Priority Algorithms for the Subset-Sum Problem

by

Yuli Ye

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Priority Algorithms for the Subset-Sum Problem

Yuli Ye

Master of Science

Graduate Department of Computer Science

University of Toronto

2006

Priority algorithms capture the key notion of "greediness" in the sense that they process the "best" data item one at a time, depending on the current knowledge of the input, while keeping a feasible solution for the output. Although priority algorithms are often simple to state, their relative power is not completely understood. In this thesis, we study priority algorithms for the Subset-Sum Problem. In particular, several variants of priority algorithms: revocable versus irrevocable, fixed versus adaptive, non-increasing order versus non-decreasing order; are analyzed and corresponding lower bounds are provided.

Dedication

Dedicate to my beloved parents Lvan Ye and Yi Jin

Acknowledgements

First of all, I would like to thank Allan Borodin, my supervisor, for introducing to me this wonderful topic, and the numerous discussions and encouragements he gave me during the study of my master's degree. I also thank him for the approximation algorithms course he taught in the winter of 2005, which has become an inspiration for this work. Very special thanks go to John Brzozowski for bringing me into the world of scientific research and giving me invaluable help during the very early stage of my research. I thank him for the four years of collaboration and friendship, and the two papers we wrote together; without him, I would miss a great deal of my life. I also thank Avner Magen for carefully reading my thesis and many comments and suggestions to make it better.

I am greatly indebted to my parents for their long-time financial and emotional support, and my family and friends at Waterloo and Toronto; without them, this thesis would become impossible.

The financial support from the Natural Sciences and Engineering Research Council of Canada and the Department of Computer Science, University of Toronto are gratefully acknowledged.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter introduces two main objects of the thesis: the Subset-Sum Problem and the Priority Model. A list of previous work done in related fields is surveyed and some motivations for this topic are provided.

1.1 Subset-Sum Problem

The *Subset-Sum Problem* (SSP) is one the most fundamental NP-complete problems [11], and perhaps the simplest of its kind. Given a set of n data items with positive weights and a capacity c, the *decision version* of SSP asks whether there exists a subset whose corresponding total weight is exactly the capacity c; the *maximization version* of SSP is to find a subset such that the corresponding total weight is maximized without exceeding the capacity c. The Subset-Sum Problem has many applications; for example, a decision version of SSP with unique solutions represents a secret message in a SSP-based cryptosystem. It also appears in more complicated combinatorial problems [25], scheduling problems [15, 16], 0-1 integer programs [9, 10], and bin packing algorithms [6, 7]. The Subset-Sum Problem is often thought of as a special case of the Knapsack Problem, where the weight of a data item is proportional to its size. Therefore, algorithms for the Knapsack Problem can be applied for SSP automatically. However, as simple

as it is, the Subset-Sum Problem has a better structure and hence sometimes admits better algorithms.

A naive algorithm with time complexity $O(n2^n)$ solves SSP, by iterating through all possible subsets, and for each subset comparing its sum with the capacity c. A better algorithm is proposed in 1974 using the Horowitz and Sahni decomposition scheme, which achieves time $O(n2^{\frac{n}{2}})$; exact algorithms for solving SSP have not been improved since then. From a practical point of view, if the capacity c is relatively small, there exist dynamic programming algorithms that can run much faster. A classic pseudo-polynomial algorithm using Bellman recursion solves SSP in both time and space $O(nc)$, and many other algorithms have been proposed under different heuristics, see [21] for a detailed comparison.

Approximation algorithms have also been studied extensively for SSP. The first FPTAS is due to Ibarra and Kim [18], which has time and space complexity $O(\frac{n}{\varepsilon^2})$ and $O(n + \frac{1}{\varepsilon^3})$ respectively. This is subsequently improved by Lawler [22], Gens and Levner [12, 13, 14]. The best known approximation algorithm is due to Kellerer et al. [20], and has time and space complexity $O(min\{\frac{n}{\varepsilon}, n + \frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon}\})$ and $O(n + \frac{1}{\varepsilon})$ respectively. The idea of the algorithm is as follows. The algorithm first separates data items into small items, data items with weight $\leq \varepsilon c$, and large items. It can be easily seen that any $(1-\varepsilon)$-approximation algorithm for the large items remains an $(1-\varepsilon)$-approximation algorithm for the whole set if we fill the small items during the end of the algorithm in a greedy way. Next, it partitions the interval $[\varepsilon c, c]$ into subintervals of equal length εc. For each of such intervals, the algorithm selects a constant number of candidates; these data items form the set of relevant items. The paper then proves that the optimal solution on the set of relevant items approximates the optimal solution on the whole set with accuracy εc. Finally, the algorithm computes the optimal solution on the set of relevant items using relaxed dynamic programming; divide-and-conquer and backtracking techniques are applied to secure the bounds on running time and required space.

In terms of "greedy" approximation algorithms, not much work has been done. The first such

algorithm is derived directly from a "greedy" algorithm of the Knapsack Problem [24], which considers the data item with maximum weight alone as a possible alternative solution, while processing the rest of the data items greedily to build the other solution. It is not hard to see that this strategy guarantees an approximation ratio $\frac{1}{2}$. An alternative greedy algorithm [24], and maybe the most intuitive one, considers data items in non-increasing order, and accepts one if it fits. This algorithm has a better average result, but the worst case approximation ratio still remains at $\frac{1}{2}$. However, by applying it n times on item sets $\{1, \ldots, n\}$, $\{2, \ldots, n\}$, $\{3, \ldots, n\}$, and so on, respectively, Martello and Toth [23] prove that the approximation ratio can be improved to $\frac{3}{4}$. Of course, the disadvantages of this approach are that it requires n passes and needs to maintain n solutions. In [19], Iwama and Taketomi manage to show that a better strategy exists for one-pass algorithms even if the ordering of data items is arbitrary. They prove a tight bound on the approximation ratio of $\frac{\sqrt{5}-1}{2} \approx 0.618$ for online revocable algorithms of SSP. These four algorithms represent the current stage of "greedy" approximation algorithms, yet not all of them fit in our definition of priority algorithms. The second and the fourth are most relevant to this thesis; the second is an irrevocable priority algorithm and the fourth is an online revocable algorithm which is a special case of revocable priority algorithms. The remaining two are not so relevant as they belong to a more general class of BT algorithms defined in [1].

1.2 Priority Model and Related Work

Greedy algorithms are among the most natural attempts in solving combinatorial optimization problems. Although they seldom yield optimal solutions for hard problems, like the class of NP-hard problems, they sometimes give good approximation ratios; for example, a simple greedy algorithm yields the best known constant approximation ratio of 2 for the vertex cover problem.

The Priority Model [5], recently established by Borodin et al., formulates a notion of "greediness". This formulation provides us a tool to assess the computational power and limitation

for greedy algorithms, especially in terms of their approximability. The key characteristics of priority algorithms are that it has an ordering function evaluating the priority of each data item, processes data items in turn according to their priorities and builds the solution incrementally. For this reason, priority algorithms can be viewed as an extension of online algorithms, where the ordering function instead of nature decides which data item comes next. The algorithms defined under this framework are usually efficient; running times are often bounded by $O(n \log n)$[1].

In addition to [5], where priority algorithms for scheduling problems are studied, several papers have been published along this line, analyzing the relative power of priority algorithms on many problems, including facility location, set cover [2], interval selection [17], and various graph problems [4, 8]. Results have shown that non-trivial lower bounds for such models exist for many NP-hard problems, and some gaps with respect to the approximation ratio are inherently difficult to close.

1.3 Motivations

The main motivation of this work comes from two papers. The first one is by Iwama and Taketomi [19], where a tight bound on the approximation ratio of $\frac{\sqrt{5}-1}{2} \approx 0.618$ has been proved for online revocable algorithms for SSP. Since priority algorithms are natural extensions of online algorithms, it is interesting to see whether the additional power of priority algorithms improves the approximation ratio. The second one is the master thesis work done by Horn [17], which extends the work from the scheduling approximation paper by Bar-Noy et al. [3]. In her thesis, one-pass algorithms with revocable acceptances are explicitly studied for the job interval selection problem. Both of these two papers mentioned above are built on an important feature that we are allowed to make revocable decisions on accepted items. This new feature brings us more power, and provides a new prospective for priority algorithms. Furthermore,

[1] Note that we do not insist upon any time complexity on the model, it is determined by the actual algorithm.

as a fundamental NP-complete problem, the Subset-Sum Problem has a very simple structure. Therefore, studying priority algorithms for SSP gives us some additional benefits:

- Lower bounds on approximation ratio of priority algorithms for SSP provide us some insights into how well priority algorithms can approximate for more complicated problems. For example, a lower bound for SSP automatically implies a lower bound for the Knapsack Problem.

- Approximability gaps are of fundamental interest to the theoretical computer science community, exhibiting such gaps for relative simple structures may reveal some of mysteries as to why those gaps are hard to close.

Nevertheless, priority algorithms for SSP themselves are interesting enough as they have never been studied, and their behavior can give us more evidence of how good greedy algorithms can be.

The rest of the thesis is organized as follows. In Chapter 2, we give formal definitions for the Subset-Sum Problem, the Priority Model and the adversarial strategy. We deal with irrevocable and revocable priority algorithms for SSP in Chapter 3 and 4 respectively. In Chapter 5, we conclude our work and discuss future directions.

Chapter 2

Preliminaries

In this chapter, we formally define the Subset-Sum Problem and the Priority Model. We also describe the adversarial strategy, which is the tool for deriving lower bounds.

2.1 Notation and Definitions

Since the basic elements of SSP are data items, we first define some of notation used for data items and sets of data items. We use **bold** font letters to denote sets of data items. For a give set $\mathbf{R}$ of data items, we use $|\mathbf{R}|$ to denote its size, i.e., the number of data items in $\mathbf{R}$; and use $\|\mathbf{R}\|$ to denote the corresponding total weight. For a data item u, we sometimes use u to represent the singleton set $\{u\}$, and $2u$ to represent the multi-set $\{u, u\}$; we also use u to represent its weight since it is the only attribute. The term u here is an overloaded term, but the meaning will become clear in the actual context. For set operations, the standard notation for set addition and set subtraction are $\cup$ and $\setminus$ respectively. In this thesis, we use $\oplus$ to denote set addition, and use $\ominus$ to denote set subtraction, as they deliver more intuitive meaning of adding and deleting data items. We keep the traditional $\cap$ to denote set intersection.

2.1.1 Definition of the Subset-Sum Problem

We provide a slightly different definition from the one in Section 1.1 for the suit of our analysis. Here, we make two additional assumptions. First of all, the weights are all scaled to their relative values to the capacity; hence we can use 1 instead of c for the capacity. Secondly, we assume each data item has weight less than or equal to 1, for otherwise the weight of that data item exceeds the capacity, hence useless to our solution. Therefore, an *instance* σ of SSP is a set $\mathbf{I} = \{u_1, u_2, \ldots, u_n\}$ of n data items with $u_i \in (0, 1]$ for all $i \in [1, n]$. The set $\mathbf{I}$ is the *input* set, and $u_1, u_2, \ldots, u_n$ are the *data items*. A *feasible* solution of σ is a subset $\mathbf{B}$ of $\mathbf{I}$ such that $\|\mathbf{B}\| \leq 1$. An *optimal* solution of σ is a feasible solution with maximum weight. We can formulate SSP as a solution of the following integer programming:

$$\text{maximize} \quad \sum_{i=1}^{n} u_i x_i \tag{2.1}$$

$$\text{subject to} \quad \sum_{i=1}^{n} u_i x_i \leq 1, \tag{2.2}$$

where $x_i \in \{0, 1\}$ and $i \in [1, n]$. Let $\mathcal{A}$ be an algorithm for SSP, for a given instance σ, we denote $\mathcal{A}(\sigma)$ the solution achieved by $\mathcal{A}$ and $\textbf{OPT}(\sigma)$, the optimal solution, then the *approximation ratio* of $\mathcal{A}$ on σ is denoted by

$$\rho(\mathcal{A}, \sigma) = \frac{\|\mathcal{A}(\sigma)\|}{\|\textbf{OPT}(\sigma)\|}.$$

Let Σ be the set of all instances of SSP, then the *approximation ratio* of $\mathcal{A}$ over Σ is denoted by

$$\rho(\mathcal{A}) = \inf_{\sigma \in \Sigma} \rho(\mathcal{A}, \sigma).$$

In a particular analysis, the algorithm and the problem instance are usually fixed. For convenience, we often use **ALG**, **OPT** and ρ to denote the algorithm's solution, the optimal solution and the approximation ratio respectively.

2.1.2 Priority Model

We base our terminology and model on that of [5], and start with the class of fixed order irrevocable priority algorithms for SSP. For a given instance σ, a *fixed (order) irrevocable* priority algorithm maintains a feasible solution $\mathbf{B}$ throughout the algorithm. At each step, it looks at a data item with the highest priority in $\mathbf{I}$ according to some pre-determined ordering, then it can either discard it, i.e., remove it from $\mathbf{I}$ without doing anything, or add it to $\mathbf{B}$, i.e., move it from $\mathbf{I}$ to $\mathbf{B}$. The structure of a fixed irrevocable priority algorithm is described as follows:

Fixed Irrevocable Priority

Ordering: Determine, without looking at $\mathbf{I}$, a total ordering of all possible data items

while $\mathbf{I}$ is not empty

$next :=$ index of the data item in $\mathbf{I}$ that comes first in the ordering

Decision: Decide whether or not to add u_{next} to $\mathbf{B}$, and then remove u_{next} from $\mathbf{I}$

end while

An *adaptive (order) irrevocable* priority algorithm is similar to a fixed one, but instead of looking at a data item according to some pre-determined ordering, the algorithm is allowed to reorder the remaining data items in $\mathbf{I}$ at each step. This gives the algorithm an advantage since now it can take into account the information that has been revealed so far to determine which is the best data item to consider next. The structure of an adaptive irrevocable priority algorithm is described as follows:

Adaptive Irrevocable Priority

while $\mathbf{I}$ is not empty

Ordering: Determine, without looking at $\mathbf{I}$, a total ordering of all possible data items

$next :=$ index of the data item in $\mathbf{I}$ that comes first in the ordering

Decision: Decide whether or not to add u_{next} to $\mathbf{B}$, and then remove u_{next} from $\mathbf{I}$

end while

The above defined priority algorithms are "irrevocable" in the sense that once a data item is admitted to the solution it cannot be removed. We can extend our notion of "fixed" and "adaptive" to the class of revocable priority algorithms, where revocable decisions on accepted data items are allowed. Accordingly, those algorithms are called *fixed (order) revocable* and *adaptive (order) revocable* priority algorithms respectively; they are formally defined as follows:

Fixed Revocable Priority

Ordering: Determine, without looking at **I**, a total ordering of all possible data items

while **I** is not empty

$next$:= index of the data item in **I** that comes first in the ordering

Decision: Decide whether or not to add u_{next} to **B** by discarding any number data items in **B**, and then remove u_{next} from **I**

end while

Adaptive Revocable Priority

while **I** is not empty

Ordering: Determine, without looking at **I**, a total ordering of all possible data items

$next$:= index of the data item in **I** that comes first in the ordering

Decision: Decide whether or not to add u_{next} to **B** by discarding any number data items in **B**, and then remove u_{next} from **I**

end while

The extension[1] to revocable acceptances does give us some additional power, for example, as shown in [19], for the online version of SSP, irrevocable algorithms do not achieve any constant bound approximation ratio, while revocable algorithms achieve a tight approximation ratio $\frac{\sqrt{5}-1}{2} \approx 0.618$.

[1] This extension applies to priority algorithms for packing problems.

2.2 Adversarial Strategy

We often utilize an adversary in proving lower bounds. For a given priority algorithm, we run the adversary against the algorithm in the following scheme. At the beginning of the algorithm, the adversary first presents a set of data items to the algorithm, possibly with some data items having the same weight. Furthermore, our adversary promises that the actual input is contained in this set[2]. Note that weight is the only criterion for which the algorithm can compare two data items, hence the algorithm cannot distinguish two data items if they have the same weight. At each step, the adversary asks the algorithm to select one data item in the remaining set and make a decision on that data item. Once the algorithm makes a decision on the selected item, the adversary then has the power to remove any number of data items in the remaining set; this repeats until the remaining set is empty, which leads to the end of the algorithm.

For convenience, we often use a diagram to illustrate an adversarial strategy. A diagram of an adversarial strategy is an acyclic directed graph, where each node represents a possible state of the strategy, and each arc indicates a possible transition. Each state of the strategy contains two boxes. The first box indicates the current solution maintained by the algorithm, the second box indicates the remaining set of data items maintained by the adversary. A state can be either terminal or non-terminal. A state is *terminal* if and only if it is a sink, in the sense that the adversary no longer need perform any additional action; we indicate a terminal state using **bold** boxes. Each transition also contains two lines of actions. The first line indicates the action taken by the algorithm and the second line indicates the action taken by the adversary; we use $\oslash$ to indicate no action. Note that to calculate a lower bound for the approximation ratio of an algorithm, it is sufficient to consider the approximation ratios achieved in all terminal states. An example is given in the next subsection.

[2]This assumption is optional. The lower bounds clearly hold for a stronger adversary.

2.2.1 An Example

Let us take a fixed revocable priority algorithm which orders data item $u_1 = 0.8$ before $u_2 = 0.5$ as an example; see Fig. 2.1.

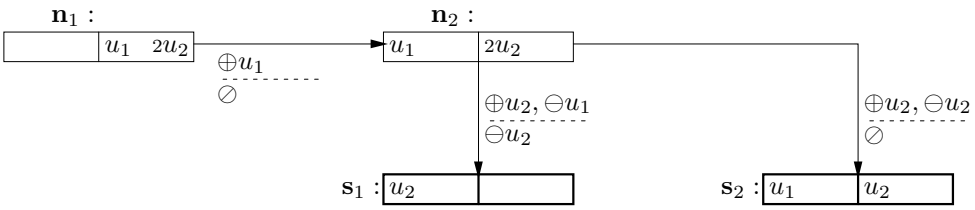

Figure 2.1: An example of an adversarial strategy.

At the beginning, the adversary presents one copy of u_1 and two copies of u_2 to the algorithm; this state is represented by the state $\mathbf{n}_1$. Since the algorithm orders u_1 before u_2, it selects u_1; this is indicated by the first line under the transition from $\mathbf{n}_1$ to $\mathbf{n}_2$. Accordingly, adversary takes no action; this is indicated by the second line under the transition from $\mathbf{n}_1$ to $\mathbf{n}_2$. The first transition leads to the state $\mathbf{n}_2$, where u_1 is in the algorithm's solution while the adversary still holds two copies of u_2 for the remaining input. There are two choices for the algorithm starting from $\mathbf{n}_2$. The first one is indicated by the transition from $\mathbf{n}_2$ to $\mathbf{s}_1$, where the algorithm takes u_2 into the solution and discards the previously accepted u_1. If this is the case, the adversary deletes u_2 from the remaining input. The second one is indicated by the transition from $\mathbf{n}_2$ to $\mathbf{s}_2$, where the algorithm just discards u_2. If this is the case, the adversary takes no action. Both $\mathbf{s}_1$ and $\mathbf{s}_2$ are terminal states since no further action is required for the adversary. If the algorithm terminates in state $\mathbf{s}_1$, it has approximation ratio $\frac{5}{8}$. If the algorithm terminates in state $\mathbf{s}_2$, it has approximation ratio 0.8. Since they are the only terminal states, the algorithm cannot achieve approximation ratio better than $\frac{5}{8}$.

Chapter 3

Irrevocable Priority Algorithms

Let $\alpha_1 \approx 0.657$ be the real root of the equation

$$2x^3 + x^2 - 1 = 0 \tag{3.1}$$

between 0 and 1. In this chapter, we prove a tight bound of α_1 on the approximation ratio of irrevocable priority algorithms for SSP. It is interesting that there is no approximability gap between fixed and adaptive irrevocable priority algorithms. Before we get into the proof, we first provide two simplifications for general priority algorithms.

3.1 General Simplifications

The two simplifications given in this section are both based on the approximation ratio, say α, we want to achieve.

3.1.1 Implicit Terminal Conditions

Since we are interested in approximation algorithms, we can terminate an algorithm at any time if the approximation ratio of α is achieved. This condition is called a *terminal condition*.

- For a fixed irrevocable priority algorithm, the terminal condition is satisfied, if at the beginning of some step of the algorithm, u is the next data item to be examined, $\mathbf{B}' = (\mathbf{B} \oplus u)$ and $\alpha \leq \|\mathbf{B}'\| \leq 1$.

- For an adaptive irrevocable priority algorithm, the terminal condition is satisfied, if at the beginning of some step of the algorithm, there exists some $u \in \mathbf{I}$ and $\mathbf{B}' = (\mathbf{B} \oplus u)$ such that $\alpha \leq \|\mathbf{B}'\| \leq 1$. In this case, u is the next data item, and the approximation ratio can be achieved.

- For a fixed revocable priority algorithm, the terminal condition is satisfied, if at the beginning of some step of the algorithm, u is the next data item to be examined, and there exists $\mathbf{B}' \subseteq (\mathbf{B} \oplus u)$ such that $\alpha \leq \|\mathbf{B}'\| \leq 1$.

- For an adaptive revocable priority algorithm, the terminal condition is satisfied, if at the beginning of some step of the algorithm, there exists some $u \in \mathbf{I}$ and $\mathbf{B}' \subseteq (\mathbf{B} \oplus u)$ such that $\alpha \leq \|\mathbf{B}'\| \leq 1$. In this case, u is the next data item, and the approximation ratio can be achieved.

It is clear that in all four cases, the algorithm can take $\mathbf{B}'$ as the final solution and immediately terminate. For any algorithm given in this thesis, we will not explicitly state the check of the terminal condition; we assume that the algorithm tests the condition whenever it considers the next data item, and will terminate if the condition is satisfied. Note that here we do not impose any time bound for checking the terminal condition in the general model. But for all the algorithms studied in this thesis, the extra check for the terminal condition does not increase much for the time complexity as the input against such test is highly restricted, and the running time is bounded by a constant.

3.1.2 Exclusion of Small and Extra Large Data Items

For the approximation ratio α, a data item u is said to be in class $\mathbf{S}$ and $\mathbf{X}$ if $0 < u \leq 1 - \alpha$ and $\alpha \leq u \leq 1$ respectively. A data item is *small* if it is in $\mathbf{S}$, and *extra large* if it is in $\mathbf{X}$; a data item is *relevant* if it is neither small nor extra large. Let Σ' be the set of instances of SSP whose input contains only relevant data items, and let $\mathcal{A}'$ be a priority algorithm over Σ'; we call $\mathcal{A}'$ a *restricted* algorithm. For a given instance $\sigma \in \Sigma$ with input $\mathbf{I}$, we let $\sigma' \in \Sigma'$ be the instance with input $\mathbf{I} \ominus \mathbf{S} \ominus \mathbf{X}$. An algorithm $\mathcal{A}$ over Σ is a *completion*[1] of $\mathcal{A}'$ if for any instance σ, either $\rho(\mathcal{A}, \sigma) \geq \alpha$ or $\mathbf{X} \cap \mathbf{I} = \emptyset$, $\mathcal{A}(\sigma) \ominus \mathbf{S} = \mathcal{A}'(\sigma')$ and $\mathcal{A}(\sigma) \cap \mathbf{S} = \mathbf{S} \cap \mathbf{I}$. This definition implies that if $\mathbf{X} \cap \mathbf{I} \neq \emptyset$, then we have $\rho(\mathcal{A}, \sigma) \geq \alpha$.

Proposition 3.1.1 *Let $\mathcal{A}$ be a completion of $\mathcal{A}'$. If $\mathcal{A}'$ achieves approximation ratio α over Σ', then $\mathcal{A}$ achieves approximation ratio α over Σ.*

Proof: Suppose that $\mathcal{A}'$ achieves approximation ratio α over Σ', but $\mathcal{A}$ achieves approximation ratio less than α over Σ, then $\mathbf{X} \cap \mathbf{I} = \emptyset$ and for a given instance σ, $\mathcal{A}(\sigma) \ominus \mathbf{S} = \mathcal{A}'(\sigma')$ and $\mathcal{A}(\sigma) \cap \mathbf{S} = \mathbf{S} \cap \mathbf{I}$. Hence, we have

$$\|\mathbf{OPT}(\sigma) \cap \mathbf{S}\| \leq \|\mathbf{S} \cap \mathbf{I}\| = \|\mathcal{A}(\sigma) \cap \mathbf{S}\|.$$

Therefore, approximation ratio of $\mathcal{A}$ on σ is given by

$$\rho(\mathcal{A}, \sigma) = \frac{\|\mathcal{A}(\sigma)\|}{\|\mathbf{OPT}(\sigma)\|} = \frac{\|\mathcal{A}(\sigma) \cap \mathbf{S}\| + \|\mathcal{A}(\sigma) \ominus \mathbf{S}\|}{\|\mathbf{OPT}(\sigma) \cap \mathbf{S}\| + \|\mathbf{OPT}(\sigma) \ominus \mathbf{S}\|} \geq \frac{\|\mathcal{A}(\sigma) \ominus \mathbf{S}\|}{\|\mathbf{OPT}(\sigma) \ominus \mathbf{S}\|}.$$

On the other hand, since the algorithm $\mathcal{A}'$ achieves approximation ratio of α on σ', we have

$$\rho(\mathcal{A}', \sigma') = \frac{\|\mathcal{A}'(\sigma')\|}{\|\mathbf{OPT}(\sigma')\|} \geq \alpha.$$

Since $\mathcal{A}'(\sigma') = \mathcal{A}(\sigma) \ominus \mathbf{S}$, and $\|\mathbf{OPT}(\sigma')\| \geq \|\mathbf{OPT}(\sigma) \ominus \mathbf{S}\|$. Therefore,

$$\rho(\mathcal{A}, \sigma) \geq \frac{\|\mathcal{A}(\sigma) \ominus \mathbf{S}\|}{\|\mathbf{OPT}(\sigma) \ominus \mathbf{S}\|} \geq \frac{\|\mathcal{A}'(\sigma')\|}{\|\mathbf{OPT}(\sigma')\|} \geq \alpha.$$

Therefore, the algorithm $\mathcal{A}$ achieves approximation ratio of α, which is a contradiction. □

[1] This is only defined on valid algorithms, i.e., both solutions of $\mathcal{A}$ and $\mathcal{A}'$ have to be feasible.

Proposition 3.1.1 shows that if we have a restricted algorithm which achieves approximation ratio α over Σ' and it has a completion, then the completion achieves approximation ratio α over Σ. This gives us some hope, yet it is not enough to exclude small and extra large data items for the following two reasons:

- We still do not know how to construct a completion from a restricted algorithm for specific class of priority algorithms.
- Such construction may alter the class of the original algorithm, hence may not be applicable.

Therefore, we need the following definition. For a given class $\mathbb{C}$ of priority algorithms, a completion is $\mathbb{C}$-*preserving* if both algorithms are in $\mathbb{C}$. Now the remaining task is to show for a specific class $\mathbb{C}$, how to construct a $\mathbb{C}$-preserving completion from a restricted algorithm. Note that for most classes of algorithms studied in this thesis, we do not impose any ordering to process data items. That means for those classes, one has the freedom to design an algorithm to process data items in any order. There are two exceptions: the non-increasing and non-decreasing order revocable priority algorithms studied in Section 4.2, each of them imposes a restricted ordering, hence we have to deal them separately.

Proposition 3.1.2 *Let $\mathbb{C}$ be the class of* [fixed | adaptive] [irrevocable | revocable] *priority algorithms, then there exists a $\mathbb{C}$-preserving completion from a restricted algorithm in $\mathbb{C}$.*

Proof: Given a restricted algorithm $\mathcal{A}'$ in $\mathbb{C}$, the algorithm $\mathcal{A}$ is constructed from $\mathcal{A}'$ with three phases:

- **Phase 1:** if there is an extra large data item then take that data item as the final solution, and terminate the algorithm; otherwise go to **Phase 2**.
- **Phase 2:** run the algorithm $\mathcal{A}'$ on σ', and then go to **Phase 3**.

- **Phase 3:** greedily fill the solution with small data items until either an overflow or the exhaustion of small data items; terminate the algorithm.

It is clear that the above construction preserves the class of fixed irrevocable priority algorithms, which is the most restricted class among the four combinations. Therefore, such construction is $\mathbb{C}$-preserving. There are three possible ways for algorithm $\mathcal{A}$ to terminate. If the algorithm $\mathcal{A}$ terminates in **Phase 1**, then it clearly achieves approximation ratio of α; otherwise, it terminates in **Phase 3**. There are two cases. If the termination is caused by an overflow of a small data item, then the total weight of the solution is $\geq \alpha$, hence the target approximation ratio is also achieved. If the termination is caused by the exhaustion of small data items, then there exists no extra large data item and the solution admits all the small data items. Furthermore, the relevant data items kept in the solution by $\mathcal{A}$ on σ are the same as those in the solution by $\mathcal{A}'$ on σ'. In all three cases, the definition of completion is satisfied. Therefore the algorithm $\mathcal{A}$ is a $\mathbb{C}$-preserving completion of $\mathcal{A}'$. □

Corollary 3.1.3 *Let $\mathbb{C}$ be the class of* [fixed | adaptive] [irrevocable | revocable] *priority algorithms, then there exists an α-approximation algorithm in $\mathbb{C}$ if and only if there exists an α-approximation restricted algorithm in $\mathbb{C}$.*

Proof: This follows immediately by Proposition 3.1.1 and 3.1.2. □

Proposition 3.1.4 *Let $\mathbb{C}$ be the class of* [non-increasing | non-decreasing] *order revocable priority algorithms, then there exists a $\mathbb{C}$-preserving completion from a restricted algorithm in $\mathbb{C}$.*

Proof: Given a restricted algorithm $\mathcal{A}'$ in $\mathbb{C}$, the algorithm $\mathcal{A}$ is constructed from $\mathcal{A}'$ as follows:

- If encounters an extra large item, then the terminal condition is satisfied.
- If encounters a relevant data item, then it acts the same as $\mathcal{A}'$ unless the terminal condition is satisfied.

- If encounters a small data item, then that data item always stays in the solution until the end of the algorithm or the terminal condition is satisfied.

It is clear that the above construction preserves the class of [non-increasing | non-decreasing] order revocable priority algorithms. Therefore, it is $\mathbb{C}$-preserving. There are two possible ways for algorithm $\mathcal{A}$ to terminate. If the termination of the algorithm is caused by a satisfaction of the terminal condition, then it clearly achieves approximation ratio of α; otherwise, there exists no extra large data item and the solution admits all the small data items. Furthermore, the relevant data items kept in the solution by $\mathcal{A}$ on σ are the same as those in the solution by $\mathcal{A}'$ on σ'. In both cases, the definition of completion is satisfied. Therefore the algorithm $\mathcal{A}$ is a $\mathbb{C}$-preserving completion of $\mathcal{A}'$. □

Corollary 3.1.5 *Let $\mathbb{C}$ be the class of* [non-increasing | non-decreasing] *order revocable priority algorithms, then there exists an α-approximation algorithm in $\mathbb{C}$ if and only if there exists an α-approximation restricted algorithm in $\mathbb{C}$.*

Proof: This follows immediately by Proposition 3.1.1 and 3.1.4. □

Note that all the algorithms studied in this thesis are covered by either Corollary 3.1.3 or 3.1.5, we can now safely assume the original input contains only relevant data items.

3.2 Fixed Irrevocable Priority Algorithms

In this section, we provide a fixed irrevocable priority algorithm that achieves approximation of α_1. This improves the $\frac{1}{2}$ approximation ratio achieved by the "intuitive" fixed irrevocable priority algorithm mentioned in Section 1.1, which sorts data items according to non-increasing weights and accepts one if it fits. This improvement shows strong evidence that a better ordering can sometimes produce a better approximation ratio for priority algorithms.

As justified earlier in Section 3.1, we only consider relevant data items for the input set; we partition all possible relevant data items into two sets: **M** and **L**. A data item u is said to be in class **M** and **L** if $1 - \alpha_1 < u \leq \alpha_1^2$ and $\alpha_1^2 < u < \alpha_1$ respectively; see Fig. 3.1.

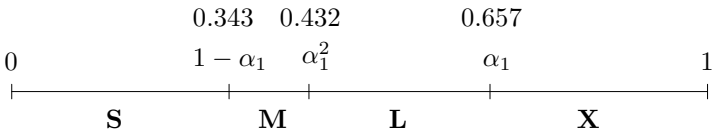

Figure 3.1: Classification for the fixed irrevocable priority algorithm.

3.2.1 Description of the Algorithm

We now specify the ordering of data items for the fixed irrevocable priority algorithm: it first orders data items in **L** non-decreasingly, and then data items in **M** non-decreasingly. The algorithm, which is described below, uses this ordering to achieve the approximation ratio α_1.

Algorithm 1

1: $\mathbf{B} := \emptyset$;
2: **for** $i := 1$ to n **do**
3: let u_i be the next data item in $\mathbf{I}$;
4: $\mathbf{I} := \mathbf{I} \ominus u_i$;
5: **if** $\|\mathbf{B}\| + u_i \leq 1$ **then**
6: $\mathbf{B} := \mathbf{B} \oplus u_i$;
7: **end if**
8: **end for**

3.2.2 Proof of the Upper Bound

Theorem 3.2.1 *There exists a fixed irrevocable priority algorithm of SSP with approximation ratio α_1.*

Proof: It is sufficient to show that Alg. 1 achieves this approximation ratio. Since the smallest relevant data item is greater than $\frac{1}{3}$, the weight of two such items is greater than $\frac{2}{3}$, and the weight of three such items is greater than 1. If **ALG** contains more than one data item, the $\|\mathbf{ALG}\| > \frac{2}{3} > \alpha_1$; hence the algorithm achieves approximation ratio α_1. Suppose now **ALG** contains only one data item u_j. If $u_j \in \mathbf{M}$, then $|\mathbf{L} \cap \mathbf{I}| = 0$ and $|\mathbf{M} \cap \mathbf{I}| = 1$. Therefore, we have $\|\mathbf{OPT}\| = \|\mathbf{ALG}\| = u_j$; the algorithm achieves approximation ratio 1. Suppose now $u_j \in \mathbf{L}$, we then consider data items in **OPT**. Note that **OPT** can contain at most two data items; there are five cases:

1. If **OPT** contains exactly one data item $u_r \in \mathbf{M}$, then $|\mathbf{L} \cap \mathbf{I}| = 0$ and hence $u_j \in \mathbf{M}$. Since $u_j \in \mathbf{L}$, this is a contradiction; this case is impossible.

2. If **OPT** contains exactly one data item $u_r \in \mathbf{L}$, then the approximation ratio is
$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} = \frac{u_j}{u_r} > \frac{\alpha_1^2}{\alpha_1} = \alpha_1.$$

3. If **OPT** contains two data items $u_r, u_s \in \mathbf{M}$ with $u_r \leq u_s$, then we have $u_j + u_s > 1$. Therefore, by equation (3.1), the approximation ratio is
$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} = \frac{u_j}{u_r + u_s} > \frac{1 - u_s}{u_r + u_s} > \frac{1 - \alpha_1^2}{2\alpha_1^2} = \frac{2\alpha_1^3}{2\alpha_1^2} = \alpha_1.$$

4. If **OPT** contains two data items u_r, u_s with $u_r \in \mathbf{M}$ and $u_s \in \mathbf{L}$, then **ALG** should have contained the smallest data item in $\mathbf{M} \cap \mathbf{I}$ and the smallest data item in $\mathbf{L} \cap \mathbf{I}$, or the smallest two data items in $\mathbf{L} \cap \mathbf{I}$, and hence a contradiction; this case is impossible.

5. If **OPT** contains two data items $u_r, u_s \in \mathbf{L}$ with $u_r \leq u_s$, then **ALG** should have contained the smallest two data items in $\mathbf{L} \cap \mathbf{I}$, and hence a contradiction; this case is impossible.

Therefore, Alg. 1 achieves the approximation ratio α_1. □

3.3 Adaptive Irrevocable Priority Algorithms

In this section, we prove a lower bound for any adaptive irrevocable priority algorithm. This lower bound matches the upper bound we have in the previous section.

3.3.1 Deriving the Lower Bound

Theorem 3.3.1 *No adaptive irrevocable priority algorithm of SSP can achieve approximation ratio better than α_1.*

Proof: We show that for any irrevocable priority algorithm and every small $\epsilon > 0$, $\rho < \alpha_1 + \epsilon$. Let

$$u_1 = \alpha_1 \approx 0.657, \qquad u_2 = 2\alpha_1^3 + \alpha_1^2\epsilon \approx 0.568,$$
$$u_3 = \alpha_1^2 \approx 0.432, \qquad u_4 = \alpha_1^2 - \alpha_1^2\epsilon \approx 0.432;$$

then for any algorithm, the adversary presents one copy of u_1, u_2, u_4 and two copies of u_3 to the algorithm. Let u be the data item with highest priority among these four types, then if the algorithm discards u on its first decision, then the adversary can remove the rest of the data items and force infinite approximation ratio. We therefore assume that the algorithm always takes the first data item into its solution, then depending on which is the first data item the algorithm selects, there are four cases:

1. If $u = u_1$, then the adversary removes two copies of u_3; since $u_1 + u_2 > 1$ and $u_1 + u_4 > 1$, the algorithm can take neither of the two, hence $\|\mathbf{ALG}\| = u_1$. On the other hand, by equation (3.1), $u_2 + u_4 = 1$, hence $\|\mathbf{OPT}\| = u_2 + u_4$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} = \frac{u_1}{u_2 + u_4} = \frac{\alpha_1}{2\alpha_1^3 + \alpha_1^2} = \alpha_1.$$

2. If $u = u_2$, then the adversary removes one copy of u_1 and one copy of u_4; since $u_2+u_3 > 1$, the algorithm can take neither of the two, hence $\|\mathbf{ALG}\| = u_2$. On the other hand, $u_2 < 2u_3 < 1$, hence $\|\mathbf{OPT}\| = 2u_3$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} = \frac{u_2}{2u_3} = \frac{2\alpha_1^3 + \alpha_1^2\epsilon}{2\alpha_1^2} = \alpha_1 + \frac{\epsilon}{2}.$$

3. If $u = u_3$, then the adversary removes one copy of u_2, u_3 and u_4; since $u_1 + u_3 > 1$, the algorithm cannot take u_1, hence $\|\mathbf{ALG}\| = u_3$. On the other hand, $u_3 < u_1 < 1$, hence $\|\mathbf{OPT}\| = u_1$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} = \frac{u_3}{u_1} = \frac{\alpha_1^2}{\alpha_1} = \alpha_1.$$

4. If $u = u_4$, then the adversary removes two copies of u_3; since $u_1 + u_4 > 1$, the algorithm cannot take u_1, hence $\|\mathbf{ALG}\| = u_4$. On the other hand, $u_4 < u_1 < 1$, hence $\|\mathbf{OPT}\| = u_1$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} = \frac{u_4}{u_1} = \frac{\alpha_1^2 - \alpha_1^2\epsilon}{\alpha_1} < \frac{\alpha_1^2}{\alpha_1} = \alpha_1.$$

Therefore, in all cases, $\rho < \alpha_1 + \epsilon$; this completes the proof. □

Theorem 3.2.1 and 3.3.1 show that for fixed irrevocable priority algorithms, the best ordering is that first selecting data items in **L** non-decreasingly, and then selecting data items in **M** non-decreasingly. This ordering seems counter-intuitive, as the common sense of a greedy algorithm is always to select the "best" possible data item at each step, and for SSP, the best item is indisputably the one with largest weight. The algorithm and the proofs show that it is not the case here, at least not in terms of approximation ratios. This is because a large data item, although gives more, also takes more space. Since the algorithm is irrevocable, once a large data item gets into the solution it blocks more data items to enter the solution.

Chapter 4

Revocable Priority Algorithms

In this chapter, we study revocable priority algorithms. More specifically, for fixed revocable priority algorithms of SSP, we prove tight approximation ratios for non-increasing and non-decreasing order. We also provide an adaptive revocable priority algorithm of SSP which achieves a better approximation ratio, and prove lower bounds on the approximation ratios for general fixed and adaptive revocable priority algorithms.

4.1 More Simplifications

In this section, we provide two more simplifications for revocable priority algorithms. These simplifications together with those in Section 3.1, allow us to focus on the core of the problem rather than dealing with unimportant minor cases.

4.1.1 Two Assumptions

The ability to make revocable acceptances gives the algorithm certain flexibility to "regret". The data items admitted into the solution are never "safe" until the termination of the algorithm. Therefore, if there is enough "room", it never hurts to accept a data item no matter how "bad" it is, as we can always reject it later at any time and with no cost. Based on this observation,

for a revocable priority algorithm, we may assume the following two scenarios do not occur, as they gain no advantage to the algorithm:

- On facing a decision on a data item, the algorithm can take a data item for free, i.e., without discarding anything in the current solution, but the algorithm decides not to take it.
- On facing a decision on a data item, the algorithm decides to remove more data items than necessary to accommodate that data item.

These two assumptions simplify our analysis as they eliminate many "irrational" possibilities when the algorithm makes a decision on a data item.

4.1.2 Four Operational Modes

As mentioned earlier in Section 2.1, a priority algorithm of SSP maintains a feasible solution $\mathbf{B}$ throughout the algorithm. In order to state our algorithms cleanly, we put more restrictions on $\mathbf{B}$; in particular, we specify how it can discard accepted items, if necessary, when accepting a new data item u from the input. The container $\mathbf{B}$ has to operate in one of the following four modes:

1. `Queue Mode:` In this mode, accepted items are discarded in the FIFO order to accommodate the new data item u.
2. `Queue_1 Mode:` In this mode, the first accepted item is never discarded, the rest data items are discarded in the FIFO order to accommodate the new data item u.
3. `Stack Mode:` In this mode, accepted items are discarded in the FILO order to accommodate the new data item u.
4. `Optimum Mode:` In this mode, accepted items are discarded to maximize $\|\mathbf{B}\|$; the new data item u may also be discarded for this purpose. For all the algorithms studied in this thesis, this mode is used only when $|\mathbf{B}|$ is bounded by a small constant.

We use $\mathbf{B}_{\text{mode}}$ to represent the operational mode of $\mathbf{B}$. A priority algorithm can switch among these four modes during the processing of data items, but no matter what operational mode it is in, from now on, we do not explicitly mention what data items are being discarded since it is well-defined under its operational mode.

4.2 Fixed Revocable Priority Algorithms

In this section, we first study non-increasing and non-decreasing order for fixed revocable priority algorithms; tight approximation ratios are given in both cases. We then prove a general lower bound for fixed revocable priority algorithms.

4.2.1 Non-Increasing Order

In this subsection, we prove a tight bound of $\alpha_1 \approx 0.657$ on the approximation ratio for non-increasing order. It is a coincident that the bound we have here is the same as the one we have for irrevocable priority algorithms, but we do not believe these two classes are equivalent. This result is an improvement for the non-increasing order irrevocable priority algorithms mentioned in Section 1.1.

Theorem 4.2.1 *No non-increasing order revocable priority algorithm of SSP can achieve approximation ratio better than α_1.*

Proof: We show that for any non-increasing order revocable priority algorithm and every small $\epsilon > 0$, $\rho < \alpha_1 + \epsilon$. Let

$$u_1 = \alpha_1 \approx 0.657, \qquad u_2 = 2\alpha_1^3 + \alpha_1^2\epsilon \approx 0.568,$$
$$u_3 = \alpha_1^2 \approx 0.432, \qquad u_4 = \alpha_1^2 - \alpha_1^2\epsilon \approx 0.432;$$

we utilize the following adversarial strategy shown in Fig. 4.1.

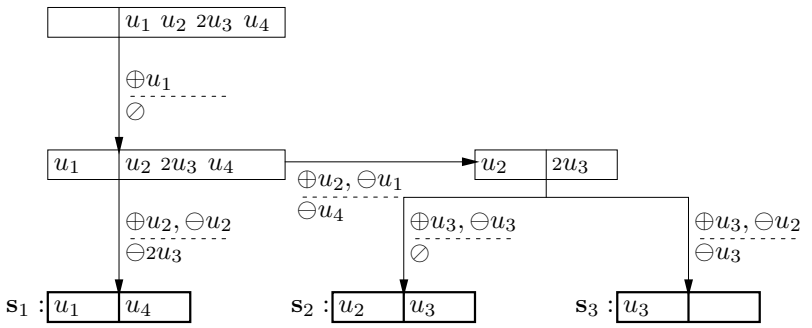

Figure 4.1: Adversarial strategy for fixed non-increasing order.

If the algorithm terminates via state $\mathbf{s}_1$, then $\|\mathbf{ALG}\| \leq u_1$. Note that, by equation (3.1), we have $u_2 + u_4 = 2\alpha_1^3 + \alpha_1^2 = 1$, hence $\|\mathbf{OPT}\| = u_2 + u_4$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} \leq \frac{u_1}{u_2 + u_4} = \frac{\alpha_1}{2\alpha_1^3 + \alpha_1^2} = \alpha_1.$$

If the algorithm terminates via state $\mathbf{s}_2$, then $\|\mathbf{ALG}\| \leq u_2$ and $\|\mathbf{OPT}\| = 2u_3$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} \leq \frac{u_2}{2u_3} = \frac{2\alpha_1^3 + \alpha_1^2\epsilon}{2\alpha_1^2} = \alpha_1 + \frac{1}{2}\epsilon.$$

If the algorithm terminates via state $\mathbf{s}_3$, then $\|\mathbf{ALG}\| = u_3$ and $\|\mathbf{OPT}\| = u_1$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} = \frac{u_3}{u_1} = \frac{\alpha_1^2}{\alpha_1} = \alpha_1.$$

In all three cases, the adversary forces the algorithm to have approximation ratio $\rho < \alpha_1 + \epsilon$; this completes the proof. □

Description of the Algorithm

Next, we give a non-increasing order revocable priority algorithm that achieves the approximation ratio α_1. Similar to Section 3.2, we partition all possible relevant data items into two sets: **M** and **L**. A data item u is said to be in class **M** and **L** if $1 - \alpha_1 < u \leq \alpha_1^2$ and $\alpha_1^2 < u < \alpha_1$ respectively; see Fig. 4.2.

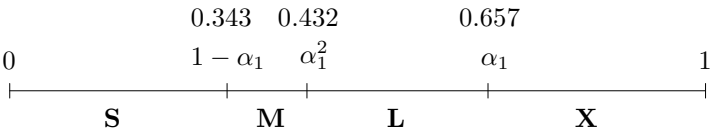

Figure 4.2: Classification for non-increasing order.

Since the approximation ratio $\alpha_1 > \frac{1}{2}$, the algorithm has to be selective on what data items to keep in the solution, it cannot always keep the largest data item and hope to improve it, as there exists the possibility of two smaller items, none of which fits in the remaining space, but has a better total weight.

Algorithm 2

1: $\mathbf{B} := \emptyset$;
2: $\mathbf{B}_{\text{mode}} :=$ `Queue`;
3: **for** $i := 1$ to n **do**
4: let u_i be the next data item in $\mathbf{I}$;
5: $\mathbf{I} := \mathbf{I} \ominus u_i$;
6: **if** $\|\mathbf{B}\| + u_i \leq 1$ **then**
7: accept u_i;
8: **else if** $u_i \in \mathbf{L}$ **then**
9: accept u_i;
10: **else**
11: reject u_i;
12: **end if**
13: **end for**

Proof of the Upper Bound

Theorem 4.2.2 *There exists a non-increasing order revocable priority algorithm of SSP with approximation ratio α_1.*

Proof: It is sufficient to show that Alg. 2 achieves this approximation ratio. We prove this by induction on i. At the end of the i^{th} iteration of the **for** loop, we let $\mathbf{ALG}_i$ and $\mathbf{OPT}_i$ be the algorithm's solution and the optimal solution respectively; these are solutions with respect to the first i data items $\{u_1, \ldots, u_i\}$. We also let ρ_i to be the current approximation ratio,

$$\rho_i = \frac{\|\mathbf{ALG}_i\|}{\|\mathbf{OPT}_i\|}.$$

It is clear that at the end of the first iteration, the algorithm has approximation ratio $\rho_1 = 1 > \alpha_1$. Suppose there is no early termination and the α_1-approximation is maintained for the first $k-1$ iterations, i.e., $\|\mathbf{ALG}_i\| < \alpha_1$ and $\rho_i \geq \alpha_1$ for all $i < k$; we examine the case for $i = k$, $k \geq 2$. Note that $|\mathbf{ALG}_i| = 1$ for all $i < k$, for otherwise $\|\mathbf{ALG}_i\| \geq \alpha_1$. The algorithm examines data items in non-increasing order; for data items in $\mathbf{L}$, it always accepts the current one into the solution, no matter whether or not the data item fits; for data items in $\mathbf{M}$, it always rejects the current one if it does not fit. In that sense, it tends to keep the smallest data item in $\mathbf{L} \cap \mathbf{I}$ and the largest data item in $\mathbf{M} \cap \mathbf{I}$. There are only three possibilities for u_k during the k^{th} iteration; it must fall into one of the lines 6, 8 and 10:

1. **Line 6:** then the algorithm accepts u_k into the solution without discarding any accepted data item in the solution. Hence we have $|\mathbf{ALG}_i| = 2$, the terminal condition is satisfied. Therefore, the algorithm terminates and achieves approximation ratio α_1.

2. **Line 8:** then at the end of $(k-1)^{\text{th}}$ iteration, $\mathbf{ALG}_{k-1}$ contains only u_{k-1}. Since the terminal condition is not satisfied during the k^{th} iteration, we have $u_{k-1} + u_k > 1$. Therefore, the algorithm accepts u_k into the solution by discarding u_{k-1}. Note that u_k and u_{k-1} are the two smallest data items examined so far and $u_{k-1} + u_k > 1$, hence $\mathbf{OPT}_k$

contains exactly one data item, say u_r, in $\mathbf{L}$. Therefore, the approximation ratio is

$$\rho_k = \frac{\|\mathbf{ALG}_k\|}{\|\mathbf{OPT}_k\|} = \frac{u_k}{u_r} \geq \frac{\alpha_1^2}{\alpha_1} = \alpha_1.$$

3. **Line 10:** then $u_k \in \mathbf{M}$ and the algorithm rejects u_k, therefore $\mathbf{ALG}_k = \mathbf{ALG}_{k-1}$. Furthermore, at the end of $(k-1)^{\text{th}}$ iteration, $\mathbf{ALG}_{k-1}$ contains exactly one data item. Observe that it cannot be a data item in $\mathbf{M}$; for otherwise, the terminal condition is satisfied. Therefore, $\mathbf{ALG}_{k-1}$ contains exactly one data item, say u_j, in $\mathbf{L}$. Since the terminal condition is not satisfied during the k^{th} iteration, we have $u_k + u_j > 1$; then by equation (3.1),

$$\|\mathbf{ALG}_k\| = \|\mathbf{ALG}_{k-1}\| = u_j > 1 - u_k \geq 1 - \alpha_1^2 = 2\alpha_1^3.$$

Note that for all the data items examined so far, u_k is the smallest in $\mathbf{M}$, u_j is the smallest in $\mathbf{L}$, and $u_k + u_j > 1$, hence $\mathbf{OPT}_k$ can either contain one data item in $\mathbf{L}$ or two data items in $\mathbf{M}$; in both cases $\|\mathbf{OPT}_k\| < 2\alpha_1^2$. Therefore, the approximation ratio is

$$\rho_k = \frac{\|\mathbf{ALG}_k\|}{\|\mathbf{OPT}_k\|} > \frac{2\alpha_1^3}{2\alpha_1^2} = \alpha_1.$$

We now have shown that in all three cases, the algorithm maintains the approximation ratio $\rho_i \geq \alpha_1$ for $i = k$; this completes the induction. Therefore, Alg. 2 achieves approximation ratio α_1. □

Theorem 4.2.2 shows that, with the aid of revocable acceptance, we can improve the non-increasing approximation ratio from $\frac{1}{2}$ to $\alpha_1 \approx 0.657$.

4.2.2 Non-Decreasing Order

One often tends to believe that non-increasing order is advantageous as it allows early exposure of large data items, but it is not the case for fixed revocable priority algorithms. This is because large data items, while not large enough to achieve the approximation ratio, need to take more

space; in other words, a fixed capacity takes less data items with large weight. Therefore, at the time of an overflow, there are less data items in the solution, and hence less choices as to which data items are rejected. Let $\alpha_2 = \frac{\sqrt{17}-1}{4} \approx 0.780$ be the real root of the equation

$$2x^2 + x - 2 = 0 \tag{4.1}$$

between 0 and 1. In this subsection, we prove a tight bound of α_2 on the approximation ratio for non-decreasing order.

Theorem 4.2.3 *No non-decreasing order revocable priority algorithm of SSP can achieve approximation ratio better than α_2.*

Proof: We show that for any non-decreasing order revocable priority algorithm and every small $\epsilon > 0$, $\rho < \alpha_2 + \epsilon$. Let

$$u_1 = 1 - \alpha_2 \approx 0.220, \quad u_2 = \tfrac{1}{2}\alpha_2 + \tfrac{1}{4}\epsilon \approx 0.390, \quad u_3 = \alpha_2 \approx 0.780;$$

we utilize the following adversarial strategy shown in Fig. 4.3.

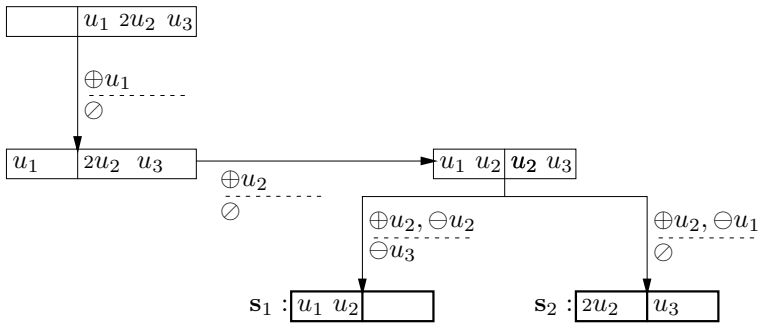

Figure 4.3: Adversarial strategy for fixed non-decreasing order.

If the algorithm terminates via state $\mathbf{s}_1$, then $\|\mathbf{ALG}\| = u_1 + u_2$ and $\|\mathbf{OPT}\| = 2u_2$. Note that, by equation (4.1), we have $u_1 + u_2 = 1 - \frac{1}{2}\alpha_2 + \frac{1}{4}\epsilon = \alpha_2^2 + \frac{1}{4}\epsilon$. Therefore, the approximation

ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} = \frac{u_1 + u_2}{2u_2} = \frac{1 - \frac{1}{2}\alpha_2 + \frac{1}{4}\epsilon}{\alpha_2 + \frac{1}{2}\epsilon} = \frac{\alpha_2^2 + \frac{1}{4}\epsilon}{\alpha_2 + \frac{1}{2}\epsilon} < \alpha_2.$$

If the algorithm terminates via state $\mathbf{s}_2$, then $\|\mathbf{ALG}\| \leq 2u_2$. Since $u_1 + u_3 = 2 = 1$, we have $\|\mathbf{OPT}\| = u_1 + u_3$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} \leq \frac{2u_2}{u_1 + u_3} = \alpha_2 + \frac{1}{2}\epsilon.$$

In both cases, the adversary forces the algorithm to have approximation ratio $\rho < \alpha_2 + \epsilon$; this completes the proof. □

Description of the Algorithm

Next, we give a non-decreasing order revocable priority algorithm that achieves the approximation ratio α_2. It is interesting that non-decreasing order can, to a large extent, improve the approximation ratio. As we show later in this thesis, no fixed priority revocable algorithm can achieve approximation ratio better than $\beta_1 \approx 0.852$; considering the fact that $\alpha_2 \approx 0.780$ and all previous priority algorithms studied have approximation ratios under the $\frac{2}{3}$ mark, this is a significant improvement.

We now describe the algorithm. For all possible relevant data items, we partition them into two sets: **M** and **L**. A data item u is said to be in class **M** and **L** if $1 - \alpha_2 < u \leq \frac{1}{2}\alpha_2$ and $\frac{1}{2}\alpha_2 < u < \alpha_2$ respectively; see Fig. 4.4.

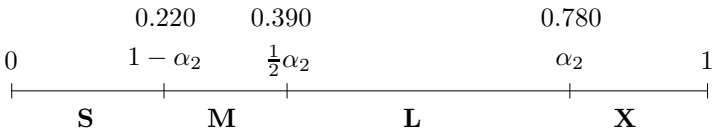

Figure 4.4: Classification for non-decreasing order.

The algorithm achieves this ratio is surprisingly simple; it basically set $\mathbf{B}_{\text{mode}}$ to `Stack` mode, and accepts each data item in turn; see Alg. 3.

Algorithm 3

1: $\mathbf{B} := \emptyset$;

2: $\mathbf{B}_{\text{mode}} :=$ `Stack`;

3: **for** $i := 1$ to n **do**

4: let u_i be the next data item in $\mathbf{I}$;

5: $\mathbf{I} := \mathbf{I} \ominus u_i$;

6: accept u_i;

7: **end for**

Proof of the Upper Bound

Theorem 4.2.4 *There exists a non-decreasing order revocable priority algorithm of SSP with approximation ratio α_2.*

Proof: It is sufficient to show that Alg. 3 achieves this approximation ratio. We prove this by induction on i. At the end of the i^{th} iteration of the **for** loop, we let $\mathbf{ALG}_i$ and $\mathbf{OPT}_i$ be the algorithm's solution and the optimal solution respectively; and let ρ_i be the current approximation ratio. It is clear that at the end of the first iteration, the algorithm has approximation ratio $\rho_1 = 1 > \alpha_2$. Suppose there is no early termination and the approximation ratio $\rho_i \geq \alpha_2$ is maintained for all $i < k$, we examine the case for $i = k$. There are two possibilities. The first case is that $\|\mathbf{B}\| + u_k \leq 1$, then the algorithm accepts u_k into the solution without discarding any accepted data item in the solution. Hence we have $\|\mathbf{OPT}_k\| \geq \|\mathbf{OPT}_{k-1}\| + u_k$; for otherwise, we can take $\mathbf{OPT}_{k-1} \oplus u_k$ as $\mathbf{OPT}_k$, which leads to a contradiction. Therefore, the approximation ratio is

$$\rho_k = \frac{\|\mathbf{ALG}_k\|}{\|\mathbf{OPT}_k\|} \geq \frac{\|\mathbf{ALG}_{k-1}\| + u_k}{\|\mathbf{OPT}_{k-1}\| + u_k} \geq \frac{\|\mathbf{ALG}_{k-1}\|}{\|\mathbf{OPT}_{k-1}\|} = \rho_{k-1} \geq \alpha_2.$$

This leaves the case when $\|\mathbf{B}\| + u_k > 1$. Note that the algorithm examines data items in non-decreasing order; u_1 is the smallest data item in $\mathbf{I}$ and u_2 is the second smallest. These

two data items are at the bottom of the stack, hence tend to stay in the stack unless we see a data item with sufficient large weight:

- **Observation 1:** if at certain stage of the algorithm, after accepting u_k, there is only one data item $u_k \neq u_1$ in the stack, then we have $u_1 + u_k > 1$; see Fig. 4.5.

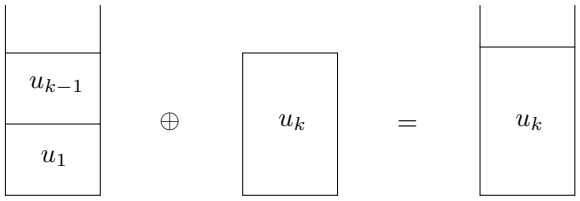

Figure 4.5: Stack with one data item.

- **Observation 2:** if at certain stage of the algorithm, after accepting u_k, there are only two data items in the stack and the top of the stack is $u_k \neq u_2$, then we have $u_1 + u_2 + u_k > 1$; see Fig. 4.6.

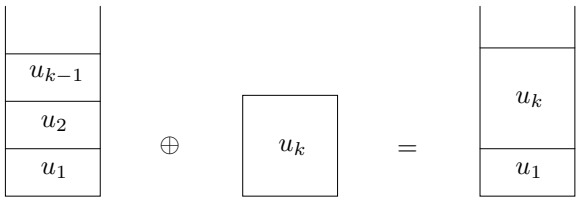

Figure 4.6: Stack with two data items.

The above two inequalities happen to be useful in our analysis. Now we consider data items in $\textbf{OPT}_k$; there are six cases:

1. If $|\textbf{OPT}_k| = 1$, then $\textbf{OPT}_k = \{u_k\}$. Since the algorithm always accepts the current data item, we have $u_k \in \textbf{ALG}_k$, and hence $\textbf{ALG}_k = \textbf{OPT}_k$. Therefore, the approximation ratio

is

$$\rho_k = \frac{\|\mathbf{ALG}_k\|}{\|\mathbf{OPT}_k\|} = 1.$$

2. If $\mathbf{OPT}_k$ contains exactly two data items $u_r, u_s \in \mathbf{M}$ with $u_r \leq u_s$, then we have both u_1 and u_2 in M; we now consider data items in $\mathbf{ALG}_k$.

   (a) If $|\mathbf{ALG}_k| = 1$, then $\mathbf{ALG}_k = \{u_k\}$; by Observation 1, we have

   $$u_1 + u_k > 1.$$

   Note that $u_r + u_s \leq \alpha_2$ and by equation (4.1), we have $1 - u_1 \geq 1 - \frac{1}{2}\alpha_2 = \alpha_2^2$. Therefore, the approximation ratio is

   $$\rho_k = \frac{\|\mathbf{ALG}_k\|}{\|\mathbf{OPT}_k\|} = \frac{u_k}{u_r + u_s} > \frac{1 - u_1}{u_r + u_s} \geq \frac{\alpha_2^2}{\alpha_2} = \alpha_2.$$

   (b) If $|\mathbf{ALG}_k| = 2$, then $\mathbf{ALG}_k = \{u_1, u_k\}$; by Observation 2, we have

   $$u_1 + u_2 + u_k > 1.$$

   Note that $u_r + u_s \leq \alpha_2$ and by equation (4.1), we have $1 - u_2 \geq 1 - \frac{1}{2}\alpha_2 = \alpha_2^2$. Therefore, the approximation ratio is

   $$\rho_k = \frac{\|\mathbf{ALG}_k\|}{\|\mathbf{OPT}_k\|} = \frac{u_1 + u_k}{u_r + u_s} > \frac{1 - u_2}{u_r + u_s} \geq \frac{\alpha_2^2}{\alpha_2} = \alpha_2.$$

   (c) If $|\mathbf{ALG}_k| \geq 3$, then $\{u_1, u_2, u_k\} \in \mathbf{ALG}$. Note that $u_1 + u_2 > 2 - 2\alpha_2$. Therefore, the approximation ratio is

   $$\rho_k = \frac{\|\mathbf{ALG}_k\|}{\|\mathbf{OPT}_k\|} \geq \frac{u_1 + u_2 + u_k}{u_r + u_s} > \frac{2 - 2\alpha_2 + u_s}{\frac{1}{2}\alpha_2 + u_s} > 1,$$

   which is a contradiction; this case is impossible.

3. If $\mathbf{OPT}_k$ contains exactly two data items $u_r \in \mathbf{M}$ and $u_s \in \mathbf{L}$, then we have u_1 in M; we now consider data items in $\mathbf{ALG}_k$.

(a) If $|\mathbf{ALG}_k| = 1$, then $\mathbf{ALG}_k = \{u_k\}$; by Observation 1, we have

$$u_1 + u_k \quad > 1. \tag{4.2}$$

Note that $u_1 + u_s \leq u_r + u_s \leq 1$. When the algorithm considers u_s, u_1 is already in the solution. Since the terminal condition is not satisfied, we have

$$u_1 + u_s \quad < \alpha_2. \tag{4.3}$$

Combining (4.2) and (4.3), we have $u_k > 1 - \alpha_2 + u_s$. Therefore,

$$\rho_k = \frac{\|\mathbf{ALG}_k\|}{\|\mathbf{OPT}_k\|} = \frac{u_k}{u_r + u_s} > \frac{1 - \alpha_2 + u_s}{\frac{1}{2}\alpha_2 + u_s} > \frac{1 - \alpha_2 + \frac{1}{2}\alpha_2}{\frac{1}{2}\alpha_2 + \frac{1}{2}\alpha_2} = \frac{1 - \frac{1}{2}\alpha_2}{\alpha_2}.$$

By equation (4.1), the approximation ratio is

$$\rho_k > \frac{1 - \frac{1}{2}\alpha_2}{\alpha_2} = \frac{\alpha_2^2}{\alpha_2} = \alpha_2.$$

(b) If $|\mathbf{ALG}_k| = 2$, then $\mathbf{ALG}_k = \{u_1, u_k\}$. Therefore

$$\rho_k = \frac{\|\mathbf{ALG}_k\|}{\|\mathbf{OPT}_k\|} = \frac{u_1 + u_k}{u_r + u_s} > \frac{1 - \alpha_2 + u_s}{\frac{1}{2}\alpha_2 + u_s} > \frac{1 - \alpha_2 + \frac{1}{2}\alpha_2}{\frac{1}{2}\alpha_2 + \frac{1}{2}\alpha_2} = \frac{1 - \frac{1}{2}\alpha_2}{\alpha_2}.$$

By equation (4.1), the approximation ratio is

$$\rho_k > \frac{1 - \frac{1}{2}\alpha_2}{\alpha_2} = \frac{\alpha_2^2}{\alpha_2} = \alpha_2.$$

(c) If $|\mathbf{ALG}_k| \geq 3$, then $\{u_1, u_2, u_k\} \in \mathbf{ALG}$. Note that $u_1 + u_2 > 2 - 2\alpha_2$. Therefore, the approximation ratio is

$$\rho_k = \frac{\|\mathbf{ALG}_k\|}{\|\mathbf{OPT}_k\|} \geq \frac{u_1 + u_2 + u_k}{u_r + u_s} > \frac{2 - 2\alpha_2 + u_s}{\frac{1}{2}\alpha_2 + u_s} > 1,$$

which is a contradiction; this case is impossible.

4. If $\mathbf{OPT}_k$ contains exactly two data items $u_r, u_s \in \mathbf{L}$ with $u_r \leq u_s$, then we have

$$\alpha_2 \leq u_r + u_{r+1} \leq u_r + u_s \leq 1.$$

When the algorithm considers u_{r+1}, u_r is already in the solution; hence the terminal condition is satisfied.

5. If $\mathbf{OPT}_k$ contains exactly three data items $u_r, u_s, u_t \in \mathbf{M}$ with $u_r \leq u_s \leq u_t$, then we have both u_1 and u_2 in M; we now consider data items in $\mathbf{ALG}_k$.

   (a) If $|\mathbf{ALG}_k| = 1$, then $\mathbf{ALG}_k = \{u_k\}$; by Observation 1, we have

   $$u_1 + u_k \quad > 1. \tag{4.4}$$

   Note that $u_1 + u_r + u_s \leq u_1 + u_s + u_{s+1} \leq u_r + u_s + u_t \leq 1$. When the algorithm considers u_{s+1}, u_1 and u_s are already in the solution. Since the terminal condition is not satisfied, we have $u_1 + u_s + u_{s+1} < \alpha_2$. Therefore,

   $$u_1 + u_r + u_s \quad < \alpha_2. \tag{4.5}$$

   Combining (4.4) and (4.5), we have $u_k > 1 - \alpha_2 + u_r + u_s$. Since both u_r and u_s are in $\mathbf{M}$, we have $u_r + u_s > 2 - 2\alpha_2 > \frac{1}{2}\alpha_2$. Therefore,

   $$\rho_k = \frac{\|\mathbf{ALG}_k\|}{\|\mathbf{OPT}_k\|} = \frac{u_k}{u_r + u_s + u_t} > \frac{1 - \alpha_2 + (u_r + u_s)}{\frac{1}{2}\alpha_2 + (u_r + u_s)} > \frac{1 - \alpha_2 + \frac{1}{2}\alpha_2}{\frac{1}{2}\alpha_2 + \frac{1}{2}\alpha_2} = \frac{1 - \frac{1}{2}\alpha_2}{\alpha_2}.$$

   By equation (4.1), the approximation ratio is

   $$\rho_k > \frac{1 - \frac{1}{2}\alpha_2}{\alpha_2} = \frac{\alpha_2^2}{\alpha_2} = \alpha_2.$$

   (b) If $|\mathbf{ALG}_k| = 2$, then $\mathbf{ALG}_k = \{u_1, u_k\}$; by Observation 2, we have

   $$u_1 + u_2 + u_k \quad > 1. \tag{4.6}$$

   - If $r = 1$, then $u_2 + u_r + u_s \leq u_1 + u_2 + u_{s+1} \leq u_r + u_s + u_t \leq 1$. When the algorithm considers u_{s+1}, u_1 and u_2 are already in the solution. Since the terminal condition is not satisfied, we have $u_1 + u_2 + u_{s+1} < \alpha_2$. Therefore,

     $$u_2 + u_r + u_s \quad < \alpha_2. \tag{4.7}$$

   - If $r > 1$, then $u_2 + u_r + u_s \leq u_2 + u_s + u_{s+1} \leq u_r + u_s + u_t \leq 1$. When the algorithm considers u_{s+1}, u_2 and u_s are already in the solution. Since the terminal condition is not satisfied, we have $u_2 + u_s + u_{s+1} < \alpha_2$. Therefore, the inequality (4.7) is also satisfied.

Therefore, in both cases we have (4.7). Combining (4.6) and (4.7), we have $u_1+u_k > 1-\alpha_2+u_r+u_s$. Since both u_r and u_s are in $\mathbf{M}$, we have $u_r+u_s > 2-2\alpha_2 > \frac{1}{2}\alpha_2$. Therefore,

$$\rho_k = \frac{\|\mathbf{ALG}_k\|}{\|\mathbf{OPT}_k\|} = \frac{u_1+u_k}{u_r+u_s+u_t} > \frac{1-\alpha_2+(u_r+u_s)}{\frac{1}{2}\alpha_2+(u_r+u_s)} > \frac{1-\alpha_2+\frac{1}{2}\alpha_2}{\frac{1}{2}\alpha_2+\frac{1}{2}\alpha_2} = \frac{1-\frac{1}{2}\alpha_2}{\alpha_2}.$$

By equation (4.1), the approximation ratio is

$$\rho_k > \frac{1-\frac{1}{2}\alpha_2}{\alpha_2} = \frac{\alpha_2^2}{\alpha_2} = \alpha_2.$$

(c) If $|\mathbf{ALG}_k| \geq 3$, then $\{u_1, u_2, u_k\} \subseteq \mathbf{ALG}$. Note that $u_1+u_r+u_s \leq u_1+u_s+u_{s+1} \leq u_r+u_s+u_t \leq 1$. When the algorithm considers u_{s+1}, u_1 and u_s are already in the solution. Since the terminal condition is not satisfied, we have $u_1+u_s+u_{s+1} < \alpha_2$. Therefore,

$$u_1+u_r+u_s \quad < \alpha_2. \tag{4.8}$$

By (4.8), we have $u_r+u_s < \alpha_2-u_1 < \alpha_2-(1-\alpha_2) = 2\alpha_2-1$. Note that $u_1+u_2 > 2-2\alpha_2$. Therefore,

$$\rho_k = \frac{\|\mathbf{ALG}_k\|}{\|\mathbf{OPT}_k\|} \geq \frac{u_1+u_2+u_k}{u_r+u_s+u_t} > \frac{2-2\alpha_2+u_t}{2\alpha_2-1+u_t} > \frac{2-2\alpha_2+1-\alpha_2}{2\alpha_2-1+1-\alpha_2} = \frac{3-3\alpha_2}{\alpha_2}.$$

Since $3-3\alpha_2 > \alpha_2^2$, the approximation ratio is

$$\rho_k > \frac{3-3\alpha_2}{\alpha_2} > \alpha_2.$$

6. If $\mathbf{OPT}_k$ contains exactly three data items $u_r, u_s \in \mathbf{M}$ and $u_t \in \mathbf{L}$ with $u_r \leq u_s$, then we have

$$\alpha_2 < 2-2\alpha_2+\frac{1}{2}\alpha_2 < u_1+u_2+u_t \leq u_r+u_s+u_t \leq 1.$$

When the algorithm considers u_t, u_1 and u_2 are already in the solution; hence the terminal condition is satisfied.

We now have shown that in all six cases, the algorithm maintains the approximation ratio $\rho_i \geq \alpha_2$ for $i = k$; this completes the induction. Therefore, Alg. 3 achieves approximation ratio α_2. □

Theorem 4.2.4 shows the existence of a simple non-decreasing order revocable priority algorithm which achieves the approximation ratio $\alpha_2 \approx 0.780$. This bound is better than any "greedy" algorithm discussed so far in this thesis, and it surpasses the multi-pass greedy algorithm by Martello and Toth mentioned in Section 1.1.

4.2.3 A General Lower Bound

Let $\beta_1 = \frac{\sqrt{145}+5}{20} \approx 0.852$ be the real root of the equation

$$10x^2 - 5x - 3 = 0 \tag{4.9}$$

between 0 and 1. In this subsection, we prove a lower bound of β_1 on the approximation ratio for general fixed revocable priority algorithms.

Theorem 4.2.5 *No fixed revocable priority algorithm of SSP can achieve approximation ratio better than* β_1.

Proof: Let

$$u_1 = \tfrac{1}{5} = 0.200, \qquad u_2 = \tfrac{1}{2}\beta_1 - \tfrac{1}{10} \approx 0.326,$$
$$u_3 = \tfrac{1}{2} = 0.500, \qquad u_4 = \tfrac{4}{5} = 0.800;$$

For a data item u, we denote by $rank(u)$ its priority assigned by algorithm $\mathcal{A}$. There are four cases:

1. If $rank(u_2) > rank(u_1)$, then we utilize the following adversarial strategy shown in Fig. 4.7; we omit the starting few states where overflow does not occur.

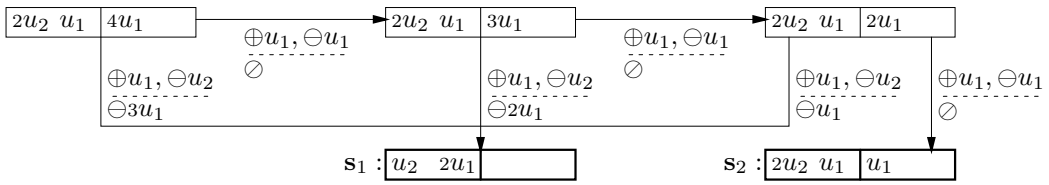

Figure 4.7: Adversarial strategy for $rank(u_2) > rank(u_1)$.

If the algorithm terminates via state $\mathbf{s}_1$, then $\|\mathbf{ALG}\| = 2u_1 + u_2$ and $\|\mathbf{OPT}\| \geq u_1 + 2u_2$. Therefore,

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} \leq \frac{2u_1 + u_2}{u_1 + 2u_2} = \frac{\frac{2}{5} + \frac{1}{2}\beta_1 - \frac{1}{10}}{\frac{1}{5} + \beta_1 - \frac{1}{5}} = \frac{\frac{1}{2}\beta_1 + \frac{3}{10}}{\beta_1} = \frac{5\beta_1 + 3}{10\beta_1}.$$

By equation (4.9), the approximation ratio is

$$\rho \leq \frac{5\beta_1 + 3}{10\beta_1} = \beta_1.$$

If the algorithm terminates via state $\mathbf{s}_2$, then $\|\mathbf{ALG}\| \leq u_1 + 2u_2$ and $\|\mathbf{OPT}\| = 5u_1$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} \leq \frac{u_1 + 2u_2}{5u_1} = u_1 + 2u_2 = \frac{1}{5} + \beta_1 - \frac{1}{5} = \beta_1.$$

Therefore, if $\mathcal{A}$ ranks u_2 before u_1, then the adversary can force it to have approximation ratio no better than β_1.

2. If $rank(u_3) > rank(u_2)$, then we utilize the following adversarial strategy shown in Fig. 4.8; we omit the starting few states where overflow does not occur.

If the algorithm terminates via state $\mathbf{s}_1$, then $\|\mathbf{ALG}\| = 2u_2$ and $\|\mathbf{OPT}\| = u_2 + u_3$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} = \frac{2u_2}{u_2 + u_3} = \frac{\beta_1 - \frac{1}{5}}{\frac{1}{2} + \frac{1}{2}\beta_1 - \frac{1}{10}} = \frac{10\beta_1 - 2}{5\beta_1 + 4} < \beta_1.$$

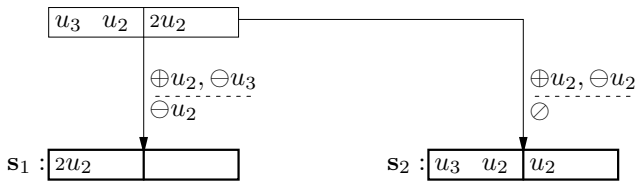

Figure 4.8: Adversarial strategy for $rank(u_3) > rank(u_2)$.

If the algorithm terminates via state $\mathbf{s}_2$, then $\|\mathbf{ALG}\| \leq u_2 + u_3$ and $\|\mathbf{OPT}\| = 3u_2$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} \leq \frac{u_2 + u_3}{3u_2} = \frac{\frac{1}{2} + \frac{1}{2}\beta_1 - \frac{1}{10}}{\frac{3}{2}\beta_1 - \frac{3}{10}} = \frac{5\beta_1 + 4}{15\beta_1 - 3} < \beta_1.$$

Therefore, if $\mathcal{A}$ ranks u_3 before u_2, then the adversary can force it to have approximation ratio no better than β_1.

3. If $rank(u_4) > rank(u_3)$, then we utilize the following adversarial strategy shown in Fig. 4.9; we omit the starting few states where overflow does not occur.

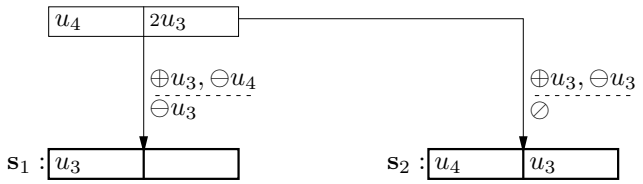

Figure 4.9: Adversarial strategy for $rank(u_4) > rank(u_3)$.

If the algorithm terminates via state $\mathbf{s}_1$, then $\|\mathbf{ALG}\| = u_3$ and $\|\mathbf{OPT}\| = u_4$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} = \frac{u_3}{u_4} < \beta_1.$$

If the algorithm terminates via state $\mathbf{s}_2$, then $\|\mathbf{ALG}\| \leq u_4$ and $\|\mathbf{OPT}\| = 2u_3$. Therefore,

the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} \leq \frac{u_4}{2u_3} = u_4 < \beta_1.$$

Therefore, if $\mathcal{A}$ ranks u_4 before u_3, then the adversary can force it to have approximation ratio no better than β_1.

4. If $rank(u_1) > rank(u_2) > rank(u_3) > rank(u_4)$, then we utilize the following adversarial strategy shown in Fig. 4.10; we omit the starting few states where overflow does not occur.

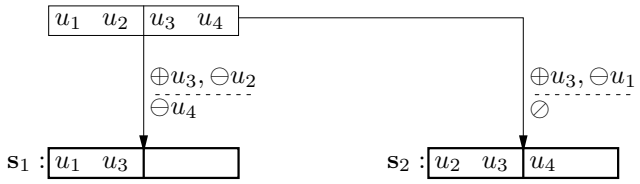

Figure 4.10: Adversarial strategy for $rank(u_1) > rank(u_2) > rank(u_3) > rank(u_4)$.

If the algorithm terminates via state $\mathbf{s}_1$, then $\|\mathbf{ALG}\| = u_1 + u_3$ and $\|\mathbf{OPT}\| = u_2 + u_3$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} = \frac{u_1 + u_3}{u_2 + u_3} = \frac{\frac{1}{5} + \frac{1}{2}}{\frac{1}{2}\beta_1 - \frac{1}{10} + \frac{1}{2}} = \frac{7}{5\beta_1 + 4} < \beta_1.$$

If the algorithm terminates via state $\mathbf{s}_2$, then $\|\mathbf{ALG}\| \leq u_2 + u_3$ and $\|\mathbf{OPT}\| = u_1 + u_4$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} \leq \frac{u_2 + u_3}{u_1 + u_4} = u_2 + u_3 = \frac{1}{2}\beta_1 - \frac{1}{10} + \frac{1}{2} < \beta_1.$$

Therefore if $\mathcal{A}$ ranks $rank(u_1) > rank(u_2) > rank(u_3) > rank(u_4)$, then the adversary can force it to have approximation ratio no better than β_1.

As a conclusion, no fixed revocable priority algorithm of SSP can achieve approximation ratio better than β_1. This completes the proof. □

This lower bound $\beta_1 \approx 0.852$ for fixed revocable priority algorithms of SSP is interesting as it shows the first gap we are unable to close. The analysis above is relatively tight in cases 1, 2 and 4, and there seems not much room left for improvement. Intuitively, though, it is quite promising that this gap can be closed, but this may require more sophisticated adversaries, which might involve a large set of data items, and more carefully designed algorithms.

4.3 Adaptive Revocable Priority Algorithms

Adaptive priority algorithms are more powerful than fixed priority algorithms in the sense that remaining data items can be reordered at each iteration. In this section, we provide an adaptive revocable priority algorithm of SSP which has a better approximation ratio than the fixed revocable priority algorithms discussed in the previous section. We also prove a lower bound on the approximation ratio for the adaptive case.

4.3.1 Deriving a Lower Bound

Let $\beta_2 = \frac{\sqrt{19}+1}{6} \approx 0.893$ be the real root of the equation

$$6x^2 - 2x - 3 = 0 \tag{4.10}$$

between 0 and 1. In this subsection, we prove a lower bound of β_2 on the approximation ratio for adaptive revocable priority algorithms.

Theorem 4.3.1 *No adaptive revocable priority algorithm of SSP can achieve approximation ratio better than β_2.*

Proof: Suppose there exists an algorithm $\mathcal{A}$ such that $\rho > \beta_2$. Let

$$u_1 = \tfrac{1}{3}\beta_2 \approx 0.298, \qquad u_3 = \tfrac{1}{2} = 0.500;$$

We utilize the following adversary strategy shown in Fig. 4.11.

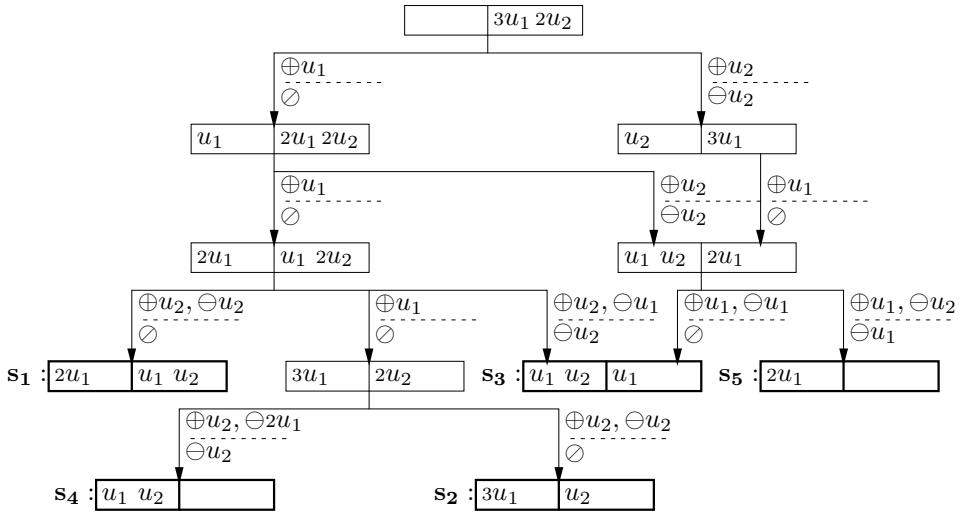

Figure 4.11: Adversarial strategy for adaptive revocable priority algorithms.

If the algorithm terminates via state $\mathbf{s}_1$ or $\mathbf{s}_2$, then $\|\mathbf{ALG}\| \leq 3u_1$ and $\|\mathbf{OPT}\| = 2u_2$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} \leq \frac{3u_1}{2u_2} = 3u_1 = \beta_2.$$

If the algorithm terminates via state $\mathbf{s}_3$ or $\mathbf{s}_4$, then $\|\mathbf{ALG}\| \leq u_1 + u_2$ and $\|\mathbf{OPT}\| = 3u_1$. Therefore,

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} \leq \frac{u_1 + u_2}{3u_1} = \frac{\frac{1}{3}\beta_2 + \frac{1}{2}}{\beta_2} = \frac{2\beta_2 + 3}{6\beta_2}.$$

By equation (4.10), the approximation ratio is

$$\rho \leq \frac{2\beta_2 + 3}{6\beta_2} = \frac{6\beta_2^2}{6\beta_2} = \beta_2.$$

If the algorithm terminates via state $\mathbf{s}_5$, then $\|\mathbf{ALG}\| = 2u_1$ and $\|\mathbf{OPT}\| = u_1 + u_2$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} = \frac{2u_1}{u_1 + u_2} = \frac{\frac{2}{3}\beta_2}{\frac{1}{3}\beta_2 + \frac{1}{2}} = \frac{4\beta_2}{2\beta_2 + 3} < \beta_2.$$

In all three cases, the adversary forces the algorithm $\mathcal{A}$ to have approximation ratio no better than β_2; this completes the proof. □

4.3.2 A Better Algorithm

In this subsection, we give an adaptive revocable priority algorithm that achieves the approximation ratio of $\alpha_3 = 0.8$. This is the most involved algorithm in this thesis. Similar to previous cases, we partition all possible relevant data items into sets: **M** and **L**. A data item u is said to be in class **M** and **L** if $0.2 < u \leq 0.4$ and $0.4 < u < 0.8$ respectively; see Fig. 4.12.

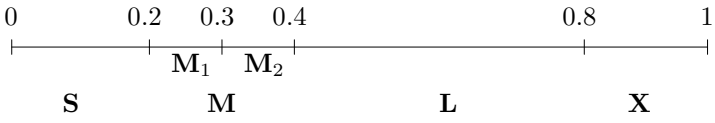

Figure 4.12: Classification for the adaptive revocable priority algorithm.

For notational convenience, we further divide the class **M** into two sets. A data item u is said to be in class $\mathbf{M}_1$, $\mathbf{M}_2$ if $0.2 < u \leq 0.3$, $0.3 < u \leq 0.4$ respectively. We call our algorithm ADAPTIVE, and use the following symbols to denote data items in different classes; see Table 4.1

Table 4.1: Corresponding symbols.

class	**M**	**L**	$\mathbf{M}_1$	$\mathbf{M}_2$
symbol	◊	⧫	▽	△

The algorithm ADAPTIVE includes two stages; data items in **M** are processed during the first stage and data items in **L** are processed during the second stage. The ordering of data items (in each stage) is determined by its distance to the middle point of **M**, i.e., the closer a data item to 0.3, the higher its priority is. The algorithm is described below.

Algorithm ADAPTIVE

- **Stage 1:**

1: $\mathbf{B} := \emptyset$;
2: **if** the first data item is in $(0.2, 0.35)$ **then**
3: $\quad \mathbf{B}_{\text{mode}} :=$ `Queue`;
4: **else**
5: $\quad \mathbf{B}_{\text{mode}} :=$ `Queue_1`;
6: **end if**
7: **while** $\mathbf{I} \cap \mathbf{M} \neq \emptyset$ **do**
8: $\quad$ let u be the next data item in $\mathbf{I}$;
9: $\quad \mathbf{I} := \mathbf{I} \ominus u$;
10: $\quad$ accept u;
11: **end while**

- **Stage 2:**

1: **if** $\mathbf{B} = \{\triangledown, \triangledown, \triangledown\}$ **then**
2: $\quad \mathbf{B}_{\text{mode}} :=$ `Stack`;
3: **else**
4: $\quad \mathbf{B}_{\text{mode}} :=$ `Optimum`;
5: **end if**
6: **while** $\mathbf{I} \cap \mathbf{L} \neq \emptyset$ **do**
7: $\quad$ let u be the next data item in $\mathbf{I}$;
8: $\quad \mathbf{I} := \mathbf{I} \ominus u$;
9: $\quad$ accept u;
10: **end while**

Note that this is an adaptive revocable priority algorithm, but we do not explicitly state the

terminal condition in the algorithm. If at a certain step of the algorithm, the terminal condition is satisfied, the data item which contributes to this satisfaction will be the next data item and interrupt the ordering. Let $\mathbf{S} = u_1, u_2, \ldots, u_n$ be the input sequence of data items according to this ordering, and let $\mathbf{S}_1 = u_1, u_2, \ldots, u_m$ and $\mathbf{S}_2 = u_{m+1}, u_{m+2}, \ldots, u_n$ be the input sequences of data items for the first stage and second stage respectively. *From now on, we assume that the algorithm* ADAPTIVE *achieves approximation ratio less than* 0.8 *and try to derive a contradiction.*

Lemma 4.3.2 *Let u be the next data item to be examined and $\mathbf{B}' = (\mathbf{B} \oplus u)$. If $\mathbf{B}'$ contains three data items u_i, u_j, u_k, with $i < j < k$, then (u_i, u_j, u_k) cannot be any of the following types:*

$$(\triangledown, \triangledown, \triangle), \qquad (\triangledown, \triangle, \triangledown),$$
$$(\triangle, \triangle, \triangledown), \qquad (\triangle, \triangledown, \triangle);$$

Proof: We show that if (u_i, u_j, u_k) is one of the above types, then $0.8 \leq u_i + u_j + u_k \leq 1$, which is a contradiction.

1. If $(u_i, u_j, u_k) = (\triangledown, \triangledown, \triangle)$, then since $j < k$, we have $0.6 \leq u_j + u_k \leq 0.7$. Since $0.2 \leq u_i \leq 0.3$, we have $0.8 \leq u_i + u_j + u_k \leq 1$.

2. If $(u_i, u_j, u_k) = (\triangledown, \triangle, \triangledown)$, then since $i < j$, we have $0.6 \leq u_i + u_j \leq 0.7$. Since $0.2 \leq u_k \leq 0.3$, we have $0.8 \leq u_i + u_j + u_k \leq 1$.

3. If $(u_i, u_j, u_k) = (\triangle, \triangle, \triangledown)$, then since $j < k$, we have $0.5 \leq u_j + u_k \leq 0.6$. Since $0.3 \leq u_i \leq 0.4$, we have $0.8 \leq u_i + u_j + u_k \leq 1$.

4. If $(u_i, u_j, u_k) = (\triangle, \triangledown, \triangle)$, then since $i < j$, we have $0.5 \leq u_i + u_j \leq 0.6$. Since $0.3 \leq u_k \leq 0.4$, we have $0.8 \leq u_i + u_j + u_k \leq 1$.

Therefore, (u_i, u_j, u_k) cannot be any of the above types. □

Lemma 4.3.3 *The input sequence $\mathbf{S}_1 = u_1, \ldots, u_m$ is a subsequence of one of the following sequence types:*

1. $\nabla, \nabla, \nabla, \ldots, \nabla, \nabla, \nabla$
2. $\triangle, \nabla, \nabla, \ldots, \nabla, \nabla, \nabla$
3. $\triangle, \triangle, \triangle, \ldots, \triangle, \triangle, \triangle$
4. $\nabla, \triangle, \triangle, \ldots, \triangle, \triangle, \triangle$

Proof: If $\mathbf{S}_1$ contains at most two data items, then it is a subsequence of one of the four sequence types above. Suppose now $\mathbf{S}_1$ contains more than two data items; there are four cases:

1. If $(u_1, u_2) = (\nabla, \nabla)$, then $\mathbf{B}$ is in `Queue Mode` for the first stage. Suppose $\triangle$ occurs after u_2, then let u_{i+1} be the first such occurrence; when the algorithm considers u_{i+1}, $\{u_{i-1}, u_i\} \subseteq \mathbf{B}$. Since $(u_{i-1}, u_i, u_{i+1}) = (\nabla, \nabla, \triangle)$, this contradicts Lemma 4.3.2.
2. If $(u_1, u_2) = (\nabla, \triangle)$, then $\mathbf{B}$ is in `Queue Mode` for the first stage. Suppose ∇ occurs after u_2, then let u_{i+1} be the first such occurrence; when the algorithm considers u_{i+1}, $\{u_{i-1}, u_i\} \subseteq \mathbf{B}$. Since either $(u_{i-1}, u_i, u_{i+1}) = (\triangle, \triangle, \nabla)$, or $(u_{i-1}, u_i, u_{i+1}) = (\nabla, \triangle, \nabla)$, this contradicts Lemma 4.3.2.
3. If $(u_1, u_2) = (\triangle, \nabla)$, then we have two cases:
   (a) If $u_1 < 0.35$, then $\mathbf{B}$ is in `Queue Mode` for the first stage. Suppose $\triangle$ occurs after u_2, then let u_{i+1} be the first such occurrence; when the algorithm considers u_{i+1}, $\{u_{i-1}, u_i\} \subseteq \mathbf{B}$. Since either $(u_{i-1}, u_i, u_{i+1}) = (\nabla, \nabla, \triangle)$, or $(u_{i-1}, u_i, u_{i+1}) = (\triangle, \nabla, \triangle)$, this contradicts Lemma 4.3.2.
   (b) If $u_1 \geq 0.35$ then $\mathbf{B}$ is in `Queue_1 Mode` for the first stage. Suppose $\triangle$ occurs after u_2, then let u_{i+1} be the first such occurrence; when the algorithm considers u_{i+1}, $\{u_1, u_i\} \subseteq \mathbf{B}$. Since $(u_1, u_i, u_{i+1}) = (\triangle, \nabla, \triangle)$, this contradicts Lemma 4.3.2.
4. If $(u_1, u_2) = (\triangle, \triangle)$, then we have two cases:

(a) If $u_1 < 0.35$, then $\mathbf{B}$ is in `Queue Mode` for the first stage. Suppose $\triangledown$ occurs after u_2, then let u_{i+1} be the first such occurrence; when the algorithm considers u_{i+1}, $\{u_{i-1}, u_i\} \subseteq \mathbf{B}$. Since $(u_{i-1}, u_i, u_{i+1}) = (\triangle, \triangle, \triangledown)$, this contradicts Lemma 4.3.2.

(b) If $u_1 \geq 0.35$, then $\mathbf{B}$ is in `Queue_1 Mode` for the first stage. Suppose $\triangledown$ occurs after u_2, then let u_{i+1} be the first such occurrence; when the algorithm considers u_{i+1}, $\{u_1, u_i\} \subseteq \mathbf{B}$. Since $(u_1, u_i, u_{i+1}) = (\triangle, \triangle, \triangledown)$, this contradicts Lemma 4.3.2.

Therefore, $\mathbf{S}_1$ must be a subsequence of one of the input sequence types above. □

Lemma 4.3.3 is useful as it eliminates many cases we need to consider. We now give two observations:

- **Observation 3:** for the input sequence types 1 and 2, the data items in $\mathbf{S}_1$ is non-increasing, and the total weight of the first three data items of $\mathbf{S}_1$ is no greater than 1.
- **Observation 4:** for the input sequence types 3 and 4, the data items in $\mathbf{S}_1$ is non-decreasing. and the total weight of the first three data items of $\mathbf{S}_1$ is greater than 0.8.

The data items in $\mathbf{S}_2$ is non-decreasing in both cases. For convenience, we denote $\textsc{alg}'$ the algorithm's solution at the end of the first stage; we also suppose $\mathbf{S}$ is non-empty, for otherwise the problem becomes trivial.

Lemma 4.3.4 $|\textsc{opt}| > 1$.

Proof: Suppose now $|\textsc{opt}| = 1$, there are two cases:

1. If $\mathbf{S}$ contains a data item of type ♦, then u_n is the largest data item in $\mathbf{S}$ and $\textsc{opt} = \{u_n\}$. No matter which operational mode $\mathbf{B}$ is in, u_n is always accepted into the solution. Since u_n is the last data item in $\mathbf{S}$, the algorithm achieves the optimal solution in this case, which is a contradiction.

2. If $\mathbf{S}$ does not contain a data item of type $\blacklozenge$, then the entire input sequence $\mathbf{S}$ contain exactly one data item of type $\lozenge$, hence the algorithm easily achieves the optimal solution, which is a contradiction.

Therefore, both cases are impossible, and hence $|\textsc{opt}| > 1$. □

Lemma 4.3.5 *No feasible solution contains two data items in* $\mathbf{S}_2$.

Proof: We prove this by contradiction. Suppose there is a feasible solution which contains two data items in $\mathbf{S}_2$, then it is clear that $0.8 \leq u_{m+1} + u_{m+2} \leq 1$, and $u_{m+1} \leq 0.5$; we exam the beginning of the second stage when the algorithm considers u_{m+1}. Note that the algorithm has to reject u_{m+1}, for otherwise, when the algorithm considers the next data item u_{m+2}, the terminal condition is satisfied.

1. If $|\textsc{alg}'| < 2$, then $\|\textsc{alg}'\| \leq 0.4$. Since $\|\textsc{alg}'\| + u_{m+1} \leq 1$, the algorithm must then accept u_{m+1} into its solution, which is a contradiction.

2. If $|\textsc{alg}'| = 2$, then $\mathbf{B}$ is in `Optimum Mode` for the second stage. Let $\textsc{alg}' = \{u_i, u_j\}$, then we have $u_i + u_j < u_j + u_{m+1} < u_{m+1} + u_{m+2} \leq 1$, the algorithm must then accept u_{m+1} into its solution, which is a contradiction.

3. If $|\textsc{alg}'| = 3$, then we have $\mathbf{S}_1$ is a subsequence of either input sequence type 1 or 2; for otherwise, by Lemma 4.3.3, either $\textsc{alg}' = \{\triangle, \triangle, \triangle\}$ or $\textsc{alg}' = \{\triangledown, \triangle, \triangle\}$, and hence $\|\textsc{alg}'\| > 0.8$, which is a contradiction.

   (a) If $\textsc{alg}' = \{\triangle, \triangledown, \triangledown\}$, then $\textsc{alg}' = \{u_1, u_{m-1}, u_m\}$. Since $u_1 \geq 0.3$, we have

   $$0.4 \leq \quad u_{m-1} + u_m \quad \leq 0.5. \tag{4.11}$$

   If there exist a data item in $[0.4, 0.5]$, then combined with (4.11), that data item will interrupt the ordering and the terminal condition is satisfied. Hence, there exists no data item in $[0.4, 0.5]$, Therefore, No feasible solution contains two data items in $\mathbf{S}_2$.

(b) If $\textsc{alg}' = \{\triangledown, \triangledown, \triangledown\}$, then during the second stage, $\mathbf{B}$ is in `Stack Mode`, hence the algorithm must then accept u_{m+1} into its solution, which is a contradiction.

Therefore, no feasible solution contains two data items in $\mathbf{S}_2$. This completes the proof. □

Lemma 4.3.6 $|\textsc{opt}| < 4 \Rightarrow \|\textsc{opt}\| < 0.8$.

Proof: We prove this by contradiction. Suppose that $|\textsc{opt}| < 4$ and $\|\textsc{opt}\| \geq 0.8$; by Lemma 4.3.4, $|\textsc{opt}| > 1$. There are two cases:

1. If $|\textsc{opt}| = 2$, then let $\textsc{opt} = \{u_r, u_s\}$ with $u_r \leq u_s$. Note that if $(u_r, u_s) = (\blacklozenge, \blacklozenge)$, then it contradicts Lemma 4.3.5. Therefore $(u_r, u_s) = (\diamondsuit, \blacklozenge)$, then u_r always enters the solution. This is because during the first stage, $\mathbf{B}$ is either in `Queue_1 Mode` or `Queue Mode`, every data item in $\mathbf{S}_1$ gets a chance to enter the solution. Since $u_r + u_s \geq 0.8$, u_s will interrupt the ordering and become the next data item to be examined when u_r is in $\mathbf{B}$, hence the terminal condition is satisfied; this is a contradiction.

2. If $|\textsc{opt}| = 3$, then let $\textsc{opt} = \{u_r, u_s, u_t\}$ with $u_r \leq u_s \leq u_t$, there are two possibilities:

   (a) If $(u_r, u_s, u_t) = (\diamondsuit, \diamondsuit, \diamondsuit)$, then consider the first three elements in $\mathbf{S}$. If $\mathbf{S}_1$ is a subsequence of either input sequence type 1 or 2, then by Observation 3, we have

   $$0.8 \leq u_r + u_s + u_t \leq u_1 + u_2 + u_3 \leq 1.$$

   Therefore, the terminal condition is satisfied when the algorithm accepts the first, second and third data item; this is a contradiction. If $\mathbf{S}_1$ is a subsequence of either input sequence type 3 or 4, then by Observation 4, we have

   $$0.8 \leq u_1 + u_2 + u_3 \leq u_r + u_s + u_t \leq 1.$$

   Therefore, the terminal condition is satisfied when the algorithm accepts the first, second and third data item; this is a contradiction.

(b) If $(u_r, u_s, u_t) = (\diamondsuit, \diamondsuit, \blacklozenge)$, then consider the input sequence $\mathbf{S}$. If $\mathbf{S}_1$ is a subsequence of either input sequence type 1 or 2, then at the end of the first stage, $\{u_{m-1}, u_m\} \subseteq \textsc{alg}'$. Since u_{m-1} and u_m are the two smallest data items, we have

$$0.8 \leq u_{m-1} + u_m + u_t \leq u_r + u_s + u_t \leq 1.$$

Therefore, the terminal condition is satisfied at the end of the first stage; this is a contradiction. If $\mathbf{S}_1$ is a subsequence of either input sequence type 3 or 4, then by Observation 4, we have

$$0.8 \leq u_1 + u_2 + u_3 \leq u_r + u_s + u_t \leq 1.$$

Therefore, the terminal condition is satisfied when the algorithm accepts the first, second and third data item; this is a contradiction.

Therefore, $\|\textsc{opt}\| < 0.8$. This completes the proof. □

Lemma 4.3.7 $\mathbf{B}_{\text{mode}} = \texttt{Optimum} \Rightarrow \|\textsc{alg}'\| < 0.8\|\textsc{opt}\|$.

Proof: Suppose that $\|\textsc{alg}'\| \geq 0.8\|\textsc{opt}\|$, then since $\mathbf{B}$ in `Optimum Mode` during the second stage, $\|\mathbf{B}\|$ is non-decreasing. Therefore $\|\textsc{alg}'\| \leq \|\textsc{alg}\|$, the approximation ratio is

$$\rho = \frac{\|\textsc{alg}\|}{\|\textsc{opt}\|} \geq \frac{\|\textsc{alg}'\|}{\|\textsc{opt}\|} \geq 0.8,$$

which is a contradiction. □

Lemma 4.3.8 $|\textsc{alg}'| \neq 0$.

Proof: Suppose that $|\textsc{alg}'| = 0$, then $\mathbf{S}_1$ is empty. By Lemma 4.3.4, $|\textsc{opt}| > 1$, hence **opt** contains at least two data items in $\mathbf{S}_2$; this contradicts Lemma 4.3.5. Therefore, $|\textsc{alg}'| \neq 0$. □

Lemma 4.3.9 $|\textsc{alg}'| \neq 1$.

Proof: Suppose that $|\textbf{ALG}'| = 1$, then $\mathbf{S}_1$ contains only one data item. By Lemmas 4.3.4 and 4.3.5, $|\textbf{OPT}| > 1$, and **OPT** contains at most one data item in $\mathbf{S}_2$. Therefore $\textbf{OPT} = \{u_1, u_i\}$. Since **B** is in `Optimum Mode`, then by Lemma 4.3.5, the algorithm achieves the optimal solution, which is a contradiction. Therefore, $|\textbf{ALG}'| \neq 1$. □

Lemma 4.3.10 $|\textbf{ALG}'| \neq 2$.

Proof: Suppose that $|\textbf{ALG}'| = 2$, then $|\textbf{OPT}| < 4$. This is because if $|\textbf{OPT}| = 4$, $\mathbf{S}_1$ is a subsequence of either input sequence type 1 or 2, and hence $|\textbf{ALG}'| = 3$, which is a contradiction. By Lemma 4.3.4, $|\textbf{OPT}| > 1$. There are two cases:

1. If $|\textbf{OPT}| = 2$, then let $\textbf{OPT} = \{u_r, u_s\}$ with $u_r \leq u_s$. By Lemma 4.3.5 and 4.3.6, u_r is in $\mathbf{S}_1$ and $\|\textbf{OPT}\| < 0.8$; there are two possibilities:

   (a) If $(u_r, u_s) = (\diamondsuit, \diamondsuit)$, then if $m = 2$, then u_r and u_s are the only two data items in $\mathbf{S}_1$ and $\textbf{ALG}' = \{u_r, u_s\}$. Since **B** is in `Optimum Mode` during the second stage, the algorithm clearly achieves the optimal solution, which is a contradiction. If $m > 2$, then $\mathbf{S}_1$ is a subsequence of either input sequence type 3 or 4; for otherwise, $|\textbf{ALG}'| > 2$. Therefore $\textbf{ALG}' = \{u_i, u_m\}$, where $u_i > 0.3$; hence we have

   $$\frac{\|\textbf{ALG}'\|}{\|\textbf{OPT}\|} = \frac{u_i + u_m}{u_r + u_s} > \frac{0.3 + u_m}{2u_m} \geq \frac{0.7}{0.8} > 0.8.$$

   Since **B** is in `Optimum Mode` during the second stage, by Lemma 4.3.7, we have $\|\textbf{ALG}'\| < 0.8\|\textbf{OPT}\|$, which is contradiction.

   (b) If $(u_r, u_s) = (\diamondsuit, \blacklozenge)$, then u_r must be the largest data item in $\mathbf{S}_1$. This is because $\|\textbf{OPT}\| = u_r + u_s < 0.8$. If u_r is not the largest data item in $\mathbf{S}_1$, we can always replace it with the largest data item in $\mathbf{S}_1$; the solution can only increase, and the feasibility is still maintained. If $m = 2$, then $u_r \in \textbf{ALG}'$. If $m > 2$, then **S** belongs to either input sequence 3 or 4; for otherwise, $|\textbf{ALG}'| > 2$. Therefore, we also have

$u_r \in \mathbf{ALG}'$. Since $\mathbf{B}$ is in `Optimum Mode` during the second stage, by Lemma 4.3.5, the algorithm achieves the optimal solution, which is also a contradiction.

2. If $|\mathbf{OPT}| = 3$, then let $\mathbf{OPT} = \{u_r, u_s, u_t\}$ with $u_r \leq u_s \leq u_t$, there are two possibilities:

   (a) If $\mathbf{ALG}' = \{\triangledown, \triangledown\}$ or $\mathbf{ALG}' = \{\triangledown, \triangle\}$, then $\mathbf{S}_1$ is a subsequence of input sequence type 1, 2 or 4, and $\mathbf{S}_1$ contains only two data items. We then have

   $$0.8 \leq u_1 + u_2 + u_3 \leq u_r + u_s + u_t \leq 1.$$

   Therefore, the terminal condition is satisfied, which is a contradiction.

   (b) If $\mathbf{ALG}' = \{\triangle, \triangle\}$, then $\mathbf{S}_1$ is a subsequence of either input sequence type 3 or 4, hence $|\mathbf{OPT}| < 3$, which is a contradiction.

Therefore, $|\mathbf{ALG}'| \neq 2$. □

Lemma 4.3.11 $|\mathbf{ALG}'| \neq 3$.

Proof: Suppose that $|\mathbf{ALG}'| = 3$, then $\mathbf{S}_1$ is a subsequence of either input sequence type 1 or 2; for otherwise, by Lemma 4.3.3, either $\mathbf{ALG}' = \{\triangle, \triangle, \triangle\}$ or $\mathbf{ALG}' = \{\triangledown, \triangle, \triangle\}$, and hence $\|\mathbf{ALG}'\| > 0.8$, the terminal condition is satisfied, which is a contradiction.

1. If $\mathbf{ALG}' = \{\triangle, \triangledown, \triangledown\}$, then $\mathbf{ALG}' = \{u_1, u_{m-1}, u_m\}$; there are two cases:

   (a) If $|\mathbf{OPT}| < 4$, then by Lemma 4.3.6, we have

   $$\|\mathbf{OPT}\| < 0.8.$$

   Since we have $\mathbf{B}$ in `Optimum Mode` during the second stage and

   $$\|\mathbf{ALG}'\| = u_1 + u_{m-1} + u_m > 0.7,$$

   this contradicts Lemma 4.3.7.

(b) If $|\mathbf{OPT}| = 4$, then if $\mathbf{B}$ in `Queue Mode` during the first stage, then $|\mathbf{ALG}'| = 4$, which is a contradiction. Therefore, $\mathbf{B}$ in `Queue_1 Mode` during the first stage. If $u_1 \in \mathbf{OPT}$, then $|\mathbf{ALG}'| = 4$, which is a contradiction. Therefore, $u_1 \notin \mathbf{OPT}$; by Observation 3, $\|\mathbf{OPT}\| \leq u_2 + u_3 + u_4 + u_5$. Note that at the same time, we have the following three inequalities:

$$u_1 \geq 0.35, \tag{4.12}$$

$$u_1 + u_2 + u_3 < 0.8, \tag{4.13}$$

$$u_2 + u_3 \geq u_4 + u_5. \tag{4.14}$$

Combining (4.12) and (4.13), we have $u_2 + u_3 < 0.45$. Therefore, by (4.14),

$$\|\mathbf{OPT}\| \leq u_2 + u_3 + u_4 + u_5 \leq 2(u_2 + u_3) < 0.9.$$

Since we have $\mathbf{B}$ in `Optimum Mode` during the second stage and

$$\|\mathbf{ALG}'\| = u_1 + u_{m-1} + u_m > 0.75,$$

this contradicts Lemma 4.3.7.

2. If $\mathbf{ALG}' = \{\triangledown, \triangledown, \triangledown\}$, then $\mathbf{ALG}' = \{u_{m-2}, u_{m-1}, u_m\}$; there are two cases:

(a) If $\mathbf{S}_2$ is empty, then there two possibilities. If $m = 3$, then the algorithm clearly achieves the optimal solution. If $m > 3$, then $\|\mathbf{ALG}\| > 0.65$. Since u_{m-2}, u_{m-1} and u_m are the three smallest data items in $\mathbf{S}$, we have $|\mathbf{OPT}| < 4$. By Lemma 4.3.6, $\|\mathbf{OPT}\| < 0.8$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} > \frac{0.65}{0.8} \geq 0.8,$$

which is a contradiction.

(b) If $\mathbf{S}_2$ is non-empty, then since u_{m-2}, u_{m-1} and u_m are the three smallest data items in $\mathbf{S}$, we have $|\mathbf{OPT}| < 4$. By Lemma 4.3.6, $\|\mathbf{OPT}\| < 0.8$. Since $\mathbf{B}$ in `Stack Mode`

during the second stage and $u_m \leq u_{m-1} \leq u_{m-2} \leq 0.3$, we then have $\|\mathbf{ALG}\| > 0.7$. Therefore, the approximation ratio is

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} > \frac{0.7}{0.8} \geq 0.8,$$

which is a contradiction.

Therefore, $|\mathbf{ALG}'| \neq 3$. □

Now we are ready to prove the main theorem. Recall that all Lemmas 4.3.8, 4.3.9, 4.3.10, and 4.3.11 are assuming the algorithm ADAPTIVE achieves approximation ratio less than 0.8.

Theorem 4.3.12 *There exists an adaptive revocable priority algorithm of SSP with approximation ratio* 0.8.

Proof: It is sufficient to show that the algorithm ADAPTIVE achieves this approximation ratio. We prove this by contradiction. Suppose the algorithm ADAPTIVE achieves approximation ratio less than 0.8, then by Lemmas 4.3.8, 4.3.9, 4.3.10, and 4.3.11, we have $|\mathbf{ALG}'| \geq 4$. This satisfies the terminal condition, hence a contradiction. Therefore, the algorithm ADAPTIVE achieves approximation ratio 0.8. □

Chapter 5

Conclusion

5.1 Summary of Results

This work studies priority algorithms for the Subset-Sum Problem with the main focus on their approximation ratios. Different classes of priority algorithms are analyzed and corresponding lower bounds are proved. Although we are unable to close the two gaps we have for fixed and adaptive revocable priority algorithms, this work gives us some basic understandings of the relative power the Priority Model, and how revocable acceptances increase the algorithm's approximation ability.

The thesis starts with irrevocable priority algorithms, shows tight approximation ratio for this class, regardless of whether it is fixed or adaptive. It then moves onto revocable priority algorithms, in the spirit of algorithms with revocable acceptances studied in [17, 19]. Fixed revocable priority algorithms with non-increasing and non-decreasing order are subsequently studied; both with tight bounds on the approximation ratios. Then a general lower bound for fixed revocable priority algorithms is given, exhibiting the first gap that we are unable to close in this thesis. Finally, the class of adaptive revocable priority algorithms, a more complex structure, is studied. A pair of unmatched upper and lower bound is provided, illustrating the second unclosed gap. A list of results is summarized in Table 5.1, where the corresponding

values are given in Table 5.2.

Table 5.1: Summary of results.

	irrevocable		revocable
fixed	$\rho = \alpha_1$		$\alpha_2 \leq \rho \leq \beta_1$
		non-increasing order	$\rho = \alpha_1$
		non-decreasing order	$\rho = \alpha_2$
adaptive			$\alpha_3 \leq \rho \leq \beta_2$

Table 5.2: Corresponding values.

name	α_1	α_2	α_3	β_1	β_2
value	≈ 0.657	≈ 0.780	$= 0.8$	≈ 0.852	≈ 0.893

Note that throughout the thesis, the power and limitations of priority algorithms in term of their approximability are of our major concern. We are not, in particular, interested in the time complexity of an algorithm. In most cases, our algorithms runs time $O(n \log n)$; the only exception is the algorithm ADAPTIVE, which is quadratic due to its checking of the terminal condition.

5.2 Questions and Future Directions

One open question of this thesis is whether or not we can close the two gaps for fixed and adaptive revocable priority algorithms. Most likely, they can be closed; but it seems requiring a large data set to construct an adversary, hence the question arises whether we can have a systematic way to generate good adversaries. One possibility might be set a target ratio, and work out a priority chain for specific data items, similar to the general lower bound proof

for fixed revocable priority algorithms. This approach only works for fixed order. The other possibility is to come up with a way to build one adversary onto the other, while lowering the lower bound. This approach is less promising as the data items of a less "successful" adversary are rarely useful for a "successful" adversary.

The other future direction we can take is of course to study priority algorithms for other NP-hard problems. It is interesting to think whether some sort of "reduction" would exist among those problems, in terms of the approximation ratio.

The third direction is the study of adversarial strategies. We have a good exposure of adversarial strategies in this thesis, and adversarial strategies have not been studied as thoroughly as algorithms. One can view an adversarial strategy as an "anti-algorithm" which takes an algorithms as its input whereas an algorithm often takes a set of data items as its input. If an algorithm can be dissembled into pieces, for example a list of instructions, then it is possible to think that an adversarial strategy is also an algorithm. The question arises whether there exists an algorithm whose adversary is itself. This may provide a new way to study the limitation of algorithms for certain class of problems.

Bibliography

[1] M. Alekhnovich, A. Borodin, J. Buresh-Oppenheim, R. Impagliazzo, A. Magen, and T. Pitassi. Toward a model for backtracking and dynamic programming. In *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, pages 308–322, 2005.

[2] S. Angelopoulos and A. Borodin. On the power of priority algorithms for facility location and set cover. *Algorithmica*, 40:271–291, 2004.

[3] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines under real-time scheduling. *SIAM Journal on Computing*, 31:331–352, 2001.

[4] A. Borodin, J. Boyar, and K. Larsen. Priority algorithms for graph optimization problems. *Lecture Notes in Computer Science*, 3351:126–139, 2005.

[5] A. Borodin, M. Nielsen, and C. Rackoff. (Incremental) priority algorithms. *Algorithmica*, 37:295–326, 2003.

[6] A. Caprara and U. Pferschy. Packing bins with minimal slack. Technical report, University of Graz, 2002.

[7] A. Caprara and U. Pferschy. Worst-case analysis of the subset sum algorithm for bin packing. *Operations Research Letters*, 32:159–166, 2004.

[8] S. Davis and R. Impagliazzo. Models of greedy algorithms for graph problems. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2004.

[9] B. Dietrich and L. Escudero. Coefficient reduction for knapsack constraints in 0-1 programs with VUBs. *Operations Research Letters*, 9:9–14, 1990.

[10] L. Escudero, S. Martello, and P. Toth. A framework for tightening 0-1 programs based on extensions of pure 0-1 KP and SS problems. *Lecture Notes in Computer Science*, 920:110–123, 1995.

[11] M. Garey and D. Johnson. *Computers and Intractability: a Guide to the Theory of NP-completeness*. Freeman, 1979.

[12] G. Gens and E. Levner. Approximation algorithms for certain universal problems in scheduling theory. *Soviet Journal of Computers and System Sciences*, 6:31–36, 1978.

[13] G. Gens and E. Levner. Fast approximation algorithms for knapsack type problems. *Lecture Notes in Control and Information Sciences*, 23:185–194, 1980.

[14] G. Gens and E. Levner. A fast approximation algorithm for the subset-sum problem. *Information Systems and Operational Research*, 32:143–148, 1994.

[15] C. Guéret and C. Prins. A new lower bound for the open-shop problem. *Annals of Operations Research*, 92:165–183, 1999.

[16] J. Hoogeveen, H. Oosterhout, and S. van de Velde. New lower and upper bounds for scheduling around a small common due date. *Operations Research*, 42:102–110, 1994.

[17] S. Horn. One-pass algorithms with revocable acceptances for job interval selection. Master's thesis, University of Toronto, 2004.

[18] O. Ibarra and C. Kim. Fast approximation algorithms for the knapsack and sum of subset problem. *Journal of the ACM*, 22:463–468, 1975.

[19] K. Iwama and S. Taketomi. Removable online knapsack problems. *Lecture Notes in Computer Science*, 2380:293–305, 2002.

[20] H. Kellerer, R. Mansini, U. Pferschy, and M. Speranza. An efficient fully polynomial approximation scheme for the subset-sum problem. *Journal of Computer and System Sciences*, 66:349–370, 2003.

[21] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[22] E. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4:339–256, 1979.

[23] S. Martello and P. Toth. Worst-case analysis of greedy algorithms and for the subset-sum problem. *Mathematical Programming*, 28:198–205, 1984.

[24] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley and Sons, 1990.

[25] D. Pisinger. An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114:528–541, 1999.