[Inner endpaper]

# HEC MONTRÉAL

## Resource-constrained Activity Sequencing Problem

par

**Xiao Li**

**Robert Pellerin**

**Polytechnique Montréal**

**Codirecteur de recherche**

**Okan Arslan**

**HEC Montréal**

**Codirecteur de recherche**

**Sciences de la gestion**

**(Spécialisation en science des données et analytique d'affaires)**

Mémoire présenté en vue de l'obtention
du grade de maîtrise ès sciences en gestion
(M. Sc.)

Juillet 2024

# Résumé

Ce mémoire étudie le problème de séquencement d'activités avec contraintes de ressources (RASP - *Resource-Constrained Activity Sequencing Problem*), un problème d'optimisation combinatoire qui implique de séquencer des activités divisibles dans un horizon de planification donné tout en respectant les contraintes de capacité liée à l'utilisation d'une ressource commune unique. L'exécution de chaque activité consomme la ressource commune et les objectifs sont de minimiser hiérarchiquement la durée totale d'exécution des activités (makespan), les coûts opérationnels totaux et la ressource inutilisée au fil du temps. Le RASP se distingue des problèmes d'ordonnancement similaires, tels que le problème d'ordonnancement de projet avec contraintes de ressources, le problème d'ordonnancement en atelier et le problème d'ordonnancement séquentiel par ses caractéristiques uniques : il permet l'exécution partielle des activités, et il existe des contraintes spécifiques sur l'utilisation et la libération de la ressource. Nous présentons un modèle mathématique et développons des algorithmes métaheuristiques pour résoudre efficacement le RASP. Les algorithmes métaheuristiques incluent un algorithme glouton et un algorithme basé sur la recherche taboue, conçu pour améliorer les résultats de l'algorithme glouton. Nous testons nos algorithmes sur des ensembles de données de différentes tailles pour évaluer leur performance. Nos résultats computationnels montrent que l'algorithme basé sur la recherche taboue est très efficace et qu'il peut trouver la durée optimale dans 85% des petites instances, pour lesquelles une solution optimale est obtenue par le modèle mathématique dans une limite de temps. Nous démontrons également son efficacité sur les grandes instances.

**Mots clés :** Problème de Séquencement d'Activités avec contraintes de Ressources ; Algorithme Glouton ; Recherche taboue ; Optimisation Combinatoire ; Activités Divisibles

# Avant-Propos

Ce mémoire est rédigé sous la forme d'un article qui sera soumis à une revue scientifique. La direction du programme de M.Sc. ainsi que les co-auteurs ont donné l'autorisation pour son utilisation.

# Table of Contents

# List of Tables and Figures

## List of Tables



## List of Figures

# List of Abbreviations and Acronyms

**JSSP** Job Shop Scheduling Problem

**RASP** Resource-constrained Activity Sequencing Problem

**RCPSP** Resource-Constrained Project Scheduling Problem

**SOP** Sequential Ordering Problem

# Acknowledgements

I would like to express my gratitude to all those who supported me throughout my master's degree and contributed to the completion of this thesis.

First, I would like to thank Professor Okan Arslan, my co-supervisor at HEC Montréal, for offering me the opportunity to work on this project and for his significant involvement in its completion. I would also like to thank Professor Robert Pellerin, my co-supervisor at Polytechnique Montréal, for his valuable expert suggestions and guidance throughout this journey.

Next, I would like to extend my thanks to Maxime Dumas and Hugo Deschênes from Croesus, whose expertise in understanding the project's challenges and their coding suggestions were important to its successful completion.

Additionally, I am deeply grateful to the MITACS scholarship for the financial support that made this research possible.

Finally, I would like to thank my mother and father for their constant support and encouragement from the beginning to the end of my master's degree.

# Chapitre 1: Introduction

L'ordonnancement et le séquencement des activités avec une ressource commune en recherche opérationnelle ont plusieurs applications dans des domaines allant de la fabrication aux industries de services. Ces problèmes sont souvent caractérisés par la nécessité d'allouer efficacement des ressources et de planifier des activités afin de minimiser les coûts ou d'améliorer la productivité. Ce mémoire introduit le problème de séquencement d'activités avec contraintes de ressources (RASP - *Resource-Constrained Activity Sequencing Problem*), un problème d'optimisation combinatoire qui implique de séquencer des activités divisibles dans un horizon de planification donné, où chaque activité augmente ou diminue le niveau d'une ressource commune unique.

Dans le RASP, l'horizon de planification est divisé en $|T|$ intervalles de temps de longueur égale, notés $[t-1, t), t = 1, ..., |T|$, appelés périodes de temps. Une ressource commune unique est utilisée, chaque activité a une durée spécifique, et elle augmente ou diminue le niveau de la ressource. Les activités qui augmentent le niveau de la ressource sont appelées activités positives, tandis que celles qui le diminuent sont appelées activités négatives. Toutes les activités doivent être entièrement complétées dans l'horizon de planification, et des exécutions partielles sont autorisées. Nous désignons chaque exécution d'une activité, qu'elle soit complète ou partielle, comme une opération. Chaque opération a une date d'engagement, marquant le début de l'exécution, et une date de règlement, signifiant la fin de l'exécution. Une opération d'activité négative consomme la ressource à sa date d'engagement, tandis qu'une opération d'activité positive augmente la ressource à sa date de règlement. De plus, chaque opération entraîne un coût fixe, appelé coût opérationnel, qui diminue la ressource commune à la date d'engagement. En conséquence, les exécutions partielles d'une activité entraînent des coûts opérationnels supplémentaires. Il n'y a pas de relations de précédence entre les opérations. Par conséquent, le RASP est défini comme un problème de séquencement plutôt qu'un problème d'ordonnancement, en se concentrant sur la séquence des actions à décider.

Pour illustrer l'application pratique du RASP, considérons le contexte d'un système de gestion d'entrepôt. Dans l'entreposage, les nouveaux produits envoyés d'une usine à l'entrepôt augmentent le stock lors de leur livraison, de la même manière que les activités positives augmentent le niveau

d'une ressource commune lors de leur achèvement à la date de règlement. À l'inverse, lorsque les clients passent des commandes, les produits sont expédiés immédiatement depuis l'entrepôt, ce qui réduit instantanément les stocks, semblable aux activités négatives diminuant le niveau de la ressource au moment de l'exécution à la date d'engagement. Le coût opérationnel peut être comparé aux pertes subies lors de la réception ou de l'expédition des produits, chaque transaction comporte un certain niveau de gaspillage. La durée d'une activité correspond au temps de traitement, ce qui signifie qu'une expédition n'est complète que lorsque le temps de traitement spécifié pour le produit est écoulé. Étant donné que le stockage des produits entraîne des frais de stockage, une gestion efficace des stocks est importante. Cela nécessite de gérer soigneusement le plan de l'offre et de la demande afin de maintenir un stock minimal sans permettre aux niveaux de stock de tomber en dessous de zéro. Pour atteindre cet équilibre, la réception partielle de produits de l'usine ou les livraisons partielles aux clients sont autorisées.

Le RASP présente un problème de séquencement d'activités difficile visant à optimiser trois objectifs hiérarchiques : (1) minimiser la durée totale d'exécution des activités (makespan), (2) minimiser les coûts opérationnels totaux et (3) minimiser la ressource inutilisée de chaque période de temps. De plus, il est essentiel de développer un algorithme efficace capable de résoudre de grands instances de ce problème en moins d'une minute afin de répondre à des applications complexes du monde réel. Ce mémoire définit formellement le RASP, souligne les différences et similitudes entre le RASP et d'autres problèmes connexes, y compris le problème d'ordonnancement de projet avec contraintes de ressources (RCPSP - *Resource-Constrained Project Scheduling Problem*), le problème d'ordonnancement en atelier (JSSP - *Job Shop Scheduling Problem*) et le problème d'ordonnancement séquentiel (SOP - *Sequential Ordering Problem*). En comparant le RASP à ces problèmes, nous illustrons que le RASP a des aspects uniques de gestion de la ressource et d'activités divisibles. Pour résoudre le RASP, nous présentons un modèle mathématique et développons des algorithmes métaheuristiques. Des expériences informatiques approfondies démontrent l'efficacité des algorithmes sur diverses instances de problèmes, montrant leur robustesse et leur évolutivité.

Ce mémoire est structuré en trois chapitres principaux. Le Chapitre 1 constitue une traduction de l'introduction de l'article scientifique. Il présente le problème de séquencement d'activités

avec contraintes de ressources (RASP), son application, et les principaux objectifs de la recherche. Le Chapitre 2 correspond à l'article lui-même, rédigé en anglais pour une soumission à une revue scientifique, où nous détaillons le modèle mathématique et les algorithmes métaheuristiques développés, ainsi que les résultats obtenus. Enfin, le Chapitre 3 est une traduction de la conclusion de l'article, en résumant les contributions principales de cette recherche et en suggérant des pistes pour des recherches futurs, notamment concernant les améliorations potentielles des algorithmes.

L'article est organisé comme suit. La Section 2 donne un aperçu de la littérature pertinente. Dans la Section 3, nous introduisons le problème, discutons des compromis dans le problème multi-objectifs et présentons un modèle mathématique. La Section 4 introduit les algorithmes métaheuristiques utilisés pour résoudre le problème. La Section 5 discute de nos résultats expérimentaux, y compris la création de jeux de données, les mesures d'évaluation et les comparaisons de performances entre les différentes méthodes. Enfin, la Section 6 résume nos principaux résultats et discute des directions futures de la recherche.

# Chapitre 2: Article

## Abstract

This paper studies the Resource-constrained Activity Sequencing Problem (RASP), a combinatorial optimization problem that involves sequencing divisible activities within a given planning horizon while respecting capacity constraints related to the use of a single common resource. The execution of each activity consumes the common resource, and the objectives are to minimize hierarchically the makespan, the total operational costs, and the unused resource over time. The RASP is different from related scheduling problems, such as the Resource-Constrained Project Scheduling Problem, Job Shop Scheduling Problem, and Sequential Ordering Problem by its unique features: it allows partial activity execution, and there are specific constraints on the use and release of resource. We present a mathematical model and develop metaheuristic algorithms to solve the RASP efficiently. Metaeuristic algorithms include a greedy algorithm and a tabu search-based algorithm, designed to enhance the results of the greedy algorithm. We test our algorithms on datasets of various instance sizes to evaluate their performance. Our computational results show that the tabu search-based algorithm is highly effective and that it can find the optimal makespan in 85% of the small instances, for which an optimal solution is obtained by the mathematical model within a time limit. We also demonstrate its efficiency on large instances.

**Keywords:** Resource-Constrained Activity Sequencing Problem; Greedy Algorithm; Tabu Search; Combinatorial Optimization; Divisible Activities

# Section 1: Introduction

The scheduling and sequencing of activities with a common resource in operations research have several applications in domains ranging from manufacturing to service industries. These problems are often characterized by the need to efficiently allocate resource and schedule activities in order to minimize costs or to improve productivity. This paper introduces the Resource-constrained Activity Sequencing Problem (RASP), a combinatorial problem involving the sequencing of divisible activities within a planning horizon, where each activity either increases or decreases a single common resource.

In RASP, the planning horizon is divided into $|T|$ equal-length time intervals, denoted as $[t-1, t), t = 1, ..., |T|$, referred to as time periods. A single common resource is utilized, each activity has a specific duration and it either increases or decreases the level of the resource. Activities that increase the resource are referred to as the positive activities, while those that decrease it are called the negative activities. All activities must be fully completed within the planning horizon, and partial execution is allowed. We refer to every execution of an activity, whether full or partial, as an operation. Each operation has a commitment date, marking the start of execution, and a settlement date, signifying the end of execution. An operation of negative activity consumes the resource at its commitment, whereas an operation of positive activity increases the resource at its settlement. Additionally, each operation incurs a fixed cost, referred to as the operational cost, which decreases the common resource at the commitment date. As the result, partial executions of an activity lead to additional operational cost. There are no precedence relationships between operations. Hence, RASP is defined as a sequencing problem rather than a scheduling problem, focusing on the sequence of actions to be decided.

To illustrate the practical application of RASP, consider the context of a warehouse management system. In warehousing, newly produced items sent from a plant to the warehouse increase inventory upon delivery, similar to how positive activities increase the common resource upon completion at the settlement date. Conversely, when customers place orders, items are shipped immediately from the warehouse, instantly reducing inventory, akin to negative activities decreasing the resource at the moment of execution on the commitment date. The operational cost can be com-

pared to the losses incurred during the receiving or shipping of items, as each transaction carries some level of wastage. The length of an activity corresponds to the processing time, meaning a shipment is only complete once the specified processing time for the item has passed. Since storing items incurs storage costs, efficient inventory management is important. This requires carefully managing the timing of both supply and demand to maintain a minimal inventory without allowing inventory levels to fall below zero. To achieve this balance, partial receipt of items from the plant or partial deliveries to customers are permitted.

RASP presents a challenging activity sequencing problem that aims to optimize three hierarchical objectives: (1) minimizing the makespan of executing activities, (2) minimizing the total operational costs, and (3) minimizing the unused resource of each time period. Additionally, it is essential to develop an efficient algorithm that is capable of solving large instances of this problem within a minute in order to address complex real-world applications. This paper formally defines the RASP, and underlines the differences and similarities between the RASP, and other related problems including the Resource-Constrained Project Scheduling Problem (RCPSP), Job Shop Scheduling Problem (JSSP), and Sequential Ordering Problem (SOP). By comparing the RASP to these problems, we illustrate that the RASP has unique aspects of resource management and divisible activities. To solve the RASP, we present a mathematical model and develop metaheuristic algorithms. Extensive computational experiments demonstrate the efficiency of the algorithms across various problem instances, showcasing its robustness and scalability.

This paper is organized as follows. Section 2 provides an overview of the relevant literature. In Section 3, we introduce the problem, discuss the trade-off in the multi-objective problem, and present a mathematical model. Section 4 introduces metaheuristic algorithms used to solve the problem. Section 5 discusses our experimental results, including dataset creation, evaluation metrics, and performance comparisons between different methods. Finally, Section 6 summarizes our main findings and discusses future research directions.

# Section 2: Literature Review

In this section, we review the related literatures and compare the RCPSP, the JSSP, and the SOP to RASP in Section 2.1, 2.2, and 2.3, respectively. We then present a summary of these comparisons to highlight the research gap in Section 2.4.

## 2.1 Resource-Constrained Project Scheduling Problem

The RCPSP was introduced in 1969 by Pritsker et al. (1969). It is a classic problem in project management that has been extensively studied for over two decades (Pellerin et al. 2020). RCPSP involves scheduling a set of activities within a given time horizon, where each activity has a specific duration and requires certain amount of resource. The primary objective is to minimize the total project duration (makespan) while respecting resource constraints and precedence relations.

Research on RCPSP has yielded various exact and metaheuristic methods. Exact methods such as branch-and-bound, branch-and-cut, and branch-and-price guarantee optimal solutions but are often computationally intensive for large-scale problems (De Reyck & Herroelen 1998, Montoya et al. 2014, Shirzadeh Chaleshtarti & Shadrokh 2014). Metaheuristic approaches, including Genetic Algorithm (Lin et al. 2020, Luo et al. 2022), Simulated Annealing (Bouleimen & Lecocq 2003, Cho & Kim 1997, Józefowska et al. 2001), and Neural Network-based approaches (Agarwal et al. 2011, Phuntsho & Gonsalves 2023), are commonly used to find near-optimal solutions for larger instances. Among these approaches, hybrid metaheuristics, which combine the strengths of different methods, offer a particularly robust approach for solving the RCPSP (Pellerin et al. 2020).

To address the complexities of real-world problems, several extensions of RCPSP have been proposed. These extensions include Multi-Mode RCPSP, Preemptive Activity Splitting, and Time-Varying Resource Availability (Hartmann & Briskorn 2022). They help model various practical constraints and scenarios encountered in project scheduling. One notable variation is the RCPSP with preemptive or non-preemptive activity splitting. This variation shares similarities with our problem, RASP, where activities can be divided into non-continuous segments (Cheng et al. 2015,

Vanhoucke & Coelho 2019).

While RCPSP, especially RCPSP with activity preemption, and RASP share similarities in minimizing makespan under resource constraints, they have distinct differences. RCPSP focuses on scheduling activities by determining feasible start times through precedence constraints. In contrast, RASP determines the sequence in which activities are performed, prioritizing resource allocation without incorporating any precedence constraints. Additionally, RCPSP uses activity splitting to minimize the total makespan, whereas RASP treats splitting as a strategy to optimize resource allocation while maintaining an optimal makespan.

## 2.2 Job Shop Scheduling Problem

The JSSP is a classic NP-hard combinatorial optimization problem introduced in 1954 in the field of machine scheduling, and it has been extensively studied since then (Johnson 1954, Lenstra et al. 1977). JSSP involves the allocation of a set of jobs that must pass through various machines in specific sequences, with each operation requiring a specific machine for a defined processing time. The main objective of JSSP is typically to minimize the makespan, which is the total time needed to complete all jobs.

The complexity of JSSP arises from the need to manage machine constraints and precedence relations between operations within each job. This inherent complexity has led to the study of various JSSP variations, including flexible job shop scheduling and dynamic job shop scheduling, both of which have received significant research attention (Xiong et al. 2022). Solutions to JSSP include both exact methods, such as branch-and-bound, and metaheuristics such as genetic algorithm, simulated annealing, and ant colony optimization (Çaliş & Bulkan 2013).

Among the various techniques, hybrid approaches that combine genetic algorithm with other methods have been particularly effective in developing efficient solutions for JSSP and its variations (Amjad et al. 2018, Asadzadeh 2015, Chen et al. 2020, Zhang et al. 2020). Specific algorithms such as the KK (Kundakcı & Kulak) heuristic (Kundakcı & Kulak 2016) and tabu search (Li & Gao 2016, Xie et al. 2023) have also been successfully applied to solve JSSP. Additionally, Iterated Greedy algorithm have proven effective in addressing JSSP variations, providing valuable insights

that could be applied to other scheduling problems (Al Aqel et al. 2019, Zhao et al. 2022).

Both JSSP and RASP involve the allocation of tasks under resource constraints. The primary difference lies in the nature of these constraints and tasks: JSSP typically deals with jobs that need to be processed on machines in a predefined sequence, while RASP involves activities that must be scheduled under resource constraints without precedence relationships.

## 2.3 Sequential Ordering Problem

The SOP is an extension of the Asymmetric Traveling Salesman Problem, which itself is a generalization of the classic Traveling Salesman Problem. It was first introduced in 1988 and has since been extensively studied in areas such as manufacturing, transportation, and logistics (Escudero 1988). The objective is to find the shortest possible path in a weighted directed graph while respecting specific constraints, where each node must be visited once and the tour must respect given precedence relations.

Exact algorithms such as the branch-and-bound method have been developed for the SOP. Innovative lower-bound techniques, based on the dynamic Hungarian algorithm and local search domination techniques, demonstrate the potential for precise solutions by efficiently pruning the search space and identifying optimal solutions (Jamal et al. 2017). Given the NP-hard nature of this problem, approximate algorithms are frequently employed to find efficient solutions in practice (Gambardella & Dorigo 2000). Among these methods, hybrid heuristic approaches are commonly used due to their ability to combine global and local search techniques, effectively balancing exploration and exploitation of the solution space. For instance, the Ant Colony System hybridized with other local search techniques (Skinderowicz 2017) and the hybrid Particle Swarm Optimization approach (Anghinolfi et al. 2011) have been developed to provide effective solutions for the SOP.

Comparing the SOP to the RASP reveals both similarities and differences. Similar to the RASP, the SOP focuses on sequencing tasks under constraints. However, the SOP primarily addresses the order of tasks without considering the possibility of splitting activities or managing resource constraints. In contrast, the RASP involves scheduling activities with a more complex consideration

of resource availability and allocation, ensuring that resource levels never fall below zero and stay as close to zero as possible.

## 2.4 Research Gap

We have reviewed three relevant problems: the RCPSP, the JSSP, and the SOP. Each of these problems has distinct characteristics that differentiate them from the RASP. The RCPSP involves precedence constraints and uses activity preemption as a technique for better makespan, the JSSP deals with machine constraints and predefined job sequences, and the SOP emphasizes order constraints without resource considerations or activity splitting. The RASP, on the other hand, focuses on sequencing activities to manage resource effectively without precedence constraints, allowing for activity splitting to optimize resource allocation and ensuring resource levels remain positive and as close to zero as possible for optimal resource use.

The literature review provides insights from the RCPSP, the JSSP, and the SOP to guide the development of efficient algorithms for RASP. Specifically, the use of greedy algorithm and their enhancement through local search has inspired our approach. These techniques offer a promising direction for addressing the unique challenges posed by the RASP and developing solutions for this combinatorial problem.

# Section 3: Problem Statement

We now present a formal description of the RASP in Section 3.1, and its mathematical model in Section 3.2.

## 3.1 Problem Definition

| Symbol | Description |
|---|---|
| **Sets** | |
| $I$ | Set of all activities |
| $I_+$ | Set of positive activities $i \in I$ where $a_i > 0$ |
| $I_-$ | Set of negative activities $i \in I$ where $a_i < 0$ |
| $I_s$ | Set of activities for split type $s \in \{0, 1, 2, 3\}$ |
| $T$ | Set of time periods |
| **Parameters** | |
| $\alpha_1, \alpha_2, \alpha_3$ | Objective function weights |
| $a_i$ | Resource amount of activity $i$ ($a_i > 0$ for Positive activity, $a_i < 0$ for Negative activity) |
| $c_i$ | Operational cost of activity $i$ |
| $l_i$ | Length of activity $i$ |
| $P$ | Maximum number of splits for $s_i = 0$ |
| $q_{initial}$ | Initial amount of available resource |
| $s_i$ | Split type of activity $i$ |
| **Variables** | |
| $q_t$ | Amount of available resource at the end of time period $t$ |
| $x_{it}$ | Proportion of activity $i$ completed at time period $t$ |
| $y_{it}$ | Binary variable indicating whether any portion of activity $i$ takes place at time period $t$ |
| $z_t$ | Binary variable indicating whether all activities are completed by time period $t$ |

Table 1 – Nomenclature

The nomenclature is presented in Table 1. Let $I$ be the sets of activities, $I_+$ and $I_-$ be the set of activities with positive and negative resource amounts, respectively. Each activity $i$ is associated with the total execution amount $a_i$, where positive activities have $a_i > 0$, and negative ones have $a_i < 0$. Each activity $i \in I$ also has a fixed operational cost $c_i$ for every execution, it is partial or full. The length of this activity $l_i$ is a fixed period of time required after the execution for it

to be fully completed. Additionally, $q_{initial}$ represents the initial amount of the common resource available at the start of the planning horizon. The amount of available resource will vary over time with the execution of activities, reflecting available resource at the end of each time period. Finally, each activity is assigned a single split type attribute $s_i$, which indicates how the activity can be split. The split types are defined as follows:

— **Split Type 0** ($s_i = 0$): The activity can be split up to a predefined limit of $P$ times at any point over the time horizon. This means the activity can be divided into up to a maximum number of $P$ segments, allowing flexibility in resource allocation while limiting excessive fragmentation.

— **Split Type 1** ($s_i = 1$): The activity can be split at any point without limitations (i.e., $P = \infty$), providing maximum flexibility of execution.

— **Split Type 2** ($s_i = 2$): The activity can be split at any point, but the next portion must wait for a fixed time period before execution. This fixed time period is equal to the length of the activity $l_i$. This introduces a delay between consecutive segments, representing cases when setup times are required between activity executions.

— **Split Type 3** ($s_i = 3$): The activity cannot be split, representing activities that must be executed fully without any splitting.

The RASP sequences the execution of all activities in $I$, within a time horizon $T$, with the primary objective of completing all activities at the earliest possible time, referred to as the makespan. Additionally, we minimize the total operational costs and the cumulative sum of resource remaining after each time period, hierarchically in the same order. We transform the three hierarchical objectives, $f_1$, $f_2$, and $f_3$, into a single objective function to use in the mathematical model by multiplying by sufficiently large values ($\alpha_1 \gg \alpha_2 > \alpha_3$). The significant difference in these weights ensures that the minimization of makespan is prioritized. Once the set of solutions with the shortest makespan is found, we then optimize the second and third objectives within this subset of solutions. The single objective is therefore $\alpha_1 f_1 + \alpha_2 f_2 + \alpha_3 f_3$.

A significant challenge in RASP is balancing the trade-off between minimizing operational costs and optimizing resource utilization across time periods. Initially, available resource is equal to

the initial amount, but in subsequent periods, they are adjusted based on executed positive and negative activities, and the incurred operational costs. Since operational costs directly affect the calculation of available resource, optimizing resource utilization often leads to higher operational costs, creating a trade-off. Observe that, prioritizing operational cost reduction leads to solutions that avoid activity splitting, whereas emphasizing resource utilization favors solutions that split activities.

## 3.2 Model Formulation

With the above problem description, we formulate the RASP as a Mixed-Integer Linear Programming (MILP) model.

$$\text{Minimize} \quad \alpha_1 \sum_{t \in T} t\ z_t + \alpha_2 \sum_{i \in I} \sum_{t \in T} c_i y_{it} + \alpha_3 \sum_{t \in T} q_t \tag{1}$$

$$\text{Subject to} \quad \sum_{t \in T} x_{it} = 1 \qquad i \in I \tag{2}$$

$$\sum_{i \in I} \sum_{w=0}^{t-l_i} x_{iw} \geq |I| z_t \qquad t \in T \tag{3}$$

$$\sum_{t \in T} z_t = 1 \tag{4}$$

$$x_{it} \leq y_{it} \qquad i \in I, t \in T \tag{5}$$

$$q_0 = q_{initial} \tag{6}$$

$$q_t = q_{t-1} + \sum_{i \in I_+ : t - l_i \geq 0} a_i x_{i,t-l_i} + \sum_{i \in I_-} a_i x_{it} - \sum_{i \in I} c_i y_{it} \qquad t \in T \setminus \{0\} \tag{7}$$

$$\sum_{t \in T} y_{it} \leq P \qquad i \in I_0 \tag{8}$$

$$y_{ik} + y_{it} \leq 1 \qquad i \in I_2, t \in T, k = t+1, \ldots, min\{t + l_i - 1, |T|\} \tag{9}$$

$$\sum_{t \in T} y_{it} = 1 \qquad i \in I_3, t \in T \tag{10}$$

$$0 \leq x_{it} \leq 1 \qquad i \in I, t \in T \tag{11}$$

$$z_t \in \{0, 1\} \qquad t \in T \tag{12}$$

$$y_{it} \in \{0, 1\} \qquad i \in I, t \in T \tag{13}$$

$$q_t \geq 0 \qquad t \in T. \tag{14}$$

The objective function (1) is composed of three terms, each representing one of our objectives and each term is associated with a different alpha value to reflect their hierarchical importance. The first term minimizes the completion time, the second term selects the solution with the least operational costs between alternative optimal solutions, and the third term minimizes the total unused resource at the end of the time periods. The non-negativity of variable $q_t$ ensures the resource levels are always non-negative.

Constraints (2) ensure that all activities are completed by the end of the planning horizon. Con-

straints (3) and (4) calculate the completion time while including the length of activity and guarantee that activities will be finished before the designated period. Constraints (5) ensure that $y_{it} = 1$ when $x_{it} > 0$, indicating any activity execution comes with a fixed operational cost. Constraints (6) and (7) are the resource balance equations considering the length of the activity. In the first period, the initial resource level is given. For subsequent periods, the available resource are calculated based on the remaining resource level from the previous period, the resource generated by positive activities from prior periods, the resource consumed by negative activities during the current period, and fixed operational costs (or execution losses, as illustrated in the warehouse management example) associated with activity execution during that period. Constraints (8) guarantee activities in type 0 splitting scenarios can be split a maximum of $P$ times across the time period. Constraints (9) ensure the next portion of activities in type 2 splitting scenarios must wait for a duration of $l_i$ time periods before execution. Constraints (10) ensure activities in type 3 splitting scenarios will not be split. Constraints (11), (12) and (13) are domain restrictions, and constraints (14) ensure that the resource is always non-negative.

The implementation details of the mathematical model are provided in Section 5. We also created specialized datasets, discussed in detail in Section 5, to test our model at various scales. Real-life problems often involve a large number of activities and need to be solved within a few minutes. However, using CPLEX for solving the MILP model, we were only able to solve small instance sizes. Therefore, we explore metaheuristic algorithms, described in the next section, to obtain near-optimal solutions.

# Section 4: Solution Methods

In large-scale problem instances, the mathematical model often becomes impractical due to the combinatorial explosion of feasible solutions. Metaheuristic methods, however, are particularly efficient for solving problems like RASP in such scenarios, as they provide feasible solutions within a reasonable time frame.

In this section, we describe two metaheuristics for RASP. Section 4.1 presents the first metaheuristic, a greedy search algorithm. Section 4.2 introduces the second metaheuristic, a search algorithm that enhances the greedy algorithm using a tabu search strategy. All computation results will be presented in Section 5.

## 4.1 Greedy Algorithm

The greedy algorithm's ability to produce a good initial solution quickly makes it a valuable starting point for more sophisticated optimization techniques. The greedy algorithm is widely utilized due to its simplicity and efficiency in handling resource-constrained scheduling problems. Several studies in JSSP and RCPSP highlight the role of greedy algorithm in project scheduling, emphasizing their effectiveness in generating initial solutions that can be further refined by local search or metaheuristic methods (Al Aqel et al. 2019, Hartmann & Briskorn 2022, Zhao et al. 2022).

In the greedy algorithm for solving RASP, activities are sequentially matched with time periods. The algorithm is designed to provide a good initial solution, with the primary focus being on optimizing the makespan by prioritizing activities based on a scoring system that reflects their strategic importance and priority. The detailed flowchart of the greedy algorithm and its dependent components are presented below. Table 2 represents a list of terminology and symbols used in the flowchart, Figure 1 shows the main structure of the algorithm, Figure 2 shows the detailed structure for positive and negative activities, and Figure 3 shows the details of handling activities of different split types. These figures are interconnected by the node with dashed line, it indicates that the following figure provides more detailed information about the processes described in these nodes.

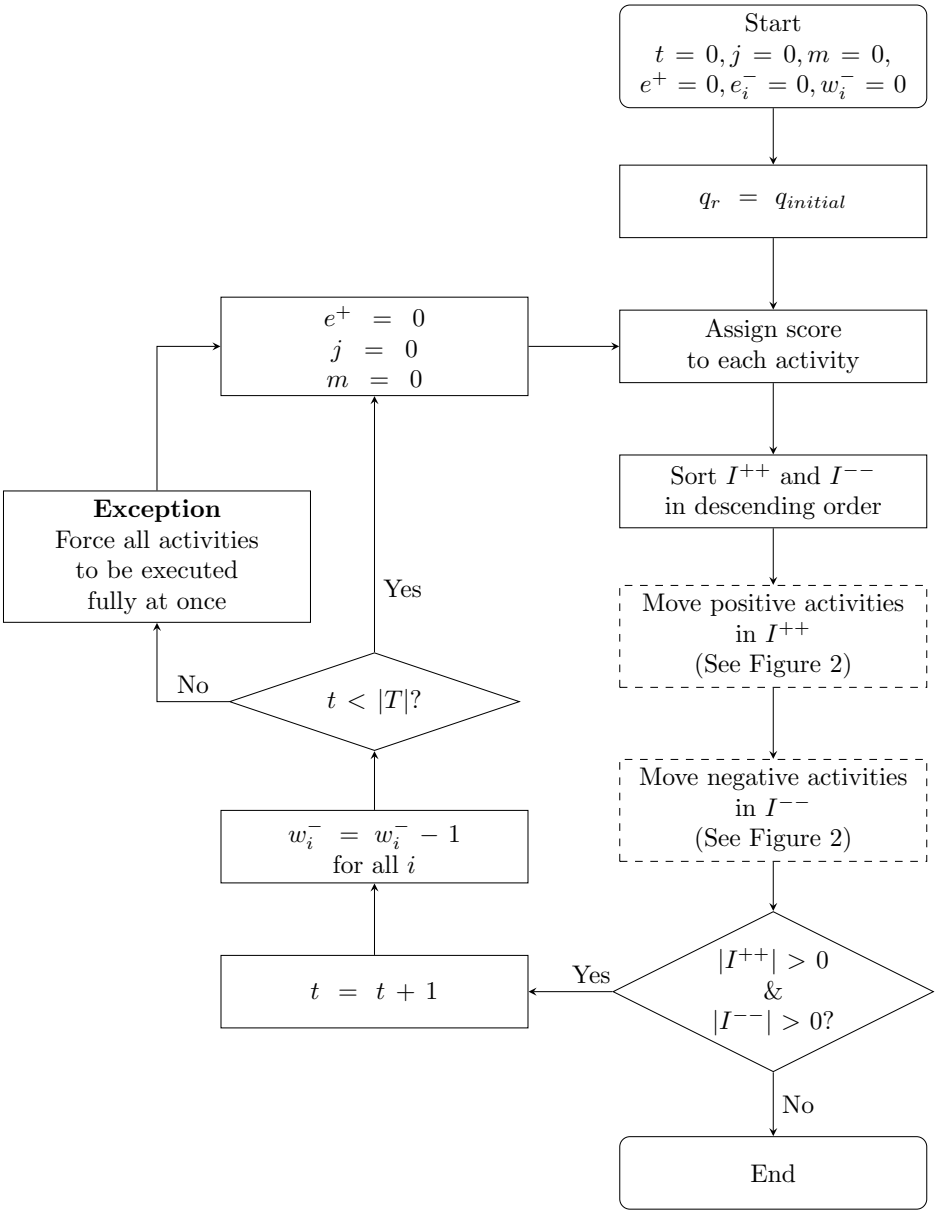
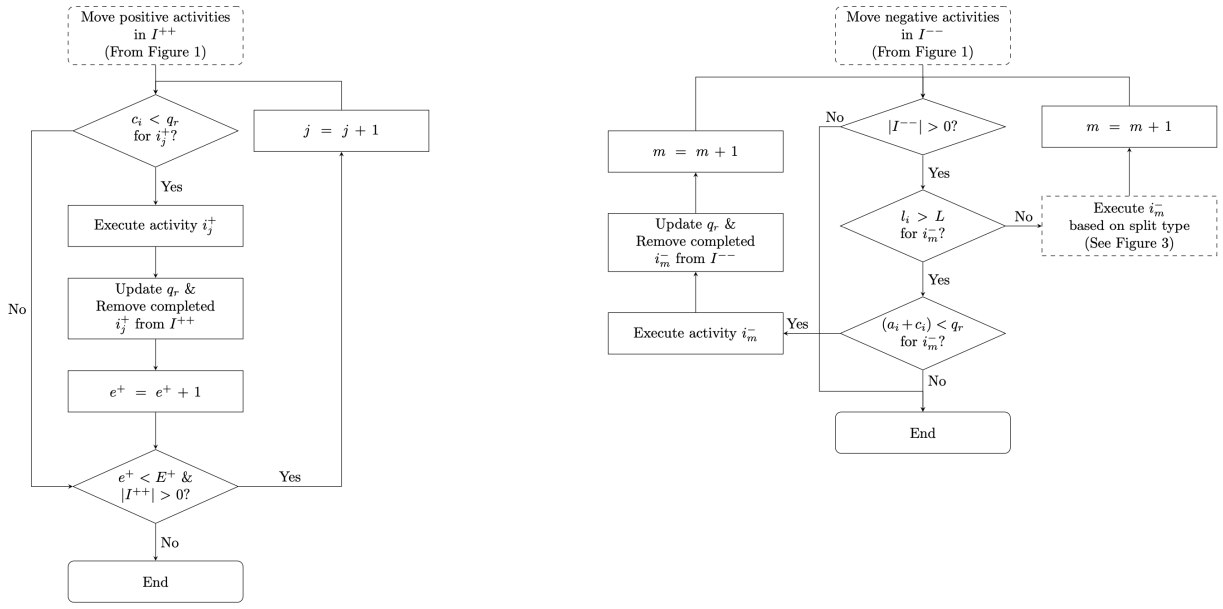



Figure 1 – Main Structure of Greedy Algorithm

| Symbol | Description |
|---|---|
| **Activity** | |
| $i$ | Single activity |
| $i_j^+$ | Selected positive activity $i$ at position $j$ |
| $i_m^-$ | Selected negative activity $i$ at position $m$ |
| **Integer** | |
| $e^+$ | Execution counter for positive activities |
| $e_i^-$ | Execution counter for negative activity $i$ |
| $j$ | Position index for positive activity |
| $m$ | Position index for negative activity |
| $t$ | Time period |
| $w_i^-$ | Wait time for negative activity $i$ |
| **Set** | |
| $I^{++}$ | Set of remaining positive activities |
| $I^{--}$ | Set of remaining negative activities |
| $T$ | Set of time periods |
| **Parameter** | |
| $a_i$ | Resource amount of activity $i$ |
| $c_i$ | Operation cost of activity $i$ |
| $E^+$ | Maximum count for positive activity execution per time period |
| $E^-$ | Maximum number of divisions for negative activities with $s = 0$ |
| $L$ | Predefined length limit |
| $l_i$ | Length of activity $i$ |
| $q_{initial}$ | Initial amount of available resource |
| $q_r$ | Remaining amount of available resource |

Table 2 – Nomenclature for Greedy Algorithm

Figure 1 illustrates the main structure of the greedy algorithm. The algorithm begins by initializing parameters time period $t$, indices for the position of positive and negative activities $j$ and $m$, the execution counter for positive activities $e^+$, specific execution counter for each negative activity $e_i^-$, the waiting measure for each negative activity $w-_i$, and the initial amount of available resource $q_r$. A score is then assigned to each activity based on its duration and impact on resource to determine their priority of execution. Positive activities are sorted into set $I^{++}$, and negative activities are sorted into set $I^{--}$, both arranged in descending order of scores. The algorithm attempts to move positive and negative activities iteratively, repeating while $t < T$ and if any set of $I^{++}$ and $I^{--}$ has any activities left to execute. At each iteration, parameters such as time period $t$ and wait




**(a) Move positive activities in $I^{++}$**

**(b) Move negative activities in $I^{--}$**

Figure 2 – Execution of Positive and Negative Activities in the Greedy Algorithm

time of each negative activity $w_i^-$ will be updated, and parameters such as execution count for positive activities $e^+$, indices for the position of positive and negative activities $j$ and $m$ will be re-initialized. Within the iteration, a mechanism for handling exceptions has been employed. The exception occurs when activities remain in a set, whether $I^{++}$ or $I^{--}$, and the maximum period has been reached $t = |T|$. This may be due to excessive splitting of the current solution, leaving insufficient resource to execute all activities. To solve this error, the algorithm has a backup plan that mandates the full execution of all activities, ensuring that the algorithm can deliver a complete and eligible solution.

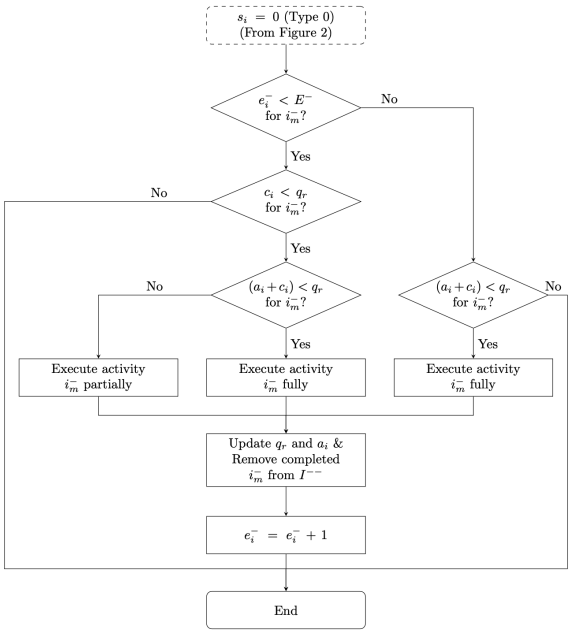
Figure 2 shows different means of moving positive and negative activities. For each positive activity $i_j^+$ selected, the algorithm checks if there is enough resource $q_r$ to consume the operational cost $c_i$. If so, the activity is executed, the resource level $q_r$ is updated, and the executed activity is removed from $I^{++}$. Otherwise, the algorithm moves on to the next positive activity. In the process, a predefined threshold $E^+$ and the execution count $e^+$ are used to limit the maximum number of positive activities per period. The execution count $e^+$ is updated after each execution, aiming to prevent all positive activities from being executed in the first period and ensuring a relatively bal-

anced distribution of positive activity executions, given their impact on resource occurs in future periods. For each negative activity $i_m^-$ selected, the algorithm first checks if there are remaining negative activities in $I^{--}$, and quickly follows by checking if the length $l_i$ of this activity is longer than the predefined length limit $L$. If the length exceeds the limit, the activity is critical and likely to impact the project's makespan significantly. To avoid such activities extending the total makespan, they are forced to be executed fully as soon as possible. If the common resource $q_r$ are sufficient to consume the operational cost $c_i$ and the full activity resource amount $a_i$, the negative activity is executed, the resource level $q_r$ is updated, and this activity is removed from $I^{--}$. If not, the algorithm moves to the next time period without considering other negative activities, this process will repeat until this one activity is executed. If the length of the activity $l_i$ does not exceed the predefined limit $L$, the execution process will be based on the activity's split type, detailed in Figure 3.

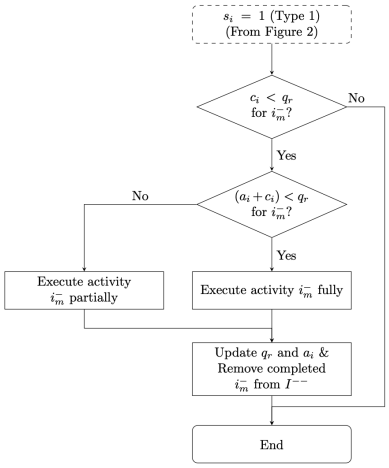
Figure 3 provides the detailed structure of how the algorithm handles each split type. In this paper, we consider four different split types to model real-world cases. In the following, we discuss how activities will be handled when resource is insufficient.

— **Split Type 0**: Split type 0 activities can be split up to a predefined maximum number of times (In the computational study of this work, we take it as three). The threshold $E^+$ and execution count for each negative activity $e_i^-$ are used to ensure the number of splitting events. For the first two executions, selected activity $i_m^-$ can proceed as long as the operational cost $c_i$ is less than the remaining resource $q_r$. If the remaining resource is sufficient to execute the full activity resource amount $a_i$, it will be fully executed; otherwise, it will be partially executed. When this activity reaches its third execution, it must wait until sufficient resource are available to execute the remaining resource amount; otherwise, it is skipped in favor of other activities. After each execution, remaining resource $q_r$, activity resource amount $a_i$, and execution count $e_i^-$ are updated. Once the activity is fully executed, it is removed from the set $I^{--}$.

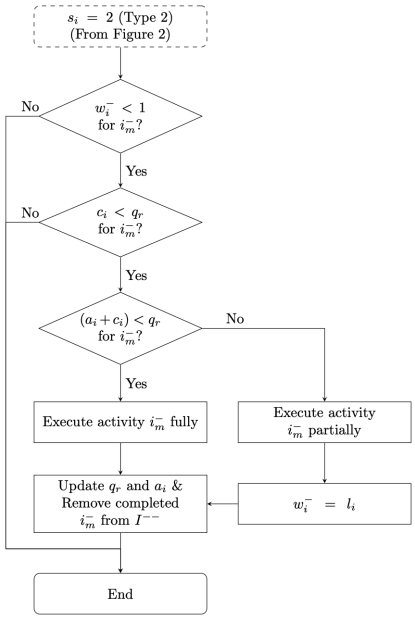
— **Split Type 1**: Split type 1 activities have no splitting restrictions. These activities can be executed as long as remaining resource $q_r$ is sufficient to consume the operational cost $c_i$. If the remaining resource is sufficient to fully execute the resource amount $a_i$, it will be fully



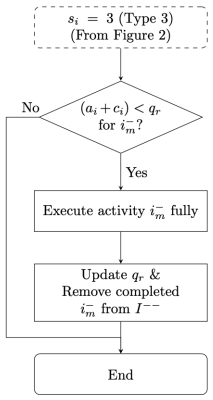

**(a) Execute split type 0 activities**




**(b) Execute split type 1 activities**




**(c) Execute split type 2 activities**




**(d) Execute split type 3 activities**

Figure 3 – Execution of Activities by Split Type in the Greedy Algorithm

executed; otherwise, it will be partially executed. After each execution, remaining resource $q_r$ and activity resource amount $a_i$ are updated. Once the activity is fully executed, it is removed from the set $I^{--}$.

— **Split Type 2**: Split type 2 activities require a waiting period before the next segment is executed. Similar to Split type 1, these activities can be fully or partially executed based on the remaining resource $q_r$, the operational cost $c_i$ and resource amount $a_i$. The key difference is the introduction of a waiting measure for each activity $w_i^-$. When an activity is partially executed, this value becomes the length of the activity $l_i$. At each iteration, whether this activity is selected to test its feasibility of execution or not, the wait time will decrease by 1, and the next segment of the activity cannot be executed until this waiting measure is less than 1.

— **Split Type 3**: Split type 3 activities cannot be split. They can only be fully executed when the remaining resource is sufficient for both the operational cost $c_i$ and resource amount $a_i$. Once executed, remaining resource $q_r$ is updated, and the activity is removed from the set $I^{--}$.

By using this greedy algorithm as a foundation, the subsequent local search algorithm can leverage the initial solution to explore the solution space more effectively, thereby improving the overall optimization.

## 4.2 Tabu Search Algorithm

Building upon the initial solution provided by the greedy algorithm, a tabu search algorithm is developed to further enhance solution quality by optimizing the second objective concerning the sum of operational costs and the third objective about unused common resource for each time period. The tabu search algorithm iteratively explores the neighborhood of the current solution, adjusting activity schedules within the solution space and validating the feasibility of each move. Each move is evaluated, and the best solution is selected after all iterations. To avoid repetition of recent moves, a tabu list is used to prevent revisiting recently moved activities or paths. The process continues until the stopping criterion, such as the maximum number of iterations, is met.

During the construction of the tabu search, several key strategy choices that could significantly impact optimization results were proposed. The details are presented in Table 3.

| | Option 1 | Option 2 |
|---|---|---|
| **Selection Criteria** | **A**: All activities of time period & Move one activity at a time | **B**: Batch of activities of time period & Move entire batch at once |
| **Activity Candidates** | **A1**: Mix of positive and negative activities | **A2**: Only positive or only negative activities |
| **Activity Treatment** | **B1**: Move all activities to one new time period $t'$ | **B2**: Move positive activities to time period ($t'$ - activity length) |
| **Solution Acceptance** | **C1**: Accept if the move is feasible and the new solution $s'$ is better than the current solution $s$ | **C2**: Accept if the move is feasible |

Table 3 – Categories of Tabu Search Strategies

The selection criteria focus on how to choose and move activities. We can either select all activities of one time period and iteratively move one activity at a time to a neighboring period for incremental improvement, or select a batch of activities from a time period and move the whole group simultaneously for collective improvement. The activity candidates determine what type of activities are selected, further refining the selection process. We can either consider a mix of positive and negative activities or focus on moving one type of activity. The activity treatment involves deciding where to move the selected activities. Since positive activities and negative activities impact resource at different time periods, we must decide whether to move all activities to the selected neighboring time period or move positive activities before the selected neighboring time period based on their length $l_i$, ensuring their settlement date coincides with the time period when negative activities will consume resource. Finally, solution acceptance specifies how to accept the new solution. We can either accept feasible and better solutions during iterations or accept all feasible solutions to overcome local optima.

Each category offers two distinct options, resulting in 16 possible combinations to fine-tune the search process. For easier representation, each option is coded as: A (for all)/B (for batch) - A1/A2 - B1/B2 - C1/C2. For example, a scenario that selects a batch of activities and moves the

entire batch to a neighboring time period, considers a mix of positive and negative activities as candidates to move, moves all of them to the selected neighboring time period, and accepts the move if it is feasible and better than the current solution would be coded as B-A1-B1-C1. This approach aims to identify the optimal combination for our tabu search and highlight the criteria with the most significant impact on optimization results.

| Symbol | Description |
|---|---|
| **Activity** | |
| $i_b$ | Selected activity $i$ at position $b$ |
| **Integer** | |
| $b$ | Position index for activity |
| $n$ | Current iteration |
| $t$ | Time period |
| $t'$ | Neighborhood time period |
| **Set** | |
| $I^*$ | Set of selected activities |
| **State** | |
| $s$ | Current solution schedule |
| $s'$ | New solution after relocation |
| **Parameter** | |
| $N$ | Stopping criteria for iteration |

Table 4 – Nomenclature for Tabu Search Algorithm

Table 4 represents the terminology and symbols used in the flowchart of the tabu search algorithm, and the structure of the tabu search algorithm is illustrated in Figure 4.

The algorithm is primarily divided into two distinct branches based on the first strategy category: selection criteria. With different ways to select activities and move them, the construction of the tabu list is different.

For the common part, the algorithm always begins by taking the initial solution $s$ and randomly selecting a time period $t$ with activities to start. Activities from this period are considered candidates for potential relocation to explore better arrangements. After deciding how to select candidate activities and move them (A or B), and the type of activities considered (A1 or A2), both branches

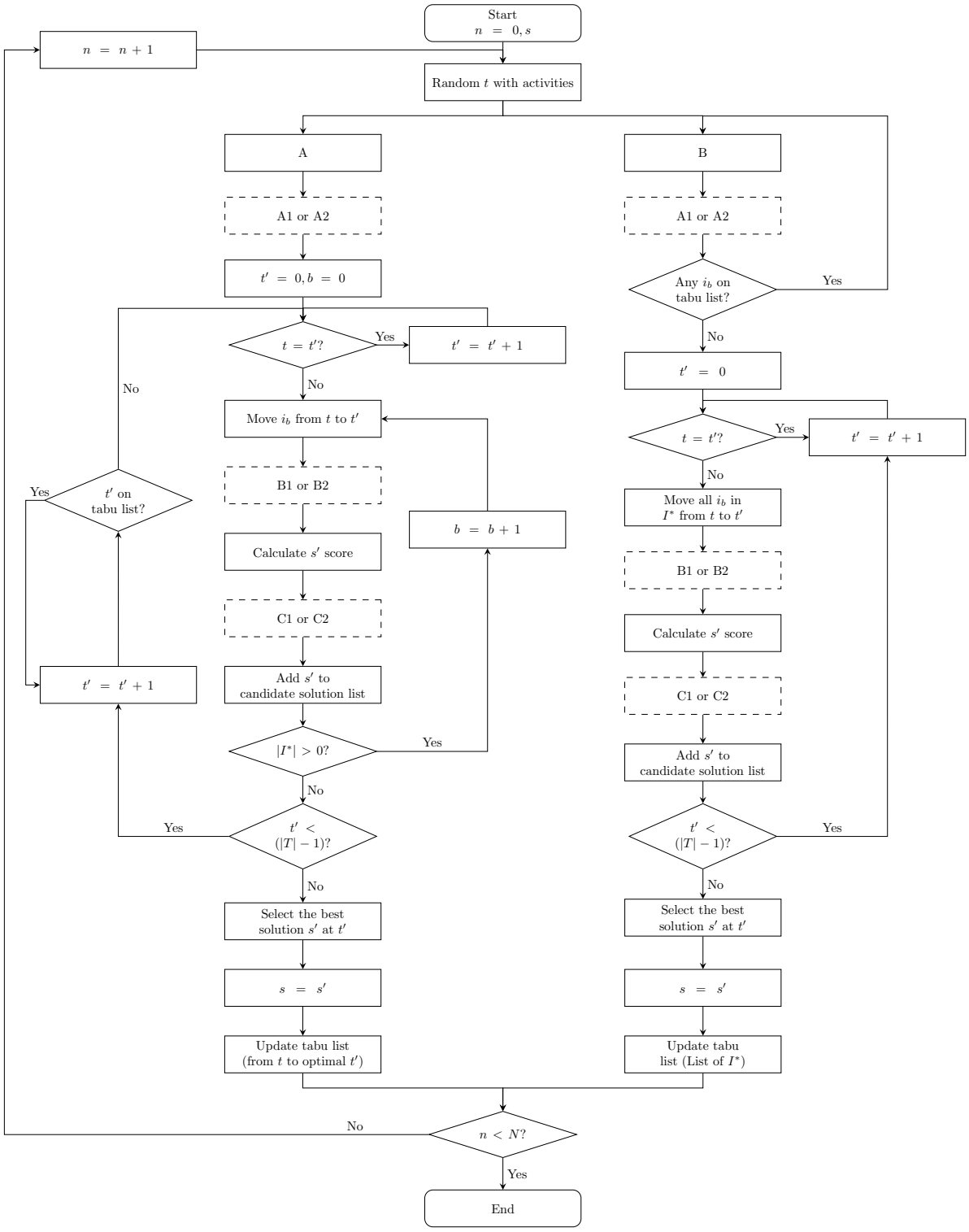



Figure 4 – Tabu Search Algorithm

will select a neighboring time period $t'$ and move candidate activities to that time period. Before the moving process, the algorithm verifies whether the neighboring time period $t'$ is the same as the original time period $t$. If so, a new neighboring time period is selected. When moving activities, we then need to decide how to treat different types of activities (B1 or B2). After moving activities, both branches go through the evaluation process by calculating the score of the new solution, comparing it with the current solution, and determining whether to accept the potential solution $s'$ (C1 or C2). The new solution is added to the candidate solution list. After testing one neighboring time period, the algorithm continues to move activities to all other neighboring time periods, providing a list of improved solutions. The best solution out of all candidate solutions is selected as the new solution for the next iteration. The tabu list is updated accordingly, and this process repeats until the stopping criteria are met.

The main difference between the two branches focuses on the tabu list integration. Both branches uses a "First-in First-out" strategy for managing the tabu tenure and criteria for lifting restrictions in order to balance exploration and exploitation. However, when we select all activities of one time period and move them, the tabu list bans the path ($fromDay$, $toDay$) from one time period to the other. If $fromDay = t$ and $toDay = t'$ is on the tabu list, this neighboring time period is skipped. When we select a batch of activities to move, the tabu list bans the batch of selected activities $I^*$, preventing those activities from being selected for following time periods until they are no longer on the list.

Overall, the tabu search-based algorithm represents a sophisticated way to further optimize solutions obtained from the greedy algorithm for RASP. Through strategic activity movements and an iterative refinement process, the algorithm demonstrates its capability in finding high-quality solutions. This approach not only optimizes the secondary objectives but also establishes a robust solution strategy for RASP, setting a benchmark for future research.

# Section 5: Computational Experiments

We tested the effectiveness of the mathematical model and the metaheuristic algorithms using various instance. All computations were performed on a personal computer with an Intel(R) Xeon(R) Silver 4216 CPU (32 cores) and 128 GB of memory to ensure consistent performance. The algorithms were coded in a C# development environment on Windows, specifically using Visual Studio 2022 and .NET 7.0. The CPLEX 22.1.0 optimization package was used for constructing and executing the mathematical model.

This section is organized into three subsections. Section 5.1 describes the datasets created for RASP. Section 5.2 details the comparison framework for different algorithms. Section 5.3 presents the computational results for both the mathematical model and the metaheuristic algorithms, along with their comparison.

## 5.1 Dataset Used

As RASP is a new problem, there are no existing benchmarks in the literature. We therefore create a series of datasets specifically designed to simulate varying complexities and scales of the problem. These datasets are used to assess the efficiency and effectiveness of the proposed algorithms.

The series of data is structured to represent a wide range of problem sizes, with activity counts of 10, 20, 50, and 100, labeled as $SET10$, $SET20$, $SET50$, and $SET100$, respectively. It is designed to simulate different problem scales. For each problem size, 40 distinct instances in JSON format were created, each corresponding to a scheduling scenario. Each instance is designated by a filename following the pattern RASP-[Size]-[Id]-[PositiveRatio]-[NegativeRatio]-[SplitRatio].json, indicating the instance size, a unique identifier, the proportion of positive and negative activities, and the percentage of activities that can be split.

Each JSON file comprises global attributes that quantify and qualify the characteristics of the instance. These include initial resource amounts, total value of positive and negative resource activities, theoretical end values, and minimal execution costs. An integral part of the instance is the "NbActivities" attribute, denoting the count of activities within the instance.

The detailed attributes of each activity in the file include:

- **Id**: A unique identifier for the activity.
- **Name**: A descriptive name for the activity.
- **Length**: The duration of the activity.
- **ActType**: An indicator of whether the activity decreases (coded as 1) or increases (coded as 2) the available resource.
- **ResourceAmount**: The quantity of the resource implicated in the activity.
- **IsSplittable**: A boolean variable indicating the divisibility of the activity (false if it cannot be split, and true if it can be split).
- **SplitInfo**: An integer (0, 1, 2, or 3) detailing the splitting behavior where 0 allows up to a maximum number of splits, 1 denotes unlimited splits at any point, 2 requires waiting periods between the execution of each segment, and 3 specifies that the activity cannot be split.
- **ExecutionCost**: The fixed cost associated with the activity when executing.

The dataset is designed to model varying conditions in a controlled experimental setup. The ratio of positive to negative activities is defined as 20/80, 40/60, 60/40, or 80/20, while the ratio of divisible activities is given as 0%, 25%, 50%, 75%, or 100%. Each series progresses through these ratios systematically, ensuring a comprehensive exploration of potential scenarios. For instance, the first instance of a series named RASP-Size-Idxac-20-80-0.json would have 20% positive activities and 80% negative activities, with 0% of activities being splittable. The next instance instance RASP-Size-Id-40-60-25.json would have 40% positive activities and 60% negative activities, with 25% of activities being splittable, and so on. To generate all possibilities, there must be a total of $4 \times 5$ instances, giving 20 combinations. The number of instances generated per series is set at 40, providing two examples of each possible combination. This comprehensive approach allows for robust testing and validation of the developed algorithms.

## 5.2 Comparison Framework

To evaluate the performance of our proposed algorithms for RASP, we established a systematic comparison framework. This framework assesses the algorithms based on three hierarchical objectives: minimizing the makespan, minimizing operational costs, and minimizing unused resource at the end of each time period.

First, we focus on the primary objective of minimizing the makespan, we compare the makespan obtained by our metaheuristic algorithms with the optimal makespan with small instances where the mathematical model can be solved. We count the number of instances where the metaheuristic achieves the same makespan as the mathematical model. For instances where the metaheuristic does not achieve the optimal makespan, we calculate the percentage gap between the metaheuristic and the optimal makespan to understand the efficiency of the metaheuristic algorithms in approximating the optimal solution.

Next, for the secondary objective of minimizing operational costs, we focus on instances where the metaheuristic achieves the same makespan as the optimal solution. Among these instances, we compare the operational costs. For instances with matching makespans but differing operational costs, we calculate the percentage gap in operational costs between the metaheuristic and the optimal solutions in order to evaluate the cost efficiency of proposed metaheuristic algorithms.

Finally, we assess the third objective of minimizing unused resource by considering instances that have matching makespan and operational cost with the optimal solution. For instances with matching makespan and operational costs but differing unused resource, we calculate the percentage gap in unused resource to assess the effectiveness of the algorithms in resource management.

For larger instances where the optimal solution is unavailable, we compare the performance of the algorithms to a baseline metaheuristic algorithm. In this evaluation, we primarily focus on the makespan objective, given its critical importance in RASP. We assess the number of instances where the proposed metaheuristic algorithms improve upon the baseline greedy algorithm and calculate the average improvement in makespan for these instances to evaluate the overall effectiveness of the metaheuristics.

This hierarchical comparison framework allows us to systematically evaluate the performance of our algorithms in solving RASP, ensuring a comprehensive understanding of their strengths and limitations across different problem sizes and objectives.

Section 5.3 presents a detailed comparison of the mathematical model and metaheuristic algorithms based on this comparison framework.

## 5.3 Results

To illustrate the computational complexity and effectiveness of various approaches for solving RASP, we conducted experiments on designed datasets with different problem sizes. A coefficient setting of 10000-500-1 was determined for each objective to run the mathematical model, and all metaheuristic algorithms were run 25 times on each instance, each time within a time limit of 15 seconds, to ensure a robust comparison. The results are divided into computations for both the mathematical model and the metaheuristic algorithms.

| **Set** | **Min (sec)** | **Max (sec)** | **Average (sec)** |
|---|---|---|---|
| SET10 | 0.1 | 90.6 | 3.4 |
| SET20 | 1.3 | 392743.5 | 20230.6 |

Table 5 – Execution Time of MILP Model with CPLEX

Table 5 shows the execution time of the MILP model with CPLEX. It quickly revealed the NP-hard nature of RASP with the significant computational resource required and the rapid increase in solution times with larger problem sizes. When the problem size increased from 10 to 20 activities, computation time increased dramatically from a few seconds to over 5.6 hours on average. This exponential growth in computation time, which could extend to days for even larger instances, highlighted the limitations of the mathematical model for practical application on a larger scale. Consequently, this confirmed the necessity of metaheuristic methods for efficient problem-solving.

Due to these computational challenges, the MILP model was tested only on instances with 10 and 20 activities and used for comparison at these sizes. For larger instances, the greedy algorithm served as the baseline for practical feasibility in comparisons.

Tables 6, 7, and 8 compare the mathematical model with the greedy algorithm and 16 variations

of the tabu search algorithm based on the greedy approach for small instances with 10 and 20 activities. The comparisons focus on the objectives of makespan, operational costs, and unused resource, respectively. In the first subtable, we count the number of instances that achieved the optimal value in each dataset. In the second subtable, we calculate the performance gap for instances with sub-optimal values for each dataset, using the total number of instances as the base, along with an overall average gap across different dataset sizes. Additionally, Table 9 presents the results for larger instances with 50 and 100 activities, using the greedy algorithm as the baseline, comparing it with the 16 variations of the greedy-based tabu search algorithm, with a focus on the makespan objective. For each method, we evaluate the number of instances the method has improved the makespan objective compared to the greedy algorithm, along with an average improvement based on those improved instances.

| **Set** | **Optimal Value Found** | **Sub-optimal Value Found** |
|---|---|---|
| SET10 | 38 / 40 (95%) | 2 / 40 (5%) |
| SET20 | 30 / 40 (75%) | 10 / 40 (25%) |
| **Average** | 34 (85%) | 6 (15%) |

| **Method** | **SET10 (%)** | **SET20 (%)** | **Average (%)** |
|---|---|---|---|
| Greedy | 0.90 | 1.56 | 1.23 |
| A - A1 - B1 - C1 | 0.71 | 1.51 | 1.11 |
| A - A1 - B1 - C2 | 0.53 | 1.40 | 0.96 |
| A - A2 - B1 - C1 | 0.77 | 1.52 | 1.14 |
| A - A2 - B1 - C2 | 0.55 | 1.53 | 1.04 |
| A - A1 - B2 - C1 | 0.71 | 1.55 | 1.13 |
| **A - A1 - B2 - C2** | **0.46** | **1.39** | **0.93** |
| A - A2 - B2 - C1 | 0.76 | 1.54 | 1.15 |
| A - A2 - B2 - C2 | 0.62 | 1.53 | 1.08 |
| B - A1 - B1 - C1 | 0.71 | 1.50 | 1.11 |
| B - A1 - B1 - C2 | 0.76 | 1.52 | 1.14 |
| B - A2 - B1 - C1 | 0.69 | 1.53 | 1.11 |
| B - A2 - B1 - C2 | 0.78 | 1.50 | 1.14 |
| B - A1 - B2 - C1 | 0.70 | 1.49 | 1.10 |
| B - A1 - B2 - C2 | 0.73 | 1.53 | 1.13 |
| B - A2 - B2 - C1 | 0.74 | 1.51 | 1.12 |
| B - A2 - B2 - C2 | 0.83 | 1.55 | 1.19 |

Table 6 – Gap Analysis on Objective 1 for Small Instances (Cases Differing on Objective 1)

Table 6 presents the results focusing on the first objective, minimizing the makespan. Among

the 80 evaluated instances with 10 or 20 activities, the greedy algorithm achieved the optimal makespan in 85% of cases on average, demonstrating the general efficiency of our metaheuristic in achieving the optimal makespan. For the remaining cases that don't have the same makespan, the greedy algorithm maintained an average gap of 1.23% from the optimal value. The tabu search algorithm based on this greedy approach further reduced this gap. Among all variations, the combination A-A1-B2-C2 is particularly good, reducing the gap to as low as 0.93% on average, and it has the lowest gap for both instances with 10 activities (0.46%) and instances with 20 activities (1.39%).

| **Set** | **Optimal Value Found** | **Sub-optimal Value Found** |
|---|---|---|
| SET10 | 7 / 38 (18.42%) | 31 / 38 (81.58%) |
| SET20 | 4 / 30 (13.33%) | 26 / 30 (86.67%) |
| **Average** | 5.5 (15.875%) | 28.5 (84.125%) |

| **Method** | **SET10 (%)** | **SET20 (%)** | **Average (%)** |
|---|---|---|---|
| Greedy | 12.26 | 10.87 | 11.57 |
| A - A1 - B1 - C1 | 6.13 | 3.80 | 4.97 |
| A - A1 - B1 - C2 | 9.25 | 6.44 | 7.84 |
| A - A2 - B1 - C1 | 5.84 | 3.82 | 4.83 |
| A - A2 - B1 - C2 | 9.03 | 6.74 | 7.89 |
| A - A1 - B2 - C1 | 5.85 | 3.85 | 4.85 |
| A - A1 - B2 - C2 | 7.90 | 5.00 | 6.45 |
| **A - A2 - B2 - C1** | **5.79** | **3.80** | **4.80** |
| A - A2 - B2 - C2 | 7.90 | 5.45 | 6.67 |
| B - A1 - B1 - C1 | 5.88 | 4.26 | 5.07 |
| B - A1 - B1 - C2 | 9.29 | 7.21 | 8.25 |
| B - A2 - B1 - C1 | 5.99 | 4.25 | 5.12 |
| B - A2 - B1 - C2 | 9.18 | 6.59 | 7.89 |
| B - A1 - B2 - C1 | 5.87 | 4.11 | 5.00 |
| B - A1 - B2 - C2 | 8.67 | 6.12 | 7.39 |
| B - A2 - B2 - C1 | 5.97 | 4.28 | 5.12 |
| B - A2 - B2 - C2 | 8.75 | 6.21 | 7.48 |

Table 7 – Gap Analysis on Objective 2 for Small Instances (Cases Matching on Objective 1, Differing on Objective 2)

Table 7 shows the results for instances with the same makespan, focusing on the second objective, operational costs. On average, 15.875% of cases achieved the optimal value with the greedy algorithm. For cases with the same makespan but differing operational costs, the greedy algorithm maintained a small gap of 11.57%. The tabu search algorithm further significantly narrowed

this gap down, especially for A-A2-B2-C1, which reduced it to as low as 4.80%, underlining the ability of proposed metaheuristic to refine solutions for the second objective while maintaining performance for the first.

| Set | Optimal Value Found | Sub-optimal Value Found |
|---|---|---|
| SET10 | 1 / 7 (14.29%) | 6 / 7 (85.71%) |
| SET20 | 0 / 4 (0%) | 4 / 4 (100%) |
| **Average** | 0.5 (7.145%) | 5 (92.855%) |

| Method | SET10 (%) | SET20 (%) | Average (%) |
|---|---|---|---|
| Greedy | 352.86 | 5269.12 | 2810.99 |
| A - A1 - B1 - C1 | 95.60 | 3807.06 | 1951.33 |
| A - A1 - B1 - C2 | 199.36 | 2613.42 | 1406.39 |
| A - A2 - B1 - C1 | 38.85 | 3883.56 | 1961.21 |
| A - A2 - B1 - C2 | 141.02 | 2979.84 | 1560.43 |
| A - A1 - B2 - C1 | 50.49 | 3795.92 | 1923.21 |
| **A - A1 - B2 - C2** | **163.69** | **2276.09** | **1219.89** |
| A - A2 - B2 - C1 | 49.53 | 3897.58 | 1973.55 |
| A - A2 - B2 - C2 | 115.98 | 3077.44 | 1596.71 |
| B - A1 - B1 - C1 | 143.00 | 3929.90 | 2036.45 |
| B - A1 - B1 - C2 | 225.31 | 4819.78 | 2522.55 |
| B - A2 - B1 - C1 | 143.00 | 3946.55 | 2044.78 |
| B - A2 - B1 - C2 | 264.02 | 4369.91 | 2316.97 |
| B - A1 - B2 - C1 | 128.75 | 3856.24 | 1992.50 |
| B - A1 - B2 - C2 | 192.41 | 4478.21 | 2335.31 |
| B - A2 - B2 - C1 | 128.24 | 3843.08 | 1985.66 |
| B - A2 - B2 - C2 | 240.25 | 3801.55 | 2020.99 |

Table 8 – Gap Analysis on Objective 3 for Small Instances (Cases Matching on Objectives 1 & 2, Differing on Objective 3)

Table 8 concentrates on cases with the same first two objectives but differing on the third, which is about minimizing unused resource at the end of each time period. The greedy algorithm found an optimal solution in some instances with 10 activities. For cases with differing unused resource, the greedy algorithm showed a relative large gap for this objective, maintaining a performance gap of 2810.99% on average with the optimal value. The tabu search algorithm significantly reduced this gap, with A-A1-B2-C2 reducing it to as low as 1219.89%. However, this combination does not provide the best result in all instance sizes. For instances with 10 activities, combinations such as A-A2-B1-C1 minimized the gap to as low as 38.85%, whereas A-A1-B2-C2 gave 163.69%. This shows that our proposed algorithm can significantly reduce the gap in unused resource, even

though the tabu search algorithm focus less on this objective, demonstrating the efficiency of our algorithm.

The considerable difference in gaps for the third objective can be attributed to several factors. First, due to the comparison metrics we used, relatively fewer qualified instances were considered in the gap analysis for the third objective. Second, this highlights the trade-off between operational costs and resource management. Minimizing costs often results in an increase in unused resource at the end of each time period. Most importantly, one of the limitations of our algorithms is the lack of incorporation of the postponement effect of positive activities on the common resource. The greedy algorithm aims to complete all activities as quickly as possible, and it lacks clear restrictions on common resource for positive activities, unlike for negative activities. This often results in most positive activities being executed at the beginning. Although we set a maximum count for positive activity execution per time period $E+$ to address this, it is not optimal. For example, in the optimal solution obtained from the mathematical model, certain positive activities with short length are executed towards the end of the planning horizon to minimize unused resource over time. Furthermore, our tabu search algorithm cannot improve upon the results of the greedy algorithm, as it moves activities from one time period to a neighboring time period for improvement. During this process, any move is rejected if the unused resource in any time period drop below zero. Therefore, moving only positive activities without moving the corresponding negative activities may not be feasible, as negative activities require resource consumption during that period. To address this gap in future research, we need to incorporate the postponement effect of positive activities into the algorithms, either by generating a better initial solution or by developing a local search algorithm that moves activities across multiple time periods simultaneously.

Table 9 assesses the performance of the first objective, makespan, of our algorithms on larger instances of 50 and 100 activities. Compared with the greedy algorithm as the baseline, all variations of tabu search are able to further improve the makespan in several instances. Among them, A-A1-B2-C2 improves in the most instances, with 4 instances in activities of 50 and 6 instances in activities of 100, achieving an average improvement of 2.68% over the greedy algorithm. This confirms the effectiveness of our tabu search algorithm in handling larger and more complex scenarios.

| Method | SET50 | | SET100 | | Average | |
|---|---|---|---|---|---|---|
| | # Imp. Inst.* ( /40) | % | # Imp. Inst.* ( /40) | % | # Imp. Inst.* ( /40) | % |
| A - A1 - B1 - C1 | 2 | 0.44 | 0 | 0.00 | 1 | 0.22 |
| A - A1 - B1 - C2 | 4 | 3.93 | 5 | 2.32 | 4.5 | 3.12 |
| A - A2 - B1 - C1 | 1 | 1.22 | 0 | 0.00 | 0.5 | 0.61 |
| A - A2 - B1 - C2 | 5 | 1.71 | 2 | 2.13 | 3.5 | 1.92 |
| A - A1 - B2 - C1 | 1 | 1.22 | 0 | 0.00 | 0.5 | 0.61 |
| **A - A1 - B2 - C2** | **4** | **3.95** | **6** | **1.41** | **5** | **2.68** |
| A - A2 - B2 - C1 | 1 | 0.87 | 0 | 0.00 | 0.5 | 0.43 |
| A - A2 - B2 - C2 | 1 | 0.24 | 0 | 0.00 | 0.5 | 0.12 |
| B - A1 - B1 - C1 | 1 | 1.22 | 0 | 0.00 | 0.5 | 0.61 |
| B - A1 - B1 - C2 | 5 | 0.84 | 1 | 2.84 | 3 | 1.84 |
| B - A2 - B1 - C1 | 1 | 0.70 | 0 | 0.00 | 0.5 | 0.35 |
| B - A2 - B1 - C2 | 3 | 0.66 | 2 | 1.55 | 2.5 | 1.10 |
| B - A1 - B2 - C1 | 1 | 1.04 | 0 | 0.00 | 0.5 | 0.52 |
| B - A1 - B2 - C2 | 3 | 0.34 | 1 | 0.65 | 2 | 0.49 |
| B - A2 - B2 - C1 | 1 | 0.52 | 0 | 0.00 | 0.5 | 0.26 |
| B - A2 - B2 - C2 | 2 | 0.35 | 1 | 0.39 | 1.5 | 0.37 |

* Number of improved instances

Table 9 – Gap Analysis on Objective 1 for Large Instances (Cases Differing on Objective 1)

Overall, several strategy combinations have been highlighted in the comparison process, but no single strategy combination of the tabu search algorithm globally dominates all others. However, the most effective tabu search algorithm overall is A-A1-B2-C2. This strategy involves considering all positive and negative activities of a time period as candidates for movement and moving one activity at a time to a neighboring time period for optimization. Specifically, positive activities are moved before the period where resource is consumed by negative activities, ensuring synchronization of resource impacts. Additionally, we adopted the strategy of accepting all feasible solutions to enhance solution exploration.

This combination of strategies demonstrated superior performance in the comparison of the first and third objectives and showed the most improvement in the first objective on average for large instances. Although this combination slightly underperformed in the comparison of the second objective (7.73% vs. 5.74% for the best-performing strategy), the difference is minimal. Considering the trade-off of multi-objectives, this strategy combination is the most balanced out of all other variations, and the small underperformance on the second objective does not detract from its

overall high performance compared to other strategies.

Through the comparison process, we were also able to identify and analyze the effects of each strategy in the tabu search algorithm:

— **Selection Criteria** (Option A vs. B): Selecting all activities and moving one activity at a time to a neighboring time period for incremental improvement (Option A) proved superior to selecting a batch of activities and moving the entire batch together (Option B). All best and near-best values were found using the first option. Only certain combinations involving the batch selection approach reached similar levels of results.

— **Activity Candidates** (Option A1 vs. A2): The decision to move a mix of positive and negative activities (Option A1) or only one type of activity (Option A2) did not yield clear insights into its effect, indicating it is the element that influences the result the least.

— **Activity Treatment** (Option B1 vs. B2): Moving positive activities to periods before the periods when negative activities consume resource (Option B2) provided slight improvements in the gap with the optimal solution for all objectives compared to moving positive and negative activities to the same time period in the process (Option B1).

— **Solution Acceptance** (Option C1 vs. C2): Accepting feasible but not necessarily better solutions during iterations (Option C2) significantly impacted the results. This strategy helped further minimize the critical objective of makespan compared to the other option, showing its capacity to overcome local optima. However, accepting only feasible and better solutions (Option C1) greatly reduced the gap for the second objective concerning operational costs.

Overall, the strategies for selection criteria and solution acceptance had the most significant influence on the results, followed by activity treatment, and the selection of activity candidates had the least impact. These findings provide valuable direction for future improvements of the tabu search algorithm.

# Section 6: Conclusion

This paper has introduced the Resource-Constrained Activity Sequencing Problem (RASP), a combinatorial optimization problem focusing on sequencing activities within a planning horizon while managing a common resource. The problem has three hierarchical objectives (1) to minimize the makespan, (2) to minimize operational costs, and (3) to reduce unused resource at the end of each time period. We have presented a formulation of the RASP as a mixed-integer linear program, combining all three hierarchical objectives into the objective function as a weighted sum. However, the computational experiments on small instances have highlighted the need for metaheuristic approaches to handle larger problem sizes effectively.

To this end, we have developed a greedy algorithm and a tabu search algorithm. These methods have generated near-optimal solutions for the RASP in terms of the first two objectives and have significantly improved solution times for large instances. We have tested various setting of the tabu search algorithm. Our computational experiments have demonstrated that the best-performing setting of tabu search algorithm is the A-A1-B2-C2 strategy, performed well in solving the RASP. This strategy considers all positive and negative activities of a time period as candidates for movement, moving one activity at a time to a neighboring time period. Positive activities are moved before the selected neighboring time period to ensure synchronization of the resource impacts of different types of activities. Additionally, feasible but non improving solutions are accepted during the process to overcome the local optima. Through extensive computational experiments, we have demonstrated the scalability of our proposed algorithms. The tabu search algorithm have found the optimal makespan in more than 85% of the small instance, and the algorithm has yielded an average gap of 0.93%. For large instances, the tabu search algorithm has improved the makespan obtained from the greedy algorithm by 2.68%. Our analysis of each strategy within the tabu search algorithm also highlighted the importance of selecting appropriate candidates for movement, the method of moving them, and the strategy for accepting new solutions during the search process.

The models and methods covered in this paper also has limitations. The MILP model is computationally challenging for large instances, resulting in a lack of results for systematically evaluating the performance of metaheuristic algorithms on instances with more than 20 activities, which are

closer to real-life case sizes. For the metaheuristic algorithms, an important issue is incorporating the postponement effect of positive activities while balancing the trade-off in the multi-objective problem between resource management and operational costs. Although our metaheuristics perform well, there remains a gap between the solutions produced and the optimal solution, particularly regarding the third objective of minimizing unused resource over time periods.

Our future research will focus on improving algorithms to minimize unused resource by incorporating the postponement effect of positive activities, thereby achieving a better balance between operational costs and resource management. Additionally, we plan to explore other metaheuristic approaches, such as genetic algorithm-based local search strategies, to address the limitations identified in our current methods. This will help us find more efficient solutions and further close the gap with optimal solutions.

# Chapitre 3: Conclusion

Ce mémoire a introduit le problème de séquencement d'activités avec contraintes de ressource (RASP), un problème d'optimisation combinatoire qui se concentre sur le séquencement des activités dans un horizon de planification tout en gérant une ressource commune. Le problème a trois objectifs hiérarchiques : (1) minimiser la durée totale d'exécution des activités, (2) minimiser les coûts opérationnels, et (3) réduire la ressource inutilisée à la fin de chaque période. Nous avons présenté une formulation du RASP sous la forme d'un programme linéaire en nombres entiers mixtes, combinant les trois objectifs hiérarchiques dans la fonction objective en tant que somme pondérée. Cependant, les expériences computationnelles sur des petites instances ont mis en évidence la nécessité d'approches métaheuristiques pour traiter efficacement des problèmes de plus grande taille.

À cette fin, nous avons développé un algorithme glouton et un algorithme de recherche taboue. Ces méthodes ont généré des solutions quasi-optimales pour le RASP en termes des deux premiers objectifs et ont considérablement amélioré les temps de résolution pour les grandes instances. Nous avons testé différents paramètres de l'algorithme de recherche taboue. Nos expériences computationnelles ont démontré que la combinaison de paramètres la plus performante de l'algorithme de recherche taboue est la stratégie A-A1-B2-C2, qui a donné de bons résultats dans la résolution du RASP. Cette stratégie considère toutes les activités positives et négatives d'une période comme candidates au déplacement, en déplaçant une activité à la fois vers une période voisine. Les activités positives sont déplacées avant la période voisine sélectionnée pour assurer la synchronisation des impacts de la ressource des différents types d'activités. De plus, les solutions faisables mais non améliorantes sont acceptées pendant le processus pour surmonter les optima locaux. Grâce à des expériences computationnelles approfondies, nous avons démontré l'évolutivité de nos algorithmes proposés. L'algorithme de recherche taboue a trouvé la durée optimale dans plus de 85% des petites instances, et l'algorithme a produit un écart moyen de 0.93%. Pour les grandes instances, l'algorithme de recherche taboue a amélioré la durée obtenue par l'algorithme glouton de 2.68%. Notre analyse de chaque stratégie au sein de l'algorithme de recherche taboue a également mis en évidence l'importance de sélectionner des candidats appropriés pour le déplacement, la

méthode de déplacement et la stratégie d'acceptation des nouvelles solutions pendant le processus de recherche.

Les modèles et méthodes couverts dans ce mémoire ont également des limitations. Le modèle MILP est computationnellement difficile pour les grandes instances, ce qui entraîne un manque de résultats pour évaluer systématiquement la performance des algorithmes métaheuristiques sur des instances de plus de 20 activités, qui sont plus proches des tailles de cas réels. Pour les algorithmes métaheuristiques, un problème important est d'incorporer l'effet de report des activités positives tout en équilibrant le compromis dans le problème multi-objectif entre la gestion de la ressource et les coûts opérationnels. Bien que nos métaheuristiques fonctionnent bien, il subsiste un écart entre les solutions produites et la solution optimale, en particulier en ce qui concerne le troisième objectif qui consiste à minimiser la ressource inutilisée au fil du temps.

Nos futures recherches se concentreront sur l'amélioration des algorithmes pour minimiser la ressource inutilisée en intégrant l'effet de report des activités positives, afin d'atteindre un meilleur équilibre entre les coûts opérationnels et la gestion de la ressource. De plus, nous prévoyons d'explorer d'autres approches métaheuristiques, telles que les stratégies de recherche locale basées sur des algorithmes génétiques, pour répondre aux limitations identifiées dans nos méthodes actuelles. Cela nous aidera à trouver des solutions plus efficaces et à réduire davantage l'écart avec les solutions optimales.

# Bibliography


Agarwal, A., Colak, S., & Erenguc, S. (2011). A neurogenetic approach for the resource-constrained project scheduling problem. *Computers & Operations Research*, *38*(1), 44−50.

Al Aqel, G., Li, X., & Gao, L. (2019). A modified iterated greedy algorithm for flexible job shop scheduling problem. *Chinese Journal of Mechanical Engineering*, *32*(1).

Amjad, M. K., Butt, S. I., Kousar, R., Ahmad, R., Agha, M. H., Faping, Z., ... Asgher, U. (2018). Recent research trends in genetic algorithm based flexible job shop scheduling problems. *Mathematical Problems in Engineering*, *2018*(1), 9270802.

Anghinolfi, D., Montemanni, R., Paolucci, M., & Maria Gambardella, L. (2011). A hybrid particle swarm optimization approach for the sequential ordering problem. *Computers & Operations Research*, *38*(7), 1076−1085.

Asadzadeh, L. (2015). A local search genetic algorithm for the job shop scheduling problem with intelligent agents. *Computers & Industrial Engineering*, *85*, 376−383.

Bouleimen, K., & Lecocq, H. (2003). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research*, *149*(2), 268−281.

Chen, R., Yang, B., Li, S., & Wang, S. (2020). A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. *Computers & Industrial Engineering*, *149*, 106778.

Cheng, J., Fowler, J., Kempf, K., & Mason, S. (2015). Multi-mode resource-constrained project scheduling problems with non-preemptive activity splitting. *Computers & Operations Research*, *53*, 275−287.

Cho, J.-H., & Kim, Y.-D. (1997). A simulated annealing algorithm for resource constrained project scheduling problems. *Journal of the Operational Research Society*, *48*(7), 736−744.

De Reyck, B., & Herroelen, W. (1998). A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research*, *111*(1), 152−174.

Escudero, L. (1988). An inexact algorithm for the sequential ordering problem. *European Journal of Operational Research*, *37*(2), 236−249.

Gambardella, L. M., & Dorigo, M. (2000). An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, *12*(3), 237−255.

Hartmann, S., & Briskorn, D. (2022). An updated survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, *297*(1), 1−14.

Jamal, J., Shobaki, G., Papapanagiotou, V., Gambardella, L., & Montemanni, R. (2017). Solving the sequential ordering problem using branch and bound. *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, 1−9.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, *1*(1), 61−68.

Józefowska, J., Mika, M., Różycki, R., Waligóra, G., & Węglarz, J. (2001). Simulated annealing for multi-mode resource-constrained project scheduling. *Annals of Operations Research*, *102*, 137−155.

Kundakcı, N., & Kulak, O. (2016). Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem. *Computers & Industrial Engineering*, *96*, 31−51.

Lenstra, J., Rinnooy Kan, A., & Brucker, P. (1977). Complexity of machine scheduling problems. *Studies in Integer Programming*, *1*, 343−362.

Li, X., & Gao, L. (2016). An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *International Journal of Production Economics*, *174*, 93−110.

Lin, J., Zhu, L., & Gao, K. (2020). A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem. *Expert Systems with Applications*, *140*, 112915.

Luo, J., Vanhoucke, M., Coelho, J., & Guo, W. (2022). An efficient genetic programming approach to design priority rules for resource-constrained project scheduling problem. *Expert Systems with Applications*, *198*, 116753.

Montoya, C., Bellenguez-Morineau, O., Pinson, E., & Rivreau, D. (2014). Branch-and-price approach for the multi-skill project scheduling problem. *Optimization Letters*, *8*, 1721−1734.

Pellerin, R., Perrier, N., & Berthaut, F. (2020). A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, *280*(2), 395−416.

Phuntsho, T., & Gonsalves, T. (2023). Maximizing the net present value of resource-constrained project scheduling problems using recurrent neural network with genetic algorithm. *2023 International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT)*, 524−530.

Pritsker, A. A. B., Waiters, L. J., & Wolfe, P. M. (1969). Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, *16*(1), 93−108.

Shirzadeh Chaleshtarti, A., & Shadrokh, S. (2014). A branch and cut algorithm for resource-constrained project scheduling problem subject to nonrenewable resources with pre-scheduled procurement. *Arabian Journal for Science and Engineering*, *39*, 8359−8369.

Skinderowicz, R. (2017). An improved ant colony system for the sequential ordering problem. *Computers & Operations Research*, *86*, 1−17.

Vanhoucke, M., & Coelho, J. (2019). Resource-constrained project scheduling with activity splitting and setup times. *Computers & Operations Research*, *109*, 230−249.

Xie, J., Li, X., Gao, L., & Gui, L. (2023). A hybrid genetic tabu search algorithm for distributed flexible job shop scheduling problems. *Journal of Manufacturing Systems*, *71*, 82−94.

Xiong, H., Shi, S., Ren, D., & Hu, J. (2022). A survey of job shop scheduling problem: The types and models. *Computers & Operations Research*, *142*, 105731.

Zhang, G., Hu, Y., Sun, J., & Zhang, W. (2020). An improved genetic algorithm for the flexible job shop scheduling problem with multiple time constraints. *Swarm and Evolutionary Computation*, *54*, 100664.

Zhao, Z., Zhou, M., & Liu, S. (2022). Iterated greedy algorithms for flow-shop scheduling problems: A tutorial. *IEEE Transactions on Automation Science and Engineering*, *19*(3), 1941−1959.

Çaliş, B., & Bulkan, S. (2013). A research survey: Review of AI solution strategies of job shop scheduling problem. *Journal of Intelligent Manufacturing*, *26*(5), 961−973.