論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Approximation algorithms for single-machine scheduling problems with a non-renewable resource
著者(和文)	橋本進
Author(English)	Susumu Hashimoto
出典(和文)	学位:博士（工学）, 学位授与機関:東京工業大学, 報告番号:甲第12274号, 授与年月日:2022年6月30日, 学位の種別:課程博士, 審査員:塩浦 昭義,松井 知己,梅室 博行,大和 毅彦,中田 和秀,水野 眞治
Citation(English)	Degree:Doctor (Engineering), Conferring organization: Tokyo Institute of Technology, Report number:甲第12274号, Conferred date:2022/6/30, Degree Type:Course doctor, Examiner:,,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Doctoral Thesis

Approximation algorithms for single-machine scheduling problems with a non-renewable resource

Susumu Hashimoto

Supervisor Prof. Akiyoshi Shioura

June, 2022

Department of Industrial Engineering and Economics
School of Engineering
Tokyo Institute of Technology

Abstract

This thesis deals with single-machine scheduling problems with a non-renewable resource and total weighted completion time criterion. Non-renewable resources refer to the resources that are consumed when processing a job. The non-renewable resource often appears in real-world scheduling because it includes typical constraints, such as raw materials. The total weighted completion time criterion is one of the most studied objective functions in researches of scheduling problems. Many of these problems are known to be NP-hard, even if they have some input restrictions. If a problem is NP-hard, there is no polynomial-time algorithm to solve the problem unless P=NP. Thus, approximation algorithms are studied for them. A polynomial-time algorithm is called an α-approximation algorithm for a minimization problem if the algorithm always outputs a solution whose objective value is α times the optimal value at most. Despite several approximation studies for the problem, no approximation algorithm is known, even with some input restrictions so far.

In this thesis, we give approximation results for the problems. To obtain the results, we establish a problem simplification technique that preserves approximation ratios of the approximation algorithms. This technique plays an important role in the proof of our results because it can fix several inputs. Based on this technique, we find approximation algorithms for two problems.

First, we prove a conjecture by Györgyi and Kis [19]. The conjecture states that an algorithm is 2-approximation for the problem such that the processing times are all equal and the weights are proportional to the resource consumption. Besides, we show that the factor 2 is tight for the algorithm.

Second, we establish a 3-approximation algorithm for the problem such that the weights are proportional to the resource consumption. The algorithm is the first approximation algorithm for the problem. Moreover, we also show that the factor 3 is tight for the algorithm.

Keywords Scheduling, List Scheduling Algorithm, Non-renewable Resource, Approximation Algorithm

Contents

Chapter 1

Introduction

This chapter gives the background and the motivation of the thesis. Section 1.1 describes the background of problems and algorithms treated in the thesis. Section 1.2 states the objective of the thesis. Finally, Section 1.3 gives the structure of the thesis.

1.1 Background

The thesis studies approximation list scheduling algorithms for single-machine scheduling problems with a non-renewable resource and the total weighted completion time criterion. This section elucidates the problems, then we explain approximation algorithms and list scheduling algorithms. At first, we introduce the most fundamental scheduling problem.

Single-machine Scheduling Problem (SSP) has jobs, a machine, and a schedule criterion or objective function. The machine can process each job, and its processing time is given. SSP aims to find the best schedule or when the machine completes (starts) each job. SSP ordinary has three constraints as follows:

- The jobs cannot be preempted.
- The machine can process the jobs after a fixed time.
- The machine cannot process multiple jobs simultaneously.

We study the total weighted completion time criterion for SSPs. This criterion gives a weight for each job to minimize the total sum of weighted completion times. Based on SSP, we explain Single-machine Scheduling Problems with a Non-renewable Resource(NR-SSP).

NR-SSP is one of the simplest SSPs dealing with non-renewable resources. Non-renewable resources are one of the machine's resources. The machine consumes non-renewable resources when it starts processing a job. NR-SSP considers the situation that has no plenty non-renewable resources and the lack resources are replenished in future. NR-SSP contains jobs, a machine, single-type non-renewable resources, supply plans of the resource, and a schedule criterion. Each job has a processing time and a resource consumption amount, and each supply plan consists of a supply date and a supply amount.

Constraints of NR-SSP are as follows:

- The jobs cannot be preempted.

- The machine can process the jobs after a fixed time.
- The machine cannot process multiple jobs simultaneously.
- Starting each job requires consuming the resources.
- On each supply date, the fixed amount of resources are replenished.
- There are no resources except to be supplied.

Note that the first three constraints are the same as SSP.

The non-renewable resource often appears in the real-world scheduling problems. We give two examples below:

Example 1. Paper factory

We consider a factory that produces papers. The papers made of pulps, and wholesalers supply them to the factory. Thus, the pulps are one of the factory's resources, and besides, paper-producing jobs consume them. Therefore, pulps can be regarded as a non-renewable resource for jobs in the factory.

Example 2. Startup company

This example focuses on a growing startup company. Various projects, such as building another office and capital expenditure, can be considered ways to expand the company's business. However, the company does not have enough cash for them, and thus, it needs to assign future profits and loans to them. In this case, projects are regarded as jobs, and money is considered as a non-renewable resource.

The first example shows that the non-renewable resources include ingredients and intermediate products of the main product. The second example implies that money can be one of the non-renewable resources in some situations. Since the former situation is common in real-world factories, NR-SSP is one of the fundamental problems for practical scheduling problems.

Next, we elucidate approximation algorithms. Most of the scheduling problems can be formulated as combinatorial optimization problems. Thus, exponential-time algorithms, such as brute-force algorithms, can solve them. However, these algorithms take an enormous amount of time except for tiny problems. For example, a SSP with 20 jobs has $20! \approx 2.43 \times 10^{18}$ possible job processing orders. Thus, primitive brute-force algorithms must check 2.4×10^{18} schedules at least. Therefore, polynomial-time algorithms are studied for scheduling problems. Although some scheduling problems can be solved in polynomial-time [8, 14, 23], many problems are known to be NP-hard [4, 10, 14, 16, 18, 20]. If a problem is NP-hard, then there is no polynomial-time algorithm to solve it unless P=NP. For NP-hard problems, many studies investigate approximation algorithms.

For a minimization problem, a polynomial-time algorithm is said to be a $\rho(n)$-approximation algorithm if the algorithm always outputs a solution whose objective value is less than or equals to $\rho(n) * OPT$, where n represents the input size and OPT denotes the optimal value. The factor $\rho(n)$ is called approximation ratio of the algorithm. Besides, $O(1)$-approximation algorithm is also called constant-factor approximation algorithm. For constant-factor approximation algorithms, if its approximation ratio can be less than $1 + \epsilon$ for any constant $\epsilon > 0$, the algorithm is called PTAS(Polynomial-Time Approximation Scheme). Moreover, if the time complexity of a PTAS is polynomial of ϵ^{-1} and the input size, the PTAS is specially called FPTAS(Fully Polynomial-Time Approximation Scheme).

Finally, we explain list scheduling algorithms. List scheduling algorithm is one of the greedy algorithms for scheduling problems. The algorithm consists of the following steps:

List scheduling algorithm(for single-machine scheduling problems)

Inputs: Jobs and other inputs of the problem
Outputs: Completion times
Step1. Sort the jobs.
Step2. Process the first job as soon as possible.
Step3. Process the next job as soon as possible while the starting time is after the completion of the previous job.
Step4. Repeat Step3 until all the jobs are completed.

Since the algorithm constructs only single schedule, it is known as one of the fast algorithms for scheduling problems. Although the algorithm is simple, there are exact and constant-factor approximation results for some scheduling problems. These results are summarized in Section 2.4.

1.2 Objective

The objective of the thesis is to establish constant-factor approximation algorithms for NR-SSPs with the total weighted completion time criterion. Györgyi and Kis [18] show that the problem is NP-hard even if it has the following two input limitations:

(A) Unit processing times,
(B) Weights proportional to resource consumption.

Besides, they prove that a list scheduling algorithm with non-increasing processing time order is a 3-approximation algorithm for the NR-SSP with (A) and (B). After that, Györgyi and Kis [19] conjecture that the algorithm is actually a 2-approximation algorithm for the problem. However, the conjecture has not been proved until now. Moreover, no constant-factor approximation algorithm is found for the more general NR-SSPs.

The thesis gives new approximation results for the next two problems:

Problem (I): NR-SSP with the total weighted completion time criterion, (A), and (B),
Problem (II): NR-SSP with the total weighted completion time criterion and (B).

For Problem (I), we show that the conjecture by Györgyi and Kis [19] is true; namely, we prove that a list scheduling algorithm with non-increasing processing time order is a 2-approximation algorithm for Problem (I). Besides, we also show that the approximation ratio is tight for the algorithm, and no instance achieves the approximation ratio exactly.

For Problem (II), we develop a 3-approximation list scheduling algorithm. Problem (II) is a generalized problem of Problem (I), and the algorithm is the first constant-factor approximation algorithm for the problem. Similar to the above, we also show that the approximation ratio is tight for the algorithm, and no instance achieves the approximation ratio exactly.

1.3 Structure

The structure of the thesis is below.

Chapter 2 explicates the formal background. Chapter 3 shows a simplification technique that is used for proving our results. This technique can fix all inputs about supply plans when we prove main results described in Section 1.2. Chapter 4 proves the main results for Problem (I) or NR-SSP with the total weighted completion time criterion, unit processing times, and weights proportional to resource consumption. Chapter 5 shows the main results for Problem (II) or NR-SSP with the total weighted completion time criterion and weights proportional to resource consumption.

Chapter 2

Preliminary

This chapter gives a formal explanation of the background of the thesis. Section 2.1 formulates NR-SSP mathematically. Section 2.2 describes notations and assumptions (with no loss of generality) used in this thesis. Section 2.3 explains a list scheduling algorithm for NR-SSP. Section 2.4 summarizes previous researches related to the thesis. Finally, Section 2.5 shows the motivation and main results of the thesis.

2.1 Problem formulation

This section shows a mathematical formulation of NR-SSP.

NR-SSP contains jobs $\mathcal{J} = (J_1, \ldots, J_n)$, a machine, single-type non-renewable resources, and q supply plans of the resources. Each job J_j has a weight $w_j > 0$, a consumption amount $a_j > 0$, and a processing time $p_j > 0$, and the ith supply plan consists of a supply date $u_i \geq 0$ and a supply amount $b_i > 0$. In this problem, any schedule is represented by an n-dimensional vector $\boldsymbol{C} = (C_1, C_2, \ldots, C_n)^\mathsf{T}$, where C_j means completion time of J_j and $(\cdot)^\mathsf{T}$ denotes transpose.

Constraints of NR-SSP are as follows:

- The jobs cannot be preempted.
- The machine can process the jobs after time 0.
- The machine cannot process multiple jobs simultaneously.
- Starting job J_j requires to consume a_j resources.
- b_i resources are replenished at time u_i.
- There are no resources except to be supplied.

The last three constraints can be expressed as the next condition:

$$\sum_{j:C_j-p_j\leq T} a_j \leq \sum_{i:u_i\leq T} b_i \text{ for any } T \geq 0 \tag{2.1}$$

For this problem, various objective functions are studied, for examples, total weighted completion times(minimizing $\sum_j w_j C_j$) and makespan (minimizing $\max_j C_j$). Depending on the objective function, NR-SSP may have extra inputs. For an instance, NR-SSP with maximum lateness objective has an n-dimensional vector $\boldsymbol{d} = (d_1, d_2, \ldots, d_n)^\mathsf{T}$ that represents due dates, and the problem aims to minimize $\max_j C_j - d_j$.

2.2 Notations and Assumptions

This section introduce notations and assumptions (with no loss of generality) for the simplicity of the thesis.

We introduce vector expressions of the inputs, $\boldsymbol{p} = (p_1, p_2, \ldots, p_n)^\mathsf{T}$, $\boldsymbol{w} = (w_1, w_2, \ldots, w_n)^\mathsf{T}$, $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)^\mathsf{T}$, $\boldsymbol{b} = (b_1, b_2, \ldots, b_q)^\mathsf{T}$, and $\boldsymbol{u} = (u_1, u_2, \ldots, u_q)^\mathsf{T}$, where $(\cdot)^\mathsf{T}$ denotes transpose. In the rest of the thesis, we deal with NR-SSPs with total weighted completion time criterion and with a constraint $\boldsymbol{w} = \boldsymbol{a}$. Then, we represent an instance by a tuple of inputs, $\langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b} \rangle$. Note that the input $\boldsymbol{w}$ is abbreviated from the expression because of the additional constraint $\boldsymbol{w} = \boldsymbol{a}$. Moreover, we denote the objective function of an instance $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b} \rangle$ by $f_S(\boldsymbol{C})$, namely, $f_S(\boldsymbol{C}) = \boldsymbol{w}^\mathsf{T}\boldsymbol{C} = \boldsymbol{a}^\mathsf{T}\boldsymbol{C}$. Let $\boldsymbol{0}_n$ be an n-dimensional zero vector $(0, 0, \ldots, 0)^\mathsf{T}$, $\boldsymbol{1}_n$ be an n-dimensional 1's vector $(1, 1, \ldots, 1)^\mathsf{T}$, and $\boldsymbol{n}_n$ be an n-dimensional vector $(1, 2, \ldots, n)^\mathsf{T}$. We define $\boldsymbol{\sigma}(\boldsymbol{v}) = (v_1, v_1 + v_2, \ldots, \sum_j^n v_j)^\mathsf{T}$, where $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)^\mathsf{T}$ is any n-dimensional vector. Moreover, we let $N = \{1, 2, \ldots, n\}$ and $\mathcal{O}$ be a set of all the permutations of $(1, 2, \ldots, n)$.

For any instance $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b} \rangle$ and any permutation $\boldsymbol{o} \in \mathcal{O}$, let $\boldsymbol{r} = (r_1, r_2, \ldots, r_n)^\mathsf{T} = (w_1/p_1, w_2/p_2, \ldots, w_n/p_n)^\mathsf{T}$ be a n-dimensional vector and $\boldsymbol{C}^*(S) = (C_1^*(S), C_2^*(S), \ldots, C_n^*(S))^\mathsf{T}$ be an optimal schedule for S. Then we impose the following assumptions without loss of generality:

Assumption 1. $\sum_{j=1}^n a_j = \sum_{i=1}^q b_i$.

Assumption 2. $\max_j \frac{a_j}{p_j} \leq 1$.

Assumption 3. *Jobs are sorted such that* $C_1^*(S) < C_2^*(S) < \cdots < C_n^*(S)$.

Assumption 1 does not lose generality because the problem becomes infeasible if $\sum_{j=1}^n a_j > \sum_{i=1}^q b_i$, and if $\sum_{j=1}^n a_j < \sum_{i=1}^q b_i$, we use only the first $\sum_{j=1}^n a_j$ resources. Assumption 2 also does not lose generality because minimizing $\sum_j a_j C_j (= \sum_j w_j C_j)$ is equivalent to minimizing $\sum_j \lambda a_j C_j$ for any $\lambda > 0$. Moreover, we can suppose Assumption 3 because we do not use any information of the optimal schedule in our algorithms.

2.3 List scheduling algorithms

Mathematically, list scheduling algorithms can be expressed as follows:

List scheduling algorithm(for single-machine scheduling problems)

Inputs: Jobs $(J_1, J_2, \ldots, J_n)$ and other inputs of the problem
Outputs: Completion times $\boldsymbol{C} = (C_1, C_2, \ldots, C_n)$
Step1. Calculate a permutation $\boldsymbol{o} = (o(1), o(2), \ldots, o(n))$ of $(1, 2, \ldots, n)$ by the inputs.
Step2. For $i = o(1), o(2), \ldots, o(n)$, process $J_{o(i)}$ at

$$C_{o(i)} = \min \left\{ t | t \geq C_{o(i-1)} + p_{o(i)}, t \text{ satisfies (2.1)} \right\},$$

where $C_{o(0)} = 0$.

Step1 calculates the order of the jobs, and Step2 processes each job at the earliest time respectively while keeping the order. For simple ordering such as non-increasing p_j order, the running time of Step1 is $O(n \log n)$. Besides, the time complexity of Step2 is $O(n)$ if each repetition runs in $O(1)$ time. For NR-SSP problems, Step2 or the list scheduling algorithm with a permutation $\boldsymbol{o} = (o(1), o(2), \ldots, o(n))$ can be written as follows:

Algorithm 1 A list-scheduling algorithm for NR-SSP

Input: NR-SSP inputs $(n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{w}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b})$ and a permutation $\boldsymbol{o} \in \mathcal{O}$
Output: Completion times $\boldsymbol{C} = (C_1, C_2, \ldots, C_n)^\mathsf{T}$

```
if $\sum_{j=1}^{n} a_j > \sum_{i=1}^{q} b_i$ then
    return INFEASIBLE                                   ▷ The instance is infeasible
end if
$A \leftarrow a_{o(1)}$
$B \leftarrow b_1$
$j \leftarrow 1$
$i \leftarrow 1$
$C_{o(0)} \leftarrow 0$
Calculate a permutation $(o'_1, o'_2, \ldots, o'_q)$ of $(1, 2, \ldots, q)$ such that $u_{o'_1} \leq u_{o'_2} \leq \cdots \leq u_{o'_q}$
while $j < n + 1$ do
    while $A \leq B$ do
        $C_{o(j)} \leftarrow \max\left\{C_{o(j-1)}, u_{o'_i}\right\} + p_{o(j)}$.
        if $j = n$ then
            return $\boldsymbol{C}$
        end if
        $j \leftarrow j + 1$
        $A \leftarrow A + a_{o(j)}$
    end while
    $i \leftarrow i + 1$
    $B \leftarrow B + b_{o'_i}$
end while
```

Since i or j increases by 1 for each repetition, the algorithm repeats the line 10 to 21 at most $n+q$ times. Therefore, the time complexity of Algorithm 1 is $O(\max\{n, q \log q\})$ because the line 9 sorts the q-dimensional vector $\boldsymbol{u}$.

In the rest of the thesis, we let $\boldsymbol{C^o}(S) = (C_1^{\boldsymbol{o}}(S), C_2^{\boldsymbol{o}}(S), \dots, C_n^{\boldsymbol{o}}(S))^\mathsf{T}$ be a schedule for an instance $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b} \rangle$ calculated by Algorithm 1 with a permutation $\boldsymbol{o} \in \mathcal{O}$.

2.4 Previous research

This section summarizes previous researches. To denote scheduling problems simply, we introduce Graham's notation [13] at first.

Graham's notation is a triplet notation $\alpha|\beta|\gamma$ by Graham et al. [13], which describes scheduling problems. In this notation, α represents the information of the machines, γ gives the objective function, and β implies the other constraints. Besides, this notation assume that the problem has n jobs, and has the constraints of SSP defined in Section 1.1 unless it has conflicting constraints. We summarize the notations of α, β, and γ used in the thesis below:

α notations

- **Single-machine**($\alpha = 1$): The problem has a single machine.
- **Identical parallel machines**(Pm): The problem has m identical parallel machines; in other words, the problem deals with m machines that have the same processing time for each job.

β notations

- **No constraints** ($\beta = [blank]$): The β field is blank if there are only the basic constraints.
- **Non-renewable resource** ($nr = k$): For the case $k = 1$, this notation gives new inputs, resource consumption amounts of the jobs $\boldsymbol{a} = (a_1, a_2, \dots, a_n)^\mathsf{T}$, the number of supply plans q, supply dates $\boldsymbol{u} = (u_1, u_2, \dots, u_q)^\mathsf{T}$, and supply amounts $\boldsymbol{b} = (b_1, b_2, \dots, b_q)^\mathsf{T}$. Besides, the problem has the constraints about non-renewable resource or (2.1). For the case $k \geq 2$, the problem has these inputs and constraints for each non-renewable resource independently.
- **Release dates** (r_j): This notation gives a new input $\boldsymbol{r} = (r_1, r_2, \dots, r_n)^\mathsf{T}$. Then each job cannot start processing until time r_i, namely, $S_j \geq r_j$ for each $j \in \{1, 2, \dots, n\}$, where S_j denotes the starting time of J_j.
- **Due dates** (d_j): This notation gives a new input $\boldsymbol{d} = (d_1, d_2, \dots, d_n)^\mathsf{T}$. Then each job must complete by its due date, that is, $C_j \leq d_j$ for each $j \in \{1, 2, \dots, n\}$.

- **Precedence constraints** ($prec$): This notation gives a new input $P \subset \{1, 2, \ldots, n\}^2$ and precedence constraints $S_l \geq C_k$ for any $(k, l) \in P$, where S_j denotes the starting time of J_j
- **Constraints for inputs**: The constraints for the inputs is expressed by represent elements. For instances, $\boldsymbol{p} = (1, 1, \ldots, 1)^\mathsf{T}$ and $\boldsymbol{w} = \boldsymbol{a}$ can be written as $p_j = 1$ and $w_j = a_j$ respectively.

γ notations

- **Minimizing makespan** ($\gamma = C_{\max}$): This objective aims to minimize the maximum completion time $\max_j C_j$.
- **Minimizing total completion times** ($\sum C_j$): This objective aims to minimize the total completion times $\sum_j C_j$.
- **Minimizing total weighted completion times** ($\sum w_j C_j$): This notation gives the problem a new input $\boldsymbol{w} = (w_1, w_2, \ldots, w_n)^\mathsf{T}$ or weights of the jobs. Then the objective aims to minimize the total weighted completion times $\sum_j w_j C_j$.
- **Minimizing maximum lateness** ($L_{\max}$): This notation gives the problem a new input $\boldsymbol{d} = (d_1, d_2, \ldots, d_n)^\mathsf{T}$ or due dates of the jobs. Then the objective aims to minimize the maximum lateness $\max_j L_j = \max_j(\max(C_j - d_j, 0))$.
- **Minimizing the number of late jobs** ($\sum U_j$): This notation gives the problem a new input $\boldsymbol{d} = (d_1, d_2, \ldots, d_n)^\mathsf{T}$ or due dates of the jobs. Then the objective aims to minimize the number of late jobs $\sum_j U_j = |\{j | C_j > d_j\}|$.
- **Minimizing total tardiness** ($\sum T_j$): This notation gives the problem a new input $\boldsymbol{d} = (d_1, d_2, \ldots, d_n)^\mathsf{T}$ or due dates of the jobs. Then the objective aims to minimize the total tardiness $\sum_j T_j = \sum_j \min((C_j - d_j), 0)$.

Next, we overview previous researches related to the thesis. Although the list scheduling algorithm is simple, it is one of the powerful algorithms for scheduling problems. Smith [24] show that a list scheduling problem with w_j/p_j non-increasing order proves $1||\sum w_j C_j$. Graham [11] shows that a list scheduling algorithm with any order is a $(2 - \frac{1}{m})$-approximation algorithm for $Pm||C_{\max}$. After that, Graham [12] proves that the list scheduling problem with p_j non-increasing order is $(\frac{4}{3} - \frac{1}{3m})$-approximation algorithm for the problem.

Several studies deal with scheduling problems with non-renewable resource(s). Carlier and Rinnooy Kan [8] establish polynomial-time algorithms for precedence constrained scheduling problems with a non-renewable resource, which can be written as $Pn|nr = 1, prec|C_{\max}$, $Pn|nr = 1, d_j, prec|C_{\max}$, and $Pn|nr = 1, prec|L_{\max}$. Note that $\alpha = Pn$ represents the problem has n machine, and thus, each job has an available machine in any time. Slowiński [23] proposes a polynomial-time algorithm with parametric linear programming approach for preemptive unrelated machine scheduling problems with (renewable) resources and a non-renewable resource. Grigoriev et al. [14] give polynomial-time algorithms for $1|nr = 1, p_j = 1|L_{max}$ and $1|nr = 1, p_j = 1|C_{\max}$. Moreover, they show that $1|nr = 2, p_j = 1|L_{\max}$, $1|nr = 2, p_j = 1|C_{\max}$, and $1|nr = 1|C_{\max}$ are NP-hard. Furthermore, they prove that list scheduling algorithms with d_j non-decreasing order for $1|nr = 2, p_j = 1|L_{\max}$ and a list scheduling algorithm with any order for $1|nr = 1|C_{\max}$ are 2-approximation algorithms. Gafarov et al. [10] verify that $1|nr = 1|L_{\max}$, $1|nr = 1|\sum U_j$, $1|nr = 1|\sum T_j$, and $1|nr = 1|\sum C_j$ are NP-hard problems. Györgyi and Kis [15] give an FPTAS and a PTAS for $1|nr = 1, q = 2|C_{\max}$ and $1|nr = 1, q = const.|C_{\max}$ respectively. Györgyi and Kis develop PTASs for $1|nr = const., q = const.|C_{\max}$, $1|nr = 1, p_j = a_j|C_{\max}$, and $1|nr = 1, p_j = a_j, r_j|C_{\max}$. Györgyi and Kis [17] establish a PTAS for $1|nr = const., r_j|C_{\max}$. Bredereck et al. [3] give many parameterized computational complexity results for NR-SSPs with makespan objective.

In contrast to NR-SSP with the makespan criterion or $1|nr = 1|C_{\max}$, there is no known constant-factor approximation algorithm for NR-SSP with the total (weighted) completion time criterion, that is, $1|nr = 1|\sum C_j$ and $1|nr = 1|\sum w_jC_j$. Thus, some researches address $1|nr = 1|\sum w_jC_j$ with input limitations. Kis [20] proves that $1|nr = 1, q = 2|\sum C_j$ is NP-hard, and proposes an FPTAS for $1|nr = 1, q = 2|\sum w_jC_j$. Györgyi and Kis [18] show that $1|nr = 1, p_j = 1, w_j = a_j, q = 2|\sum w_jC_j$ and $1|nr = 1, p_j = w_j = a_j, q = 2|\sum w_jC_j$ are NP-hard problems. Besides, they establish a 2-approximation list scheduling algorithm with non-decreasing p_j order for $1|nr = 1, p_j = w_j = a_j|\sum w_jC_j$, and a PTAS for $1|nr = 1, p_j = w_j, q = const.|\sum w_jC_j$. Györgyi and Kis [19] verify that a list scheduling algorithm with non-decreasing p_j order is a 2-approximation algorithm for $1|nr = 1, a_j = const.|\sum C_j$. Moreover, they prove that a list scheduling algorithm with a_j non-increasing order for $1|nr = 1, p_j = 1, w_j = a_j|\sum w_jC_j$ is a 3-approximation algorithm, and it becomes 2-approximation algorithm when $q = 2$. Bérczi et al. [4] prove that $1|nr = 1, a = 1|\sum C_j$ is an NP-hard problem, and a list scheduling problem with p_j non-decreasing order is a 1.5-approximation algorithm for the problem. Besides, they establish a 6-approximation list scheduling algorithm and a PTAS for $1|nr = 1, p_j = 0|\sum w_jC_j$.

Inventory Constrained Scheduling Problem (ICSP) is related to NR-SSP. ICSP deals with a non-renewable resource, and it does not have any supply plans by contrast. In ICSP, some jobs obtain resources instead of consumption. Briskorn et al. [5] prove that $1|inv|L_{\max}$, $1|inv|\sum C_j$, $1|inv|\sum w_jC_j$, and $1|inv|\sum U_j$ with some input limitations are NP-hard, where inv in β-field represents ICSP. Moreover, they study input limitations allow us to solve these problems in polynomial time. Kononov and Lin [21] address the more general situation of $1|inv|\sum w_jC_j$, whose jobs can both consume and supply resources. They establish a 2-approximation algorithm for the problem with input limitations. Briskorn et al. [6] study the optimal conditions of $1|inv|\sum w_jC_j$, and they propose a branch and bound method and a dynamic programming for the problem. Briskorn and

Leung [7] show some properties of optimal schedules for $1|inv|L_{\max}$, and develop a branch and bound algorithm for the problem. Morsy and Pesch [22] propose 2-approximation algorithms for $1|inv|\sum w_jC_j$ with two cases of input limitations, which are proved to be NP-hard by [5]. Bazgosha et al. [2] design a linear integer programming model of $Pm|r_j, inv|C_{\max}$, and they establish metaheuristic algorithms for the problem. Davari et al. [9] show that $1|r_j, inv|C_{\max}$ and deciding the feasibility of $1|inv|C_{\max}$ are NP-hard. Moreover, they propose mixed integer programming formulations and heuristic algorithms for $1|r_j, inv|C_{\max}$.

2.5 Motivation and main contribution

As described the previous section, no approximation algorithm has been found for NR-SSP with the total weighted completion time criterion until now. This thesis aims to find better approximation results for the problem with input limitations. As a result, we prove that a list scheduling algorithm with a_j non-increasing order is a 2-approximation algorithm for the problem with unit processing times($\boldsymbol{p} = \mathbf{1}_n$) and weights proportional to resource consumption ($\boldsymbol{w} = \boldsymbol{a}$), which is conjectured by [19]. Besides, we establish a 3-approximation list scheduling algorithm for the above problem without unit processing time condition, which has been found no approximation algorithm so far.

Formally, we prove the next two theorems in the thesis.

Theorem 1. *List scheduling algorithm with non-increasing order of resource consumption is a 2-approximation algorithm for NR-SSP with $\boldsymbol{p} = \mathbf{1}_n$, $\boldsymbol{a} = \boldsymbol{w}$, and total weighted completion time criterion (or $1|nr = 1, p_j = 1, w_j = a_j|\sum w_jC_j$).*

Theorem 2. *A list scheduling algorithm is a 3-approximation algorithm for NR-SSP with $\boldsymbol{a} = \boldsymbol{w}$ and total weighted completion time criterion (or $1|nr = 1, w_j = a_j|\sum w_jC_j$).*

Moreover, we prove these approximation ratios are tight for each algorithm, and there are no instance that achieves the approximation ratio precisely. In the rest of the thesis, Chapter 4 proves Theorem 1, and Chapter 5 solves Theorem 2. Before these, Chapter 3 introduces a simplification technique used for both proofs.

Chapter 3

Problem simplification

This chapter introduces a simplification theorem used to prove Theorems 1 and 2. Suppose any instance S of NR-SSP with the total weighted completion time criterion and weights proportional to resource consumption ($\boldsymbol{w} = \boldsymbol{a}$), which can be written as $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b} \rangle$. Then the theorem allows us to fix all the inputs of supply plans in the proofs.

At first, we describe notations used in the rest of the thesis. We define

$$A^*_j = \sum_{k=j}^{n} a_k \text{ and } A^{\boldsymbol{o}}_j = \sum_{k=j}^{n} a_{o(k)} \text{ for } j \in N.$$

For simplicity of discussion, let $A^*_j = A^{\boldsymbol{o}}_j = 0$ for any $j \geq n+1$. Then we define

$$\lambda_j = \max\{k \in N | A^{\boldsymbol{o}}_k \geq A^*_j\} \text{ for any } j \in \{1, 2, \ldots, n+1\}.$$

From the definition, we see that

$$A^{\boldsymbol{o}}_{\lambda_j+1} < A^*_j \leq A^{\boldsymbol{o}}_{\lambda_j} \text{ for any } \ j \in N. \tag{3.1}$$

To apply the simplification theorem, the list scheduling algorithm must satisfy the following assumption:

Assumption 4. *Let $\boldsymbol{o}$ be a permutation of the list scheduling algorithm for any NR-SSP instance $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b} \rangle$. Then $A^{\boldsymbol{o}}_{\lambda_j} \leq 2A^*_j$ for any $j \in \{1, 2, \ldots, n+1\}$.*

Using this assumption, we prove the next theorem:

Theorem 3. *Let $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b} \rangle$ and define $S^\dagger = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, n, (\boldsymbol{\sigma}(\boldsymbol{p}) - \boldsymbol{p}), \boldsymbol{a} \rangle$. Suppose $\boldsymbol{o} \in \mathcal{O}$ such that $A^{\boldsymbol{o}}_{\lambda_j} \leq 2A^*_j$ for any $j \in \{1, 2, \ldots, n+1\}$. Then for any $\alpha \geq 2$, $f_{S^\dagger}(\boldsymbol{C}^{\boldsymbol{o}}(S^\dagger)) < \alpha f_{S^\dagger}(\boldsymbol{C}^*(S^\dagger))$ implies $f_S(\boldsymbol{C}^{\boldsymbol{o}}(S)) < \alpha f_S(\boldsymbol{C}^*(S))$.*

This theorem states that to fix $q = n$, $\boldsymbol{u} = \boldsymbol{\sigma}(\boldsymbol{p}) - \boldsymbol{p}$, and $\boldsymbol{b} = \boldsymbol{a}$ preserves approximation ratio $\alpha \geq 2$ of list scheduling algorithms for any instance $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b} \rangle$ under Assumption 4. In this chapter, we divide Theorem 3 into the next two theorems.

Theorem 4. *Let* $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b}\rangle$ *be any instance of NR-SSP. Define* $S' = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, n, (\boldsymbol{C}^*(S) - \boldsymbol{p}), \boldsymbol{a}\rangle$. *Suppose* $\boldsymbol{o} \in \mathcal{O}$ *such that* $A^{\boldsymbol{o}}_{\lambda_j} \le 2A^*_j$ *for any* $j \in \{1, 2, \dots, n+1\}$. *Then for any* $\alpha \ge 2$, $f_{S'}(\boldsymbol{C}^{\boldsymbol{o}}(S')) < \alpha f_{S'}(\boldsymbol{C}^*(S'))$ *implies* $f_S(\boldsymbol{C}^{\boldsymbol{o}}(S)) < \alpha f_S(\boldsymbol{C}^*(S))$.

Theorem 5. *Let* $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, n, (\boldsymbol{C}' - \boldsymbol{p}), \boldsymbol{a}\rangle$ *be any instance of NR-SSP, where* $\boldsymbol{C}' = (C'_1, C'_2, \dots, C'_n)^\mathsf{T}$ *such that* $C'_1 \ge p_1$ *and* $C'_{j+1} - C'_j \ge p_{j+1}$ *for any* $j \in \{1, 2, \dots, n-1\}$. *Let* $\bar{S} = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, n, (\boldsymbol{\sigma}(\boldsymbol{p}) - \boldsymbol{p}), \boldsymbol{a}\rangle$ *and suppose* $\boldsymbol{o} \in \mathcal{O}$ *such that* $A^{\boldsymbol{o}}_{\lambda_j} \le 2A^*_j$ *for any* $j \in \{1, 2, \dots, n+1\}$. *Then for any* $\alpha \ge 2$, $f_{\bar{S}}(\boldsymbol{C}^{\boldsymbol{o}}(\bar{S})) < \alpha f_{\bar{S}}(\boldsymbol{C}^*(\bar{S}))$ *implies* $f_S(\boldsymbol{C}^{\boldsymbol{o}}(S)) < \alpha f_S(\boldsymbol{C}^*(S))$.

Theorem 4 can fix $q = n$, $\boldsymbol{u} = \boldsymbol{C}^*(S) - \boldsymbol{p}$, and $\boldsymbol{b} = \boldsymbol{a}$ for any instance of NR-SSP $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b}\rangle$ to prove Theorems 1 and 2. In other words, the theorem allows us to suppose that there is a Just-In-Time optimal schedule $\boldsymbol{C}^*(S)$. Besides, Theorem 5 can assume $\boldsymbol{C}^*(S) = \boldsymbol{\sigma}(\boldsymbol{p})$ in addition to the above because $\boldsymbol{C}^*(S)$ satisfies the assumption of the theorem. Thus, these two theorems can prove Theorem 3. In the rest of this chapter, we prove these two theorems.

Proof of Theorem 4

To prove Theorem 4, we verify the following lemmas:

Lemma 1. *Let* $\boldsymbol{a} = (a_1, a_2, \dots, a_n)^\mathsf{T}$, $\boldsymbol{v} = (v_1, v_2, \dots, v_n)^\mathsf{T}$, $\boldsymbol{b} = (b_1, b_2, \dots, b_q)^\mathsf{T}$, *and* $\boldsymbol{u} = (u_1, u_2, \dots, u_q)^\mathsf{T}$ *be non-negative vectors such that* $\sum_{i=1}^n a_i = \sum_{i=1}^q b_i$. *If*

$$\sum_{i:v_i \le t} a_i \le \sum_{i:u_i \le t} b_i \quad \textit{for any} \quad t \ge 0,$$

then $\boldsymbol{b}^\mathsf{T}\boldsymbol{u} \le \boldsymbol{a}^\mathsf{T}\boldsymbol{v}$.

Proof. Define two functions f_a and f_b of $t \ge 0$ by

$$f_a(t) = \sum_{i:v_i \le t} a_i, \quad f_b(t) = \sum_{i:u_i \le t} b_i.$$

Define a permutation $\boldsymbol{o} = (o_1, \dots, o_n)$ of $(1, 2, \dots, n)$ by non-decreasing order of $\boldsymbol{v}$ and a permutation $\boldsymbol{\pi} = (\pi_1, \dots, \pi_q)$ of $(1, 2, \dots, q)$ by non-decreasing order of $\boldsymbol{u}$. Let $\bar{t} = \max\{v_{o_n}, u_{\pi_q}\}$. Then

$$\begin{aligned}\int_0^{\bar{t}} f_a(t)dt &= a_{o_1}(v_{o_2} - v_{o_1}) + (a_{o_1} + a_{o_2})(v_{o_3} - v_{o_2}) + \cdots \\ &\quad + (a_{o_1} + \cdots + a_{o_{n-1}})(v_{o_n} - v_{o_{n-1}}) \\ &\quad + (a_{o_1} + \cdots + a_{o_n})(\bar{t} - v_{o_n}) \\ &= \bar{t}\sum_{i=1}^n a_i - \boldsymbol{a}^\mathsf{T}\boldsymbol{v},\end{aligned}$$

$$\int_0^{\bar{t}} f_b(t)dt = \bar{t}\sum_{i=1}^{q} b_i - \boldsymbol{b}^\mathsf{T}\boldsymbol{u}.$$

From the assumption of the lemma, we see

$$f_a(t) \le f_b(t) \text{ for any } 0 \le t \le \bar{t}.$$

Thus, we have

$$\bar{t}\sum_{i=1}^{n} a_i - \boldsymbol{a}^\mathsf{T}\boldsymbol{v} = \int_0^{\bar{t}} f_a(t)dt \le \int_0^{\bar{t}} f_b(t)dt = \bar{t}\sum_{i=1}^{q} b_i - \boldsymbol{b}^\mathsf{T}\boldsymbol{u}.$$

Since $\sum_{i=1}^{n} a_i = \sum_{i=1}^{q} b_i$, we obtain $\boldsymbol{b}^\mathsf{T}\boldsymbol{u} \le \boldsymbol{a}^\mathsf{T}\boldsymbol{v}$. □

Lemma 2. *Let $S' = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, n, (\boldsymbol{C} - \boldsymbol{p}), \boldsymbol{a}\rangle$ be any instance of NR-SSP, where $\boldsymbol{C} = (C_1, C_2, \ldots, C_n)^\mathsf{T}$ such that $C_1 \ge p_1$ and $C_{j+1} - C_j \ge p_{j+1}$ for any $j \in \{1, 2, \ldots, n-1\}$, then $\boldsymbol{C}$ is optimal for S'.*

Proof. It is clear that $\boldsymbol{C}$ is a feasible schedule for S'. Let $\boldsymbol{C}^v$ be any feasible schedule for S'. Then we have that

$$\sum_{j:C_j^v - p_j \le T} a_j \le \sum_{i:u_i \le T} b_i \text{ for any } T \ge 0$$

from (2.1), where $\boldsymbol{u} = \boldsymbol{C} - \boldsymbol{p}$ and $\boldsymbol{b} = \boldsymbol{a}$. We obtain

$$\boldsymbol{a}^\mathsf{T}(\boldsymbol{C} - \boldsymbol{p}) \le \boldsymbol{a}^\mathsf{T}(\boldsymbol{C}^v - \boldsymbol{p}),$$

by applying Lemma 1 for $\boldsymbol{a}$, $\boldsymbol{v} = \boldsymbol{C}^v - \boldsymbol{p}$, $\boldsymbol{b} = \boldsymbol{a}$, and $\boldsymbol{u} = \boldsymbol{C} - \boldsymbol{p}$. Since the object function value is $\boldsymbol{a}^\mathsf{T}\boldsymbol{C}^v$, $\boldsymbol{C}$ is optimal for S'. □

Lemma 3. *Let $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b}\rangle$ and $\tilde{S} = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q', \boldsymbol{u}', \boldsymbol{b}'\rangle$ be two instances of NR-SSP.If*

$$\sum_{i:u'_i \le T} b'_i \le \sum_{i:u_i \le T} b_i \text{ for any } T \ge 0, \tag{3.2}$$

then $\boldsymbol{C^o}(S) \le \boldsymbol{C^o}(\tilde{S})$ and $f_S(\boldsymbol{C^o}(S)) \le f_{\tilde{S}}(\boldsymbol{C^o}(\tilde{S}))$ for any $\boldsymbol{o} \in \mathcal{O}$.

Proof. From condition (3.2), Algorithm 1 for any $\boldsymbol{o} \in \mathcal{O}$ outputs $\boldsymbol{C^o}(S)$ for S and $\boldsymbol{C^o}(\tilde{S})$ for $\tilde{S}$ so that

$$\boldsymbol{C^o}(S) \le \boldsymbol{C^o}(\tilde{S}).$$

Since $\boldsymbol{w} > \boldsymbol{0}_n$ from the assumption, we have

$$\begin{aligned} f_S(\boldsymbol{C^o}(S)) &= \boldsymbol{w}^T\boldsymbol{C^o}(S) \\ &\le \boldsymbol{w}^T\boldsymbol{C^o}(\tilde{S}) \\ &= f_S(\boldsymbol{C^o}(\tilde{S})). \end{aligned}$$

□

Finally, we prove Theorem 4.

Proof of Theorem 4. Recall that $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b} \rangle$ and $S' = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, n, (\boldsymbol{C}^*(S) - \boldsymbol{p}), \boldsymbol{a} \rangle$ in Theorem 4. From Lemma 2, $\boldsymbol{C}^*(S)$ is an optimal schedule for S'. Since $\boldsymbol{C}^*(S)$ is a feasible schedule for S, (2.1) holds for every $T \geq 0$, namely,

$$\sum_{j : C^*(S)_j - p_j \leq T} a_j \leq \sum_{i : u_i \leq T} b_i \text{ for any } T \geq 0.$$

Hence $f_S(\boldsymbol{C}^{\boldsymbol{o}}(S)) \leq f_{S'}(\boldsymbol{C}^{\boldsymbol{o}}(S'))$ holds for any $\boldsymbol{o} \in \mathcal{O}$ from Lemma 3. If $f_{S'}(\boldsymbol{C}^{\boldsymbol{o}}(S')) < \alpha f_{S'}(\boldsymbol{C}^*(S'))$, then we see that

$$\begin{aligned} f_S(\boldsymbol{C}^{\boldsymbol{o}}(S)) &\leq f_{S'}(\boldsymbol{C}^{\boldsymbol{o}}(S')) \\ &< \alpha f_{S'}(\boldsymbol{C}^*(S')) \\ &= \alpha \boldsymbol{a}^\mathsf{T} \boldsymbol{C}^*(S) \\ &= \alpha f_S(\boldsymbol{C}^*(S)). \end{aligned}$$

□

Proof of Theorem 5

We prove Theorem 5 by the next lemma.

Lemma 4. *Let $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, n, (\boldsymbol{C} - \boldsymbol{p}), \boldsymbol{a} \rangle$ be any instance of NR-SSP, where $\boldsymbol{C} = (C_1, C_2, \ldots, C_n)^\mathsf{T}$ such that $C_1 \geq p_1$ and $C_{j+1} - C_j \geq p_{j+1}$ for any $j \in \{1, 2, \ldots, n-1\}$. Let $\boldsymbol{d}_k = (\boldsymbol{0}_k^\mathsf{T} \ \boldsymbol{1}_{n-k}^\mathsf{T})^\mathsf{T}$ be a vector whose first k elements are 0 and the latter $n - k$ elements are 1 for an integer $k \in \{0, 1, \ldots, n-1\}$. Define $S^\triangle = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, n, (\boldsymbol{C} + L\boldsymbol{d}_k - \boldsymbol{p}), \boldsymbol{a} \rangle$ for any $L \geq 0$. Suppose $\boldsymbol{o} \in \mathcal{O}$ such that $A^{\boldsymbol{o}}_{\lambda_j} \leq 2A^*_j$ for any $j \in \{1, 2, \ldots, n+1\}$. Then for any $\alpha \geq 2$, $f_S(\boldsymbol{C}^{\boldsymbol{o}}(S)) < \alpha f_S(\boldsymbol{C}^*(S))$ implies $f_{S^\triangle}(\boldsymbol{C}^{\boldsymbol{o}}(S^\triangle)) < \alpha f_{S^\triangle}(\boldsymbol{C}^*(S^\triangle))$.*

Proof. Clearly $\boldsymbol{C}$ is a feasible schedule for S. By Lemma 2, we have $\boldsymbol{C}^*(S) = \boldsymbol{C}$ and $\boldsymbol{C}^*(S^\triangle) = \boldsymbol{C} + L\boldsymbol{d}_k$. Then we see

$$\begin{aligned} f_{S^\triangle}(\boldsymbol{C}^*(S^\triangle)) - f_S(\boldsymbol{C}^*(S)) &= \boldsymbol{a}^\mathsf{T}(L\boldsymbol{d}_k) \\ &= L \sum_{j=k+1}^{n} a_j \\ &= LA^*_{k+1}. \end{aligned}$$

Since supply dates $C_j - p_j, j \in \{1, 2, \ldots, k\}$ in $S^\triangle$ are equal to those in S, we see from Algorithm 1 that

$$C^{\boldsymbol{o}}_{o(j)}(S) = C^{\boldsymbol{o}}_{o(j)}(S^\triangle) \quad \text{for any} \quad j \in \{1, 2, \ldots, \beta_k\},$$

where

$$\begin{aligned}\beta_k &= \max\{j \in N | a_{o(1)} + \cdots + a_{o(j)} \le a_1 + \cdots + a_k\} \\ &= \min\{j \in N | a_{o(j+1)} + \cdots + a_{o(n)} \ge a_{k+1} + \cdots + a_n\} \\ &= \lambda_{k+1} - 1.\end{aligned}$$

Then $\boldsymbol{C}' = (C'_1, C'_2, \ldots, C'_n)$ define by

$$C'_{o(j)} = \begin{cases} C^{\boldsymbol{o}}_{o(j)}(S) & \text{for any} \quad j \in \{1, 2, \ldots, \lambda_{k+1} - 1\} \\ C^{\boldsymbol{o}}_{o(j)}(S) + L & \text{for any} \quad j \in \{\lambda_{k+1}, \ldots, n\} \end{cases}$$

is a feasible schedule for $S^{\triangle}$ since $\boldsymbol{C^o}(S)$ is feasible for S. We easily see that $\boldsymbol{C^o}(S^{\triangle}) \le \boldsymbol{C}'$ and $f_{S^{\triangle}}(\boldsymbol{C^o}(S^{\triangle})) \le f_{S^{\triangle}}(\boldsymbol{C}')$ from Algorithm 1. Then we have that

$$\begin{aligned}f_{S^{\triangle}}(\boldsymbol{C^o}(S^{\triangle})) - f_S(\boldsymbol{C^o}(S)) &\le f_{S^{\triangle}}(\boldsymbol{C}') - f_S(\boldsymbol{C^o}(S)) \\ &= \boldsymbol{a}^{\mathsf{T}}(\boldsymbol{C}' - \boldsymbol{C^o}(S)) \\ &= L \sum_{j=\lambda_{k+1}}^{n} a_{o(j)} \\ &= L A^{\boldsymbol{o}}_{\lambda_{k+1}} \\ &\le 2 L A^*_{k+1} \\ &= 2(f_{S^{\triangle}}(\boldsymbol{C}^*(S^{\triangle})) - f_S(\boldsymbol{C}^*(S))),\end{aligned}$$

where the second inequality follows from the assumption. Therefore if $f_S(\boldsymbol{C^o}(S)) < \alpha f_S(\boldsymbol{C}^*(S))$, then we have that

$$\begin{aligned}f_{S^{\triangle}}(\boldsymbol{C^o}(S^{\triangle})) \le& f_S(\boldsymbol{C^o}(S)) + 2(f_{S^{\triangle}}(\boldsymbol{C}^*(S^{\triangle})) - f_S(\boldsymbol{C}^*(S))) \\ <& \alpha f_{S^{\triangle}}(\boldsymbol{C}^*(S^{\triangle})).\end{aligned}$$

□

Proof of Theorem 5. Recall that $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, n, (\boldsymbol{C}' - \boldsymbol{p}), \boldsymbol{a}\rangle$ and $\bar{S} = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, n, (\boldsymbol{\sigma}(\boldsymbol{p}) - \boldsymbol{p}), \boldsymbol{a}\rangle$ in Theorem 5. We can express that

$$\begin{aligned}\boldsymbol{C}' - \boldsymbol{\sigma}(\boldsymbol{p}) =& (C'_1 - C'_0 - p_1)\boldsymbol{d}_0 + (C'_2 - C'_1 - p_2)\boldsymbol{d}_1 \\ &+ \cdots + (C'_n - C'_{n-1} - p_n)\boldsymbol{d}_{n-1} \\ =& \sum_{k=0}^{n-1} L_k \boldsymbol{d}_k,\end{aligned}$$

where $C'_0 = 0$, $\boldsymbol{d}_k = (\boldsymbol{0}_k^{\mathsf{T}} \ \boldsymbol{1}_{n-k}^{\mathsf{T}})^{\mathsf{T}}$, and $L_k = C'_{k+1} - C'_k - p_{k+1}$ for any $k \in \{0, 1, \ldots, n-1\}$. From the conditions in the theorem, we have that $L_k \ge 0$ for any $k \in \{0, 1, \ldots, n-1\}$. We define $\boldsymbol{D}_0 = \boldsymbol{\sigma}(\boldsymbol{p})$ and

$$\boldsymbol{D}_k = \boldsymbol{D}_{k-1} + L_{k-1}\boldsymbol{d}_{k-1}$$

for any $k \in N$. Then we see that

$$\boldsymbol{D}_n = \boldsymbol{C}'.$$

We also define $(n+1)$-instances $S^k = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, n, (\boldsymbol{D}_k - \boldsymbol{p}), \boldsymbol{a} \rangle$ for $k \in \{0, 1, \ldots, n\}$. Then we have that $S^0 = \bar{S}$ and $S^n = S$.

Now we consider two instances $S^0 = \bar{S}$ and S^1. If $f_{S^0}(\boldsymbol{C}^{\boldsymbol{o}}(S^0)) < \alpha f_{S^0}(\boldsymbol{C}^*(S^0))$, we have $f_{S^1}(\boldsymbol{C}^{\boldsymbol{o}}(S^1)) < \alpha f_{S^1}(\boldsymbol{C}^*(S^1))$ from Lemma 4. For $k = 2, \ldots, n$, we repeat this procedure for two instances S^{k-1} and S^k, then we finally obtain that $f_{S^n}(\boldsymbol{C}^{\boldsymbol{o}}(S^n)) < \alpha f_{S^n}(\boldsymbol{C}^*(S^n))$, where $S^n = S$.

□

Chapter 4

A 2-approximation algorithm for a single-machine scheduling problem with a non-renewable resource and unit processing times

This chapter proves a conjecture by Györgyi and Kis [19], in other words, we show that a list scheduling algorithm with a non-increasing order of resource consumption is a 2-approximation algorithm for NR-SSP with the total weighted completion time criterion, unit processing times ($\boldsymbol{p} = \mathbf{1}_n$), and weights proportional to resource consumption ($\boldsymbol{w} = \boldsymbol{a}$). This problem can be written as $1|nr = 1, p_j = 1, w_j = a_j|\sum w_j C_j$ by Graham's notation. Furthermore, we show the tightness of the approximation ratio. Section 4.1 introduces the conjecture and our main results. Section 4.2 proves the conjecture, and Section 4.3 shows the tightness of the ratio.

4.1 Main results

This section introduces our main results, and give an overview of its proof. We prove the following conjecture:

Conjecture 1 (Györgyi and Kis [19]). *List scheduling algorithm with non-increasing order of a_j is a 2-approximation algorithm for NR-SSP with the total weighted completion time criterion, $\boldsymbol{p} = \mathbf{1}_n$, and $\boldsymbol{w} = \boldsymbol{a}$ (or $1|nr = 1, p_j = 1, w_j = a_j|\sum w_j C_j$).*

Note that the conjecture is equivalent to Theorem 1. In addition to prove the conjecture, we verify that the approximation ratio is tight, and there are no instance that achieves the tight ratio.

Recall that any instance of the problem can be written as $\langle n, \mathcal{J}, \mathbf{1}_n, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b}\rangle$.

Theorem 6. *For any instance $S = \langle n, \mathcal{J}, \mathbf{1}_n, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b}\rangle$ of NR-SSP, $f_S(\boldsymbol{C}^{\boldsymbol{o}_a}(S)) <$*

$2f_S(\boldsymbol{C}^*(S))$ *holds.*

Theorem 7. *For any* $\epsilon > 0$*, there exists an instance* $S = \langle n, \mathcal{J}, \mathbf{1}_n, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b}\rangle$ *of NR-SSP such that* $f_S(\boldsymbol{C}^{\boldsymbol{o}_a}(\tilde{S})) \geq (2-\epsilon) f_S(\boldsymbol{C}^*(\tilde{S}))$*.*

Theorem 6 can prove Conjecture 1 or Theorem 1, and show that there are no instance that achieve the approximation ratio. Theorem 7 guarantees that the ratio is tight for the algorithm. To solve Theorem 6, we simplify the problem while preserving the upper bound of the approximation ratio by Theorem 3. Then we prove the theorem for the simplified problem. In the proof of Theorem 7, we establish a parameterized instance that proves the theorem.

Before the proofs, we define a notation used in the proof of Theorem 6 (and Chapter 5). For any instance S of the simplified problem can be written as $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, n, (\boldsymbol{\sigma}(\boldsymbol{p}) - \boldsymbol{p}), \boldsymbol{a}\rangle$. Then for any $j \in N$, we denote an NR-SSP instance obtained by removing every jth input from S by S_j, that is, $S_j = \langle n-1, \mathcal{J}_j, \boldsymbol{p}_j, \boldsymbol{a}_j, n-1, (\boldsymbol{\sigma}(\boldsymbol{p}_j) - \boldsymbol{p}_j), \boldsymbol{a}_j\rangle$ where $\mathcal{J}_j = (J_1, J_2, \dots, J_{j-1}, J_{j+1}, \dots, J_n)$, $\boldsymbol{p}_j = (p_1, p_2, \dots, p_{j-1}, p_{j+1}, \dots, p_n)^\mathsf{T}$, $\boldsymbol{a}_j = (a_1, a_2, \dots, a_{j-1}, a_{j+1}, \dots, a_n)^\mathsf{T}$.

4.2 Proof of Theorem 6

This section proves Theorem 6. First, we check that the list scheduling algorithm satisfies Assumption 4 for any instance of the problem.

Proposition 1. *Let* $S = \langle n, \mathcal{J}, \mathbf{1}_n, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b}\rangle$ *be any instance of NR-SSP. For any* a_j *non-increasing order* $\boldsymbol{o}_a$*, Assumption 4 holds, i.e.,* $A^{\boldsymbol{o}_a}_{\lambda_j} \leq 2A^*_j$ *for any* $j \in \{1, 2, \dots, n+1\}$*.*

Proof. If $j = n+1$, the proposition clearly holds. Suppose $j \in N$. Then we have $A^{\boldsymbol{o}_a}_{\lambda_j+1} < A^*_j$ from (3.1). This inequality implies that the set $\{J_j, J_{j+1}, \dots, J_n\}$ is not a subset of $\{J_{o_a(\lambda_j+1)}, J_{o_a(\lambda_j+2)}, \dots, J_{o_a(n)}\}$. Hence there exists $j' \geq j$ such that $J_{j'} \in \{J_{o_a(1)}, J_{o_a(2)}, \dots, J_{o_a(\lambda_j)}\}$. Since $\boldsymbol{o}_a$ is non-increasing order of a_j, we see that

$$A^*_j \geq a_{j'} \geq \min\{a_{o_a(1)}, a_{o_a(2)}, \dots, a_{o_a(\lambda_j)}\} = a_{o_a(\lambda_j)}.$$

Thus we have that

$$A^{\boldsymbol{o}_a}_{\lambda_j} = a_{o_a(\lambda_j)} + A^{\boldsymbol{o}_a}_{\lambda_j+1} \leq 2A^*_j.$$

□

Next, we show the lemma used in the proof of Theorem 6.

Lemma 5. *Let* $k \geq 2$ *be any integer. Suppose that Theorem 6 is true for* $n = k-1$*. Then Theorem 6 holds for any instance* $S = \langle k, \mathcal{J}, \mathbf{1}_n, \boldsymbol{a}, k, (\boldsymbol{n}_k - \mathbf{1}_k), \boldsymbol{a}\rangle$ *if there exists an index* $j' \in \{1, 2, \dots, k-1\}$ *such that*

$$2j' a_{j'} \geq f_S(\boldsymbol{C}^{\boldsymbol{o}_a}(S)) - f_{S_{j'}}(\boldsymbol{C}^{\boldsymbol{o}_a}(S_{j'})). \tag{4.1}$$

(Recall that $S_{j'}$ *is obtained by removing every* j'*th element from* S*.)*

Proof. From Lemma 2, the optimal schedule for $S_{j'}$ is $\boldsymbol{C}^*(S_{j'}) = \boldsymbol{n}_{k-1} + (\boldsymbol{0}_{j'-1}^\mathsf{T}\ \boldsymbol{1}_{k-j'}^\mathsf{T})^\mathsf{T}$. By applying Theorem 6 for $S_{j'}$, we have that

$$\begin{aligned} f_{S_{j'}}(\boldsymbol{C}^{\boldsymbol{o}_a}(S_{j'})) &< 2 f_{S_{j'}}(\boldsymbol{C}^*(S_{j'})) \\ &= 2\left(\sum_{j=1}^{j'-1} j a_j + \sum_{j=j'+1}^{k} j a_j\right). \end{aligned}$$

Hence, if (4.1) holds, we have that

$$\begin{aligned} f_S(\boldsymbol{C}^{\boldsymbol{o}_a}(S)) &\le 2j' a_{j'} + f_{S_{j'}}(\boldsymbol{C}^{\boldsymbol{o}_a}(S_{j'})) \\ &< 2\sum_{j=1}^{k} j a_j \\ &= 2 f_S(\boldsymbol{C}^*(S)), \end{aligned}$$

where $\boldsymbol{C}^*(S) = \boldsymbol{n}_k$ is the optimal schedule for S. □

Finally, we prove Theorem 6.

Proof of Theorem 6. We prove the result by induction on n. When $n = 1$, obviously $\boldsymbol{C}^{\boldsymbol{o}_a}(S) = \boldsymbol{C}^*(S)$. Hence the approximation ratio is 1 and the result holds.

Suppose that the result holds when $n = k - 1$ for $k \ge 2$. Consider the case $n = k$. By Theorem 3 and Proposition 1, it suffices to prove it for any instance $S = \langle k, \mathcal{J}, \boldsymbol{1}_n, \boldsymbol{a}, k, (\boldsymbol{n}_k - \boldsymbol{1}_k), \boldsymbol{a}\rangle$. From Lemma 2, the optimal schedule for S is $\boldsymbol{C}^*(S) = \boldsymbol{n}_k$.

If $\boldsymbol{C}^{\boldsymbol{o}_a}(S) = \boldsymbol{C}^*(S)$, then the result is obvious, so we assume that $\boldsymbol{C}^{\boldsymbol{o}_a}(S) \ne \boldsymbol{n}_k$. We also assume that all the elements of $\boldsymbol{C}^{\boldsymbol{o}_a}(S)$ are integer without loss of generality. Then there exists $l' \in \{1, 2, \ldots, k\}$ such that

$$C^{\boldsymbol{o}_a}_{o_a(l')}(S) \ge C^{\boldsymbol{o}_a}_{o_a(l'-1)}(S) + 2 \tag{4.2}$$

and

$$C^{\boldsymbol{o}_a}_{o_a(j)}(S) = C^{\boldsymbol{o}_a}_{o_a(j-1)}(S) + 1 \quad \text{for any} \quad j \in \{l'+1, \ldots, k\}, \tag{4.3}$$

where $C^{\boldsymbol{o}_a}_{o_a(0)}(S) = 0$ if $l' = 1$. Let $l = C^{\boldsymbol{o}_a}_{o_a(l')}(S)$.

By the definition of l, the process of the job $J_{o_a(l')}$ starts at time $l - 1$ by Algorithm 1. Hence we have that

$$\sum_{j=1}^{l-1} a_j < \sum_{j=1}^{l'} a_{o_a(j)} \le \sum_{j=1}^{l} a_j. \tag{4.4}$$

We also see that

$$a_{l-1} < a_{o_a(l')}, \tag{4.5}$$

otherwise Algorithm 1 starts the job $J_{o_a(l')}$ at time $l - 2$ by (4.2). However, the latter case violates the definition of l. From (4.3) we have that

$$C^{\boldsymbol{o}_a}_{o_a(j)}(S) = l + j - l' \quad \text{for any} \quad j \in \{l', l'+1, \ldots, k\}. \tag{4.6}$$

Define 4 subsets

$$\begin{aligned} P &:= \{J_1, J_2, \dots, J_{l-1}\},\\ P^c &:= \{J_l, J_{l+1}, \dots, J_k\},\\ Q &:= \{J_{o_a(1)}, J_{o_a(2)}, \dots, J_{o_a(l')}\},\\ Q^c &:= \{J_{o_a(l'+1)}, J_{o_a(l'+2)}, \dots, J_{o_a(k)}\}. \end{aligned}$$

Then we consider three cases: (I) $l' < k-l+1$, (Ⅱ) $l' > k-l+1$, and (Ⅲ) $l' = k-l+1$. Note that $l' = |Q|$ and $k-l+1 = |P^c|$.

(I) Assume that $l' < k-l+1$. Then the set P^c is not included in Q. Therefore we see that

$$P^c \cap Q^c \neq \emptyset.$$

We define

$$i = \arg\min\{a_l, a_{l+1}, \dots, a_k\}. \tag{4.7}$$

Obviously, $i \geq l$ and $J_i \in P^c$. Since $a_{o_a(j)}$ is non-increasing order, $P^c \cap Q^c \neq \emptyset$ implies $J_i \in P^c \cap Q^c$. Let $o_a(i') = i$, then (4.7) implies that

$$a_{o_a(i')} \leq a_j \quad \text{for any} \quad j \in \{l, l+1, \dots, k\}.$$

Define $S^i = \langle k, \mathcal{J}^i, \mathbf{1}_k, \boldsymbol{a}^i, k, (\boldsymbol{n}_k - \mathbf{1}_k), \boldsymbol{a}^i \rangle$ for $\mathcal{J}^i = (J_1, J_2, \dots, J_{i-1}, J_{i+1}, J_{i+2}, \dots, J_k, J_i)$ and $\boldsymbol{a}^i = (a_1, a_2, \dots, a_{i-1}, a_{i+1}, a_{i+2}, \dots, a_k, a_i)^\mathsf{T}$. Since $a_i \leq a_j$ for every $j \in \{i, i+1, \dots, k\}$, we have that

$$\boldsymbol{C}^{\boldsymbol{o}_a}(S^i) = \boldsymbol{C}^{\boldsymbol{o}_a}(S) \tag{4.8}$$

from Algorithm 1 and (4.6). We also have from Lemma 1 that

$$(\boldsymbol{a}^i)^\mathsf{T} \boldsymbol{n}_k \leq \boldsymbol{a}^\mathsf{T} \boldsymbol{n}_k. \tag{4.9}$$

Note that $\boldsymbol{n}_k$ is the optimal schedule for both S and S^i. Define S^i_k by removing every kth element from S^i. From Algorithm 1 for S^i and S^i_k, we have that

$$C^{\boldsymbol{o}_a}_{o_a(j)}(S^i_k) = \begin{cases} C^{\boldsymbol{o}_a}_{o_a(j)}(S^i) & (1 \leq j \leq i'-1) \\ C^{\boldsymbol{o}_a}_{o_a(j)}(S^i) - 1 & (i'+1 \leq j \leq k). \end{cases}$$

Hence we see that

$$\begin{aligned} f_{S^i}(\boldsymbol{C}^{\boldsymbol{o}_a}(S^i)) - f_{S^i_k}(\boldsymbol{C}^{\boldsymbol{o}_a}(S^i_k)) &= a_{o_a(i')} C^{\boldsymbol{o}_a}_{o_a(i')}(S^i) + \sum_{j=i'+1}^{k} a_{o_a(j)} \\ &\leq (l + i' - l') a_{o_a(i')} + \sum_{j=i'+1}^{k} a_{o_a(i')} \qquad (4.10) \\ &\leq (l - l' + k) a_i \\ &\leq 2k a_i, \end{aligned}$$

where (4.10) follows from (4.6). Since a_i is the kth element of $\boldsymbol{a}^i$, we have $2f_{S^i}(\boldsymbol{C}^*(S^i)) > f_{S^i}(\boldsymbol{C}^{\boldsymbol{o}_a}(S^i))$ from Lemma 5. Using (4.8) and (4.9), we obtain that

$$\begin{aligned}2f_S(\boldsymbol{C}^*(S)) &= 2\boldsymbol{a}^\mathsf{T}\boldsymbol{n}_k \\ &\geq 2(\boldsymbol{a}^i)^\mathsf{T}\boldsymbol{n}_k \\ &= 2f_{S^i}(\boldsymbol{C}^*(S^i)) \\ &> f_{S^i}(\boldsymbol{C}^{\boldsymbol{o}_a}(S^i)) \\ &= f_S(\boldsymbol{C}^{\boldsymbol{o}_a}(S)).\end{aligned}$$

(II) Assume that $l' > k - l + 1$. From (4.5), $a_{l-1} < a_{o_a(l')}$ holds. Hence there exists $\bar{l} \in \{l'+1, \dots, k\}$ such that $o_a(\bar{l}) = l - 1$. Define an instance S_{l-1} by removing every $(l-1)$th element from S. Then we see from Algorithm 1 that

$$\begin{array}{lll} C^{\boldsymbol{o}_a}_{o_a(j)}(S) \leq C^{\boldsymbol{o}_a}_{o_a(j)}(S_{l-1}) & \text{for} & j \in \{1, 2, \dots, \bar{l} - 1\}, \\ C^{\boldsymbol{o}_a}_{o_a(j)}(S) \leq C^{\boldsymbol{o}_a}_{o_a(j)}(S_{l-1}) + 1 & \text{for} & j \in \{\bar{l} + 1, \dots, k\}. \end{array}$$

Hence we obtain that

$$\begin{aligned} f_S(\boldsymbol{C}^{\boldsymbol{o}_a}(S)) - f_{S_{l-1}}(\boldsymbol{C}^{\boldsymbol{o}_a}(S_{l-1})) &\leq a_{o_a(\bar{l})}C^{\boldsymbol{o}_a}_{o_a(\bar{l})}(S) + \sum_{j=\bar{l}+1}^{k} a_{o_a(j)} \\ &\leq (l + \bar{l} - l')a_{o_a(\bar{l})} + \sum_{j=\bar{l}+1}^{k} a_{o_a(\bar{l})} \qquad (4.11) \\ &= (l - l' + k)a_{o_a(\bar{l})} \\ &\leq 2(l-1)a_{l-1}, \qquad (4.12) \end{aligned}$$

where (4.11) follows from (4.6) and (4.12) follows from $l' \geq k - l + 2$. Therefore we can apply Lemma 5, and consequently, the theorem holds for this case.

(III) Assume that $l' = k - l + 1$. If $P^c \cap Q^c \neq \emptyset$, we can prove the result as in the case of (I). So we assume that $P^c \cap Q^c = \emptyset$, which implies $Q \subseteq P^c$. Since $l' = k - l + 1$ is equivalent to $|Q| = |P^c|$, we have that $Q = P^c$. So we see that

$$\sum_{j=l}^{k} a_k = \sum_{j=1}^{l'} a_{o_a(j)} \qquad (4.13)$$

and

$$\begin{aligned}
f_S(\boldsymbol{C}^{\boldsymbol{o}_a}(S)) &= \sum_{j=1}^{k} a_{o_a(j)} C^{\boldsymbol{o}_a}_{o_a(j)}(S) \\
&= \sum_{j=1}^{l'} a_{o_a(j)} C^{\boldsymbol{o}_a}_{o_a(j)}(S) + \sum_{j=l'+1}^{k} a_{o_a(j)} C^{\boldsymbol{o}_a}_{o_a(j)}(S) \\
&= \sum_{j=1}^{l'} a_{o_a(j)} C^{\boldsymbol{o}_a}_{o_a(j)}(S) + l \sum_{j=l'+1}^{k} a_{o_a(j)} + \sum_{j=l'+1}^{k} a_{o_a(j)}(j-l'), \qquad (4.14)
\end{aligned}$$

where (4.14) follows from (4.6). On the other hand, we have that

$$\begin{aligned}
2f_S(\boldsymbol{C}^*(S)) &= 2\boldsymbol{a}^\mathsf{T}\boldsymbol{n}_k \\
&\geq 2\left(\sum_{j=1}^{l-1} a_j j + l\sum_{j=l}^{k} a_j\right) \\
&\geq \sum_{j=1}^{l-1} a_j j + l\sum_{j=l}^{k} a_j + l\sum_{j=l}^{k} a_j. \qquad (4.15)
\end{aligned}$$

Comparing the first term in (4.15) and the third term in (4.14), we have

$$\sum_{j=1}^{l-1} a_j j \geq \sum_{j=l'+1}^{k} a_{o_a(j)}(j-l'),$$

since $a_{o_a(j)}$ are non-increasing order and $k - l' = l - 1$. Comparing the second term in (4.15) and the second term in (4.14), we have

$$l\sum_{j=l}^{k} a_j > l\sum_{j=l'+1}^{k} a_{o_a(j)}$$

from (4.4). Comparing the third term in (4.15) and the first term in (4.14), we have

$$l\sum_{j=l}^{k} a_j = l\sum_{j=1}^{l'} a_{o_a(j)} \geq \sum_{j=1}^{l'} a_{o_a(j)} C^{\boldsymbol{o}_a}_{o_a(j)}(S)$$

from (4.13), since we have $l \geq C^{\boldsymbol{o}_a}_{o_a(j)}(S)$ for every $j \in \{1, 2, \dots, l'\}$ from the definition of l. Hence we obtain that

$$2f_S(\boldsymbol{C}^*(S)) \geq f_S(\boldsymbol{C}^{\boldsymbol{o}_a}(S)).$$

□

4.3 Proof of Theorem 7

This section gives an instance that can solve Theorem 7.
Proof of Theorem 7. Let $S = \langle n, \mathcal{J}, \mathbf{1}_n, \boldsymbol{a}, n, (\boldsymbol{n}_n - \mathbf{1}_n), \boldsymbol{a}\rangle$ for $\boldsymbol{a} = (2^n, 1, 1, \ldots, 1, 2^n + n)^\mathsf{T}$. We easily see that

$$\begin{aligned}
&\boldsymbol{C}^*(S) = (1, 2, \ldots, n)^\mathsf{T}, \\
&\boldsymbol{o}_a = (n, 1, 2, \ldots, n-1), \\
&\boldsymbol{C}^{\boldsymbol{o}_a}(S) = (n+1, n+2, \ldots, 2n-1, n)^\mathsf{T}.
\end{aligned}$$

Then

$$\frac{f_S(\boldsymbol{C}^{\boldsymbol{o}_a}(S))}{f_S(\boldsymbol{C}^*(S))} = \frac{(n+1)2^n + \sum_{j=n+2}^{2n-1} j + n(2^n + n)}{2^n + \sum_{j=2}^{n-1} j + n(2^n + n)} \to 2$$

as $n \to \infty$.
Hence we obtain the result. □

Note that Györgyi and Kis [19] show that the instance $S = \langle 2, (J_1, J_2), (w - e, w), 2, (0, w), (w - e, w)\rangle$ also achieves approximation ratio 2 as $e \to +0$.

Chapter 5

A 3-approximation algorithm for a single-machine scheduling problem with a non-renewable resource

This chapter establishes a 3-approximation list scheduling algorithm for NR-SSP with the total weighted completion time criterion and weights proportional to resource consumption ($\boldsymbol{w} = \boldsymbol{a}$). This problem can be written as $1|nr = 1, w_j = a_j| \sum w_j C_j$ by Graham's notation. Furthermore, we show the tightness of the approximation ratio, and prove that no instance achieves the approximation ratio 3 exactly. Section 5.1 explains our algorithm and results. Section 5.2 provides an additional simplification for the problem other than Theorem 3. Section 5.3 shows proofs of the results.

5.1 Algorithm and main results

This section deals with some permutations defined as follows:

Definition 1. *For $\boldsymbol{a}$ and $\boldsymbol{p}$ in any instance $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b} \rangle$, we define a set $O(\boldsymbol{a}, \boldsymbol{p}) \subset \mathcal{O}$ of permutations $\boldsymbol{o}$ satisfying the following conditions:*

(i) $a_{o(n)} = \min_{k \in N} a_k$,

(ii) for any $j \in \{1, 2, \ldots, n-1\}$, if $a_{o(j)} > A^{\boldsymbol{o}}_{j+1}$, then

$$a_{o(i)} \geq a_{o(j)} \textit{ for any } i \in \{1, 2, \ldots, j\},$$

(iii) for any $j \in \{1, 2, \ldots, n-1\}$, if $a_{o(j)} \leq A^{\boldsymbol{o}}_{j+1}$, then

$$r_{o(i)} \geq r_{o(j)} \textit{ for any } i \in \{1, 2, \ldots, j\} \textit{ such that } a_{o(i)} \leq A^{\boldsymbol{o}}_{j+1}.$$

Recall that $A^{\boldsymbol{o}}_{j+1} = \sum_{h=j}^{n} a_{o(h)}$. The next algorithm calculates a permutation that satisfies Definition 1.

Algorithm 2 A job-ordering algorithm for NR-SSP

Input: $\boldsymbol{a}, \boldsymbol{p}$ in any instance $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b}\rangle$ of NR-SSP
Output: $\boldsymbol{o} = (o(1), o(2), \ldots, o(n)) \in \mathcal{O}$

```
 1: Initialize:
        $SUMW \leftarrow 0$
        $N^{REST} \leftarrow N$
 2: for $i = n$ to 1 do                                        ▷ Inverse order
 3:     $N' \leftarrow \{j \in N^{REST} | a_j \leq SUMW\}$
 4:     if $N' = \emptyset$ then
 5:         $o(i) \leftarrow \underset{j \in N^{REST}}{\arg\min}\, a_j$( ties are broken arbitrarily)
 6:     else
 7:         $o(i) \leftarrow \underset{j \in N'}{\arg\min}\, r_j$( ties are broken arbitrarily)
 8:     end if
 9:     $SUMW \leftarrow SUMW + a_{o(i)}$
10:     $N^{REST} \leftarrow N^{REST} \setminus \{o(i)\}$
11: end for
```

The time complexity of this algorithm is $O(n \log n)$ by using Red-black tree [1] as two containers of jobs sorted by a_j and r_j. Thus, the whole time complexity of the list scheduling algorithm is $O(\max\{n \log n, q \log q\})$. Algorithm 2 is motivated by the 6-approximation list scheduling algorithm for NR-SSP with the total weighted completion time criterion and zero processing times, or $1|nr = 1, p_j = 0| \sum w_j C_j$ [4]. Their algorithm uses "arg min w_j/a_j" in the line 7 of Algorithm 2 instead of "arg min r_j"(= arg min w_j/p_j).

The main results in this section are the following two theorems:

Theorem 8. *For any instance $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b}\rangle$ of NR-SSP and any permutation $\boldsymbol{o} \in O(\boldsymbol{a}, \boldsymbol{p})$, $f_S(\boldsymbol{C}^{\boldsymbol{o}}(S)) < 3 f_S(\boldsymbol{C}^*(S))$ holds.*

Theorem 9. *For any $\epsilon > 0$, there exists an instance $\tilde{S} = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b}\rangle$ of NR-SSP and an $\boldsymbol{o} \in O(\boldsymbol{a}, \boldsymbol{p})$ such that $f_S(\boldsymbol{C}^{\boldsymbol{o}}(\tilde{S})) \geq (3 - \epsilon) f_S(\boldsymbol{C}^*(\tilde{S}))$.*

Theorem 8 states that a list scheduling algorithm with any order calculated by Algorithm 2 is a 3-approximation algorithm for NR-SSP with the total weighted completion time criterion and weights proportional to resource consumption, which implies Theorem 2. Moreover, the theorem proves that no instance can achieve approximation ratio 3 exactly. Besides, Theorem 9 shows that the approximation ratio is tight.

5.2 Additional problem simplification

This section shows the following theorem, which provides an additional simplification for the NR-SSP other than Theorem 3.

Theorem 10. *Let $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, n, (\boldsymbol{\sigma}(\boldsymbol{p}) - \boldsymbol{p}), \boldsymbol{a}\rangle$. Define $S^\dagger = \langle n, \mathcal{J}, \boldsymbol{a}, \boldsymbol{a}, n, (\boldsymbol{\sigma}(\boldsymbol{a}) -$*

$\boldsymbol{a}), \boldsymbol{a}\rangle$. Then $O(\boldsymbol{a}, \boldsymbol{p}) \subset O(\boldsymbol{a}, \boldsymbol{a})$ holds, and for any $\boldsymbol{o} \in O(\boldsymbol{a}, \boldsymbol{p})$, $f_{S^\dagger}(\boldsymbol{C}^{\boldsymbol{o}}(S^\dagger)) < 3f_{S^\dagger}(\boldsymbol{C}^*(S^\dagger))$ implies $f_S(\boldsymbol{C}^{\boldsymbol{o}}(S)) < 3f_S(\boldsymbol{C}^*(S))$.

Recall that Theorem 3 can fix $q = n$, $\boldsymbol{u} = \boldsymbol{\sigma}(p) - \boldsymbol{p}$, and $\boldsymbol{b} = \boldsymbol{a}$ for any instance $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b}\rangle$ when we prove Theorem 8. Theorem 10 provides an additional input restriction $\boldsymbol{p} = \boldsymbol{a}$ in the proof.

First, we verify that the list scheduling algorithm with any order calculated by Algorithm 2 satisfies Assumption 4, which allows us to apply Theorem 3 for the proof.

Proposition 2. *Let $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b}\rangle$ be any instance of NR-SSP and $\boldsymbol{o} \in O(\boldsymbol{a}, \boldsymbol{p})$, then $A^{\boldsymbol{o}}_{\lambda_j} \le 2A^*_j$ for any $j \in \{1, 2, \ldots, n+1\}$.*

Proof. If $j = n + 1$, then $A^{\boldsymbol{o}}_{\lambda_{n+1}} = 2A^*_{n+1} = 0$. Thus we assume that $j \in N$. From the definition of λ_j,

$$A^{\boldsymbol{o}}_{\lambda_j+1} < A^*_j = \sum_{k=j}^{n} a_k. \tag{5.1}$$

If $a_{o(\lambda_j)} \le A^{\boldsymbol{o}}_{\lambda_j+1}$, then $A^{\boldsymbol{o}}_{\lambda_j} = a_{o(\lambda_j)} + A^{\boldsymbol{o}}_{\lambda_j+1} < 2A^*_j$; therefore we assume $a_{o(\lambda_j)} > A^{\boldsymbol{o}}_{\lambda_j+1}$. From Definition 1, we see $a_{o(\lambda_j)} \le a_{o(l)}$ for any $l \in \{1, 2, \ldots, \lambda_j\}$. The inequality (5.1) implies that there is a term $a_{o(i')}$ that is not in $A^{\boldsymbol{o}}_{\lambda_j+1} = \sum_{i=\lambda_j+1}^{n} a_{o(i)}$ but in $A^*_j = \sum_{k=j}^{n} a_k$. Hence we see $a_{o(\lambda_j)} \le a_{o(i')} \le A^*_j$. From this inequality and (5.1), we obtain $A^{\boldsymbol{o}}_{\lambda_j} < 2A^*_j$. □

Then we prove Theorem 10 using the next proposition.

Proposition 3. *Let $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, q, \boldsymbol{u}, \boldsymbol{b}\rangle$ be any instance of NR-SSP and $\boldsymbol{o} \in O(\boldsymbol{a}, \boldsymbol{p})$. For any $k, l \in N$,*

(i) if $r_{o(k)} \ge r_{o(l)}$, then $a_{o(l)}(p_{o(k)} - a_{o(k)}) \le a_{o(k)}(p_{o(l)} - a_{o(l)})$,

(ii) if $k < l$ and $r_{o(k)} < r_{o(l)}$, then $a_{o(k)} \ge a_{o(l)}$ and $a_{o(k)} \ge A^{\boldsymbol{o}}_{l+1}$.

Proof.

(i) For any $k, l \in N$, if $r_{o(k)} \ge r_{o(l)}$, then we have that

$$\begin{aligned} \frac{a_{o(l)}(p_{o(k)} - a_{o(k)})}{p_{o(l)}p_{o(k)}} &= r_{o(l)}(1 - r_{o(k)}) \\ &\le r_{o(k)}(1 - r_{o(l)}) \\ &= \frac{a_{o(k)}(p_{o(l)} - a_{o(l)})}{p_{o(l)}p_{o(k)}}. \end{aligned}$$

(ii) Suppose that $k < l$ and $r_{o(k)} < r_{o(l)}$ for $k, l \in N$. If $a_{o(l)} > A^{\boldsymbol{o}}_{l+1}$, then we have from Definition 1-(ii) that

$$a_{o(k)} \ge a_{o(l)} > A^{\boldsymbol{o}}_{l+1}.$$

Otherwise, $a_{o(l)} \le A^{\boldsymbol{o}}_{l+1}$. If $a_{o(k)} \le A^{\boldsymbol{o}}_{l+1}$, we see that $r_{o(k)} \ge r_{o(l)}$ from Definition 1-(iii), which contradicts to the assumption $r_{o(k)} < r_{o(l)}$. Hence we have that

$$a_{o(k)} > A^{\boldsymbol{o}}_{l+1} \ge a_{o(l)}.$$

□

Proof of Theorem 10. Recall that $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, n, (\boldsymbol{\sigma}(\boldsymbol{p}) - \boldsymbol{p}), \boldsymbol{a}\rangle$ and $S^\dagger = \langle n, \mathcal{J}, \boldsymbol{a}, \boldsymbol{a}, n, (\boldsymbol{\sigma}(\boldsymbol{a}) - \boldsymbol{a}), \boldsymbol{a}\rangle$. We define $\boldsymbol{\delta} = (\delta_1, \delta_2, \dots, \delta_n)^\mathsf{T}$ by

$$\delta_j = p_j - a_j \text{ for each } j \in N.$$

From Assumption 2, $\delta_j \geq 0$ for any $j \in N$. Then we easily see that $O(\boldsymbol{a}, \boldsymbol{p}) \subset O(\boldsymbol{a}, \boldsymbol{a})$. For any $\boldsymbol{o} \in O(\boldsymbol{a}, \boldsymbol{p})$, we need to prove

$$f_{S^\dagger}(\boldsymbol{C^o}(S^\dagger)) < 3 f_{S^\dagger}(\boldsymbol{C}^*(S^\dagger)) \Rightarrow f_S(\boldsymbol{C^o}(S)) < 3 f_S(\boldsymbol{C}^*(S)). \tag{5.2}$$

Define $D^*(S) = f_S(\boldsymbol{C}^*(S)) - f_{S^\dagger}(\boldsymbol{C}^*(S^\dagger))$ and $D^{\boldsymbol{o}}(S) = f_S(\boldsymbol{C^o}(S)) - f_{S^\dagger}(\boldsymbol{C^o}(S^\dagger))$. To prove (5.2), it suffices to show that

$$D^{\boldsymbol{o}}(S) \leq 3D^*(S). \tag{5.3}$$

Since the weight vector $\boldsymbol{w} = \boldsymbol{a}$ is common both in S and $S^\dagger$, we have that

$$\begin{aligned} D^*(S) &= \boldsymbol{a}^\mathsf{T}(\boldsymbol{C}^*(S) - \boldsymbol{C}^*(S^\dagger)), \\ D^{\boldsymbol{o}}(S) &= \boldsymbol{a}^\mathsf{T}(\boldsymbol{C^o}(S) - \boldsymbol{C^o}(S^\dagger)). \end{aligned}$$

From Lemma 2, $\boldsymbol{C}^*(S) = \boldsymbol{\sigma}(\boldsymbol{p})$ and $\boldsymbol{C}^*(S^\dagger) = \boldsymbol{\sigma}(\boldsymbol{a})$. Then we see that

$$\begin{aligned} D^*(S) &= \boldsymbol{a}^\mathsf{T}(\boldsymbol{\sigma}(\boldsymbol{p}) - \boldsymbol{\sigma}(\boldsymbol{a})) \\ &= \sum_{j=1}^{n} a_j \sum_{k=1}^{j} \delta_k \\ &= \sum_{k=1}^{n} \delta_k \sum_{j=k}^{n} a_j \\ &= \sum_{k=1}^{n} \delta_{o(k)} \sum_{j=o(k)}^{n} a_j \\ &= \sum_{k=1}^{n} A^*_{o(k)} \delta_{o(k)}. \end{aligned} \tag{5.4}$$

Next we evaluate $D^{\boldsymbol{o}}(S)$. Comparing $S^\dagger$ with S, the processing time vector and the supply date vector are different. Let $k \in N$. When a_k increases by δ_k, j-th element of the supply date vector $\boldsymbol{u} = \boldsymbol{\sigma}(\boldsymbol{a}) - \boldsymbol{a}$ increases by δ_k for each $j \in \{k+1, \dots, n\}$. Then $C^{\boldsymbol{o}}_{o(1)}, \dots, C^{\boldsymbol{o}}_{o(\alpha_k)}$ do not change and $C^{\boldsymbol{o}}_{o(\alpha_k+1)}, \dots, C^{\boldsymbol{o}}_{o(n)}$ may increase at most δ_k, where

$$\begin{aligned} \alpha_k &= \max\{j \in N | a_{o(1)} + \cdots + a_{o(j)} \leq a_1 + \cdots + a_k\} \\ &= \lambda_{k+1} - 1. \end{aligned}$$

Besides, when $o(k)$-th element of processing time vector increases by $\delta_{o(k)}$, not only $C^{\boldsymbol{o}}_{o(k)}$ but also $C^{\boldsymbol{o}}_{o(k+1)}, \ldots, C^{\boldsymbol{o}}_{o(n)}$ may increase at most $\delta_{o(k)}$. From these facts, we have that

$$\begin{aligned}
D^{\boldsymbol{o}}(S) &\leq \sum_{k=1}^{n} \delta_k \sum_{l=\alpha_k+1}^{n} a_{o(l)} + \sum_{k=1}^{n} \delta_{o(k)} \sum_{l=k}^{n} a_{o(l)} \\
&= \sum_{k=1}^{n} A^{\boldsymbol{o}}_{\alpha_k+1} \delta_k + \sum_{k=1}^{n} A^{\boldsymbol{o}}_k \delta_{o(k)} \\
&= \sum_{k=1}^{n} A^{\boldsymbol{o}}_{\lambda_{k+1}} \delta_k + \sum_{k=1}^{n} A^{\boldsymbol{o}}_k \delta_{o(k)} \\
&= \sum_{k=1}^{n} A^{\boldsymbol{o}}_{\lambda_{o(k)+1}} \delta_{o(k)} + \sum_{k=1}^{n} A^{\boldsymbol{o}}_k \delta_{o(k)} \\
&\leq \sum_{k=1}^{n} \left(2A^{*}_{o(k)+1} + A^{\boldsymbol{o}}_k \right) \delta_{o(k)},
\end{aligned} \tag{5.5}$$

where the last inequality follows from Theorem 2. Comparing (5.5) with (5.4), the inequality

$$\sum_{k=1}^{n} A^{\boldsymbol{o}}_k \delta_{o(k)} \leq \sum_{k=1}^{n} \left(2a_{o(k)} + A^{*}_{o(k)} \right) \delta_{o(k)},$$

implies (5.3). The inequality above is proved in Lemma 6 below. □

Lemma 6. *For any instance $S = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, n, (\boldsymbol{\sigma}(\boldsymbol{p}) - \boldsymbol{p}), \boldsymbol{a} \rangle$ and $\boldsymbol{o} \in O(\boldsymbol{p}, \boldsymbol{a})$, we define*

$$M^{*}(S) = \sum_{k=1}^{n} \left(2a_{o(k)} + A^{*}_{o(k)} \right) \delta_{o(k)}$$

and

$$M^{\boldsymbol{o}}(S) = \sum_{k=1}^{n} A^{\boldsymbol{o}}_k \delta_{o(k)},$$

where $\delta_j = p_j - a_j$ for each $j \in N$. Then we have that

$$M^{\boldsymbol{o}}(S) \leq M^{*}(S). \tag{5.6}$$

Proof. We show (5.6) by induction on n. In the case $n = 1$, the result is trivial. Then we prove (5.6) for S, $\boldsymbol{o}$, and $n \geq 2$ under the assumption that (5.6) holds for $n - 1$.

We define $S^{-} = S_{o(1)}$. Recall that $S_{o(1)}$ is an NR-SSP obtained by removing every $o(1)$th input from S, namely, $S_{o(1)} = \langle n-1, \mathcal{J}^{-}, \boldsymbol{p}^{-}, \boldsymbol{a}^{-}, n-1, (\boldsymbol{\sigma}(\boldsymbol{p}^{-}) - \boldsymbol{p}^{-}), \boldsymbol{a}^{-} \rangle$ such that $\mathcal{J}^{-} = (J_1, J_2, \ldots, J_{o(1)-1}, J_{o(1)+1}, \ldots, J_n)$, $\boldsymbol{p}^{-} = (p_1, p_2, \ldots, p_{o(1)-1}, p_{o(1)+1}, \ldots, p_n)^{\mathsf{T}}$, and $\boldsymbol{a}^{-} = (a_1, a_2, \ldots, a_{o(1)-1}, a_{o(1)+1}, \ldots, a_n)^{\mathsf{T}}$. We also define $\boldsymbol{o}^{-} = (o(2), o(3), \ldots, o(n))$,

$\Delta^* = M^*(S) - M^*(S^-)$, and $\Delta^{\boldsymbol{o}} = M^{\boldsymbol{o}}(S) - M^{\boldsymbol{o}^-}(S^-)$. It is obvious that $\boldsymbol{o}^-$ satisfies the conditions in Definition 1. Under the assumption $M^{\boldsymbol{o}^-}(S^-) \le M^*(S^-)$, we need to show

$$\Delta^{\boldsymbol{o}} \le \Delta^* \tag{5.7}$$

to prove (5.6) for n.

Next we show that (5.7) holds. We have that

$$M^*(S^-) = \sum_{k=2}^{n} \left(2a_{o(k)} + \sum_{o(k) \le j \le n, j \ne o(1)} a_j \right) \delta_{o(k)}$$

and

$$\begin{aligned}\Delta^* &= M^*(S) - M^*(S^-) \\ &= 2a_{o(1)}\delta_{o(1)} + A^*_{o(1)}\delta_{o(1)} + \sum_{o(k) < o(1)} a_{o(1)}\delta_{o(k)}.\end{aligned}$$

We also have that

$$M^{\boldsymbol{o}^-}(S^-) = \sum_{k=2}^{n} \delta_{o(k)} \sum_{j=k}^{n} a_{o(j)}$$

and

$$\begin{aligned}\Delta^{\boldsymbol{o}} &= M^{\boldsymbol{o}}(S) - M^{\boldsymbol{o}^-}(S^-) \\ &= \sum_{j=1}^{n} a_{o(j)}\delta_{o(1)} \\ &= \sum_{j=1}^{j^*-1} a_{o(j)}\delta_{o(1)} + a_{o(j^*)}\delta_{o(1)} + A^{\boldsymbol{o}}_{j^*+1}\delta_{o(1)},\end{aligned}$$

where

$$j^* = \min\{j \in N | r_{o(j)} > r_{o(1)}\}.$$

(In the case of $r_{o(j)} \le r_{o(1)}$ for any $j \in N$, we define $j^* = n + 1$ and $a_{o(j^*)} = 0$.) Since $r_{o(j)} \le r_{o(1)}$ for any $j < j^*$, we have from Theorem 3-(i) that

$$\begin{aligned}\sum_{j=1}^{j^*-1} a_{o(j)}\delta_{o(1)} &= \sum_{1 \le j \le j^*-1, o(j) \ge o(1)} a_{o(j)}\delta_{o(1)} + \sum_{1 \le j \le j^*-1, o(j) < o(1)} a_{o(j)}\delta_{o(1)} \\ &\le \sum_{1 \le j \le j^*-1, o(j) \ge o(1)} a_{o(j)}\delta_{o(1)} + \sum_{1 \le j \le j^*-1, o(j) < o(1)} a_{o(1)}\delta_{o(j)} \\ &\le A^*_{o(1)}\delta_{o(1)} + \sum_{o(k) < o(1)} a_{o(1)}\delta_{o(k)}.\end{aligned}$$

Since $j^* > 1$ and $r_{o(j^*)} > r_{o(1)}$, we have from Theorem 3-(ii) that

$$a_{o(j^*)}\delta_{o(1)} + A^{\boldsymbol{o}}_{j^*+1}\delta_{o(1)} \leq 2a_{o(1)}\delta_{o(1)}.$$

Hence $\Delta^{\boldsymbol{o}} \leq \Delta^*$ holds. □

5.3 Proofs

This section shows that Theorems 8 and 9.

Proof of Theorem 8. By Theorems 3 and 10, it suffices to prove it for any instance $S = \langle n, \mathcal{J}, \boldsymbol{a}, \boldsymbol{a}, n, (\boldsymbol{\sigma}(\boldsymbol{a}) - \boldsymbol{a}), \boldsymbol{a}\rangle$ and $\boldsymbol{o} \in O(\boldsymbol{a}, \boldsymbol{p})$. From Lemma 2, the optimal schedule for S is $\boldsymbol{C}^*(S) = \boldsymbol{\sigma}(\boldsymbol{a})$. Define $Q = \sum_{j=1}^{n} a_j$. Then we see that

$$\begin{aligned} f_S(\boldsymbol{C}^*(S)) &= \boldsymbol{a}^{\mathsf{T}}\boldsymbol{\sigma}(\boldsymbol{a}) \\ &= \sum_{j=1}^{n} a_j \sum_{i=1}^{j} a_i \\ &= \frac{1}{2}Q^2 + \frac{1}{2}\sum_{j=1}^{n} a_j^2. \end{aligned}$$

Since all the resources are available after the time $t = Q - a_{o(n)}$, we have that

$$C^{\boldsymbol{o}}_{o(j)}(S) \leq Q - a_{o(n)} + \sum_{i=1}^{j} a_{o(i)} < Q + \sum_{i=1}^{j} a_{o(i)}$$

for each $j \in N$. Thus we see that

$$\begin{aligned} f_S(\boldsymbol{C}^{\boldsymbol{o}}(S)) &= \sum_{j=1}^{n} a_{o(j)} C^{\boldsymbol{o}}_{o(j)}(S) \\ &< \sum_{j=1}^{n} a_{o(j)} \left(Q + \sum_{i=1}^{j} a_{o(i)} \right) \\ &= Q^2 + \sum_{j=1}^{n} a_{o(j)} \sum_{i=1}^{j} a_{o(i)} \\ &= Q^2 + \frac{1}{2}Q^2 + \frac{1}{2}\sum_{j=1}^{n} a_j^2 \\ &< 3f_S(\boldsymbol{C}^*(S)). \end{aligned}$$

□

Proof of Theorem 9. Let $0 < e < 0.1$ and $\tilde{S} = \langle n, \mathcal{J}, \boldsymbol{p}, \boldsymbol{a}, n, (\boldsymbol{\sigma}(\boldsymbol{p}) - \boldsymbol{p}), \boldsymbol{a} \rangle$ for $n = 3$, $\boldsymbol{p} = (e, e, 1)^\mathsf{T}$, and $\boldsymbol{a} = (1 - e, 1, 1 + e)^\mathsf{T}$. We easily see that

$$\begin{aligned}
&\boldsymbol{C}^*(\tilde{S}) = \boldsymbol{\sigma}(\boldsymbol{p}) = (e, 2e, 1 + 2e)^\mathsf{T}, \\
&\boldsymbol{o} = (3, 2, 1), \\
&\boldsymbol{C}^{\boldsymbol{o}}(\tilde{S}) = (1 + 3e, 1 + 2e, 1 + e)^\mathsf{T}.
\end{aligned}$$

Then

$$\begin{aligned}
&\lim_{e \to 0^+} \frac{f_{\tilde{S}}(\boldsymbol{C}^{\boldsymbol{o}}(\tilde{S}))}{f_{\tilde{S}}(\boldsymbol{C}^*(\tilde{S}))} \\
&= \lim_{e \to 0^+} \frac{(1 - e)(1 + 3e) + (1 + 2e) + (1 + e)(1 + e)}{(1 - e)e + 2e + (1 + e)(1 + 2e)} \\
&= 3.
\end{aligned}$$

Hence we obtain the result. □

Chapter 6

Conclusion

This thesis addresses Single-machine Scheduling Problem with a Non-renewable Resource(NR-SSP) with the total weighted completion time criterion. This problem is known to be NP-hard [10], and no constant-factor approximation algorithm has been found so far. This thesis develops (i) a simplification technique that preserves approximation ratios of some list scheduling algorithms for the NR-SSP with an input limitation, weights proportional to resource consumption ($\boldsymbol{w} = \boldsymbol{a}$). Then we show approximation results for

(ii) The NR-SSP with unit processing times($\boldsymbol{p} = \mathbf{1}_n$) and $\boldsymbol{w} = \boldsymbol{a}$,
(iii) The NR-SSP with $\boldsymbol{w} = \boldsymbol{a}$,

by applying the technique. These problems are also known to be NP-hard [4, 18].

(i) We develop a simplification technique for NR-SSP with the total weighted completion time criterion and the input limitation $\boldsymbol{w} = \boldsymbol{a}$. This simplification allows us to fix all the inputs of supply plans, the number of supplies q, supply dates $\boldsymbol{u}$, and supply amounts $\boldsymbol{b}$, when we prove some approximabilities of list scheduling algorithms.

(ii) We show that a list scheduling algorithm with non-increasing order of resource consumption is a 2-approximation algorithm for the problem, which Györgyi and Kis [19] conjecture. Besides, we verify that the approximation ratio is tight for the algorithm, and no instance can achieve the approximation ratio precisely.

(iii) We establish a 3-approximation list scheduling algorithm for the problem. This is the first (constant-factor) approximation algorithm for the problem. Similar to (ii), we also prove that the approximation ratio is tight for the algorithm, and no instance can achieve the ratio.

Appendix A

Computational remarks

In this chapter, we compare the performance of the algorithms with other simple list scheduling algorithms by numerical experiments with random NR-SSP instances. We show that (i) a list scheduling algorithm with a_j non-increasing order is a 2-approximation algorithm for NR-SSP with $\boldsymbol{w} = \boldsymbol{a}$ and $\boldsymbol{p} = \mathbf{1}_n$ and (ii) a list scheduling algorithm with any order calculated by Algorithm 2 is a 3-approximation algorithm for NR-SSP with $\boldsymbol{w} = \boldsymbol{a}$. However, these results only guarantee the worst performances. Thus, we verify the performances of the algorithms by numerical experiments. We only investigate random instances with $q = n$, $\boldsymbol{u} = \boldsymbol{\sigma}(\boldsymbol{p}) - \boldsymbol{p}$, and $\boldsymbol{b} = \boldsymbol{a}$ because we can find an optimal schedule by Theorem 4.

We consider six scenarios of the random instances. For the job number n, we experiment with three patterns $n \in \{10, 30, 100\}$. For the processing times, we take into account two cases (a) unit processing times ($\boldsymbol{p} = \mathbf{1}_n$) and (b) uniformly distributed from $\{1, 2, \dots, 1000\}^n$. For each instance, $\boldsymbol{a}$ is uniformly distributed from $\{1, 2, \dots, 1000\}^n$. We generate 1,000,000 instances for each scenario. Then we compare the approximation ratio of the instances (=[objective value]/[optimal value]) with list scheduling algorithms.

For the case (a), we experiment with the following list scheduling algorithms:

- w_j non-increasing order (2-approximation algorithm),
- w_j non-decreasing order,
- random order.

Note that Algorithm 2 always outputs a w_j non-increasing order for this case. For the case (b), we employ list scheduling algorithms with the following orders:

- Algorithm 2,
- $\frac{w_j}{p_j}(= r_j)$ non-increasing order,
- $\frac{w_j}{p_j}(= r_j)$ non-decreasing order,
- w_j non-increasing order,
- w_j non-decreasing order,
- p_j non-increasing order,
- p_j non-decreasing order,
- random order.

Table A.1. Computational results of list scheduling algorithms for random instances with unit processing times. The values denoted by bold letter indicate the best one among the three algorithms.

Order	average	min	max	stdev
(n=10, unit processing times)				
w_j non-increasing order (2-approximation algorithm)	**1.1773**	1.0062	**1.6260**	**0.0486**
w_j non-decreasing order	1.2957	**1.0004**	3.0023	0.1573
random	1.2345	1.0045	2.4442	0.0937
(n=30, unit processing times)				
w_j non-increasing order (2-approximation algorithm)	**1.0848**	**1.0302**	**1.1879**	**0.0159**
w_j non-decreasing order	1.3197	1.0542	2.0985	0.0919
random	1.1414	1.0321	1.7198	0.0485
(n=100, unit processing times)				
w_j non-increasing order (2-approximation algorithm)	**1.0440**	**1.0212**	**1.0899**	**0.0072**
w_j non-decreasing order	1.3289	1.1481	1.6489	0.0505
random	1.0775	1.0225	1.2859	0.0249

In Tables A.1 and A.2, the second column (”min”) shows the best approximation ratios, the third column (”max”) the worst approximation ratios, the fourth column (”average”) the average of the approximation ratios, and the fifth column (”stdev”) the standard deviation of the approximation ratios.

From Table A.1, w_j non-increasing order is the best except ”min” in $n = 10$, and w_j non-decreasing order is the worst except ”min” in $n = 10$. This shows that list scheduling algorithm with w_j non-increasing order is effective for this situation. From Table A.2, w_j/p_j non-increasing order or w_j non-increasing order is the best except ”max” in $n = 10$. Besides, Algorithm 2 always better than the other list scheduling algorithms except ”min” in $n = 100$. Recall that Algorithm 2 is a combination of w_j/p_j non-increasing order or w_j non-increasing order. The result shows that Algorithm 2 obtains the performance guarantee by slightly reducing the performance on average compared to the two list scheduling algorithms.

Table A.2. Computational results of list scheduling algorithms for random instances. The values denoted by bold letter indicate the best one among the eight algorithms.

Order	average	min	max	stdev
(n=10)				
Algorithm 2 (3-approximation algorithm)	1.2222	1.0005	**1.6266**	0.0596
w_j/p_j non-increasing order	**1.2102**	**1.0000**	1.7085	**0.0555**
w_j/p_j non-decreasing order	1.4619	1.0083	6.0158	0.2456
w_j non-increasing order	1.2310	1.0021	2.0348	0.0698
w_j non-decreasing order	1.3746	1.0020	5.9468	0.2200
p_j non-increasing order	1.4443	1.0071	5.9929	0.2232
p_j non-decreasing order	1.2402	1.0115	2.0071	0.0740
random	1.3057	1.0054	4.8395	0.1352
(n=30)				
Algorithm 2 (3-approximation algorithm)	1.1246	1.0442	1.3919	0.0314
w_j/p_j non-increasing order	1.1193	1.0453	**1.2701**	**0.0238**
w_j/p_j non-decreasing order	1.4669	1.0851	2.7828	0.1430
w_j non-increasing order	**1.1174**	**1.0419**	1.4054	0.0266
w_j non-decreasing order	1.3351	1.0527	2.5792	0.1338
p_j non-increasing order	1.3812	1.0578	2.6710	0.1318
p_j non-decreasing order	1.1672	1.0492	1.5518	0.0425
random	1.1898	1.0494	1.9552	0.0692
(n=100)				
Algorithm 2 (3-approximation algorithm)	1.1005	1.0330	1.2642	0.0233
w_j/p_j non-increasing order	1.0711	1.0322	**1.1561**	0.0134
w_j/p_j non-decreasing order	1.4761	1.1874	2.0203	0.0779
w_j non-increasing order	**1.0590**	**1.0264**	1.1655	**0.0121**
w_j non-decreasing order	1.3313	1.0672	1.8093	0.0756
p_j non-increasing order	1.3487	1.0544	1.8798	0.0745
p_j non-decreasing order	1.1226	1.0441	1.2929	0.0259
random	1.1065	1.0311	1.3938	0.0357

Bibliography

[1] Rudolf Bayer. Symmetric binary b-trees: Data structure and maintenance algorithms. *Acta informatica*, 1(4):290–306, 1972.

[2] Atiyeh Bazgosha, Mohammad Ranjbar, and Negin Jamili. Scheduling of loading and unloading operations in a multi stations transshipment terminal with release date and inventory constraints. *Computers & Industrial Engineering*, 106:20–31, 2017.

[3] Matthias Bentert, Robert Bredereck, Péter Györgyi, Andrzej Kaczmarczyk, and Rolf Niedermeier. A multivariate complexity analysis of the material consumption scheduling problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11755–11763. AAAI Press, 2021.

[4] Kristóf Bérczi, Tamás Király, and Simon Omlor. Scheduling with non-renewable resources: Minimizing the sum of completion times. In *Proceedings of the 6th International Symposium on Combinatorial Optimization*, volume 12176 of *Lecture Notes in Computer Science*, pages 167–178. Springer, 2020.

[5] Dirk Briskorn, Byung-Cheon Choi, Kangbok Lee, Joseph Leung, and Michael Pinedo. Complexity of single machine scheduling subject to nonnegative inventory constraints. *European Journal of Operational Research*, 207(2):605–619, 2010.

[6] Dirk Briskorn, Florian Jaehn, and Erwin Pesch. Exact algorithms for inventory constrained scheduling on a single machine. *Journal of scheduling*, 16(1):105–115, 2013.

[7] Dirk Briskorn and Joseph YT Leung. Minimizing maximum lateness of jobs in inventory constrained scheduling. *Journal of the Operational Research Society*, 64(12):1851–1864, 2013.

[8] Jacques Carlier and AHG Rinnooy Kan. Scheduling subject to nonrenewable-resource constraints. *Operations Research Letters*, 1(2):52–55, 1982.

[9] Morteza Davari, Mohammad Ranjbar, Patrick De Causmaecker, and Roel Leus. Minimizing makespan on a single machine with release dates and inventory constraints. *European Journal of Operational Research*, 286(1):115–128, 2020.

[10] Evgeny R Gafarov, Alexander A Lazarev, and Frank Werner. Single machine scheduling problems with financial resource constraints: Some complexity results and properties. *Mathematical Social Sciences*, 62(1):7–13, 2011.

[11] Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell system technical journal*, 45(9):1563–1581, 1966.

[12] Ronald L Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

[13] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Op-

timization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979.

[14] Alexander Grigoriev, Martijn Holthuijsen, and Joris van de Klundert. Basic scheduling problems with raw material constraints. *Naval Research Logistics*, 52(6):527–535, 2005.

[15] Péter Györgyi and Tamás Kis. Approximation schemes for single machine scheduling with non-renewable resource constraints. *Journal of Scheduling*, 17(2):135–144, 2014.

[16] Péter Györgyi and Tamás Kis. Approximability of scheduling problems with resource consuming jobs. *Annals of Operations Research*, 235(1):319–336, 2015.

[17] Péter Györgyi and Tamás Kis. Approximation schemes for parallel machine scheduling with non-renewable resources. *European Journal of Operational Research*, 258(1):113–123, 2017.

[18] Péter Györgyi and Tamás Kis. Minimizing total weighted completion time on a single machine subject to non-renewable resource constraints. *Journal of Scheduling*, 22(6):623–634, 2019.

[19] Péter Györgyi and Tamás Kis. New complexity and approximability results for minimizing the total weighted completion time on a single machine subject to non-renewable resource constraints. *Discrete Applied Mathematics*, 311:97–109, 2022.

[20] Tamás Kis. Approximability of total weighted completion time with resource consuming jobs. *Operations Research Letters*, 43(6):595–598, 2015.

[21] Alexander V Kononov and Bertrand MT Lin. Minimizing the total weighted completion time in the relocation problem. *Journal of Scheduling*, 13(2):123–129, 2010.

[22] Ehab Morsy and Erwin Pesch. Approximation algorithms for inventory constrained scheduling on a single machine. *Journal of Scheduling*, 18(6):645–653, 2015.

[23] Roman Slowiński. Preemptive scheduling of independent jobs on parallel machines subject to financial constraints. *European Journal of Operational Research*, 15(3):366–373, 1984.

[24] Wayne E Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.