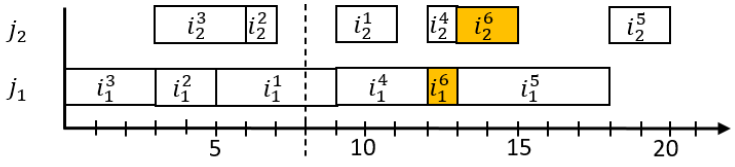
$$\underline{S}_1 = \{(i_1^1, j_1, 5, 9), (i_1^2, j_1, 3, 5), (i_2^2, j_2, 6, 7), (i_1^3, j_1, 0, 3), (i_2^3, j_2, 3, 6)\}$$

Here, tasks i_1^3, i_2^3, i_1^2, and i_2^2 are completed, while i_1^1 is still in progress.

Figure 4.5 shows reactive scheduling results from the original work, and Figure 4.6 displays my results under Policies 1–3.

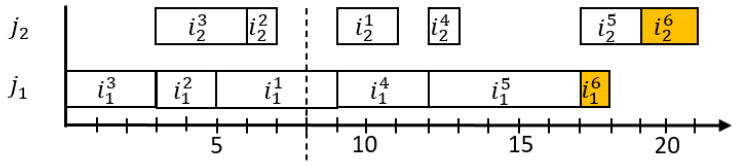


(A): FIFO-based reactive schedule from the literature.

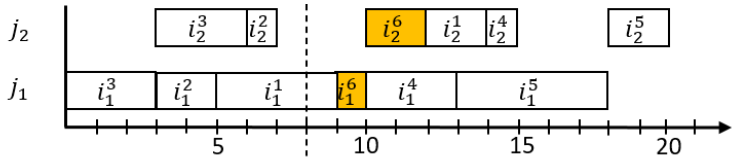


(B): MSR-based schedule from the literature.

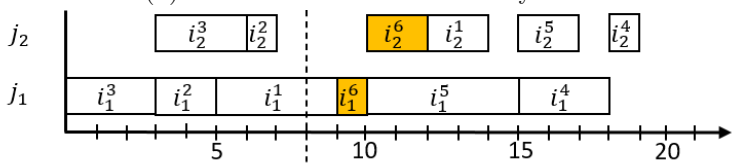
FIGURE 4.5: Reactive schedules from the original study [141].



(A): Reactive S-graph solution under Policy 1 and Policy 2.1.



(B): Reactive schedule under Policy 2.2.



(C): Reactive schedule under Policy 3.

FIGURE 4.6: Proposed S-graph-based reactive schedules for Example 1.

As seen in Figure 4.6(a), Policy 1 yields a schedule identical to the FIFO method (Figure 4.5(a)) due to the lack of idle time in j_1. Policy 2.1 also results in the same outcome. Under Policy 2.2 (Figure 4.6(b)), the tasks may be shifted slightly to insert new tasks, producing a solution equivalent to the MSR-based one in Figure 4.5(b). Policy 3 (Figure 4.6(c)) achieves an improved makespan (19), owing to its flexibility in rescheduling tasks not yet started.

4.2.2 Example 2

The second benchmark is drawn from Gao et al. [142], which proposes a two-stage artificial bee colony (TABC) algorithm for rescheduling with job insertion. The initial schedule for order o_1, shown in Figure 4.7, is used as the baseline in the experiments.

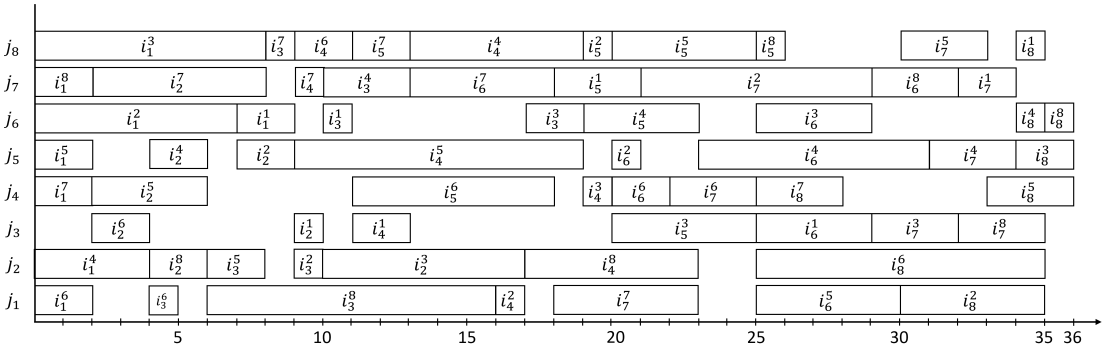
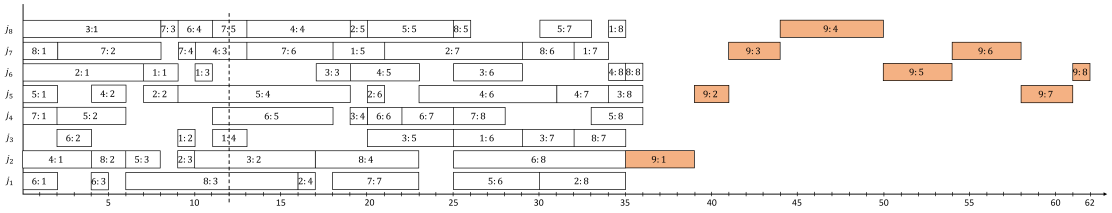
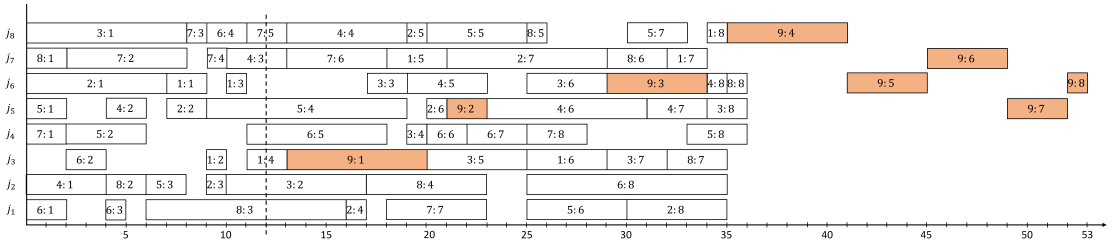


FIGURE 4.7: Initial schedule based on the TABC algorithm [142].

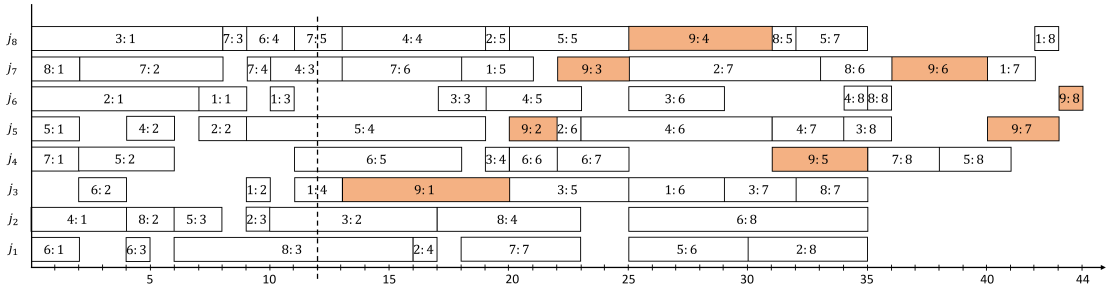
At time $t_2 = 12$, a new job o_2 arrives, consisting of eight tasks. Three strategies are used in the original work, corresponding to the Policies 1, 2.1, and 3. Figure 4.8 shows my scheduling outcomes under these policies. To improve readability, task identifiers in Figures 4.8(a)–(d) are shown using a simplified $k:\ell$ notation instead of the original indexed form.



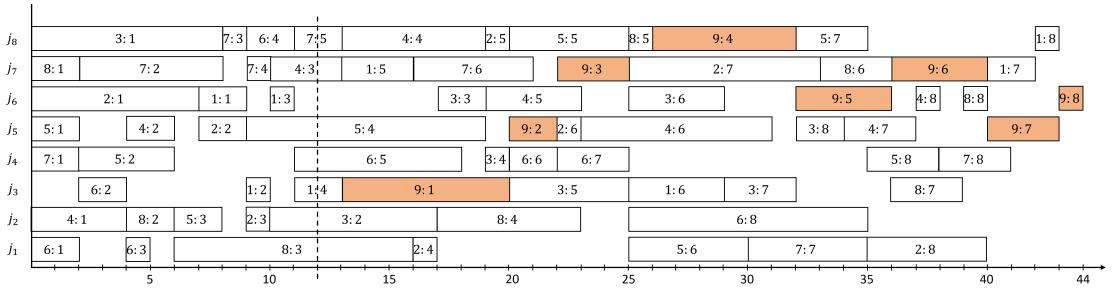
(A): Policy 1 reactive S-graph schedule.



(B): Policy 2.1 reactive S-graph schedule.



(C): Policy 2.2 reactive S-graph schedule.



(D): Policy 3 reactive S-graph schedule.

FIGURE 4.8: Reactive scheduling results for Example 2.

The makespans and computational times are compared in Table 4.2. My Policy 1 solution slightly outperforms Strategy I in makespan. Policies 2.1 and 2.2 improve schedule flexibility and reduce idle time. Notably, Policy 3 matches the optimal result of Strategy II with a makespan of 44, but its complexity leads to higher CPU usage in the task-based method.

TABLE 4.2: Comparison of the results for Example 3 (Strategies I, II, and III are equivalent to Policy 1, 3, and 2.1, respectively, and Strategy III is similar to Policy 2.2 and equivalent to Policy 2.1).

Algorithm	Strategy	CPU time [s]	Makespan
TABC algorithm [142]	Strategy I	0.235	63
	Strategy II	2.343	44
	Strategy III	0.419	53
Task-based	Policy 1	0.013	62
	Policy 2.1	0.083	53
	Policy 2.2	0.073	44
	Policy 3	487.253	44
Equipment-based	Policy 1	0.005	62
	Policy 2.1	0.156	53
	Policy 2.2	2.307	44
	Policy 3	2.270	44

4.2.3 Additional Literature Comparisons

This section presents additional literature examples in a condensed form. These problems do not fully align with the classes of problems previously examined. Most of the examples are multi-objective optimization problems; therefore, they only serve as the foundation for the initial schedule used in each case. Furthermore, multiple types of events were taken into account, e.g., new order arrivals, machine breakdowns, changes in due dates, rush order arrivals, or order cancellations. Because the events occurred at different times in the examples, only the new job arrivals were considered. A summary of the results can be seen in Table 4.3. The makespans of the examples only indicate that the proposed S-graph method can generate solutions that are not worse than the formerly published ones.

A greedy randomized adaptive search for dynamic rescheduling is presented in Baykasoğlu et al. [143]. They solved flexible job shop problems (FJSPs) and dynamic flexible job shop problems (DFJSPs) with machine capacity limitations and a sequence-dependent setup time. A lexicographic method was used for the four objectives, minimizing the mean tardiness, the instability, the makespan, and the mean flow time, respectively.

Duan et al. [144] used a robust optimization technique with dynamic events, such as machine failures and arrival of new jobs, using makespan minimization as the objective function. The paper contains multiple examples; we used the DFJSP example for

comparison. Full rescheduling (Policy 3) with the S-graph method was able to obtain the same result.

A multi-objective mathematical model for DFJSPs was published by Fattahi et al. [145]. The effectiveness and the stability of the schedules are taken into account as two goals. However, the published method can handle multiple objective functions; the authors present an example where they only minimized the makespan. This example has been selected for the purpose of comparison. Four strategies (Strategy 1, 2, 3, and 4) were presented in the paper to solve the problem.

- The new jobs are scheduled after the existing schedule is finished in the case of Strategy 1. Strategy 1 is more strict than Policy 1 because, despite the free capacity of equipments, the new jobs cannot start before finishing the whole initial schedule.
- Using Strategy 2, each equipment must complete all operations in the initial schedule before executing the new jobs. Strategy 2 is equivalent to Policy 1.
- The third strategy means simultaneously rescheduling the new jobs and the ongoing operations based on a single objective. Strategy 3 is equivalent to Policy 3.
- Strategy 4 is the same as Strategy 3 but it is based on the bi-criteria model.

Focusing on job shop rescheduling problems in a dynamic environment with random job arrivals and machine breakdowns, Hao et al. [146] provided a rescheduling solution utilizing mathematical modeling, as well as an interactive adaptive-weight evolutionary algorithm. Three jobs and three equipments make up the proposed problem. Because the original example has two objectives, the S-graph generated makespan is better than the published one.

Li et al. [147] proposed an energy-conscious optimization method for flexible machining job shop problems taking into account dynamic events. Actual machining and equipment idling/stand-by are taken into account by the method when calculating equipment tool energy consumption. The non-dominated sorting genetic algorithm II (NSGA-II) approach is used to obtain a solution that takes into account the overall energy consumption and the makespan. The presented test case differs from the previous ones because the recipe is not sequential, i.e., an operation can have multiple preceding operations. In the paper, two strategies were created. Strategy 1 prevents changing the original assignment and sequence of the unprocessed operations (equivalent to Policy 2.1), and Strategy 2 allows for changing the original schedule (equivalent to Policy 3). The solutions for the two strategies are equivalent to the solutions of the S-graph approach.

Moghaddam et al. [148] proposed a mixed integer programming model with rescheduling based on the concept of dynamic pegging in multi-level production for new order

arrival. The information about how jobs are assigned to orders is known as pegging. Pegging is generated at the start of a planning cycle and is fixed until the start of a new planning cycle, i.e., fixed pegging. On the other hand, in dynamic pegging, the job order assignment may change due to interruptions, like the arrival of new orders during the planning horizon. Products are assumed to have multi-level structures that are represented by product structure trees. The objective is to minimize the equipments' overall idle time costs and the total makespan. Both fixed pegging–rescheduling (FPRS) and fixed pegging–fixed schedule (FPFS) are used. A new schedule for all orders (original and new) is created in FPRS based on the pegging of the original schedule upon the arrival of a new order. In FPFS, a new schedule is made specifically for a new order upon its arrival using the idle time of the equipments in the initial schedule.

Muhamad et al. [149] discussed a clonal selection principle-based rescheduling method for job shop and flexible job shop scheduling problems. Machine breakdown and the arrival of new jobs were the two causes that necessitated rescheduling. Four strategies were introduced in the paper, with example data for new job arrivals.

i. Scheduling the new jobs after the current jobs are finished (Policy 1).

ii. Scheduling the new jobs immediately and continuing with current jobs after new jobs are finished (rush orders).

iii. Inserting the new jobs into idle time while the current jobs are being finished (Policy 2.1).

iv. Scheduling the new jobs immediately and inserting the current jobs into idle time during completion of current jobs (rush order combined with Policy 2.2).

The S-graph reactive method can produce equivalent results for the same policies.

Caldeira et al. [150] proposed a slack-based insertion rescheduling strategy to handle new job arrivals while minimizing makespan, energy consumption, and instability. Every idle time of the initial schedule is considered during rescheduling, including the idle time, partial slack, and total slack of each operation. The partial slack time of an operation is the time window when the operation can start without delaying other operations. The total slack time of an operation is the time window when the operation can start without increasing the makespan. The authors' proposed rescheduling strategy is equivalent to Policy 2.2.

TABLE 4.3: Results of literature examples.

		Example		S-Graph Reactive Method		
Paper	**Approach**	**Objectives**	**Makespan**	**Policy**	**Makespan**	**Ex.time[s])**
[143]	Greedy randomized adaptive search	mean tardiness, instability, makespan, mean flow time	19	Policy 1	24	0.001
				Policy 2.1	24	0.002
				Policy 2.2	19	0.11
				Policy 3	19	0.12
[144]	Swarm algorithm	makespan	22	Policy 1	28	0.001
				Policy 2.1	28	0.001
				Policy 2.2	27	0.001
				Policy 3	22	0.032
[145]	Genetic algorithm	effectiveness, stability	Strategy 1: 28 Strategy 2: 23 Strategy 3: 19 Strategy 4: 19	Policy 1	23	0.005
				Policy 2.1	22	0.003
				Policy 2.2	20	0.005
				Policy 3	19	0.019
[146]	Evolutionary algorithm	makespan, disruption	117	Policy 1	113	0.001
				Policy 2.1	113	0.001
				Policy 2.2	113	0.002
				Policy 3	113	0.01
[147]	Genetic algorithm	energy consumption, makespan	Strategy 1: 40.1 Strategy 2: 37.6	Policy 1	40.1	0.001
				Policy 2.1	40.1	0.008
				Policy 2.2	40.1	0.009
				Policy 3	37.6	0.075
[148]	MIP model	overall idle time, makespan	FPFS: 24 FPRS: 24	Policy 1	25	0.001
				Policy 2.1	23.5	0.001
				Policy 2.2	23.5	0.004
				Policy 3	23.5	0.016
[149]	Clonal selection principle	completion time	Strategy i:24 Strategy ii:25 Strategy iii:24 Strategy iv:22	Policy 1	24	0.002
				Policy 2.1	24	0.002
				Policy 2.2	22	0.002
				Policy 3	22	0.003
[150]	Backtracking search algorithm	makespan, energy consumption, instability	52	Policy 1	57	0.001
				Policy 2.1	54	0.003
				Policy 2.2	50	0.11
				Policy 3	50	4.923

4.3 Summary

This chapter presented a comprehensive exploration of reactive scheduling within the S-graph framework, addressing the challenges posed by dynamic and uncertain production environments—particularly the unexpected arrival of new orders. The methodology extends the conventional S-graph approach through the development of three reactive

policies, each offering a different level of scheduling flexibility. These policies range from appending new tasks after the existing schedule (Policy 1), to allowing task insertion without altering ongoing operations (Policy 2), and full rescheduling of unstarted tasks (Policy 3).

The proposed approach enables dynamic adaptation without the need to regenerate the entire schedule, significantly improving responsiveness and resource utilization. Through case studies and benchmark comparisons, the S-graph-based reactive method demonstrated competitive—often superior—performance in terms of makespan and flexibility compared to state-of-the-art scheduling strategies from the literature.

Key strengths of the approach include:

- Versatility: Compatible with both S-graph and non-S-graph initial schedules.
- Scalability: Capable of handling large-scale, complex production systems.
- Efficiency: Offers policy-driven trade-offs between computational load and scheduling optimality.
- Practicality: Applicable to real-world scenarios such as waste recycling facilities, where order arrivals are frequent and unpredictable.

By formalizing and integrating reactive scheduling within a structured graph-based representation, this chapter contributes a robust, adaptable, and industry-ready solution to the ongoing challenge of maintaining efficient production in volatile environments.

Chapter 5

Deterministic Scheduling with Due Date and Storage Objectives

This chapter investigates flexible job shop problems (FJSP) arising in manufacturing environments where each product consists of multiple, interdependent tasks. A distinctive feature of this problem is the flexibility in task-to-equipment assignments: each task can be executed on one of several eligible equipments, with processing times depending on the specific equipment selected. This flexibility, combined with product-specific due dates and precedence constraints between tasks, significantly increases the complexity of the scheduling process. The formal problem definition is presented in Section 3.2, which addresses the optimization of E/T penalties under different intermediate storage policies [151].

5.1 Methodology

5.1.1 Extension of the S-graph Framework for Earliness/Tardiness Minimization

The original S-graph framework [128], designed for makespan minimization in batch processing, has been extended in this study to address the minimization of earliness and tardiness (E/T) penalties. These enhancements include modifications of the objective function and the incorporation of additional scheduling considerations, where due dates are modeled through earliness and tardiness penalty terms rather than as hard constraints, alongside time-related sequencing constraints. While maintaining the structural advantages of the original approach, the extended framework supports a richer and more realistic scheduling environment.

In the original B&B algorithm, each node of the enumeration tree corresponds to an S-graph representing a partial assignment of tasks to equipments. The root node begins with a recipe-graph that contains all task precedence constraints but no equipment assignments. As branching progresses, tasks are assigned to specific equipments, producing child subproblems with updated execution constraints. Because processing times are equipment-dependent, assigning a task alters the weights of arcs in the graph, which subsequently affects start and finish times.

To incorporate due dates, each subproblem now also includes estimated temporal parameters—start and finish times—for all scheduled tasks. These are derived using product-specific due dates and guide the algorithm to prioritize assignments that align task completions with target deadlines. In each branching step, two decisions are made:

- Assigning a task to a specific equipment.
- Determining the sequence position of the task.

To make the operation of the extended framework explicit, Algorithm 1 summarizes the core conflict resolution and timing adjustment procedure applied at each branch-and-bound node. The algorithm computes task start and finish times backward from product due dates, detects equipment conflicts induced by sequencing decisions, and resolves them through controlled temporal shifts. If no feasible temporal adjustment exists, the corresponding branch is pruned.

Algorithm 1 Conflict Resolution in S-graph with E/T Minimization

Require: Partial S-graph $G = (N, A_1, A_2)$, due dates $\{d_p\}$, storage policy
Ensure: Feasible timing or infeasibility flag

Initialize start-time matrix using backward propagation from due dates
for all products $p \in P$ **do**
    Set $t^f_{\hat{i}_p} \leftarrow d_p$
    Propagate start times backward along recipe arcs
end for
repeat
    conflictFound $\leftarrow$ **false**
    for all equipment j **do**
        for all task pairs (i, i') scheduled on j **do**
            if $t^s_{i'} < t^f_i$ **then**
                conflictFound $\leftarrow$ **true**
                Compute overlap $\Delta \leftarrow t^f_i - t^s_{i'}$
                if backward shift of i is feasible **then**
                    Shift task i and its predecessors by $-\Delta$
                else if forward shift of i' is feasible **then**
                    Shift task i' and its successors by $+\Delta$
                else
                    return infeasible
                end if
            end if
        end for
    end for
until *conflictFound* = **false**
return feasible timing

Algorithm 1 corresponds directly to the implementation in the S-Graph class. Backward propagation from due dates is realized through the graph start-time matrix construction, while conflict detection and resolution are performed by iteratively adjusting start times along recipe and schedule arcs. The preference for backward or forward shifts reflects feasibility checks implemented in the timing adjustment routines.

The modification of arc weights caused by equipment assignment affects downstream timing. Given that the final task $\hat{i}_p$ of a product p must complete by its due date d_p, its finish time is constrained as:

$$t^f_{\hat{i}_p} = d_p \quad \forall p \in P. \tag{5.1}$$

The corresponding start time of any task i is derived from its finish time and the arc weight connecting it to its successor:

$$t_i^s = t_i^f - c(i, i') \quad \forall (i, i') \in A_1. \tag{5.2}$$

For the final task of each product:

$$t_{\hat{i}_p}^s = t_{\hat{i}_p}^f - c(\hat{i}_p, p) \quad \forall (\hat{i}_p, p) \in A_1. \tag{5.3}$$

And for tasks preceding others within a product, the finish time must match the earliest start time among all immediate successors:

$$t_i^f = \min_{(i,i') \in A_1} t_{i'}^s \quad \forall i \in I. \tag{5.4}$$

Feasibility checks are conducted at each node of the B&B tree using a cycle detection algorithm to prevent precedence violations. If a partial problem is feasible, a lower bound for the E/T metric is calculated.

In addition to recipe constraints, the analysis must also maintain equipment constraints via schedule-arcs. If a task is scheduled on an equipment where another task has already been assigned earlier, overlap or resource conflict may occur. The framework recursively checks all predecessor and successor tasks in both the recipe and schedule sequences to ensure feasibility. When conflicts arise, two outcomes are possible:

- The task's execution is delayed, increasing its start and finish times.
- The preceding tasks' execution intervals are shifted earlier while preserving their processing times.

If no feasible shift is found, the modification is discarded. This dual-layer sequencing ensures that both precedence (via recipe-arcs) and equipment capacity (via schedule-arcs) are respected throughout the solution process.

Storage policies also influence timing:

- Under the **No Intermediate Storage (NIS)** policy, equipments are occupied during material wait times between tasks:

$$t_i^s = t_i^f - (c(i, i') + is_i) \quad \forall (i, i') \in A_1, \tag{5.5}$$

  where is_i denotes the intermediate storage time associated with task i. Under the NIS policy, this storage takes place directly on the equipment, meaning that

the equipment remains occupied during is_i. Consequently, intermediate storage time acts as blocking time in the temporal model, even though it is conceptually identical to storage time.

- Under the **Unlimited Intermediate Storage (UIS)** policy, tasks can start independently:

$$t^s_{i'} \geq t^f_i = t^s_i + c(i, i') \quad \forall (i, i') \in A_1. \tag{5.6}$$

In this case, intermediate storage does not occupy the equipment and therefore does not contribute to equipment blocking.

These conditions integrate material handling dynamics into the temporal modeling and enable more realistic scheduling behavior in constrained environments.

5.1.2 Minimization of Intermediate Storage Time and Earliness/Tardiness

While minimizing E/T is critical, doing so without coordinating task execution may lead to inefficient schedules. Specifically, if only the final task of a product aligns with its due date, earlier tasks may be arbitrarily spaced, leading to excessive idle time and increased intermediate storage time (IST). This situation increases material holding costs and decreases equipment utilization.

To address this issue, the extended S-graph framework jointly minimizes both E/T and IST. This dual-objective model is expressed as:

$$FJ \mid prec, \text{IST-policy} \mid \sum (E_p + T_p) + IS_{\text{total}}. \tag{5.7}$$

The multi-objective optimization function is:

$$\min \left(\alpha \cdot IS_{\text{total}} + \beta \cdot (E_{\text{total}} + T_{\text{total}}) \right), \tag{5.8}$$

where:

- α, β are non-negative weights with $\alpha + \beta = 1$ (or user-defined).
- $IS_{\text{total}} = \sum_{i \in I} is_i$
- $E_{\text{total}} = \sum_{p \in P} \max(0, d_p - t^f_{i^p})$
- $T_{\text{total}} = \sum_{p \in P} \max(0, t^f_{i^p} - d_p)$.

The recipe and equipment sequence examinations introduced in Section 5.1.1 are extended to include intermediate storage minimization. During branching, if task i has nonzero storage time ($is_i > 0$), the framework attempts to reduce it to zero by applying local schedule adjustments—modifying the start and finish times of neighboring tasks without changing the scheduling order. If this reduction is infeasible due to machine or precedence constraints, the algorithm determines and applies the minimal feasible value of is_i that preserves overall feasibility.

Additional scenarios may arise when recipe constraints allow earlier start times, but equipment constraints do not. In such cases, a gap analysis is performed to assess whether task i can be rescheduled without violating feasibility. If the available gap is insufficient, the system attempts to adjust is_i values of adjacent tasks to create enough room for a valid placement.

Balancing E/T and IST requires trade-offs. Minimizing IST improves throughput but may cause E/T penalties, especially under NIS constraints. Conversely, focusing solely on E/T may lead to uncoordinated, storage-heavy schedules. The extended S-graph framework enables flexible prioritization depending on production needs, providing a robust and adaptable scheduling solution.

5.1.3 Integration of the S-graph Framework and an LP Solver for Scheduling

This section introduces a hybrid scheduling approach that combines the graph-theoretic structure of the S-graph framework with the optimization capabilities of a linear programming (LP) solver. The S-graph effectively captures task precedence and sequencing constraints, while the LP solver optimizes equipment assignments and task timings under evolving scheduling conditions. This integration leverages both discrete and continuous optimization techniques, enabling efficient solutions for complex job shop scheduling problems.

The integration process begins by transforming the current S-graph, reflecting the scheduling state at a given decision point—into an LP model. This graph includes not only the recipe-based precedence arcs (A_1), but also equipment sequencing arcs (A_2) that encode decisions made so far. The LP model aims to minimize the total E/T penalties, while observing all relevant constraints on task start and finish times, processing durations, precedence, and equipment sequencing.

The base LP model for minimizing E/T penalties is formulated as:

$$\text{Minimize} \quad \sum_{p \in P} (E_p + T_p) \tag{5.9}$$

Subject to:

$$E_p + t^f_{\hat{i}^p} \geq d_p \qquad \forall p \in P \tag{5.10}$$

$$-T_p + t^f_{\hat{i}^p} \leq d_p \qquad \forall p \in P \tag{5.11}$$

$$t^s_{i'} - t^f_i \geq 0 \qquad \forall (i, i') \in A_1 \cup A_2 \tag{5.12}$$

$$t^f_i = t^s_i + c(i, i') \qquad \forall (i, i') \in A_1 \tag{5.13}$$

$$t^f_{\hat{i}^p} = t^s_{\hat{i}^p} + c(\hat{i}^p, p) \qquad \forall p \in P, (\hat{i}^p, p) \in A_1 \tag{5.14}$$

$$t^s_i \geq 0 \qquad \forall i \in I \tag{5.15}$$

$$E_p, T_p \geq 0 \qquad \forall p \in P. \tag{5.16}$$

The objective minimizes the total E/T penalties across all products. Equations (5.10) and (5.11) define the conditions for earliness and tardiness. The sequencing constraint (5.12) ensures that no task starts before its predecessors are completed. Task finish times are determined by (5.13), while product completion times follow from (5.14).

The hybrid approach is operationalized within the branch-and-bound (B&B) algorithm of the S-graph framework. Each node in the B&B tree corresponds to a subproblem with partial equipment assignments and task sequencing. At each node, the LP model is updated and solved to assess the quality of the partial schedule and inform branching decisions.

Key integration mechanisms include:

- **Equipment Assignment**: When task i is assigned to equipment j, the processing time is fixed in the LP model as $c(i, i') = pt_{i,j}$ for all $(i, i') \in A_1$ involving task i. This updates the relevant constraints (e.g., (5.13), (5.14)).
- **Insertion of Schedule-Arcs**: When tasks i and i' are assigned to the same equipment and a sequencing decision is made (e.g., i precedes i'), a constraint of the form $t^s_{i'} \geq t^f_i$ is added to the LP model (see (5.12)).

To incorporate intermediate storage time (IST) minimization into the LP model, the objective and constraints are expanded:

$$\text{Minimize} \quad \sum_{p \in P} (E_p + T_p + IS_p) \tag{5.17}$$

Subject to:

$$E_p + t^f_{\hat{i}^p} \geq d_p \qquad \forall p \in P \tag{5.18}$$

$$-T_p + t^f_{\hat{i}^p} \leq d_p \qquad \forall p \in P \tag{5.19}$$

$$t^s_{i'} - t^f_i \geq 0 \qquad \forall (i, i') \in A_1 \cup A_2 \tag{5.20}$$

$$t^f_i = t^s_i + c(i, i') + is_i \qquad \forall (i, i') \in A_1 \tag{5.21}$$

$$t^f_{\hat{i}^p} = t^s_{\hat{i}^p} + c(\hat{i}^p, p) + is_{\hat{i}^p} \qquad \forall p \in P, (\hat{i}^p, p) \in A_1 \tag{5.22}$$

$$IS_p = \sum_{i \in I_p} is_i \qquad \forall p \in P \tag{5.23}$$

$$t^s_i, is_i \geq 0 \qquad \forall i \in I \tag{5.24}$$

$$E_p, T_p \geq 0 \qquad \forall p \in P. \tag{5.25}$$

Here, is_i represents the intermediate storage time after task i, and IS_p is the total storage time for product p. Equations (5.21) and (5.22) update finish time calculations to include storage time. Constraint (5.23) aggregates the IST over a product's task sequence.

It is important to note that constraints (5.12) and (5.20) represent the general sequencing relations corresponding to the UIS policy, allowing a successor task to start after the completion of its predecessor. Under the NIS policy, blocking behavior is enforced through zero-wait arcs in the S-graph. For such arcs, the corresponding LP constraint is specialized to an equality of the form

$$t^s_{i'} = t^f_i \quad \forall (i, i') \in A_{\text{zw}}, \tag{5.26}$$

where $A_{\text{zw}} \subseteq A_1 \cup A_2$ denotes the set of zero-wait arcs. Thus, the LP model itself remains unchanged; the distinction between UIS and NIS is captured by the type of arcs present in the S-graph and the resulting equality or inequality constraints added to the LP.

At each node of the B&B tree, the LP model is solved to determine optimal start and finish times based on the current partial schedule. The solution provides valuable insights, such as:

- Identification of critical paths and potential bottlenecks.
- Feasibility assessment of partial assignments.

- Guidance for subsequent branching decisions in the S-graph framework.

This iterative integration ensures that each decision made by the S-graph is grounded in optimal timing and feasibility analysis from the LP model. As the B&B process unfolds, the LP model is dynamically updated, providing a tight lower bound at each node and guiding the algorithm toward high-quality, feasible solutions that jointly minimize E/T and IST penalties.

5.2 Results

The purpose of this section is to thoroughly evaluate the performance and applicability of the proposed scheduling methodology across a series of problem instances. These experiments are designed to demonstrate how the method handles the dual objectives of E/T and IST minimization under different scheduling constraints, task structures, and storage policies. By analyzing results obtained from multiple configurations, the section aims to establish both the effectiveness and the practical limitations of the approach in solving real-world instances of the FJSP with due dates.

All computations were carried out on a personal computer equipped with an Intel(R) Core(TM) i7-7820HQ CPU operating at 2.9 GHz and 16 GB of RAM. The S-graph framework was implemented in C++, and linear programming functionality was provided by the GNU Linear Programming Kit (GLPK) [152], which is integrated into the framework and called directly by the S-graph algorithms.

To demonstrate the model's behavior in detail, a representative example problem is first introduced. Table 5.1 summarizes the parameters of this instance, including task-to-equipment compatibilities and due dates for each product. Each row corresponds to a specific task, identifying which pieces of equipment can process it and the associated processing times. Due dates (d_p) for the final product are also provided. This structure serves as the input for the scheduling scenarios analyzed in the following figures.

The outcomes of applying the proposed methodology to this example are shown in Figures 5.1 and 5.2. These figures present the resulting schedule graphs and Gantt charts under two optimization objectives: E/T minimization alone, and combined E/T and IST minimization, respectively. The schedule graphs display both recipe arcs (capturing task dependencies) and schedule arcs (capturing equipment assignments), while the Gantt charts offer a clear view of the temporal allocation of tasks across equipments. Red markers are used to highlight the start times of tasks, and due dates are labeled for each product.

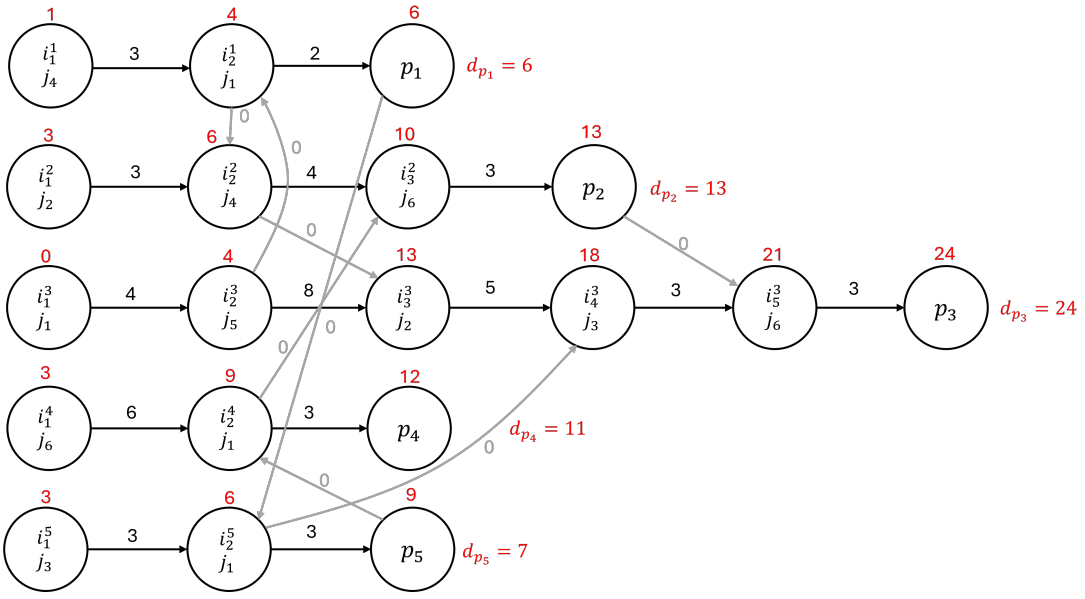
TABLE 5.1: Parameters of problem given in Figure 2.1.

products (p)	task	equipment						d_p
		j_1	j_2	j_3	j_4	j_5	j_6	
product 1	i_1^1	-	-	-	3	7	-	6
	i_2^1	2	-	-	-	-	-	
product 2	i_1^2	-	3	-	-	-	-	13
	i_2^2	-	-	-	4	8	-	
	i_3^2	-	-	3	-	-	3	
product 3	i_1^3	4	-	-	-	-	-	24
	i_2^3	-	-	-	4	8	-	
	i_3^3	-	5	-	-	-	-	
	i_4^3	-	-	3	-	-	-	
	i_5^3	-	-	-	-	-	3	
product 4	i_1^4	-	-	-	-	-	6	11
	i_2^4	3	-	-	-	-	-	
product 5	i_1^5	-	-	3	-	-	-	7
	i_2^5	3	-	-	-	-	-	

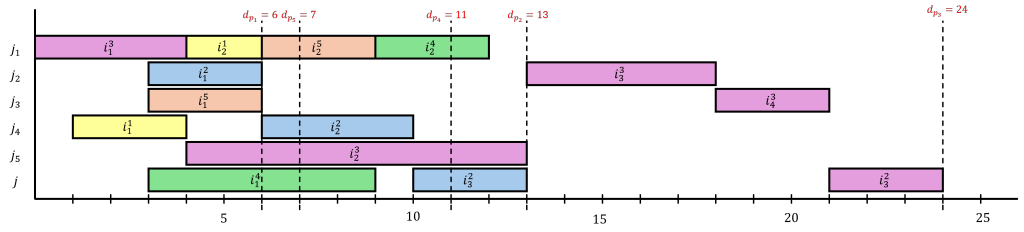
In Figure 5.1, which corresponds to the case of minimizing only E/T, the solution focuses on aligning task completion as closely as possible with product due dates. However, this sometimes comes at the cost of introducing idle periods between tasks. One such case can be observed in the scheduling of task i_3^3. In this solution, task i_3^3 begins at time 13, following a one-unit gap after the completion of its predecessor i_2^3. This gap introduces an intermediate storage period on equipment j_5, representing material that must wait before further processing. The final objective function value for this solution is 3, which consists of a tardiness of 1 for product p_4 and 2 for product p_5.

In contrast, Figure 5.2 shows the result of solving the same problem while also minimizing IST. The optimizer adjusts the schedule to start i_3^3 earlier, thereby eliminating the storage gap between i_2^3 and i_3^3. This leads to a cascading effect, as subsequent tasks for product p_3 must also begin earlier. While this strategy successfully removes IST, it introduces an earliness of 1 and maintains a total tardiness of 3. The resulting objective function value increases to 4. Despite this, the schedule is more compact and avoids unnecessary storage. Interestingly, this same solution is produced by the hybrid S-graph + LP approach, as described in Section 5.1.3, validating the consistency of the methods.

To quantitatively compare the different solution methods, Table 5.2 presents a summary of results for the four main configurations: (1) E/T-only optimization using the S-graph, (2) E/T-only optimization using the S-graph + LP hybrid, (3) combined E/T and IST optimization using the S-graph, and (4) combined E/T and IST optimization



(A): Schedule-graph.



(B): Gantt chart.

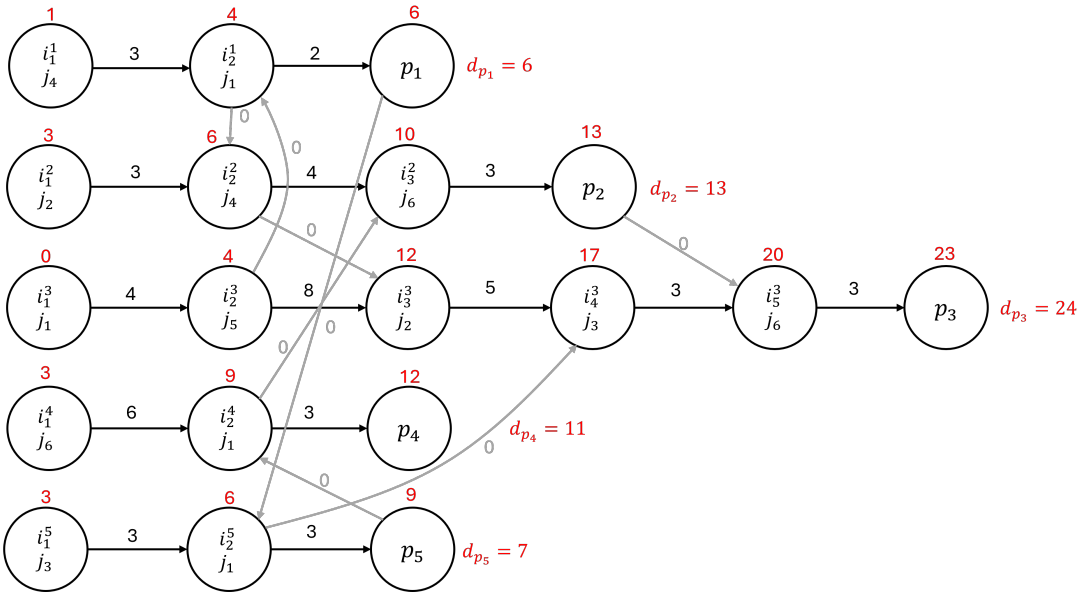
FIGURE 5.1: E/T solution in NIS case of problem Figure 2.1.

TABLE 5.2: The results of problem given in Figure 2.1.

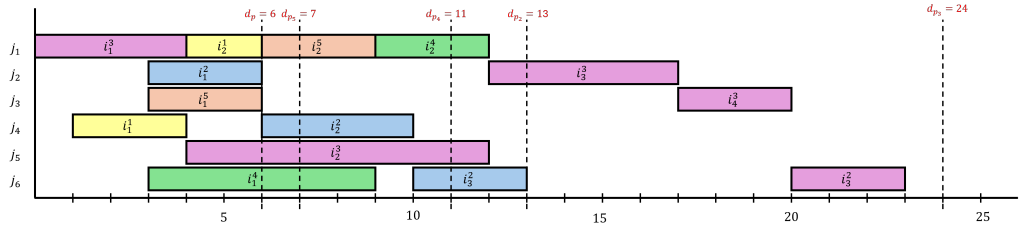
Objective	Method	Time (s)	Branches	$\sum E_p$	$\sum T_p$	$\sum IST$
E/T	S-graph	0.015	522	0	3	-
	S-graph + LP	0.069	679	0	3	-
E/T + IST	S-graph	0.017	735	1	3	0
	S-graph + LP	0.079	726	1	3	0

using the hybrid approach. The table includes total computation time, number of B&B branches explored, and resulting values for earliness, tardiness, and IST.

The results in Table 5.2 reveal several important trends. First, both S-graph and hybrid methods are capable of solving this moderate-sized instance efficiently. The hybrid versions explore more branches, indicating that the additional LP-based refinement of task timing introduces more branching decisions, although solution quality remains consistent. When IST minimization is introduced, the objective value increases slightly due to the added constraint of avoiding intermediate gaps. However, the IST value is successfully reduced to zero, demonstrating that the model effectively handles this added layer of complexity.



(A): Schedule-graph.



(B): Gantt chart.

FIGURE 5.2: E/T and IST solution in NIS case of problem Figure 2.1.

To evaluate the method against external benchmarks, we compare results with a published MILP-based approach proposed by Vital-Soto et al. [153]. The problem structure, processing times, and results from that study are summarized in Table 5.3. A key characteristic of their model is the allowance for parallel precedence structures, increasing the solution space and scheduling flexibility.

The Vital-Soto et al. [153] model reports a cumulative tardiness of 32 units and zero earliness, resulting in a weighted tardiness score of 97. This value is influenced by product-specific weights (w_p), shown in the final column of Table 5.3. The fact that zero earliness was achieved suggests a just-in-time focus, yet the solution incurs considerable lateness for certain products. The comparison is made using the same weight configuration and processing data.

TABLE 5.3: Processing times of tasks on different equipments, product due dates (d_p), and product results (E_p, T_p) [153].

products (p)	task	equipment			d_p	E_p	T_p	w_p
		j_1	j_2	j_3				
product 1	i_1^1	10	-	3	27	0	1	3
	i_2^1	4	-	11				
	i_3^1	-	11	5				
product 2	i_1^2	7	10	13	36	0	19	1
	i_2^2	-	11	-				
	i_3^2	13	6	13				
	i_4^2	9	7	-				
product 3	i_1^3	14	3	-	29	0	3	4
	i_2^3	-	12	8				
	i_3^3	-	-	12				
	i_4^3	6	11	9				
product 4	i_1^4	11	-	15	31	0	9	7
	i_2^4	13	-	13				
	i_3^4	-	7	7				
	i_4^4	9	15	-				

TABLE 5.4: Performance results for different objectives, methods, and IST policies, showing earliness ($\sum E_p$), tardiness ($\sum T_p$), intermediate storage time ($\sum$IST), computation time, and number of branches.

Objective	**Method**	**IST policy**	$\sum w_p E_p$	$\sum w_p T_p$	$\sum$IST	**Time (s)**	**Branches**
E/T	S-graph	UIS	0	97	-	0.222	22431
		NIS	0	144	-	1.112	107178
	S-graph + LP	UIS	0	97	-	7.089	91161
		NIS	0	144	-	36.234	258679
E/T + IST	S-graph	UIS	0	97	17	0.683	54034
		NIS	0	144	0	0.789	52475
	S-graph + LP	UIS	0	97	17	12.301	142589
		NIS	0	144	0	20.211	216408

Table 5.4 shows that the proposed S-graph method is able to reproduce the same weighted objective value of 97, but with a significantly lower computational time (0.222

seconds vs. 1.6 seconds reported by Vital-Soto et al. [153]). This demonstrates the competitive advantage of the S-graph-based approach in terms of runtime. The hybrid LP versions perform similarly but require more computation time due to the overhead of solving LPs at each B&B node. Moreover, incorporating IST minimization introduces modest overhead but results in tighter, more compact schedules without idle periods.

The computational experiments were performed on a machine equipped with an Intel(R) Core(TM) i7-7820HQ CPU at 2.90 GHz and 16 GB of RAM, whereas the original experiments in Vital-Soto et al. [153] were run on a system with an Intel(R) Core(TM) i7-10510U CPU at 1.80 GHz and 16 GB of RAM. While the hardware configurations are not identical, the difference in processor performance does not fully account for the observed gap in runtime, indicating a genuine algorithmic efficiency improvement.

TABLE 5.5: Problem instance parameters.

instance	**nproduct**	**nmac**	**nop**	**nmo**	**proc**	**dd**
dd_1	3	5	3-5	1-3	6-9	27-42
dd_2	3	5	4-5	2-4	6-9	32-43
dd_3	3	5	5-5	2-4	6-9	40-43
dd_4	4	5	5-5	1-3	6-9	40-46
dd_5	4	5	5-5	2-4	6-9	40-46
dd_6	5	4	2-4	1-3	6-9	27-40
dd_7	5	4	2-4	2-3	6-9	27-40
dd_8	5	5	5-5	1-3	6-9	40-46
dd_9	5	5	3-4	2-4	7-10	50-58
dd_10	6	5	4	2	1-3	14-20

To assess scalability and performance under more diverse conditions, a broader set of instances is defined in Table 5.5. Each instance varies key parameters including the number of products, task count per product, equipment flexibility, processing time range, and due date span. The instance set progresses from simpler (dd_1) to more complex scenarios (dd_10). Here, *nproduct* denotes the number of products, *nmac* the number of available equipment units, *nop* the number of tasks per product, *nmo* the minimum and maximum number of eligible equipments per task, *proc* the processing time range, and *dd* the due-date interval.

TABLE 5.6: Comparison of computation times and objective values for various scheduling methods and IST policies across different problem instances.

		S-graph & ET		S-graph + LP & ET		S-graph & ET + IST		S-graph + LP & ET + IST	
		Time (s)	**Obj.**	**Time (s)**	**Obj.**	**Time (s)**	**Obj.**	**Time (s)**	**Obj.**
dd_1	UIS	0.008	12	0.062	12	0.01	21	0.088	21
	NIS	0.014	12	0.068	12	0.01	21	0.093	21
dd_2	UIS	0.016	2	1.043	2	0.07	6	0.542	4
	NIS	0.114	3	0.897	3	0.195	4	0.532	4
dd_3	UIS	0.014	0	0.107	0	0.181	3	0.952	3
	NIS	0.045	3	0.225	3	0.183	1	0.876	1
dd_4	UIS	0.25	4	2.795	4	0.321	11	1.624	11
	NIS	0.3	4	2.824	4	0.345	11	1.568	11
dd_5	UIS	0.246	0	0.494	0	264.5	6	10.749	6
	NIS	0.338	0	0.707	0	1.8	7	11.645	7
dd_6	UIS	0.087	15	4.991	15	0.339	25	6.618	25
	NIS	1.231	27	10.503	27	1.167	33	8.521	35
dd_7	UIS	0.121	11	0.725	11	4.466	22	474.239	22
	NIS	10.801	11	682.1	11	1.874	14	593.4	14
dd_8	UIS	3.684	19	133.662	19	7.953	45	54.215	44
	NIS	3.968	19	154.023	19	8.361	45	27.232	44
dd_9	UIS	25.574	4	921.866	4	1228.531	13	13354.1	13
	NIS	37.958	6	986.038	6	887.501	16	8452.61	15
dd_10	UIS	11.992	2	294.5	2	646.1	4	480.9	4
	NIS	84.327	5	652.4	5	843.02	6	1099.91	6

Finally, Table 5.6 summarizes the performance of the four scheduling methods across all five instances and both storage policies (UIS and NIS). The table includes both computation time and objective value, allowing for comprehensive comparison. Several trends can be observed:

- **Computation Time:** Runtime increases with instance size. This is especially evident for LP-based approaches, which show significant time growth between `dd_7` and `dd_8`.
- **Objective Value:** The hybrid methods often achieve slightly better or equal objective values compared to the base S-graph. However, improvements are small relative to the increase in computation time.
- **Storage Policies:** UIS generally yields lower tardiness due to relaxed material flow constraints, but at the cost of higher IST. NIS leads to stricter scheduling, often increasing E/T values but reducing IST.

- **IST Inclusion:** When IST is included in the objective function, overall scores increase slightly, reflecting the added optimization burden. However, the resulting schedules tend to be more compact and storage-efficient.

These findings demonstrate that the S-graph methodology, with or without LP integration, offers a flexible and computationally efficient tool for addressing real-world job shop scheduling challenges. While LP methods provide fine-grained control over timing, their computational cost grows rapidly with instance size. The choice between UIS and NIS policies should be made based on system constraints: NIS is stricter but reduces handling complexity, while UIS offers scheduling flexibility at the expense of potential idle storage.

5.3 Discussion

This research presents a comprehensive analysis of four distinct methodological approaches to the FJSP, each representing a progressive enhancement in addressing complex scheduling scenarios. The study systematically explores the capabilities and limitations of these methods, offering a nuanced understanding of their applicability in modern manufacturing contexts.

The analysis begins with the application of the S-graph framework for E/T minimization. This foundational approach leverages the structural strengths of the S-graph in representing complex task dependencies and equipment assignments. It enables efficient navigation of the solution space by incorporating both precedence and resource constraints. The results demonstrate that the S-graph method consistently yields high-quality solutions across a variety of small to medium-sized problem instances.

Building upon this baseline, the methodology was extended to include IST minimization alongside E/T. This extension addresses a more comprehensive set of scheduling objectives by considering both timing accuracy and material handling efficiency. The experimental results show that the extended S-graph framework effectively balances these objectives. While the inclusion of IST introduces additional complexity, the framework remains capable of producing feasible and competitive solutions, particularly when storage considerations play a critical role in the production environment.

To further enhance scheduling performance, a hybrid approach was introduced that integrates the S-graph framework with a LP solver for E/T minimization. This combination leverages the graphical structure of the S-graph with the optimization precision of LP, enabling refined control over task timing. The results show that, although the hybrid approach can match or slightly improve solution quality, it generally incurs greater

computational costs. For instance, in the `dd_8` instance under the UIS policy, both the S-graph and hybrid methods achieved an objective value of 19, but the hybrid method required 133.662 seconds compared to 3.684 seconds for the standalone S-graph. This disparity suggests that the additional complexity introduced by LP integration may not be justified for single-objective E/T optimization, especially in cases where computational efficiency is paramount.

The most advanced variant of the methodology involves the hybrid S-graph and LP approach for simultaneous E/T and IST minimization. This integrated strategy is designed to manage multiple, and often conflicting, scheduling objectives. The results indicate that this approach can yield superior scheduling outcomes in terms of overall objective values. For example, in the `dd_8` instance, the hybrid method achieved an objective value of 44 compared to 45 from the standalone approach. However, this improvement comes at a significant computational cost, with runtime increasing from 7.953 seconds to 54.215 seconds under the UIS policy. These findings reinforce the trade-off between solution quality and computational effort inherent in multi-objective scheduling problems.

Across all tested problem instances, several trends emerge. As problem complexity increases—from `dd_1` to `dd_10`—computational times grow significantly across all methods, particularly for hybrid approaches. The choice of intermediate storage policy also plays a crucial role. The UIS policy tends to result in lower E/T penalties due to greater flexibility in scheduling, though often at the cost of increased storage time. In contrast, the NIS policy enforces stricter sequencing, often leading to higher E/T penalties but reducing IST.

While the hybrid methods occasionally produce slightly better objective values, they consistently require substantially more computational resources. This trade-off becomes increasingly pronounced with problem size, suggesting that pure S-graph approaches may be more suitable for time-sensitive or resource-limited contexts, whereas hybrid methods are better suited to scenarios where solution quality is prioritized over computational cost.

The inclusion of IST in the objective function typically leads to more balanced and realistic production schedules by avoiding idle periods between tasks. However, the increased computational complexity underscores the challenges of addressing multiple objectives simultaneously. The results show that while E/T minimization focuses on aligning task completions with due dates, IST minimization encourages tighter sequencing and improved resource utilization. Balancing these goals is essential for effective scheduling.

Importantly, the FJSP with due dates is NP-complete, and its complexity increases exponentially with the number of products and tasks. The problem instances used in this study were selected to allow for detailed evaluation while maintaining computational feasibility. Despite their moderate scale, these instances retain the core characteristics of realistic scheduling problems, including equipment flexibility, precedence constraints, and heterogeneous due dates.

In conclusion, the results provide strong evidence of the proposed methodology's effectiveness. The S-graph framework, with or without LP integration, offers a robust foundation for addressing a wide range of scheduling objectives. While LP-based methods can enhance optimization precision, they also increase runtime significantly. Therefore, the choice of method should align with specific production priorities—whether minimizing computation time, improving schedule adherence, or optimizing storage use.

Future research should explore heuristic and metaheuristic approaches to improve scalability, particularly for large-scale industrial applications. Techniques such as genetic algorithms, simulated annealing, or parallel computation could extend the applicability of the proposed framework to broader and more complex scheduling environments. Nevertheless, this study lays a solid foundation for further exploration of hybrid scheduling strategies in flexible manufacturing systems.

Chapter 6

Integrated Reactive Scheduling for Due Dates and Storage

This chapter develops a unified conceptual framework that combines reactive scheduling with due-date optimization and intermediate storage considerations, formulated within a FJSP setting. The proposed methodology integrates dynamic order arrivals, equipment flexibility, and multi-objective trade-offs into a single theoretical model, establishing a foundation for adaptive and storage-aware scheduling in modern production systems. The comprehensive problem formulation underlying this framework is presented in Section 3.3, which combines the reactive scheduling extensions with due date optimization and IST minimization. It is important to emphasize that this chapter develops a theoretical framework; no computational implementation or empirical results are provided at this stage. The objective is to rigorously formulate and analyze the concepts, laying the groundwork for future implementation and testing.

6.1 Methodology: Reactive Scheduling with Due-Date and Storage-Aware Optimization

The proposed methodology employs a two-stage reactive framework that integrates static scheduling principles with real-time adaptability for dynamically arriving orders. The framework supports both due-date constraints and intermediate storage policies (NIS/UIS), and is implemented through an S-graph-based formalism capable of representing temporal, precedence, and resource constraints.

6.1.1 Reactive Scheduling Framework

Upon receipt of the first order $o_1 = (I_1, t_1 = 0)$, an initial schedule S_1 is constructed. This schedule minimizes due-date deviation (earliness and tardiness) and respects either the NIS or UIS storage policy. It is derived using the traditional S-graph approach—either equipment-based or task-based branching—extended to compute start and finish times that align as closely as possible with each job's due date d_p.

When a new order $o_{k+1} = (I_{k+1}, t_{k+1})$ arrives, the current schedule S_k is partitioned into:

$$\underline{S}_k = \{(i, j, t^s, t^f) \in S_k \mid t^s < t_{k+1}\}, \quad \overline{S}_k = S_k \setminus \underline{S}_k, \tag{6.1}$$

with corresponding task sets $\underline{I}_k^*$ (started) and $\overline{I}_k^*$ (not yet started).

Any revised schedule S_{k+1} must ensure that the updated schedule S_{k+1} adheres to the following criteria:

1. Retain all assignments in $\underline{S}_k$.
2. Ensure that no task of I_{k+1} begins before time t_{k+1}.
3. Preserve precedence, storage, and machine-task feasibility.
4. Continue minimizing total $\sum(w_p^E E_p + w_p^T T_p)$, evaluating completion times C_p under either NIS or UIS storage constraints.

6.1.2 Reactive Policies under Due-Date Minimization

Depending on the desired trade-off between computational efficiency and schedule quality, the rescheduling policy for $\overline{I}_k^*$ can be selected from the following levels of flexibility

- **Policy 1:** $\overline{I}_k^*$ remains unaltered. New tasks I_{k+1} are appended. While simple and fast, delays incurred by new orders may increase tardiness penalties.
- **Policy 2.1:** Sequencing of $\overline{I}_k^*$ is fixed, and start times are unchanged. New tasks can only be inserted into idle time windows. This strategy safeguards due-date alignment but may limit flexibility if idle time is scarce.
- **Policy 2.2:** Sequence is fixed, but start times of $\overline{I}_k^*$ may shift, facilitating insertions without reassignments. This improves due-date optimization at the cost of modest computation.
- **Policy 3:** Full rescheduling of $\overline{I}_k^* \cup I_{k+1}$ is permitted. This offers the greatest flexibility for due-date equality and storage time reduction but is the most computationally intensive.

Policy	Flexibility	Computation	E/T Performance
1	Low	Low	Suboptimal
2.1	Medium	Medium	Moderate
2.2	Medium-High	Medium	Good
3	High	High	Optimal

TABLE 6.1: Reactive policy trade-offs.

Each policy balances computational effort and schedule flexibility (see Table 6.1). Policy 1 minimizes complexity, while Policy 3 allows higher-quality solutions at greater computational cost.

6.1.3 S-Graph Augmentation for Reactive Due-Date Scheduling

The S-graph for rescheduling must encode both the due-date objective and the constraints imposed by reactive policies:

- An artificial root node n_0 is introduced to connect to each task via arcs weighted by their respective order arrival times, thereby enforcing earliest start time constraints (Figure 4.1).
- Zero-wait arcs fix the start times of tasks in $\underline{I}_k^*$.
- According to the chosen policy:
  - **Policy 1 & 2.1:** Add zero-wait arcs to all tasks in $\overline{I}_k^*$, freezing their start times.
  - **Policy 2.2:** Only add sequence-preserving schedule-arcs between $\overline{I}_k^*$, allowing temporal flexibility.
  - **Policy 3:** Do not add any arcs among $\overline{I}_k^*$; full rescheduling is allowed.

The graph structure supports integration of task precedence, equipment assignment, and temporal restrictions.

B&B search traverses assignments and sequencing steps to build schedule S_{k+1}:

- Each node corresponds to a partial S-graph including current assignments and arcs.
- The algorithm evaluates completion times C_p for all affected products and computes new E_p, T_p, along with storage delays if applicable.
- Lower bounds on the total weighted due-date penalty are computed to prune dominated branches.

- Feasible extensions—such as task insertions or reassignment—are performed in accordance with the constraints imposed by the selected reactive policy.

The impact of the storage policy is also reflected in how intermediate storage times influence equipment utilization and schedule evaluation. Under the NIS policy, intermediate storage is disallowed, meaning that delays between successive operations on different equipment units effectively block the upstream unit, thereby influencing the effective finish time C_p of a product. In contrast, the UIS policy permits off-equipment storage, allowing subsequent tasks to be delayed without impeding upstream machine utilization. However, such delays may still affect due-date compliance through increased intermediate storage time.

To operationalize the reactive scheduling methodology, the following steps are executed upon each new order arrival:

1. Begin with S_1 that minimizes due-date penalties under a chosen storage policy.
2. For each new arrival o_{k+1}, build the current-state S-graph based on policy and timing.
3. Initialize $\underline{S}_k$ as fixed assignments; generate candidate S_{k+1} using the B&B procedure.
4. Evaluate and select the best candidate schedule, then commit it.
5. Repeat on subsequent arrivals.

By integrating reactive scheduling with due-date objectives and storage restrictions, this methodology provides systematic control over dynamic production environments, ensuring that schedule adaptability and performance optimization co-evolve harmoniously.

6.1.4 Reactive Scheduling with Due-Date and Intermediate Storage Optimization

In dynamic manufacturing systems, characterized by the unpredictable arrival of new orders, ensuring schedule feasibility while minimizing performance penalties—such as earliness, tardiness, and intermediate storage duration—poses additional challenges. This section extends the due-date–driven scheduling methodology by incorporating the challenges of real-time order arrivals, while also minimizing IST under different storage policies.

Upon the arrival of a new order o_{k+1} at time t_{k+1}, the scheduling system must generate an updated solution S_{k+1} that reflects the revised set of active and pending tasks. The

objective of this rescheduling step is to minimize both the cumulative intermediate storage time (IST) and the aggregate earliness/tardiness (E/T) penalties associated with all products in the system.

This trade-off is captured through the following multi-objective cost function:

$$\min\left(\alpha \cdot IS_{\text{total}} + \beta \cdot (E_{\text{total}} + T_{\text{total}})\right), \tag{6.2}$$

where $\alpha, \beta \in [0, 1]$ are weighting coefficients reflecting the relative importance of storage efficiency and due-date compliance. These parameters are typically normalized such that $\alpha + \beta = 1$, enabling a balanced and interpretable optimization criterion.

The penalty components are defined as follows. Earliness and tardiness are computed for each product $p \in P$ based on the completion time of its final task $\hat{i}_p$, denoted by $t^f_{\hat{i}_p}$, and the corresponding due date d_p:

$$E_{\text{total}} = \sum_{p \in P} \max(0, d_p - t^f_{\hat{i}_p}), \tag{6.3}$$

$$T_{\text{total}} = \sum_{p \in P} \max(0, t^f_{\hat{i}_p} - d_p). \tag{6.4}$$

The total intermediate storage time across the system is given by:

$$IS_{\text{total}} = \sum_{i \in I} is_i, \tag{6.5}$$

where is_i denotes the storage delay incurred between the completion of task i and the start of its immediate successor.

This formulation is applied iteratively upon each order arrival, allowing the schedule to be adapted in real time while preserving feasibility with respect to precedence, resource, and storage constraints. By embedding this cost structure into the S-graph–based branch-and-bound rescheduling process, the methodology enables systematic trade-offs between responsiveness, storage efficiency, and due-date adherence in dynamic manufacturing environments.

6.1.4.1 Storage Policy and Temporal Feasibility

Reactive scheduling introduces significant complications to IST minimization. Under NIS policy, a task can only begin if the downstream machine is immediately available, necessitating tight coordination between operations. In contrast, UIS permits decoupling of operations, enabling earlier task completions but increasing the likelihood of storage buildup.

When revising the schedule, tasks that have already started ($\underline{I}_k^*$) remain fixed, while tasks in $\overline{I}_k^* \cup I_{k+1}$ are re-evaluated to minimize both storage and due-date deviation. However, excessive temporal flexibility—while beneficial for due-date compliance—can result in elevated IST.

6.1.4.2 Storage-Aware Schedule Adjustments

At each branching step within the S-graph framework, tasks are assigned with respect to both machine availability and storage restrictions. Specifically, the system checks:

- Whether the recipe sequence of a task allows for immediate downstream operation;
- Whether the machine sequence permits execution without conflict;
- Whether intermediate storage may need to be reduced while maintaining feasibility.

If a task i would incur positive intermediate storage time $is_i > 0$, the scheduler investigates if $is_i = 0$ can still yield a valid schedule. If not, a search procedure identifies the minimal feasible is_i. This is applied recursively to neighboring tasks to balance the cumulative IST across the product flow.

In cases where a time gap exists between preceding and succeeding tasks on a machine, the algorithm checks whether reducing is_i of adjacent tasks can expand the usable gap sufficiently. If so, a refined assignment is committed, minimizing storage across multiple tasks.

The reactive E/T and IST optimization approach imposes additional trade-offs:

- Tight E/T prioritization may induce unnecessary storage overheads by scheduling tasks earlier than needed.
- Excessive IST minimization may defer task starts and degrade E/T performance.
- NIS policy intensifies this trade-off by reducing scheduling flexibility; UIS policy provides leeway but increases storage costs.

Balancing these criteria ensures the scheduler avoids both machine underutilization and due-date violations. This method fosters the development of schedules that are not only feasible in real time but also cost-efficient and operationally sound.

This extended methodology enables dynamic adaptation of production schedules while optimizing for both due-date adherence and resource-efficient processing. By incorporating intermediate storage analysis into reactive scheduling policies, the framework offers a

robust foundation for real-world implementations in environments where time constraints and resource coordination are equally critical.

6.2 Algorithmic Framework: Reactive Scheduling with Due-Date and IST Optimization

This section develops a unified algorithmic framework extending the reactive S-graph methodology to jointly minimize E/T penalties and IST. The scheduler dynamically responds to newly arriving orders while enforcing storage policies—NIS or UIS—via a unified B&B procedure.

The root node is seeded from the existing schedule S_k, augmented with zero-wait and schedule-arcs marking fixed tasks in the S-graph. At each node, the scheduler selects an unscheduled task $i \in \overline{I}_k^* \cup I_{k+1}$ and considers all feasible placements on machines $j \in J_i$, subject to the selected policy (Policies 1–3). Children nodes are created by assigning i to j and choosing feasible insertion points.

6.2.1 Node Expansion and Temporal Assignment Under Storage Constraints

Upon branching, each child node corresponds to the assignment of a new task $i \in I$ to a selected equipment unit $j \in J_i$. The extension of this node involves determining a feasible execution interval $[t_i^s, t_i^f]$ for task i, in compliance with the underlying constraints and the selected intermediate storage policy. The algorithmic progression at each branch node unfolds through a structured sequence of operations, from identifying feasible execution windows to evaluating and pruning based on multi-objective performance. The following steps elaborate the decision-making mechanics underlying each node expansion.

Algorithm 2 Reactive Scheduling Control Flow

Require: Current schedule S_k, reschedulable tasks $\overline{I}_k^*$, new tasks I_{k+1}, policy
Ensure: Updated schedule or pruned branch

for all tasks $i \in \overline{I}_k^* \cup I_{k+1}$ **do**
  for all eligible equipment j **do**
    Step 1: $(t_i^s, t_i^f) \leftarrow$ ComputeTentativeInterval(i, j)
    if infeasible **then**
      continue
    end if
    Step 2: $is_i \leftarrow$ EstimateStorageTime(i)
    if ConflictExists(i, j) **then**
      Step 3: result $\leftarrow$ RedistributeSlackIfConflict(i, j)
      if result $=$ Unresolved **then**
        continue
      end if
    end if
    Commit task i to schedule on equipment j
    Step 4: $Z \leftarrow$ EvaluatePartialObjective(S)
    Step 5: PruneBranch(S, Z)
    break
  end for
end for
return updated schedule

At each branch-and-bound node, one task $i \in \overline{I}_k^* \cup I_{k+1}$ is selected according to the branching strategy; Algorithm 2 illustrates the evaluation process for such a selected task.

Algorithm 2 summarizes the overall control flow of the reactive scheduling procedure. For each reschedulable or newly arrived task, a tentative execution interval is first computed (Step 1). The intermediate storage implication of the tentative placement is then evaluated (Step 2). If the placement induces an equipment-level conflict that cannot be resolved by direct insertion into an idle window, local slack redistribution is attempted (Step 3). Once a task is successfully placed, the partial objective value is updated (Step 4), and a pruning decision is made using a lower-bound estimate (Step 5). This structured flow ensures that feasibility, storage behavior, and optimization criteria are jointly respected during branch-and-bound search.

6.2.1.1 Step 1: Tentative Execution Interval Computation

This procedure determines the earliest feasible execution interval $[t_i^s, t_i^f]$ for a given task i on a selected equipment unit j, considering precedence constraints, equipment availability, reactive policy settings, and the task's order release time. It is invoked during node expansion in the branch-and-bound algorithm to guide feasibility testing and ensure compliance with temporal and policy constraints in dynamic scheduling environments. This procedure is always the first feasibility test applied during node expansion.

procedure ComputeTentativeInterval(task i, equipment j, policy, schedule S_k)
  $t_{\text{earliest}} \leftarrow \max\{t_{i'}^f \mid i' \in I_i^-\}$ ▷ Precedence: all predecessors must finish
  idle_windows $\leftarrow$ FindIdleWindows$(j, S_k, t_{\text{earliest}})$ ▷ All idle intervals of j starting after t_{earliest}
  if idle_windows $= \emptyset$ **then**
    return Infeasible
  end if
  if policy $\in \{1, 2.1\}$ **then**
    $t_{\text{earliest}} \leftarrow \max(t_{\text{earliest}}, t_i^s)$ ▷ Freeze start time (if already known)
  else if policy $= 2.2$ **then**
    // Sequence is fixed, allow flexible start respecting predecessors
    $t_{\text{earliest}} \leftarrow \max(t_{\text{earliest}}, \text{EarliestStartInSequence}(i))$
  end if
  if $i \in \overline{I}_k^* \cup I_{k+1}$ **then**
    $t_{\text{earliest}} \leftarrow \max(t_{\text{earliest}}, t_{k+1})$ ▷ Order arrival constraint
  end if
  for all $w \in$ idle_windows **do**
    if length$(w) \geq pt_{i,j}$ and w.start $\geq t_{\text{earliest}}$ **then**
      $t_i^s \leftarrow w$.start
      $t_i^f \leftarrow t_i^s + pt_{i,j}$
      return (t_i^s, t_i^f)
    end if
  end for
  return Infeasible
end procedure

The function `FindIdleWindows` returns a list of time intervals during which equipment j is idle and can accommodate a task starting no earlier than the specified time. The function `EarliestStartInSequence` retrieves the earliest allowable start time for task i based on the preserved task sequence, when applicable under the current policy.

This procedure ensures:

- All immediate predecessors of task i have completed before t_i^s (precedence constraints).
- The task does not begin before its order's release time t_{k+1}.
- All tasks whose execution has not yet started at the reactive event time t_{k+1}—including tasks of earlier orders under rescheduling policies—are constrained to start no earlier than t_{k+1}, ensuring temporal consistency and preventing retroactive execution.
- The selected idle window has sufficient duration to fit the task's processing time.
- Policy-specific conditions (e.g., frozen timings or fixed sequences) are respected.

6.2.1.2 Step 2: Estimation of Intermediate Storage Time

This procedure estimates the intermediate storage time is_i incurred after task i and before its immediate successor i^+ begins. It is used during task placement to evaluate the impact of scheduling decisions under different storage policies (NIS or UIS). The procedure is called after computing the tentative execution interval of i, and requires either a scheduled successor or a placeholder when the successor is not yet placed.

procedure EstimateStorageTime(task i, policy, successor i^+)
  if i^+ not scheduled **then**
    $is_i \leftarrow 0$ ▷ Temporary placeholder if successor not yet placed
  else
    $is_i \leftarrow t_{i^+}^s - t_i^f$
  end if
  if policy = NIS **then**
    if $is_i > 0$ **then**
      AttemptStorageReduction(i) ▷ Try to reduce is_i to 0 if possible
    end if
  else if policy = UIS **then**
    $is_i \leftarrow \max(0, is_i)$ ▷ Clip temporary infeasible value during tentative scheduling
  end if
  return is_i
end procedure

The procedure `AttemptStorageReduction` is invoked exclusively under the NIS policy, because NIS explicitly prohibits positive intermediate storage times between consecutive tasks. Under this policy, any positive value of is_i indicates blocking on the same equipment, which must be eliminated whenever feasible. Conceptually, `AttemptStorageReduction` attempts to remove this blocking by locally adjusting task start times. This

is achieved either by delaying the start of the successor task i^+ or by shifting the execution of task i and its recipe predecessors backward in time, while preserving all precedence and equipment constraints. If neither adjustment is feasible without violating temporal or resource constraints, the current partial schedule is deemed infeasible and the corresponding branch is pruned. Under the UIS policy, such corrective action is unnecessary, since positive intermediate storage times are permitted and do not imply equipment blocking. Therefore, `AttemptStorageReduction` is not applied in the UIS case.

The intermediate storage time is_i quantifies the temporal gap between the finish of task i and the start of its immediate successor i^+:

$$is_i = t^s_{i^+} - t^f_i. \tag{6.6}$$

If the successor task has not yet been scheduled, a temporary value of $is_i = 0$ is assumed for feasibility checks. During tentative scheduling and before conflict resolution is completed, it is possible that $t^s_{i^+} < t^f_i$, resulting in a negative intermediate value. Such values indicate a transient precedence violation rather than physical storage and are corrected during schedule adjustment.

Policy-specific behavior:

- **NIS (No Intermediate Storage):** The scheduler attempts to enforce $is_i = 0$. If a positive value occurs, a local adjustment is triggered to eliminate storage delay via `AttemptStorageReduction`.
- **UIS (Unlimited Intermediate Storage):** Positive storage times are allowed. Any negative intermediate value—arising from tentative task placement—is clipped to zero, ensuring causal feasibility of the partial schedule.

This procedure ensures that storage implications are properly evaluated during schedule construction under each storage policy.

6.2.1.3 Step 3: Conflict Resolution via Storage-Gap Adjustment

This procedure attempts to resolve equipment-level conflicts when task i cannot be placed into its intended time slot on equipment j due to overlaps with other tasks already assigned to the same equipment. The method explores whether adjusting the timing of neighboring tasks—using available intermediate storage slack—can free enough space to accommodate task i. It is invoked during schedule construction, primarily under the NIS policy or when idle windows are too narrow to directly insert the task. This procedure

is invoked exclusively by the reactive task insertion controller (Algorithm 2) when a tentative placement results in an equipment-level overlap that cannot be resolved by direct scheduling.

Notation clarification. In this subsection, task adjacency is defined with respect to the processing sequence on a given equipment. Let $i_j^{\prec}$ and $i_j^{\succ}$ denote the immediately preceding and immediately succeeding tasks of task i on equipment j, respectively. This equipment-based adjacency is distinct from recipe-based predecessor and successor relations, which are defined by technological precedence constraints.

procedure RedistributeSlackIfConflict(task i, equipment j)
  Identify immediate temporal neighbors of i on equipment j: $i_j^{\prec}$ (preceding), $i_j^{\succ}$ (succeeding)
  Compute available slack: $\delta = \min(is_{i_j^{\prec}}, is_{i_j^{\succ}})$
  Set minimum slack threshold $\varepsilon > 0$
  while $\delta > \varepsilon$ **and** conflictExists(i, j) **do**
    Shift the end time of $i_j^{\prec}$ earlier by $\delta/2$
    Shift the start time of $i_j^{\succ}$ later by $\delta/2$
    Update $is_{i_j^{\prec}}$ and $is_{i_j^{\succ}}$
    if all precedence and storage constraints remain satisfied **then**
      return Adjusted
    end if
    Recompute $\delta = \min(is_{i_j^{\prec}}, is_{i_j^{\succ}})$
  end while
  return Unresolved
end procedure

Although the slack redistribution step reduces the available storage gap symmetrically by a fraction of the current slack, the procedure is not intended to asymptotically drive the slack to zero. Instead, each iteration aims to eliminate a finite overlap between tasks. The loop is therefore bounded by a minimum slack threshold ε, below which further adjustments are considered ineffective. In practice, the conflict is either resolved after a finite number of iterations or deemed infeasible once the available slack is exhausted. This guarantees termination and prevents infinite cycling.

Although the pseudocode refers to shifting only the immediate neighboring tasks $i_j^{\prec}$ and $i_j^{\succ}$ on equipment j, these shifts are not applied in isolation. Any temporal modification of a task is propagated along its corresponding recipe chain to preserve technological precedence constraints. Specifically, when the end time of $i_j^{\prec}$ is shifted earlier, the same backward shift is recursively applied to all its recipe predecessors. Likewise, when the start time of $i_j^{\succ}$ is shifted later, the delay is propagated forward to

all its recipe successors. During propagation, all affected tasks are re-evaluated with respect to both precedence constraints and equipment availability. If propagation would violate feasibility (e.g., negative start times or equipment conflicts on other units), the adjustment is rejected and the procedure continues with the next admissible slack value.

If task i cannot be placed on equipment j due to insufficient idle time between $i_j^{\prec}$ and $i_j^{\succ}$, this routine attempts to redistribute available intermediate storage slack locally. The slack values $is_{i_j^{\prec}}$ and $is_{i_j^{\succ}}$ represent temporal flexibility arising from previously computed storage gaps.

The algorithm symmetrically shifts the neighboring tasks in time and propagates these shifts along their respective recipe chains to maintain technological feasibility. After each adjustment, both technological precedence constraints (recipe arcs) and equipment capacity constraints (schedule arcs) are re-evaluated. If a feasible configuration is found, the modified timing is accepted; otherwise, the process continues until the available slack is exhausted.

This localized adjustment mechanism is particularly relevant under the NIS policy, where intermediate storage is realized on the equipment itself. In this case, the equipment remains occupied during waiting periods between consecutive tasks, making precise temporal coordination necessary to avoid blocking and infeasibility.

6.2.1.4 Step 4: Partial Objective Evaluation

This procedure computes the value of the partial objective function, denoted as Z, for the current state of the schedule. It is used within the branch-and-bound algorithm to evaluate and compare partial solutions based on two criteria: (1) total intermediate storage time, and (2) total earliness/tardiness penalties. The function is called whenever a new task is scheduled and a new partial schedule is generated.

The objective function for a partial schedule is:

$$Z = \alpha \sum_{i \in I_{\text{sched}}} (t_{i^+}^s - t_i^f) + \beta \sum_{p \in P_{\text{sched}}} (E_p + T_p),$$

where:

- I_{sched}: tasks whose successors are also scheduled,
- P_{sched}: products whose final tasks have been scheduled,
- $E_p = \max(0, d_p - C_p)$, $T_p = \max(0, C_p - d_p)$, and $C_p = t_{i_p}^f$,
- $\alpha, \beta \in [0, 1]$ are weights with $\alpha + \beta = 1$.

procedure EVALUATEPARTIALOBJECTIVE(partial schedule S, weights α, β)
  $\mathrm{IST}_{\mathrm{total}} \leftarrow 0$
  for all $i \in S$.scheduled_tasks **do**
    if successor i^+ is scheduled **then**
      $\mathrm{IST}_{\mathrm{total}} \leftarrow \mathrm{IST}_{\mathrm{total}} + (t^s_{i^+} - t^f_i)$
    end if
  end for
  $\mathrm{ET}_{\mathrm{total}} \leftarrow 0$
  for all $p \in P$ such that final task $\hat{i}_p$ is scheduled **do**
    $C_p \leftarrow t^f_{\hat{i}_p}$
    $E_p \leftarrow \max(0, d_p - C_p)$
    $T_p \leftarrow \max(0, C_p - d_p)$
    $\mathrm{ET}_{\mathrm{total}} \leftarrow \mathrm{ET}_{\mathrm{total}} + E_p + T_p$
  end for
  $Z \leftarrow \alpha \cdot \mathrm{IST}_{\mathrm{total}} + \beta \cdot \mathrm{ET}_{\mathrm{total}}$
  return Z
end procedure

This function evaluates only the scheduled part of the current solution — intermediate storage time is calculated only for tasks whose successors are already placed, and earliness/tardiness is calculated only for completed products. This ensures efficiency and feasibility in the incremental evaluation process of branch-and-bound search. The resulting objective value is used both for node comparison and for bounding in the subsequent pruning step.

6.2.1.5 Step 5: Pruning via Bounding and Dominance

This procedure evaluates whether a given partial schedule (represented by a search tree node) can be safely discarded ("pruned") in the branch-and-bound process. It estimates a lower bound Z_{LB} for the objective value of the node (combining optimistic E/T and IST assumptions) and compares it with the best-known solution so far.

The procedure is invoked after a new partial schedule (node) is generated. If the estimated lower bound already exceeds the cost of the best complete solution, the branch is pruned to avoid unnecessary search.

procedure PRUNEBRANCH(current_node, best_solution)
  $Z_{\mathrm{LB}} \leftarrow$ current_node.Z_{current} ▷ Objective of current partial schedule
  for all unscheduled product p **do**
    $(C_{\mathrm{early}}, C_{\mathrm{late}}) \leftarrow$ EstimateCompletionBounds(p)

```
        if d_p < C_early then
            penalty ← C_early − d_p
        else if d_p > C_late then
            penalty ← d_p − C_late
        else
            penalty ← 0
        end if
        Z_LB ← Z_LB + β · penalty
    end for
    Z_LB ← Z_LB + α · 0                    ▷ Assume perfect future IST minimization
    if Z_LB ≥ best_solution.Z then
        PruneNode
    else
        ContinueExpansion
    end if
end procedure
```

The procedure `EstimateCompletionBounds` computes optimistic bounds on the completion time of an unscheduled product p, based on the current partial schedule. Its purpose is not to predict an exact completion time, but to provide valid lower and upper bounds that can be safely used for pruning decisions.

Let $\hat{i}_p$ denote the final task of product p. Two completion bounds are computed:

- C_{early}: an optimistic (earliest possible) completion time, assuming immediate availability of required equipment and minimum processing times.
- C_{late}: a pessimistic (latest admissible) completion time, assuming maximal delays caused by already scheduled tasks. The value C_{late} accounts only for delays induced by tasks already fixed in the current partial schedule and does not assume any additional blocking from yet-unscheduled tasks.

Both bounds are derived using the current S-graph state and respect technological precedence constraints. No speculative rescheduling of already fixed tasks is performed.

procedure EstimateCompletionBounds(product p)

Identify remaining tasks I_p^{unsched}

$C_{\text{early}} \leftarrow$ earliest feasible completion of $\hat{i}_p$ assuming immediate equipment availability

$C_{\text{late}} \leftarrow$ latest feasible completion of $\hat{i}_p$ given current schedule and fixed task assignments

return $(C_{\text{early}}, C_{\text{late}})$
end procedure

Both bounds are guaranteed to be admissible: C_{early} never overestimates the best achievable completion time, while C_{late} never underestimates the worst-case completion time under the current partial schedule.

The lower bound Z_{LB} is obtained by extending the current partial objective value Z with these optimistic completion-time estimates for unscheduled products, while assuming zero additional intermediate storage cost for tasks not yet placed.

The pruning logic relies on a lower-bound estimate of the final objective, computed as:

- **E/T Bounding:** For each unscheduled product, the procedure estimates the earliest and latest possible completion times using optimistic assumptions (e.g., no delays, immediate machine availability). These bounds determine the minimum possible penalty that product p could contribute.
- **IST Bounding:** The intermediate storage cost of unscheduled tasks is assumed to be zero — an optimistic best case — which helps eliminate only those nodes that are clearly suboptimal.
- **Dominance Criterion:** If the optimistic lower bound Z_{LB} still exceeds the cost of the best known full solution, the current branch cannot yield a better outcome and is pruned.

This strategy ensures that the algorithm avoids exploring hopeless paths, improving overall computational efficiency.

6.2.2 Storage-Gap Adjustment Procedure

To facilitate tight task placement while minimizing IST, the algorithm employs a dedicated storage-gap adjustment routine whenever a newly scheduled task i conflicts with existing assignments on its allocated equipment unit j. This routine proceeds through the following steps:

1. **Conflict Localization:** Identify the immediate predecessor and successor tasks on unit j relative to the tentative interval $[t_i^s, t_i^f]$. These define the neighboring time window into which i must potentially be inserted.

2. **Storage Slack Redistribution:** Investigate whether adjacent tasks can partially relinquish their intermediate storage durations is_{i^-}, is_{i^+} by a slack margin $\delta > 0$. This may free up enough room to schedule i without violating timing constraints.

3. **Precedence Validation:** For every attempted adjustment, verify recursively that the modified execution intervals of the affected tasks respect the precedence constraints defined in the recipe-graph by the recipe-arcs A_1. Any violation renders the adjustment infeasible.

4. **Feasibility Commitment:** If a valid insertion interval is created via slack redistribution, the corresponding updates are committed to the schedule, and the B&B expansion proceeds. Otherwise, the start time t_i^s is incremented to the next feasible point, and the placement attempt is retried.

This routine provides localized reoptimization, enabling the scheduler to balance IST reduction with machine availability and precedence integrity—especially under reactive and storage-sensitive conditions.

6.2.3 Lower Bounding

To improve computational tractability, the branch-and-bound procedure integrates bounding and pruning mechanisms that eliminate suboptimal branches early.

- **Optimistic Due-Date Bound:** For each unscheduled product, the procedure assumes that the remaining tasks can be scheduled in such a way that the product finishes exactly at its due date, and that no intermediate storage is incurred. While this results in zero penalty for those products, it does not evaluate feasibility, rather, it serves as a hypothetical best-case scenario to derive an optimistic lower bound for early pruning. This bounding mechanism does not compute a realistic schedule, but estimates the "minimum possible objective value" under idealized assumptions.

- **Critical-Path Estimation:** A more conservative bound is obtained by estimating the longest feasible delay path through the task-precedence arcs (recipe structure) for each product. This includes task durations and respects resource and precedence constraints. It allows a partial projection of the earliest and latest completion times, including potential storage-related delays, thereby offering a more informed bound than the optimistic case.

If the combined lower-bound estimate of the total objective value exceeds that of the best-known complete solution, the corresponding branch is pruned. This focuses the search on promising parts of the solution space and reduces computational overhead.

6.2.4 Reactive Adaptation and Policy Enforcement

The algorithm maintains a high degree of adaptability by honoring the designated rescheduling policy when new orders o_{k+1} arrive. Each policy governs the degree to which the current schedule may be modified:

- **Policy 1:** The existing schedule remains fixed; new tasks are appended after all previously planned tasks.
- **Policy 2.1:** Both the sequence and timing of $\overline{I}_k^*$ remain fixed; new tasks can only be inserted in pre-existing idle gaps.
- **Policy 2.2:** The task sequence of $\overline{I}_k^*$ is preserved, but timings can shift to allow more compact scheduling.
- **Policy 3:** Full rescheduling is allowed—both sequence and timing of $\overline{I}_k^*$ can be modified.

To avoid recomputation, the algorithm selectively retains parts of the existing search tree that remain valid under the newly expanded order set, enabling efficient resumption of the scheduling process.

6.2.5 Computational Efficiency and Complexity Considerations

Despite the increased complexity introduced by multi-objective optimization and reactive dynamics, the algorithm employs several techniques to maintain practical performance:

- **Restricted Search Scope:** Only tasks in $\overline{I}_k^* \cup I_{k+1}$ are considered for recomputation, limiting the branching factor.
- **Heuristic-Guided Search:** Lower bounds, slack propagation, and critical-path approximations guide the exploration away from infeasible or suboptimal branches.
- **Warm-Start Initialization:** The prior schedule S_k serves as the initial reference, preserving previously verified decisions.
- **Policy Constraints:** Rescheduling flexibility is bounded by the chosen policy, limiting the degrees of freedom and thus reducing computational effort.

These strategies collectively indicate that the algorithm scales favorably in practice, delivering optimal or near-optimal solutions for medium-sized instances within tractable runtimes.

6.2.6 Summary of Methodology

Although unimplemented, the proposed S-graph–based branch-and-bound framework lays a strong theoretical foundation for real-time, policy-aware scheduling systems, particularly in production contexts with stringent intermediate storage or responsiveness constraints, thereby addressing scheduling challenges in dynamic production environments. It theoretically integrates:

- Real-time adaptability to new job arrivals,
- Multi-objective optimization of earliness/tardiness and intermediate storage,
- Policy-aware control over rescheduling flexibility,
- Efficient search pruning and heuristic bounding.

Taken together, these conceptual features form the basis for a system that could generate high-quality, policy-compliant, and storage-efficient schedules in practical industrial settings once implemented and validated.

6.3 Discussion on Trade-offs

The integration of reactive scheduling with due-date objectives and intermediate storage minimization inevitably introduces a spectrum of trade-offs that must be navigated with care. These trade-offs emerge from the interplay between flexibility, resource efficiency, and responsiveness to dynamic events, particularly under varying storage policies and policy-based rescheduling strategies.

Core trade-offs manifest in three dimensions:

- **E/T vs Storage:** Advancing tasks reduces tardiness but increases intermediate storage costs, particularly under UIS policy where decoupled operations accumulate storage time is_i. Conversely, delaying tasks to minimize storage under NIS risks violating due dates through blocking effects, where downstream equipment unavailability delays upstream completions [154].
- **Flexibility vs Computation:** Policy 3 (full rescheduling) is capable of producing optimal solutions; however, its computational demands make it unsuitable for real-time applications with frequent order arrivals [155]. In contrast, Policies 1 and 2.1 provide significantly better computational efficiency, but this comes at the cost of reduced solution quality due to limited flexibility in task placement [156].

- **Reactivity vs Stability:** Frequent rescheduling (e.g., Policy 3) improves responsiveness but causes "schedule nervousness," disrupting operator workflows and material flow predictability [157]. Infrequent adjustments (Policy 1) enhance stability but risk accumulating delays from unaddressed disruptions [158].

The choice of reactive policy directly influences the degrees of freedom available during rescheduling. Policy 1 minimizes disruption but may produce suboptimal schedules as new tasks are simply appended. Policy 3, while offering full flexibility and potentially better global solutions, introduces significantly higher computational complexity. Intermediate policies (2.1 and 2.2) provide a structured compromise, preserving useful schedule segments while enabling targeted adaptations.

Under NIS policy, task coordination must be tight, increasing the difficulty of finding feasible insertions. This may lead to schedule fragmentation or infeasibility in high-load scenarios. The UIS policy allows more decoupled execution but at the cost of elevated storage usage and potential congestion in material handling or buffer zones. The scheduler must thus evaluate which policy better aligns with the physical limitations and performance objectives of the production system.

While reactivity enables responsiveness to unexpected order arrivals, it may compromise schedule stability, especially under policies that permit shifting of previously planned tasks. In real-world settings, excessive reshuffling may impact downstream planning, personnel coordination, or quality control. Therefore, the degree of reactivity must be balanced with organizational requirements for predictability and control.

Overall, these trade-offs underscore the necessity of a flexible and parameterizable scheduling engine—one capable of adapting to varying production contexts while maintaining transparency and controllability of decision criteria.

6.4 Summary and Insights

This chapter has presented a unified algorithmic framework that extends S-graph-based scheduling to incorporate both dynamic order arrivals and multi-objective optimization. By integrating E/T minimization with IST reduction, the model addresses a complex and realistic variant of the FJSP.

The methodology introduces a reactive scheduling approach that adapts incrementally to new tasks, guided by structured policy options that determine the permissible extent of rescheduling. The algorithm utilizes a B&B engine equipped with heuristic bounding, storage-aware placement, and dynamic conflict resolution routines to ensure feasibility

and efficiency. Both NIS and UIS policies are supported, allowing the model to align with various operational environments.

Key insights from this work include:

- The ability to jointly optimize due-date adherence and storage efficiency yields more balanced production schedules.
- Flexible reactive policies (e.g., Policy 2.2 and 3) can significantly improve schedule quality, provided computational resources allow.
- Storage-aware local adjustments (e.g., the storage-gap routine) are critical to handling tight machine availability without violating precedence constraints.
- Strategic pruning and reusability of partial schedules are essential for maintaining scalability under dynamic conditions.

Altogether, the proposed framework offers a robust, extensible foundation for modern production scheduling systems, capable of responding to real-time disruptions while optimizing multiple, often conflicting, performance metrics.

This framework fundamentally advances reactive scheduling by:

1. Unifying due-date optimization with storage constraints
2. Formalizing policy-aware rescheduling mechanics
3. Introducing storage-gap adjustment for IST minimization.

This chapter establishes the conceptual and algorithmic groundwork for a unified reactive scheduling framework. All models, pseudo-codes, and algorithmic procedures discussed here serve as formal theoretical constructs intended to guide subsequent implementation. Future work will focus on realizing these algorithms in computational systems, conducting experiments, and refining heuristics based on empirical performance analysis. Additionally, the LP-based lower bounding scheme introduced in Chapter 4 may be integrated as a formal bounding component within the B&B engine to enhance pruning efficiency and provide tighter guarantees for due-date performance.

Chapter 7

Conclusions

This dissertation has systematically addressed the critical challenges of manufacturing scheduling in dynamic environments characterized by uncertainty, competing objectives, and evolving operational constraints. The research has developed novel scheduling frameworks that bridge the gap between theoretical models and practical implementation requirements, providing robust solutions for maintaining operational efficiency under disruption. By combining baseline scheduling with policy-driven reactive adaptation capabilities, the proposed methodologies offer significant advancements in balancing due date adherence, storage efficiency, and operational flexibility within unified architectures.

7.1 Summary of Contributions

The core contributions of this dissertation are organized across three interconnected domains.

For reactive scheduling in dynamic environments with due date and storage objectives, this research developed novel branch-and-bound algorithms enhanced with storage-gap adjustment procedures and policy-driven rescheduling strategies. These approaches handle dynamic job arrivals under various rescheduling policies (ranging from fixed to fully flexible), enabling efficient adaptation to disruptions while managing schedule stability. The formal mathematical formulation for flexible job shops with dynamic arrivals establishes a robust foundation for responding to unpredictable manufacturing demands.

In the domain of deterministic scheduling with integrated due date and storage constraints, this work proposed a unified modeling approach that combines intermediate storage policies (NIS/UIS) with due date objectives. The resulting storage-aware scheduling algorithms, employing bounded waiting time mechanisms and local adjustments,

demonstrated substantial reductions in earliness and tardiness while ensuring storage feasibility across different operational scenarios. This contribution directly addresses the critical industry challenge of balancing customer service requirements with internal operational constraints in tightly controlled production systems.

Finally, the multi-objective framework introduced here represents the culmination of this research, integrating S-graph models with policy-aware reactive adaptation. This hybrid architecture simultaneously addresses customer service performance (due date adherence), operational efficiency (through reduced intermediate storage), and system responsiveness to dynamic events. By formalizing the combination of due date objectives, storage minimization, and rescheduling flexibility, the framework lays a strong conceptual groundwork for future computational implementations and industrial integration, offering a holistic solution beyond traditional single-objective scheduling approaches.

7.2 Thesis Statements

1st thesis group: *Reactive scheduling in flexible job shops minimizes makespan by applying targeted S-graph modifications—zero-wait arc insertion and selective release-node locking—under three policies (append-only, partial insertion with strict or flexible delay, and full rescheduling), seamlessly integrated via equipment- or task-based branching.*

Supporting publications:

- **[P1]** Krisztián Attila Bakon, Tibor Holczinger, Zoltán Süle, Szilárd Jaskó, and János Abonyi: *Scheduling Under Uncertainty for Industry 4.0 and 5.0*, IEEE Access, 2022.
- **[P2]** Krisztián Attila Bakon, Tibor Holczinger: *S-Graph-Based Reactive Scheduling with Unexpected Arrivals of New Orders*, Machines, 2024.

This approach allows for dynamic order integration while maintaining computational efficiency and makespan optimization. The methodology classifies tasks into started, unscheduled, and new sets, constructing a root subproblem by inserting zero-wait arcs to fix start times where required. Policy-specific graph edits preserve or free task sequences and timings: the append-only policy locks all unscheduled tasks; strict insertion allows idle-gap placement only; flexible delay insertion shifts existing tasks to create slots; and full rescheduling frees all pending tasks. Equipment-based branching handles machine assignments sequentially, while task-based branching leverages activity lists, both requiring only minimal adaptation—such as gap constraints—to enforce policy rules. By unifying release-node locking, zero-wait arcs, and precedence arcs within the S-graph framework, the approach enables dynamic

order integration with controlled flexibility, balancing computational efficiency and makespan optimization.

2nd thesis group: *The extended S-graph framework jointly minimizes earliness/-tardiness penalties and intermediate storage time in flexible job shop scheduling with due-date constraints, leveraging zero-wait arc adjustments, equipment- and recipe-arc sequencing, and hybrid integration with an LP solver to balance timeliness and material handling under NIS and UIS policies.*

Supporting publications:

– **[P3]** Krisztián Attila Bakon, Tibor Holczinger: *Addressing Due Date and Storage Restrictions in the S-Graph Scheduling Framework*, Machines, 2025. DOI: 10.3390/machines13020131.

This framework extends the S-graph model to address both due-date adherence and intermediate storage minimization in flexible job shop scheduling. It employs zero-wait arc adjustments to enforce strict start-time locking for in-progress tasks, while recipe- and schedule-arcs capture precedence and resource constraints. The dual-objective function combines total earliness, tardiness, and intermediate storage, guiding branching decisions. Local schedule adjustments reduce idle storage when feasible, and a hybrid approach transforms each partial S-graph into an LP model to compute tight lower bounds and refine task timings. This integrated strategy ensures feasible, compact schedules that respect due dates and minimize holding costs, offering a modular, adaptable solution for complex scheduling environments.

3rd thesis group: *A unified reactive S-graph framework integrates due-date optimization and intermediate storage management for flexible job shops, enabling dynamic order integration under NIS/UIS policies through policy-driven graph augmentations and branch-and-bound search with storage-aware conflict resolution.*

Supporting publications:

– **[P4]** Krisztián Attila Bakon, Tibor Holczinger: *Reactive Scheduling under Due-Date and Intermediate Storage Constraints: A Conceptual Multi-Objective Approach*, in Proceedings of the Central European Conference on Information and Intelligent Systems (CECIIS), Varaždin, Croatia, 2025.

This framework extends the S-graph model to simultaneously address due-date optimization and intermediate storage management in flexible job shop scheduling. It employs policy-driven graph augmentations—such as zero-wait arcs for strict start-time locking and selective release-node locking—to enable dynamic order integration under both NIS and UIS policies. The branch-and-bound search incorporates storage-aware conflict resolution, redistributing storage slack to resolve conflicts while maintaining schedule feasibility. By integrating reactive adaptability with

multi-objective optimization, this framework provides a robust solution for managing dynamic manufacturing environments characterized by uncertainty and competing objectives.

7.3 Practical Implications and Industrial Relevance

The methodologies developed in this research offer tangible benefits for manufacturing operations operating under dynamic market conditions and production uncertainties. The proposed reactive scheduling framework enables production systems to effectively adapt to disruptions such as unexpected order arrivals, maintaining delivery performance and schedule stability under varying rescheduling policies. In parallel, the storage-aware scheduling approach minimizes intermediate storage requirements, thereby reducing work-in-process inventory costs and supporting tighter inventory control.

Moreover, by explicitly analyzing the trade-offs between due date adherence, intermediate storage utilization, scheduling flexibility, and computational effort, this research provides practical insights into how manufacturers can tailor scheduling strategies to their specific operational priorities. The ability to adjust policy parameters allows companies to balance responsiveness with stability, and solution quality with computational efficiency—key considerations in modern production environments.

These advancements are particularly relevant in manufacturing contexts that face increasing demands for customization, shorter lead times, and frequent order changes. By jointly optimizing customer service objectives and internal resource utilization within a unified multi-objective framework, this work lays a robust conceptual foundation for future industrial scheduling systems, enabling manufacturers to enhance both operational agility and efficiency in complex, volatile production scenarios.

7.4 Limitations and Future Research Directions

While this research provides significant advancements in reactive and multi-objective scheduling, several limitations warrant further investigation. The current models assume perfect information about production parameters and disruptions, whereas real-world environments often involve uncertainty in disruption durations, arrival patterns, and resource availability. Future work should incorporate stochastic or fuzzy modeling approaches to enhance decision-making under incomplete information. Additionally, while the proposed B&B framework employs pruning and heuristic bounding to manage complexity, scaling the methodology to ultra-large problems involving thousands of operations remains computationally challenging and merits further study.

Building on the findings of this dissertation, several promising research directions emerge:

- Extending the reactive and storage-aware scheduling models to explicitly incorporate uncertainty, such as stochastic disruptions and variable resource capacities, enhancing robustness under realistic operating conditions.
- Investigating advanced heuristic or metaheuristic strategies to further improve computational efficiency, enabling practical application in large-scale manufacturing systems.
- Implementing and testing the proposed scheduling framework in various industrial domains to validate its generalizability and practical impact across different production settings.
- Developing real-time scheduling approaches that leverage streaming data and rapid reoptimization to dynamically adjust schedules in response to evolving production conditions.

Pursuing these directions will not only strengthen the resilience and adaptability of scheduling models but also contribute to the development of more intelligent manufacturing systems capable of operating effectively in increasingly complex and dynamic environments.

7.5 Conclusion

This research has established a unified scheduling framework that integrates reactive adaptability with multi-objective optimization of due date adherence and intermediate storage. By simultaneously addressing customer service objectives and operational constraints within a flexible, policy-driven architecture, the developed methodologies offer robust conceptual solutions for maintaining high performance in dynamic manufacturing environments. This dissertation advances theoretical understanding of reactive scheduling under multi-criteria objectives and lays a strong foundation for future practical implementations that can enhance production resilience and competitiveness amidst increasing complexity and disruption.

7.6 Publications of the author

The core results of this doctoral research have been disseminated through peer-reviewed international journals and presented at both national and international conferences.

This section lists the author's publications, grouped according to their relevance to the thesis. The thesis group associated with each publication is indicated in parentheses. Publications are arranged in chronological order of appearance:

Publications directly related to the thesis:

[**P1**] **Krisztián Attila Bakon**, Tibor Holczinger, Zoltán Süle, Szilárd Jaskó, and János Abonyi: *Scheduling Under Uncertainty for Industry 4.0 and 5.0*, IEEE Access, vol. 10, pp. 74977–75017, 2022. DOI: 10.1109/ACCESS.2022.3191426.
Impact Factor: 3.367

[**P2**] **Krisztián Attila Bakon**, Tibor Holczinger: *S-Graph-Based Reactive Scheduling with Unexpected Arrivals of New Orders*, Machines, 2024. DOI: 10.3390/machines12070446.
Impact Factor: 2.1

[**P3**] **Krisztián Attila Bakon**, Tibor Holczinger: *Addressing Due Date and Storage Restrictions in the S-Graph Scheduling Framework*, Machines, 2025. DOI: 10.3390/machines13020131.
Impact Factor: 2.1

[**P4**] **Krisztián Attila Bakon**, Tibor Holczinger: *Reactive Scheduling under Due-Date and Intermediate Storage Constraints: A Conceptual Multi-Objective Approach*, in Proceedings of the Central European Conference on Information and Intelligent Systems (CECIIS), Varaždin, Croatia, 2025.

Other publications by the author (not directly related to the dissertation):

[**N1**] **Krisztián Attila Bakon**, Tibor Holczinger, Szilárd Jaskó: *Demonstration of Maturity Questions with Multipurpose Digital Factory*, Logistics – Information Technology – Management, Abstract book of conference presentations, p. 13, 2021.

[**N2**] **Krisztián Attila Bakon**, Tibor Holczinger, Szilárd Jaskó, Nikoletta Kaszás: *Tourism 4.0 and the Path of Achievement*, VI. International Scientific Conference on Tourism and Security, pp. 225–237, 2022. ISBN: 9789633962237

[**N3**] **Krisztián Attila Bakon**, Tibor Holczinger, Szilárd Jaskó: *Production Flow of Customized Products in a Digital Factory*, Procedia Computer Science, vol. 200, pp. 1201–1208, 2022. DOI: 10.1016/j.procs.2022.01.320.

[N4] Máté Hegyháti, **Krisztián Attila Bakon**, Tibor Holczinger: *Optimization with Uncertainties: A Scheduling Example*, Central European Journal of Operational Research, 2023. DOI: 10.1007/s10100-023-00854-4.
Impact Factor: 2.407

[N5] **Krisztián Attila Bakon**: *Agile Circularity: Reactive Scheduling Approaches*, in Mastering Time, Chapter 5, IntechOpen, Rijeka, 2025. DOI: 10.5772/intechopen.1006167.

Appendix A

Nomenclature

P is a finite set of products

I is a finite set of tasks

J is a finite set of equipments (or machines)

$I_p \subseteq I$ is the set of tasks needed to be carried out to produce product $p \in P$

$I_i^- \subseteq I$ is the set of prerequisite tasks of task $i \in I$

$J_i \subseteq J$ is the set of units capable of performing task $i \in I$

$I_j \subseteq I$ is the set of tasks that can be performed by unit $j \in J$

$pt_{i,j}$ is the processing time of task $i \in I$ performed by unit $j \in J$

d_p is the due date of product $p \in P$

t_i^s is the start time of task $i \in I$

t_i^f is the finish time of task $i \in I$

C_p is the completion time of product $p \in P$

E_p is the earliness time of product $p \in P$

T_p is the tardiness time of product $p \in P$

IS_p is the intermediate storage time of product $p \in P$

is_i is the intermediate storage time of task $i \in I$

$\hat{i}_p$ is the final task of product $p \in P$

w_p^E is the product-specific penalty weight assigned to earliness of product $p \in P$

w_p^T is the product-specific penalty weight assigned to tardiness of product $p \in P$

N is the set of nodes

$N^t \subset N$ is the set of task-nodes, where $|I| = |N^t|$

$N^p \subset N$ is the set of product-nodes, where $|P| = |N^p|$

A_1 is the set of recipe-arcs

A_2 is the set of schedule-arcs

A_{zw} is the set of zero-weight arcs, where $A_{zw} \subseteq A_1 \cup A_2$

$c(i_1, i_2) \in [0, \infty]$ is the weight of arc $(i_1, i_2) \in A_1 \cup A_2$

$G(N, A_1, A_2)$ is an S-graph

$G(N, A_1, \emptyset)$ is a recipe-graph

O is a finite ordered set of orders, where $o_{k_1} \in O$ precedes $o_{k_2} \in O$ if and only if $t_{k_1} < t_{k_2}$

t_k is the arrival time of order $o_k \in O$

$I_k \subseteq I$ is the set of tasks that have to be performed to fulfill order o_k, where $I = \bigcup_{o_k \in O} I_k$

$I_k^* = \bigcup_{k' \leq k} I_{k'}$ is the set of tasks that must be completed to fulfill orders $o_1, \ldots, o_k$

S_k is a feasible schedule of orders $o_1, \ldots, o_k \in O$

$(i, j, t^s, t^f) \in S_k$ is an assignment where task $i \in I$ is performed by unit $j \in J$, starting at t^s and finishing at t^f

$\underline{S}_k \subseteq S_k$ is the set of assignments (i, j, t^s, t^f) with $t^s < t_{k+1}$

$\overline{S}_k \subseteq S_k$ is the set of assignments (i, j, t^s, t^f) with $t^s \geq t_{k+1}$

$\underline{I}_k^* \subseteq I_k^*$ is the set of tasks started before t_{k+1} in schedule S_k

$\overline{I}_k^* \subseteq I_k^*$ is the set of tasks not started before t_{k+1} in schedule S_k

$G_k(N_k, A_{1k}, \emptyset)$ is the recipe-graph of order o_k

$n_k \in N_k$ is the arrival node of order o_k

t^s is the start time of a task i in the scheduling formulation

t^f is the generic finish time of a task i in the scheduling formulation

α is the weight coefficient assigned to intermediate storage minimization in the objective function

β is the weight coefficient assigned to earliness/tardiness minimization in the objective function

Z is the total objective value combining intermediate storage and earliness/tardiness penalties

$Z_{\mathbf{LB}}$ is the lower bound on the objective value used for branch-and-bound pruning

$Z_{\mathbf{best}}$ is the best-known value of the objective function in the current search

$I_{\mathbf{sched}} \subseteq I$ is the set of tasks with fixed execution intervals in the current partial schedule

$P_{\mathbf{sched}} \subseteq P$ is the set of products whose final task is already scheduled

i^- the task that immediately precedes task i in the recipe graph (i.e., the direct predecessor within the same product)

i^+ the task that immediately follows task i in the recipe graph (i.e., the direct successor within the same product)

δ is the slack value used to reduce intermediate storage of neighboring tasks during conflict resolution

Bibliography

[1] Wenqiang Zhang, Xuan Bao, Xinchang Hao, and Mitsuo Gen. "Metaheuristics for multi-objective scheduling problems in industry 4.0 and 5.0: a state-of-the-arts survey". In: Frontiers in Industrial Engineering Volume 3 - 2025 (2025). ISSN: 2813-6047. DOI: 10.3389/fieng.2025.1540022.

[2] Zheng-Wei Sun, Dan-Yang Lv, Cai-Min Wei, and Ji-Bo Wang. "Flow Shop Scheduling with Shortening Jobs for Makespan Minimization". In: Mathematics 13.3 (2025). ISSN: 2227-7390. DOI: 10.3390/math13030363.

[3] Krisztián Bakon, Tibor Holczinger, Zoltán Süle, Szilárd Jaskó, and János Abonyi. "Scheduling Under Uncertainty for Industry 4.0 and 5.0". In: IEEE Access 10 (2022), pp. 74977–75017. DOI: 10.1109/ACCESS.2022.3191426.

[4] Zohreh Shakeri, Khaled Benfriha, Mohsen Varmazyar, Esma Talhi, and Anthony Quenehen. "Production scheduling with multi-robot task allocation in a real industry 4.0 setting". In: Scientific Reports 15.1 (Jan. 2025), p. 1795. ISSN: 2045-2322. DOI: 10.1038/s41598-024-84240-3.

[5] Asif Ullah, Muhammad Younas, and Mohd Shahneel Saharudin. "Digital Twin Framework Using Real-Time Asset Tracking for Smart Flexible Manufacturing System". In: Machines 13.1 (2025). ISSN: 2075-1702. DOI: 10.3390/machines13010037.

[6] Yuxin Li, Qingzheng Wang, Xinyu Li, Liang Gao, Ling Fu, Yanbin Yu, and Wei Zhou. "Real-Time Scheduling for Flexible Job Shop With AGVs Using Multiagent Reinforcement Learning and Efficient Action Decoding". In: IEEE Transactions on Systems, Man, and Cybernetics: Systems 55.3 (2025), pp. 2120–2132. DOI: 10.1109/TSMC.2024.3520381.

[7] Feng Zhao and Lingli Wang. "Research and Implementation of Adaptive Production Scheduling Algorithm in Digital Transformation and Upgrading of Manufacturing Industry". In: International Journal of High Speed Electronics and Systems (2025), p. 2540228.

[8] A. Antunes, M. Matos, A. Rocha, L. Costa, and L. Varela. "A statistical comparison of metaheuristics for unrelated parallel machine scheduling problems with setup times". In: Mathematics 10 (14 2022), p. 2431. DOI: 10.3390/math10142431.

[9] Weiming Shen. "Distributed manufacturing scheduling using intelligent agents". In: IEEE Intelligent Systems 17.1 (2002), pp. 88–94. DOI: 10.1109/5254.988492.

[10] Michael R Garey, David S Johnson, and Ravi Sethi. "The complexity of flowshop and jobshop scheduling". In: Mathematics of operations research 1.2 (1976), pp. 117–129.

[11] Pablo Moscato, Alexandre Mendes, and Carlos Cotta. "Scheduling and Production & Control: MA". In: *New Optimization Techniques in Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 655–680. ISBN: 978-3-540-39930-8. DOI: 10.1007/978-3-540-39930-8_27.

[12] Danyu Bai. "Asymptotic analysis of online algorithms and improved scheme for the flow shop scheduling problem with release dates". In: International Journal of Systems Science 46.11 (2015), pp. 1994–2005.

[13] Molin Liu, Jun Lv, Shichang Du, Yafei Deng, Xiaoxiao Shen, and Yulu Zhou. "Multi-resource constrained flexible job shop scheduling problem with fixture-pallet combinatorial optimisation". In: Computers & Industrial Engineering 188 (2024), p. 109903.

[14] Teofilo Gonzalez and Sartaj Sahni. "Open Shop Scheduling to Minimize Finish Time". In: J. ACM 23.4 (Oct. 1976), pp. 665–679. ISSN: 0004-5411. DOI: 10.1145/321978.321985.

[15] Stéphane Dauzère-Pérès, Junwen Ding, Liji Shen, and Karim Tamssaouet. "The flexible job shop scheduling problem: A review". In: European Journal of Operational Research 314.2 (2024), pp. 409–432.

[16] Nurul Izah Anuar, Muhammad Hafidz Fazli Md Fauadi, and Adi Saptari. "Investigation of Flexible Job Shop with Machine Availability Constraints". In: Journal of Engineering Technology and Applied Physics 6.1 (2024), pp. 55–66.

[17] Haresh Gurmani and Pravin K. Johri. "A study of performance objectives in the scheduling of manufacturing lines". In: Journal of Manufacturing Systems 10.5 (1991), pp. 422–429. ISSN: 0278-6125. DOI: 10.1016/0278-6125(91)90059-B.

[18] Marco Ghirardi and Chris N Potts. "Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach". In: European Journal of Operational Research 165.2 (2005), pp. 457–467.

[19] John N Hooker. "Planning and scheduling to minimize tardiness". In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2005, pp. 314–327.

[20] P. Mohapatra, A. Nayak, S.K. Kumar, and M.K. Tiwari and. "Multi-objective process planning and scheduling using controlled elitist non-dominated sorting genetic algorithm". In: International Journal of Production Research 53.6 (2015), pp. 1712–1735. DOI: 10.1080/00207543.2014.957872.

[21] Hua Gong, Wanning Xu, Wenjuan Sun, and Ke Xu. "Multi-Objective Flexible Flow Shop Production Scheduling Problem Based on the Double Deep Q-Network Algorithm". In: Processes 11.12 (2023). ISSN: 2227-9717. DOI: 10.3390/pr11123321.

[22] Wook-Yeon Hwang and Jong-Seok Lee. "Compromising Multiple Objectives in Production Scheduling: A Data Mining Approach". In: Management Science and Financial Engineering 20.1 (2014), p. 1.

[23] Ali Allahverdi, Harun Aydilek, and Asiye Aydilek. "No-wait flowshop scheduling problem with two criteria; total tardiness and makespan". In: European Journal of Operational Research 269.2 (2018), pp. 590–601. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2017.11.070.

[24] Feidi Dang, Wei Li, and Honghan Ye. "An efficient constructive heuristic to balance trade-offs between makespan and flowtime in permutation flow shop scheduling". In: Procedia Manufacturing 26 (2018). 46th SME North American Manufacturing Research Conference, NAMRC 46, Texas, USA, pp. 40–48. ISSN: 2351-9789. DOI: 10.1016/j.promfg.2018.07.005.

[25] Mageed Ghaleb, Hossein Zolfagharinia, and Sharareh Taghipour. "Real-time production scheduling in the Industry-4.0 context: Addressing uncertainties in job arrivals and machine breakdowns". In: Computers & Operations Research 123 (2020), p. 105031. ISSN: 0305-0548. DOI: 10.1016/j.cor.2020.105031.

[26] D. Rahmani. "A new proactive-reactive approach to hedge against uncertain processing times and unexpected machine failures in the two-machine flow shop scheduling problems". In: Scientia Iranica 24.3 (2017), pp. 1571–1584. DOI: 10.24200/sci.2017.4136.

[27] S. Zhang and Q. Xu. "Reactive scheduling of short-term crude oil operations under uncertainties". In: Industrial and Engineering Chemistry Research 53.31 (2014), pp. 12502–12518. DOI: 10.1021/ie501588r.

[28] Di Wang, Weihua Liu, Xinran Shen, and Wanying Wei. "Service order allocation under uncertain demand: Risk aversion, peer competition, and relationship strength". In: Transportation Research Part E: Logistics and Transportation Review 130 (2019), pp. 293–311. ISSN: 1366-5545. DOI: 10.1016/j.tre.2019.09.005.

[29] E.N. Pistikopoulos, T.V. Thomaidis, A. Melin, and M.G. Ierapetritou. "Flexibility, reliability and maintenance considerations in batch plant design under uncertainty". In: Computers & Chemical Engineering 20 (1996), S1209–S1214. ISSN: 0098-1354. DOI: 10.1016/0098-1354(96)00209-8.

[30] Jon C. Helton. "Treatment of Uncertainty in Performance Assessments for Complex Systems". In: Risk Analysis 14.4 (1994), pp. 483–511. DOI: 10.1111/j.1539-6924.1994.tb00266.x.

[31] G Klir. "Uncertainty theories, measures, and principles: an overview of personal views and contributions". In: Mathematical Research 99 (1997), pp. 27–43.

[32] Daniel Kahneman and Amos Tversky. "Variants of uncertainty". In: Cognition 11.2 (1982), pp. 143–157. DOI: 10.1016/0010-0277(82)90023-3.

[33] K. Wang and S.H. Choi. "A decomposition-based approach to flexible flow shop scheduling under machine breakdown". In: International Journal of Production Research 50.1 (2012), pp. 215–234. DOI: 10.1080/00207543.2011.571456.

[34] Mark A. J. Huijbregts. "Application of uncertainty and variability in LCA". In: The International Journal of Life Cycle Assessment 3.5 (Sept. 1998), pp. 273–280. ISSN: 1614-7502. DOI: 10.1007/BF02979835.

[35] Georg Geisler, Stefanie Hellweg, and Konrad Hungerbühler. "Uncertainty Analysis in Life Cycle Assessment (LCA): Case Study on Plant-Protection Products and Implications for Decision Making (9 pp + 3 pp)". In: The International Journal of Life Cycle Assessment 10.3 (May 2005), pp. 184–192. ISSN: 1614-7502. DOI: 10.1065/lca2004.09.178.

[36] H.R. Maier, J.C. Ascough II, M. Wattenbach, C.S. Renschler, W.B. Labiosa, and J.K. Ravalico. "Chapter Five Uncertainty in Environmental Decision Making: Issues, Challenges and Future Directions". In: *Environmental Modelling, Software and Decision Support*. Ed. by A.J. Jakeman, A.A. Voinov, A.E. Rizzoli, and S.H. Chen. Vol. 3. Developments in Integrated Environmental Assessment. Elsevier, 2008, pp. 69–85. DOI: 10.1016/S1574-101X(08)00605-4.

[37] Guo Jin Hu, Pin Xin Fu, Mei Lin Wang, Qing Yun Dai, and Jian Cong Song. "A Review on Dynamic Production Scheduling and Solvable Algorithms". In: *Manufacturing Science and Technology III*. Vol. 622. Advanced Materials Research. Trans Tech Publications Ltd, Feb. 2013, pp. 1815–1820. DOI: 10.4028/www.scientific.net/AMR.622-623.1815.

[38] Djamila Ouelhadj and Sanja Petrovic. "A survey of dynamic scheduling in manufacturing systems". In: Journal of scheduling 12.4 (2009), pp. 417–431. DOI: 10.1007/s10951-008-0090-8.

[39] Willy Herroelen and Roel Leus. "Project scheduling under uncertainty: Survey and research potentials". In: European Journal of Operational Research 165.2 (2005), pp. 289–306. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2004.04.002.

[40] Der-Chang Li, Long-Sheng Chen, and Yao-San Lin and. "Using Functional Virtual Population as assistance to learn scheduling knowledge in dynamic manufacturing environments". In: International Journal of Production Research 41.17 (2003), pp. 4011–4024. DOI: 10.1080/0020754031000149211.

[41] E. J. Yellig and G.T. Mackulak and. "Robust deterministic scheduling in stochastic environments: The method of capacity hedge points". In: International Journal of Production Research 35.2 (1997), pp. 369–379. DOI: 10.1080/002075497195803.

[42] Zukui Li and Marianthi Ierapetritou. "Process scheduling under uncertainty: Review and challenges". In: Computers & Chemical Engineering 32.4 (2008). Festschrift devoted to Rex Reklaitis on his 65th Birthday, pp. 715–727. ISSN: 0098-1354. DOI: 10.1016/j.compchemeng.2007.03.001.

[43] Haldun Aytug, Mark A Lawley, Kenneth McKay, Shantha Mohan, and Reha Uzsoy. "Executing production schedules in the face of uncertainties: A review and some future directions". In: European Journal of Operational Research 161.1 (2005), pp. 86–110.

[44] Xiuli Wu, Zheng Cao, and Shaomin Wu. "Real-time hybrid flow shop scheduling approach in smart manufacturing environment". In: Complex System Modeling and Simulation 1.4 (2021), pp. 335–350.

[45] L.H. Wu, Xin Chen, X.D. Chen, and Qing Xin Chen. "The Research on Proactive-Reactive Scheduling Framework Based on Real-Time Manufacturing Information". In: *Advances in Materials Manufacturing Science and Technology XIII Volume I*. Vol. 626. Materials Science Forum. Trans Tech Publications Ltd, Nov. 2009, pp. 789–794. DOI: 10.4028/www.scientific.net/MSF.626-627.789.

[46] Florian Pleier and Johannes Schilp. "Predictive-Reactive Scheduling to Increase Robustness of Production Systems". In: *2024 4th International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*. 2024, pp. 1–6. DOI: 10.1109/ICECCME62383.2024.10796054.

[47] Hugo Zupan, Niko Herakovič, and Janez Žerovnik. "A Robust Heuristics for the Online Job Shop Scheduling Problem". In: Algorithms 17.12 (2024). ISSN: 1999-4893. DOI: 10.3390/a17120568.

[48] Charles A Holloway and Rosser Thornton Nelson. "Job shop scheduling with due dates and variable processing times". In: Management Science 20.9 (1974), pp. 1264–1275.

[49] Mohamed K Omar, Siew C Teo, and Yasothei Suppiah. "Scheduling with Setup Considerations: An MIP Approach". In: Multiprocessor Scheduling (2007), p. 241.

[50] Sönke Hartmann and Andreas Drexl. "Project scheduling with multiple modes: A comparison of exact algorithms". In: Networks: An International Journal 32.4 (1998), pp. 283–297.

[51] Erik L Demeulemeester and Willy S Herroelen. "A branch-and-bound procedure for the generalized resource-constrained project scheduling problem". In: Operations Research 45.2 (1997), pp. 201–212.

[52] Tao Ren, Meiting Guo, Lin Lin, and Yunhui Miao. "A local search algorithm for the flow shop scheduling problem with release dates". In: Discrete dynamics in nature and society 2015.1 (2015), p. 320140.

[53] Manoj Kumar Das. "Selected heuristic algorithms for solving job shop and flow shop scheduling problems". PhD thesis. National Institute of Technology, Rourkela, 2014.

[54] X Liu. "Research on the job shop real-time scheduling based on adaptive dispatching rule". In: Modular Mach. Tool Autom. Manuf. Technique 2 (2014), pp. 157–160.

[55] Jelle Adan. "A hybrid genetic algorithm for parallel machine scheduling with setup times: A comparative study of metaheuristics on large problem instances". In: Journal of Intelligent Manufacturing 33.7 (2022), pp. 2059–2073.

[56] Krisztián Mihály and Gyula Kulcsár. "A new many-objective hybrid method to solve scheduling problems". In: International Journal of Industrial Engineering and Management 14.4 (2023), pp. 326–335.

[57] Johann Hurink, Bernd Jurisch, and Monika Thole. "Tabu search for the job-shop scheduling problem with multi-purpose machines". In: Operations-Research-Spektrum 15 (1994), pp. 205–215.

[58] Jung-Sing Jwo, Cheng-Hsiung Lee, Jian-Tan Chen, Ching-Sheng Lin, Chun-Yu Lin, Wen-Kai Cheng, Chun-Hao Chang, and Jen-Kai King. "Application of Tabu Search for Job Shop Scheduling Based on Manufacturing Order Swapping". In: Engineering Proceedings 55.1 (2023). ISSN: 2673-4591. DOI: 10.3390/engproc2023055051.

[59] Daniel A Finke, Deborah J Medeiros, and Mark T Traband. "Shop scheduling using Tabu search and simulation". In: *Proceedings of the Winter Simulation Conference*. Vol. 1. IEEE. 2002, pp. 1013–1017.

[60] Mohamed Karim Hajji, Oumayma Hamlaoui, and Hatem Hadda. "A simulated annealing metaheuristic approach to hybrid flow shop scheduling problem". In: Advances in Industrial and Manufacturing Engineering 9 (2024), p. 100144. ISSN: 2666-9129. DOI: 10.1016/j.aime.2024.100144.

[61] Wenxin Wang. "Improved Simulated Annealing Algorithm for the Scheduling of Hybrid Flow-shop Problem". In: *2022 3rd International Conference on Computer Vision, Image and Deep Learning & International Conference on Computer Engineering and Applications (CVIDL & ICCEA)*. 2022, pp. 1176–1179. DOI: 10.1109/CVIDLICCEA56201.2022.9824804.

[62] Zhibo Feng, Jiguang Xue, Sitong Dong, Ye Tang, and Xuliang Zhao. "Simulated annealing algorithm-based scheduling optimization of lean manufacturing line and evaluation of the calibration effect of electrical measuring instruments". In: J. COMBIN. MATH. COMBIN. COMPUT 127 (2025), pp. 9197–9217.

[63] S. M. Mousavi, M. Zandieh, and M. Yazdani. "A simulated annealing/local search to minimize the makespan and total tardiness on a hybrid flowshop". In: The International Journal of Advanced Manufacturing Technology 64.1 (Jan. 2013), pp. 369–388. ISSN: 1433-3015. DOI: 10.1007/s00170-012-4031-5.

[64] Jinbao Zhang, Niansong Zhang, Aimin Wang, and Lulu Zhang. "Research on Hybrid Job Shop Scheduling Optimization Based on Genetic Algorithms and Simulated Annealing". In: *2024 IEEE International Conference on Mechatronics and Automation (ICMA)*. 2024, pp. 1141–1146. DOI: 10.1109/ICMA61710.2024.10633005.

[65] Dhananjay Thiruvady, Christian Blum, and Andreas T. Ernst. "Solution Merging in Matheuristics for Resource Constrained Job Scheduling". In: Algorithms 13.10 (2020). ISSN: 1999-4893. DOI: 10.3390/a13100256.

[66] Jianxin Tang and Wilfred V. Huang. "A decomposition method for generation scheduling of hydro systems with delays and unpredictable changes in natural inflows". In: Computers & Industrial Engineering 22.2 (1992), pp. 147–155. ISSN: 0360-8352. DOI: 10.1016/0360-8352(92)90041-H.

[67] Tatsushi Nishi, Masato Wakatake, and Masahiro Inuiguchi. "Decomposition of timed automata for solving scheduling problems". In: *2008 IEEE International Conference on Systems, Man and Cybernetics*. 2008, pp. 2459–2464. DOI: 10.1109/ICSMC.2008.4811664.

[68] José Miguel Laínez, Máté Hegyháti, Ferenc Friedler, and Luis Puigjaner. "Using S-graph to address uncertainty in batch plants". In: Clean Technologies and Environmental Policy 12 (2010), pp. 105–115.

[69] Zhengyang Hu and Guiping Hu. "A multi-stage stochastic programming for lot-sizing and scheduling under demand uncertainty". In: Computers & Industrial Engineering 119 (2018), pp. 157–166. ISSN: 0360-8352. DOI: 10.1016/j.cie.2018.03.033.

[70] Hanyu Shi and Fengqi You. "Robust optimisation for integrated planning, scheduling and dynamic optimisation of continuous processes under uncertainty". In: *Chemical engineering transactions*. Italian Association of Chemical Engineering-AIDIC, 2015, pp. 859–864.

[71] Hanyu Shi and Fengqi You. "Adjustable Robust Optimization for Scheduling of Batch Processes under Uncertainty". In: *26th European Symposium on Computer Aided Process Engineering*. Ed. by Zdravko Kravanja and Miloš Bogataj. Vol. 38. Computer Aided Chemical Engineering. Elsevier, 2016, pp. 547–552. DOI: 10.1016/B978-0-444-63428-3.50096-5.

[72] Dipak Laha. "Heuristics and metaheuristics for solving scheduling problems". In: *Handbook of Computational Intelligence in Manufacturing and Production Management*. IGI Global, 2008, pp. 1–18.

[73] Mariam Bouzid, Oussama Masmoudi, and Alice Yalaoui. "Exact Methods and Heuristics for Order Acceptance Scheduling Problem under Time-of-Use Costs and Carbon Emissions". In: Applied Sciences 11.19 (2021). ISSN: 2076-3417. DOI: 10.3390/app11198919.

[74] Xiaohui Li, Farouk Yalaoui, Lionel Amodeo, and Hicham Chehade. "Metaheuristics and exact methods to solve a multiobjective parallel machines scheduling problem". In: Journal of Intelligent Manufacturing 23.4 (Aug. 2012), pp. 1179–1194. ISSN: 1572-8145. DOI: 10.1007/s10845-010-0428-x.

[75] Susana Fernandes. "Optimised search heuristics: combining metaheuristics and exact methods to solve scheduling problems". PhD thesis. Universidade do Algarve (Portugal), 2008.

[76] Gyula Kulcsár, Mónika Kulcsárné Forrai, and Ákos Cservenák. "Objective Functions for Minimizing Rescheduling Changes in Production Control". In: Automation 6.3 (2025). ISSN: 2673-4052. DOI: 10.3390/automation6030030.

[77] Sofia Holguin Jimenez, Wajdi Trabelsi, and Christophe Sauvey. "Multi-Objective Production Rescheduling: A Systematic Literature Review". In: Mathematics 12.20 (2024). ISSN: 2227-7390. DOI: 10.3390/math12203176.

[78] B Shnits, J Rubinovitz*, and D Sinreich. "Multicriteria dynamic scheduling methodology for controlling a flexible manufacturing system". In: International Journal of Production Research 42.17 (2004), pp. 3457–3472.

[79] EO Oyetunji. "Minimizing the sum of makespan and maximum tardiness on single machine with release dates". In: *International Conference on Industrial Engineering and Operations Management, Istanbul, Turkey*. 2012.

[80] Fei Luan, Hongxuan Zhao, Shi Qiang Liu, Yixin He, and Biao Tang. "Enhanced NSGA-II for multi-objective energy-saving flexible job shop scheduling". In: Sustainable Computing: Informatics and Systems 39 (2023), p. 100901. ISSN: 2210-5379. DOI: 10.1016/j.suscom.2023.100901.

[81] Sheng Su and Haijie Yu. "Minimizing tardiness in data aggregation scheduling with due date consideration for single-hop wireless sensor networks". In: Wirel. Netw. 21.4 (May 2015), pp. 1259–1273. ISSN: 1022-0038. DOI: 10.1007/s11276-014-0853-4.

[82] Derya Eren Akyol and G. Mirac Bayhan. "Multi-machine earliness and tardiness scheduling problem: an interconnected neural network approach". In: The International Journal of Advanced Manufacturing Technology 37.5 (May 2008), pp. 576–588. ISSN: 1433-3015. DOI: 10.1007/s00170-007-0993-0.

[83] Pedro Palominos, Mauricio Mazo, Guillermo Fuertes, and Miguel Alfaro. "An Improved Marriage in Honey-Bee Optimization Algorithm for Minimizing Earliness/Tardiness Penalties in Single-Machine Scheduling with a Restrictive Common Due Date". In: Mathematics 13.3 (2025). ISSN: 2227-7390. DOI: 10.3390/math13030418.

[84] Vasiliki Lazari, Athanasios Chassiakos, and Stylianos Karatzas. "Multi-Objective Resource-Constrained Scheduling in Large and Repetitive Construction Projects". In: Algorithms 17.8 (2024). ISSN: 1999-4893. DOI: 10.3390/a17080351.

[85] Sujay N. Hegde, D. B. Srinivas, M. A. Rajan, Sita Rani, Aman Kataria, and Hong Min. "Multi-objective and multi constrained task scheduling framework for computational grids". In: Scientific Reports 14.1 (Mar. 2024), p. 6521. ISSN: 2045-2322. DOI: 10.1038/s41598-024-56957-8.

[86] Zhen Quan, Yan Wang, and Zhicheng Ji. "Multi-objective optimization scheduling for manufacturing process based on virtual workflow models". In: Applied Soft Computing 122 (2022), p. 108786.

[87] Sri Hartini, Ayusdilla Risvianni, Diana Puspita Sari, and Faradhina Azzahra. "A bibliometric analysis and visualization of sustainable production scheduling". In: World Journal of Advanced Engineering Technology and Sciences (2024). DOI: 10.30574/wjaets.2024.13.2.0536.

[88] Y Mauergauz. "Cost-efficiency method for production scheduling". In: *Proceedings of the world congress on engineering*. Vol. 1. 2013, pp. 3–5.

[89] Ali Jahan, Kevin L Edwards, and Marjan Bahraminasab. *Multi-criteria decision analysis for supporting the selection of engineering materials in product design*. Butterworth-Heinemann, 2016.

[90] Shuguang Li, Yong Sun, and Muhammad Ijaz Khan. "Single machine Pareto scheduling with positional deadlines and agreeable release and processing times." In: Electronic Research Archive 31.5 (2023).

[91] Qianying Cao, Shanqing Liu, Alan John Varghese, Jerome Darbon, Michael Triantafyllou, and George Em Karniadakis. *Automatic selection of the best neural architecture for time series forecasting via multi-objective optimization and Pareto optimality conditions*. 2025.

[92] Fan Yang. "Leveraging Mobile Interaction Technologies for Real-Time Decision Making in Enterprise Management Systems". In: International Journal of Interactive Mobile Technologies (iJIM) 19.02 (Jan. 2025), pp. 65–78. DOI: 10.3991/ijim.v19i02.53743.

[93] "The Role of Business Intelligence and Artificial Intelligence in Real-Time Decision Making". In: International Journal of Scientific Research and Management (IJSRM) 13.01 (Jan. 2025), pp. 1902–1916. DOI: 10.18535/ijsrm/v13i01.ec04.

[94] Nirwana Hendrastuty, M Setiawansyah, Fitrah Amalia Rahmadianti, Very Hendra Saputra, and Miftahur Rahman. "G2M weighting: a new approach based on multi-objective assessment data (case study of MOORA method in determining supplier performance evaluation)". In: Indonesian Journal of Electrical Engineering and Computer Science 38.1 (2025), pp. 403–416.

[95] Zhijie Li, Jianan Qi, and Jingquan Wang. "Multi-Objective Optimization Design of PCS Box Girder Bridges with Small and Medium Spans Using Genetic Algorithms". In: Buildings 15.3 (2025). ISSN: 2075-5309. DOI: 10.3390/buildings15030361.

[96] Marco Antonio Montufar-Benítez, Jaime Mora-Vargas, José Ramón Corona-Armenta, Gustavo Erick Anaya-Fuentes, Héctor Rivera-Gómez, and Mayra Rivera-Anaya. "An Actual Case Study of a Deterministic Multi-Objective Optimization Model in a Defined Contribution Faculty Pension System". In: Computation 13.2 (2025). ISSN: 2079-3197. DOI: 10.3390/computation13020025.

[97] Moheb Mottaghi and Saeed Mansour. "A multi-objective robust optimization model to sustainable closed-loop lithium-ion battery supply chain network design under uncertainties". In: Computers & Chemical Engineering 195 (2025), p. 109008. ISSN: 0098-1354. DOI: 10.1016/j.compchemeng.2025.109008.

[98] Bingjie Zhong, Chang Lin, Zhijie Huang, and Tianhua Lin. "Lightweight design of double-head machine tool beam based on the adaptive multi-objective method". In: Journal of Vibroengineering 27.2 (Jan. 2025), pp. 377–389. ISSN: 2538-8460. DOI: 10.21595/jve.2025.24641.

[99] Sona Babu and B.S. Girish. "Pareto-optimal front generation for the bi-objective JIT scheduling problems with a piecewise linear trade-off between objectives". In: Operations Research Perspectives 12 (2024), p. 100299. ISSN: 2214-7160. DOI: 10.1016/j.orp.2024.100299.

[100] Philipp Melchiors, Rainer Kolisch, and John J. Kanet. "The performance of priority rules for the dynamic stochastic resource-constrained multi-project scheduling problem: an experimental investigation". In: Annals of Operations Research 338.1 (July 2024), pp. 569–595. ISSN: 1572-9338. DOI: 10.1007/s10479-024-05841-9.

[101] M. I. Afrianto, F. Fauziah, and Y. F. Wijaya. "Kombinasi Algoritma Priority Scheduling dan Earliest Due Date untuk Sistem Penjadwalan Slitting Produk Berbasis Web". In: TEKNOKOM 7.1 (2024), pp. 180–186. DOI: 10.31943/teknokom.v7i1.176.

[102] Nameet Kumar Sethy and Dhiren Kumar Behera. "Resilient Heuristics for Permutation FS Scheduling: A Comparative Study of NEH And CDS Techniques with Priority and Tie-Breaking Rules". In: *Proceedings of the 3rd International Conference on Optimization Techniques in the Field of Engineering (ICOFE-2024)*. Nov. 2024. DOI: 10.2139/ssrn.5089007.

[103] Ali Soofastaei. "Intelligent Scheduling: How AI and Advanced Analytics Are Revolutionizing Time Optimization". In: *Mastering Time*. Ed. by Ali Soofastaei. Rijeka: IntechOpen, 2025. Chap. 2. DOI: 10.5772/intechopen.1008531.

[104] Zhangliang Wei, Zipeng Yu, Renzhong Niu, Qilong Zhao, and Zhigang Li. "Research on Flexible Job Shop Scheduling Method for Agricultural Equipment Considering Multi-Resource Constraints". In: Agriculture 15.4 (2025). ISSN: 2077-0472. DOI: 10.3390/agriculture15040442.

[105] Elizabeth Jordan, Arko Chatterjee, Ruochen Yang, Katrina Groth, and Shapour Azarm. "On a Sampling-Based Method for Multi-Objective Robust Optimization1". In: Journal of Computing and Information Science in Engineering 25.4 (Feb. 2025), p. 044501. ISSN: 1530-9827. DOI: 10.1115/1.4067706.

[106] Krisztián Attila Bakon and Tibor Holczinger. "S-Graph-Based Reactive Scheduling with Unexpected Arrivals of New Orders". In: Machines 12.7 (2024). ISSN: 2075-1702. DOI: 10.3390/machines12070446.

[107] Amritpal Singh Raheja and Velusamy Subramaniam. "Reactive schedule repair of job shops". In: (2002).

[108] Jun Yan, Tianzuo Zhao, Tao Zhang, Hongyan Chu, Congbin Yang, and Yueze Zhang. "A Dynamic Scheduling Method Combining Iterative Optimization and Deep Reinforcement Learning to Solve Sudden Disturbance Events in a Flexible Manufacturing Process". In: Mathematics 13.1 (2025). ISSN: 2227-7390. DOI: 10.3390/math13010004.

[109] Taicheng Zheng, Dan Li, and Jie Li. "A Rescheduling Strategy for Multipurpose Batch Processes with Processing Time Variation and Demand Uncertainty". In: Processes 13.2 (2025). ISSN: 2227-9717. DOI: 10.3390/pr13020312.

[110] Juan Novas and Gabriela Henning. "A Reactive Scheduling Framework for Batch Plants: Balancing Schedule Stability and Efficiency Performance Measures". In: Oct. 2012.

[111] Zied Bahroun, Abdulrahim Shamayleh, Rami As'ad, and Rim Zakaria. "Integrated proactive-reactive tool for dynamic scheduling of parallel machine operations". In: International Journal of Engineering Business Management 16 (2024), p. 18479790241301164. DOI: 10.1177/18479790241301164.

[112] Betul Kayisoglu, Seyma Bekli, Ayse Sena Sahin, Gamze Gul Akyurek, Ruveyda Aydinli, Sevda Nur Copur, and Tugba Ekinci. "Parallel Machine Scheduling with Re-entrant Jobs with Consideration of Set up Times". In: The Eurasia Proceedings of Science Technology Engineering and Mathematics 32 (2024), pp. 311–319. DOI: 10.55549/epstem.1602789.

[113] Andy J. Figueroa, Raul Poler, and Beatriz Andres. "Adaptive Production Rescheduling System for Managing Unforeseen Disruptions". In: Mathematics 12.22 (2024). ISSN: 2227-7390. DOI: 10.3390/math12223478.

[114] Daniel Ovalle, Joshua L. Pulsipher, Yixin Ye, Kyle Harshbarger, Scott Bury, Carl D. Laird, and Ignacio E. Grossmann. *Optimal Reactive Operation of General Topology Supply Chain and Manufacturing Networks under Disruptions*. 2024.

[115] Engelbert Pasieka and Sebastian Engell. "Reactive Real-time Scheduling Using Simulation-Optimization and Evolutionary Algorithms". In: *2024 10th International Conference on Control, Decision and Information Technologies (CoDIT)*. 2024, pp. 2470–2475. DOI: 10.1109/CoDIT62066.2024.10708608.

[116] Kyeongho Kim, Minjoo Choi, Haram Seo, Jaekyeong Lee, Jihong Kim, and Shinhyo Kim. "A Mixed Integer Linear Programming Model for Rapid Rescheduling in Ship and Offshore Unit Design Projects". In: Journal of Marine Science and Engineering 13.2 (2025). ISSN: 2077-1312. DOI: 10.3390/jmse13020222.

[117] Yulong Yang, Han Yan, Jiaqi Wang, Weiyang Liu, and Zhongwen Yan. "System Optimization Scheduling Considering the Full Process of Electrolytic Aluminum Production and the Integration of Thermal Power and Energy Storage". In: Energies 18.3 (2025). ISSN: 1996-1073. DOI: 10.3390/en18030598.

[118] H. T. Madan, H. M. Manjunatha, Pradeep Nagaraja Rao, and M. Vidyashankar. "CPU Scheduling Algorithms Performance Analysis in the RISC-V xv6 Operating System Environment". In: Journal of Integrated Science and Technology 13.3 (Dec. 2024), p. 1053. DOI: 10.62110/sciencein.jist.2025.v13.1053.

[119] Juan M Novas and Gabriela P Henning. "Reactive scheduling framework based on domain knowledge and constraint programming". In: Computers & Chemical Engineering 34.12 (2010), pp. 2129–2148.

[120] Shu-Hsing Chung, Ming-Hsien Yang, and Ching-Kuei Kao. "Reactive scheduling to minimize makespan of parallel-machine problem with job arrival in uncertainty". In: African Journal of Business Management 6.27 (2012), p. 7995.

[121] M. Geetha, R. Chandra Guru Sekar, and M. K. Marichelvam. "A Hybrid Honey Badger Algorithm to Solve Energy-Efficient Hybrid Flow Shop Scheduling Problems". In: Processes 13.1 (2025). ISSN: 2227-9717. DOI: 10.3390/pr13010174.

[122] L. H. Wu, X. Chen, X. D. Chen, and Q. X. Chen. "The Research on Proactive-Reactive Scheduling Framework Based on Real-Time Manufacturing Information". In: Materials Science Forum 626–627 (2009), pp. 789–794. DOI: 10.4028/www.scientific.net/MSF.626-627.789.

[123] J. Kale. "A Hybrid Approach to Multi-Objective Task Scheduling in Cloud Computing: Merging Estimation of Distribution and Genetic Algorithms". In: Indian Scientific Journal of Research in Engineering and Management 09.01 (2025), pp. 1–9. DOI: 10.55041/ijsrem40743.

[124] F. Zaman, S. Elsayed, R. Sarker, D. Essam, and C. A. C. Coello. "Pro-Reactive Approach for Project Scheduling Under Unpredictable Disruptions". In: IEEE Transactions on Cybernetics 52.11 (2022), pp. 11299–11312. DOI: 10.1109/TCYB.2021.3097312.

[125] Jianguo Duan, Fanfan Liu, Qinglei Zhang, Jiyun Qin, and Ying Zhou. "Genetic programming hyper-heuristic-based solution for dynamic energy-efficient scheduling of hybrid flow shop scheduling with machine breakdowns and random job arrivals". In: Expert Systems with Applications 254 (2024), p. 124375. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2024.124375.

[126] Yajie Wu, Shiming Yang, Leilei Meng, Weiyao Cheng, Biao Zhang, and Peng Duan. "A novel hybrid algorithm of cooperative variable neighborhood search and constraint programming for flexible job shop scheduling problem with sequence dependent setup time". In: International Journal of Industrial Engineering Computations 16.1 (2025), pp. 21–36.

[127] Songling Tian, Taiyong Wang, Lei Zhang, and Xiaoqiang Wu. "Real-time shop floor scheduling method based on virtual queue adaptive control: Algorithm and experimental results". In: Measurement 147 (2019), p. 106689. ISSN: 0263-2241. DOI: 10.1016/j.measurement.2019.05.080.

[128] E. Sanmartí, L. Puigjaner, T. Holczinger, and F. Friedler. "Combinatorial framework for effective scheduling of multipurpose batch plants". In: AIChE Journal 48.11 (2002), pp. 2557–2570. DOI: 10.1002/aic.690481115.

[129] J. Smidla, I. Heckl, and F. Friedler. "S-graph Based Parallel Algorithm to the Scheduling of Multipurpose Batch Plants." In: Chemical Engineering Transactions 21 (Aug. 2010), pp. 937–942. DOI: 10.3303/CET1021157.

[130] Máté Hegyháti, Tibor Holczinger, and Olivér Ősz. "Addressing storage time restrictions in the S-graph scheduling framework". In: Optimization and Engineering 22.4 (Dec. 2021), pp. 2679–2706. ISSN: 1573-2924. DOI: 10.1007/s11081-020-09548-1.

[131] Oliver Osz and Mate Hegyhati. "An S-graph Based Approach for Multi-Mode Resource-Constrained Project Scheduling with Time-Varying Resource Capacities". In: Chemical Engineering Transactions 70 (Aug. 2018), pp. 1165–1170. DOI: 10.3303/CET1870195.

[132] J Romero, L Puigjaner, T Holczinger, and F Friedler. "Scheduling Intermediate Storage Multipurpose Batch Plants Using the S-Graph". In: AIChE Journal 50(2) (2004), pp. 403–417. DOI: 10.1002/aic.10036.

[133] Mate Hegyháti, Tibor Holczinger, András Szoldatics, and Ferenc Friedler. "Combinatorial approach to address batch scheduling problems with limited storage time". In: CHEMICAL ENGINEERING 25 (2011), p. 495. DOI: 10.3303/CET1125083.

[134] R Adonyi, T Holczinger, and F Friedler. "Novel branching procedure for S-graphs based scheduling of batch processes". In: *Proceedings of 19th Polish conference of chemical and process engineering. Oficyna Wydawnicza Politechniki Wroclawskiej.* 2007, p. 9.

[135] Róbert Adonyi. "Batch process scheduling with the extensions of the S-graph framework". PhD thesis. Doctoral School of Information Science and Technology, University of Pannonia, 2008.

[136] Marcos Singer and Michael Pinedo. "A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops". In: IIE Transactions 30.2 (Feb. 1998), pp. 109–118. ISSN: 1573-9724. DOI: 10.1023/A:1007405915227.

[137] Jens Kuhpfahl. *Job shop scheduling with consideration of due dates: Potentials of local search based solution techniques.* Springer, 2015.

[138] SAMUEL Eilon and I. G. Chowdhury. "Due dates in job shop scheduling". In: International Journal of Production Research 14.2 (1976), pp. 223–237. DOI: 10.1080/00207547608956596.

[139] Imen Essafi, Yazid Mati, and Stéphane Dauzère-Pérès. "A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem". In: Computers & Operations Research 35.8 (2008). Queues in Practice, pp. 2599–2616. ISSN: 0305-0548. DOI: 10.1016/j.cor.2006.12.019.

[140] Paolo Brandimarte. "Routing and scheduling in a flexible job shop by tabu search". In: Annals of Operations Research 41.3 (Sept. 1993), pp. 157–183. ISSN: 1572-9338. DOI: 10.1007/BF02023073.

[141] Donya Rahmani and Mahdi Heydari. "Robust and stable flow shop scheduling with unexpected arrivals of new jobs and uncertain processing times". In: Journal of Manufacturing Systems 33.1 (2014), pp. 84–92. ISSN: 0278-6125. DOI: 10.1016/j.jmsy.2013.03.004.

[142] Kai Zhou Gao, Ponnuthurai Nagaratnam Suganthan, Tay Jin Chua, Chin Soon Chong, Tian Xiang Cai, and Qan Ke Pan. "A two-stage artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job insertion". In: Expert Systems with Applications 42.21 (2015), pp. 7652–7663. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2015.06.004.

[143] Adil Baykasoğlu, Fatma S. Madenoğlu, and Alper Hamzadayı. "Greedy randomized adaptive search for dynamic flexible job-shop scheduling". In: Journal of Manufacturing Systems 56 (2020), pp. 425–451. ISSN: 0278-6125. DOI: 10.1016/j.jmsy.2020.06.005.

[144] Jianguo Duan and Jiahui Wang. "Robust scheduling for flexible machining job shop subject to machine breakdowns and new job arrivals considering system reusability and task recurrence". In: Expert Systems with Applications 203 (2022), p. 117489. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2022.117489.

[145] Parviz Fattahi and Alireza Fallahi. "Dynamic scheduling in flexible job shop systems by considering simultaneously efficiency and stability". In: CIRP Journal of Manufacturing Science and Technology 2.2 (2010), pp. 114–123. ISSN: 1755-5817. DOI: 10.1016/j.cirpj.2009.10.001.

[146] Xinchang Hao and Lin Lin. "Job shop rescheduling by using multi-objective genetic algorithm". In: *The 40th International Conference on Computers & Indutrial Engineering*. 2010, pp. 1–6. DOI: 10.1109/ICCIE.2010.5668422.

[147] Yufeng Li, Yan He, Yulin Wang, Fei Tao, and John W. Sutherland. "An optimization method for energy-conscious production in flexible machining job shops with dynamic job arrivals and machine breakdowns". In: Journal of Cleaner Production 254 (2020), p. 120009. ISSN: 0959-6526. DOI: 10.1016/j.jclepro.2020.120009.

[148] Shokraneh K. Moghaddam and Kazuhiro Saitou. "On optimal dynamic pegging in rescheduling for new order arrival". In: Computers & Industrial Engineering 136 (2019), pp. 46–56. ISSN: 0360-8352. DOI: 10.1016/j.cie.2019.07.012.

[149] Ahmad Shahrizal Muhamad, Zalmiyah Zakaria, and Safaai Deris. "Rescheduling for JSSP and FJSSP using Clonal Selection Principle Approach–A Theory". In: Journal of Information 1.1 (2016), pp. 10–20.

[150] Rylan H. Caldeira, A. Gnanavelbabu, and T. Vaidyanathan. "An effective backtracking search algorithm for multi-objective flexible job shop scheduling considering new job arrivals and energy consumption". In: Computers & Industrial Engineering 149 (2020), p. 106863. ISSN: 0360-8352. DOI: 10.1016/j.cie.2020.106863.

[151] Krisztián Attila Bakon and Tibor Holczinger. "Addressing Due Date and Storage Restrictions in the S-Graph Scheduling Framework". In: Machines 13.2 (2025). ISSN: 2075-1702. DOI: 10.3390/machines13020131.

[152] Andrew Makhorin. *GNU Linear Programming Kit: Reference Manual*. Version 4.65. Department for Applied Informatics, Moscow Aviation Institute. 2018. URL: http://www.gnu.org/software/glpk/.

[153] Alejandro Vital-Soto and Jessica Olivares-Aguila. "Flexible Job-Shop Scheduling Problem with Sequencing Flexibility for Just-In-Time Production Systems". In: *Proceedings of the 11th Annual International Conference on Industrial Engineering and Operations Management (IEOM)*. Mar. 2021. DOI: 10.46254/AN11.20210968.

[154] Gregory A. Kasapidis, Dimitris C. Paraskevopoulos, Ioannis Mourtos, and Panagiotis P. Repoussis. "A unified solution framework for flexible job shop scheduling problems with multiple resource constraints". In: European Journal of Operational Research 320.3 (2025), pp. 479–495. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2024.08.010.

[155] David R Karger, Cliff Stein, and Joel Wein. "Scheduling algorithms". In: Algorithms and theory of computation handbook 1 (1999), pp. 20–20.

[156] Guilherme E. Vieira, Jeffrey W. Herrmann, and Edward Lin. "Rescheduling Manufacturing Systems: A Framework of Strategies, Policies, and Methods". In: Journal of Scheduling 6.1 (Jan. 2003), pp. 39–62. ISSN: 1099-1425. DOI: 10.1023/A:1022235519958.

[157] Ned Freed and Fred Glover. "LINEAR PROGRAMMING AND STATISTICAL DISCRIMINATION THE LP SIDE". In: Decision Sciences 13.1 (1982), pp. 172–175. DOI: 10.1111/j.1540-5915.1982.tb00140.x.

[158] Jeffrey W Herrmann. *Handbook of production scheduling*. Vol. 89. Springer Science & Business Media, 2006.