SCHEDULING AND BIN PACKING
A STUDY OF THE WORST-CASE PERFORMANCE BOUNDS

Weizhen Mao

(Thesis)

CS-TR-284-90

October 1990

SCHEDULING AND BIN PACKING

— A STUDY OF THE WORST-CASE PERFORMANCE BOUNDS

Weizhen Mao

A DISSERTATION
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE
BY THE DEPARTMENT OF
COMPUTER SCIENCE

OCTOBER 1990

Acknowledgements

I wish to thank most sincerely my adviser Andrew C. Yao, without whose valuable advice, constructive comments, and enthusiastic encouragement the work in this dissertation can never be accomplished. I am appreciative for all he did for me during my stay at Princeton.

I am thankful to my readers, Bernard Chazelle and Joel Friedman, for their helpful comments and suggestions on this dissertation and for their guidance and encouragement to my career in computer science. And also I am grateful to Ken Steiglitz, who is always so kind and helpful. Further thanks go to Sharon Rodgers, Winnie Waring, Melissa Lawson and Gene Davidson, who are always available for help.

I also wish to express my gratitude to my officemates (especially Marios Dikaiakos), my fellow graduate students (especially Mark Greenstreet), and the faculty members in the department for making my four years at Princeton pleasant and fruitful. A very special thank you goes to my husband Jie Chen for his love, help and patience. I am also grateful to my brother Weidong Mao for just being together for all these years.

Needless to say, studying in the Department of Computer Science at Princeton is a lifetime experience that I will cherish forever.

to my parents

Abstract

In this dissertation, we study the worst-case performance bounds of various algorithms of scheduling problem and bin packing problem.

For the scheduling problem with m parallel machines and precedence constraints, let ω be the total elapsed time of the execution of all tasks in the optimal non-preemptive schedule, and let ω' be the total elapsed time in the optimal preemptive schedule. C. L. Liu conjectured twenty years ago that the ratio $\frac{\omega}{\omega'}$ is no greater than $\frac{2m}{m+1}$ and this bound is also the best possible. We will prove the conjecture for a number of cases, such as when the precedence constraint is empty, or when the task system satisfies certain conditions; for other cases the previous result $\frac{\omega}{\omega'} \leq \frac{2m-1}{m}$ given by R. R. Muntz will be substantially improved.

The bin packing, a special scheduling problem, is to pack a list of reals in $(0, 1]$ into unit-capacity bins using the minimum number of bins. Let $R[A]$ be the limiting worst-case value for the ratio $A(L)/L^*$ as $L^* \to \infty$, where $A(L)$ denotes the number of bins used when the approximation algorithm A is applied to the list L, and L^* denotes the minimum number of bins needed to pack L. For Next-k-Fit (NkF) and Best-k-Fit (BkF), which are linear-time approximation algorithms for bin packing, it was known that both $R[NkF]$ and $R[BkF]$ are in the interval $[1.7 + \frac{3}{10k}, 2]$. In this dissertation, two precise bounds $R[NkF] = 1.7 + \frac{3}{10(k-1)}$ and $R[BkF] = 1.7 + \frac{3}{10k}$ are proved.

Table of Contents

1
Introduction

Introduction

1.1 Deterministic Scheduling

The extensive study of deterministic scheduling problems since early 1950's is mainly motivated by questions that arise in production planning, computer system controlling, or even everyday activities such as scheduling an efficient working day in a company, planning examination period in a university and so on. As pointed out by E. L. Lawler *et al* in [23], scheduling is about all situations in which scarce resources have to be allocated to tasks or jobs over time. By deterministic we mean that all information that defines a problem instance is known with certainty in advance. Deterministic scheduling is in fact part of combinatorial optimization.

In a computer system, we have processors or machines which can execute tasks in a parallel manner, and tasks or jobs which need to be executed on the processors. At any time, a processor can perform at most one task, and a task can be worked on by at most one processor. Based on a number of prespecified requirements concerning the processor environment and the task characteristics, we wish to find an efficient algorithm for sequencing the tasks on the processors to optimize or tend to optimize some desired performance measure. We call a schedule optimal if it minimizes a given performance measure. A scheduling model can be described by its resources, task systems, sequencing constraints, and performance measures [5].

Resources

Resources in a computer system usually include processors. In the simplest case, the resources consist of a set of m processors $P = \{P_1, P_2, \ldots, P_m\}$, where the processors may be identical, or identical in functional capability but different in speed, or different in both function and speed, depending on different scheduling problems. In more complicated cases, we may also have additional resources $R = \{R_1, R_2, \ldots, R_s\}$ involved in the scheduling, with q_i denoting the amount of resource of type R_i available in the scheduling model, for $i = 1, 2, \ldots, s$. For example, such additional resources may represent primary or secondary memory, input/output devices and library routines in a computer system.

Task Systems

In the task system, there is a set of tasks denoted as $T = \{T_1, T_2, \ldots, T_n\}$. Let p_j be the processing time for task T_j, for $j = 1, 2, \ldots, n$, then the total processing time X of the task system is defined to be $\sum_{j=1}^{n} p_j$. We also let partial order $\prec$ represent the precedence constraints among the tasks. By $T_i \prec T_j$ we mean that task T_i must be completed before task T_j can be started, and T_i is called the predecessor of T_j, and T_j the successor of T_i.

When additional resources $R = \{R_1, R_2, \ldots, R_s\}$ also have to be considered, we use $R_i(T_j)$ to represent the amount of resource of type R_i required throughout the execution of T_j. We always assume $R_i(T_j) \leq q_i$ for all i's and j's. In some cases, each task T_j may also have a due date d_j, which denotes the deadline when T_j must be finished, a release date r_j, which denotes

the time when T_j is available for execution, or a weight w_j, which can be interpreted as the deferral cost.

A task system can be represented by a directed acyclic graph (DAG) with no (redundant) transitive arcs. For each task T_i, we create a node labeled T_i/p_i, where p_i is the processing time of T_i. There is an edge from T_i to T_j if and only if $T_i \prec T_j$ is in the partial order. The longest path C, which is also called chain, is defined to be the directed path in the DAG with the maximum sum of the processing time of all the tasks on the path. Suppose $C = T_{i1} \rightarrow T_{i2} \rightarrow \cdots \rightarrow T_{ik}$, then it is not hard to see that $T_{i1} \prec T_{i2} \prec \cdots \prec T_{ik}$, and that the length of the chain $|C| = \sum_{j=1}^{k} p_{ij}$. Many interesting results have been obtained for some special precedence constraints, such as tree-like partial order and empty partial order. Figure 1.1 shows the DAG representation of a task system.

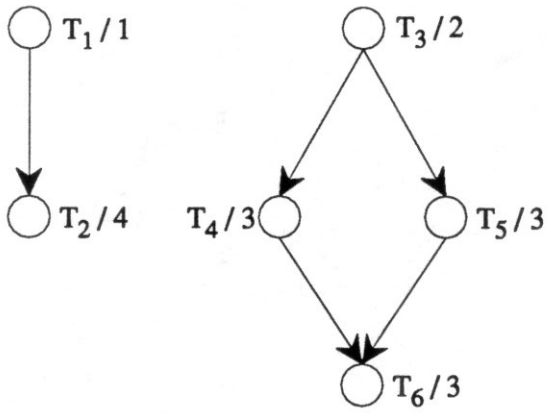


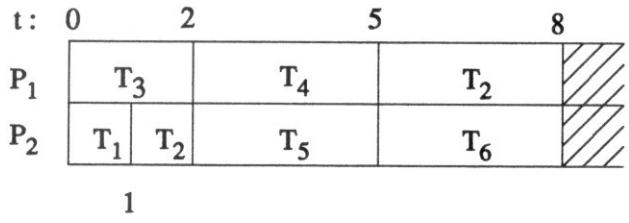
Figure 1. 1 A task system.

Sequencing Constraints

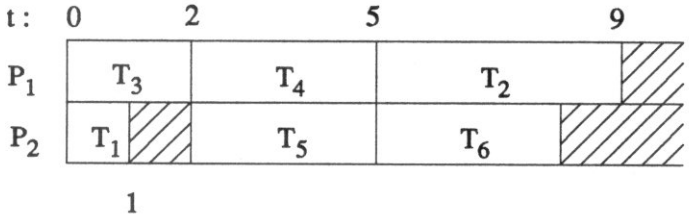
By "sequencing constraints" we mean certain restrictions that scheduling algorithms must follow. There are mainly two such restrictions. The first one is based on whether preemption of a task is allowed. In the non-preemptive scheduling, once a task has begun execution on a processor, it will keep executing until it finishes completely; while in the preemptive scheduling, a task can be interrupted during its execution, and resumed at a later time from the point at which it was last preempted. The second one is called list scheduling, which is based on an ordered priority list of tasks $L = (T_{i1}, T_{i2}, \ldots, T_{in})$ given in advance. Whenever a processor becomes idle, we check the list to find out the first unexecuted executable task in it, and the idle processor will be allocated to that task. By executable we mean that all the predecessors of the task have been finished and that all the required resources are available at that moment. In list scheduling, preemptions are not considered; thus list schedules form a subset of non-preemptive schedules.

The simplest way of graphically specifying a schedule is to use a timing diagram, also known as the Gantt Chart, which has, for each processor, a time axis with intervals marked off and labeled with the name of the task being processed in that interval. Symbol ϕ or shaded area is used to denote the idle period. Notice that in a schedule a processor might be left idle either because there are no executable tasks at that time or because it is an intentional choice. It is never necessary nor beneficial in a schedule to leave all processors idle at the same time. Figure 1.2 shows the Gantt Chart's of the preemtive, non-preemptive and list schedulings of the task system given

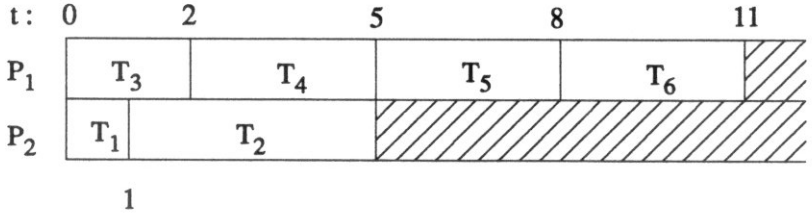
in Figure 1.1. (The Gantt Chart is formally discussed in [4].)



(a)



(b)



(c)

Figure 1.2 For the task system in Figure 1.1 and two identical processors, (a) the preemptive scheduling; (b) the non-preemptive scheduling; (c) the list scheduling for L=(T_1 , T_2 , T_3 , T_4 , T_5 , T_6).

Performance Measures

For each task T_j in the system, let C_j be its completion time in a schedule.

Suppose that C_{max} is the completion time of the task that completes last, i.e., $C_{max} = \max_{1 \leq j \leq n}\{C_j\}$. Obviously, C_{max} is in fact the total elapsed time for the execution of all the tasks in the system. Most of the time, we want to find a schedule which minimizes C_{max}. Such a schedule is called the optimal schedule. There are other performance measures of interest. For instance, let d_j be T_j's due date or deadline, then $L_j = C_j - d_j$ is called the lateness, $TA_j = \max\{0, L_j\}$ is called the tardiness, and $W_j = C_j - p_j$ is called the waiting time. In some scheduling problems, we may want to minimize the maximum lateness, tardiness or waiting time.

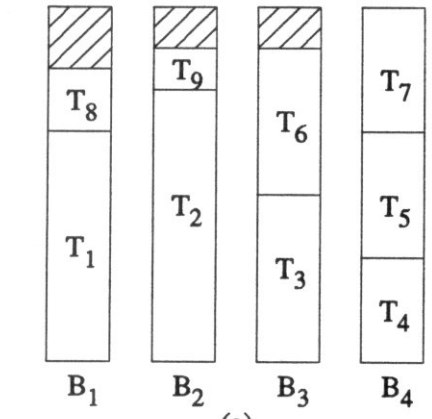
The above four parameters: resources, task systems, sequencing constraints and performance measures, are usually needed to define a scheduling problem. Different choices of the parameters will yield different types of the scheduling problems. For instance, a lot of research have been done toward certain special versions of scheduling problems, such as single-processor environment [3][21], tree-like precedence constraints [13][26], and unit-processing-time tasks [27]. Even though we have employed many restrictions on these parameters, we still have a tremendous amount of problem types of scheduling. MSPCLASS is a program written by B. J. Lageweg *et al* [19][20], which collects 4,536 scheduling problems, and records their complexity results. Among these problems, 416 are known to be solvable in polynomial time, 3,817 are proven to be NP-complete, and 303 are still open. That is why people believe that the field of scheduling problems is still rich and worth exploring even after so many years of study.

1.2 Bin Packing

Given a list of positive real numbers and a sequence of bins of capacity no less than any number in the list, the bin packing problem is to pack all the numbers in the list into the bins such that no bin contains a sum exceeding its capacity and that the number of bins used is minimized. Efficient algorithms for obtaining optimal or near-optimal packings have obvious practical applications—for example, table formatting, file allocation and so on.

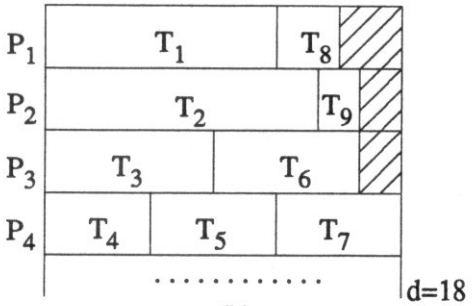
We can see from the following analysis that bin packing is actually a special version of scheduling. When studying scheduling problems, we sometime consider those in which due dates must be respected. In theses problems, we arrange the schedule such that every task T_j is completed before its due date d_j, i.e., $C_j \leq d_j$, for $j = 1, 2, \ldots, n$. The due dates are also called deadlines. One such problem to which we pay considerable attention assumes $d_j = d$, for $j = 1, 2, \ldots, n$; the precedence constraint $\prec$ is empty, i.e., the tasks are independent of each other; in addition processors are identical and no additional resources are involved. The goal is to minimize the number of processors required to meet the common deadline d.

We should be aware that this problem is not a scheduling problem because the optimization goal is the number of processors instead of time. However, this problem is equivalent to the well-known bin packing problem which packs a sequence of numbers $p_1, p_2, \ldots, p_n$ into the minimum number of bins of capacity d such that no bin contains a total exceeding d. Figure 1.3 (a) and (b) show such equivalence.



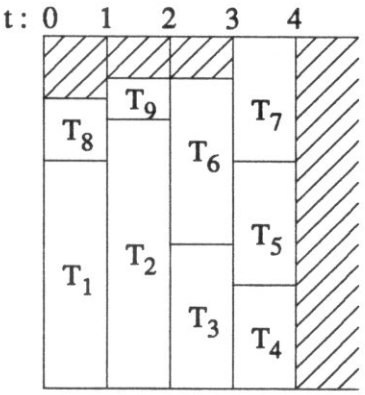
(a)

$T=\{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9\}$
$\{p_j\}=\{13, 15, 9, 6, 6, 8, 6, 3, 2\}$
bin capacity is 18
minimize the number of bins



(b)

$T=\{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9\}$
$\{p_j\}=\{13, 15, 9, 6, 6, 8, 6, 3, 2\}$
independent, no additional resourses
minimize the number of processors



(c)

$T=\{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9\}$
$p_j=1$, for $j=1, 2, ..., n$
no precedence constraints
$\{R(T_j)\}=\{13, 15, 9, 6, 6, 8, 6, 3, 2\}$
number of resource R is 18
minimize C_{max}

Figure 1. 3 (a) Packing $\{T_1, T_2, \ldots, T_n\}$ or precisely $\{p_1, p_2, \ldots, p_n\}$ into the minimum number of bins of capacity 18; (b) Scheduling $\{T_1, T_2, \ldots, T_n\}$ to meet deadline d=18 on the minimum number of processors; (c) Scheduling $\{T_1, T_2, \ldots, T_n\}$ with single-resource requirement to minimize schedule length.

Let us consider the following scheduling problem. Suppose we have m identical and parallel processors $P = \{P_1, P_2, \ldots, P_m\}$, and n unit-processing-time tasks $T = \{T_1, T_2, \ldots, T_n\}$ with $p_j = 1$, for $j = 1, 2, \ldots, n$. Assume that $m \geq n$, and that the precedence constraint among the tasks $\prec$ is empty. And also, we have one additional resource type R that needs to be considered in the scheduling. Suppose the total number of resource R is d, and task T_j needs $R(T_j)$ of resource R during its execution for $j = 1, 2, \ldots, n$. The optimization goal is to minimize the scheduling length C_{max}.

Because $m \geq n$, we do not have to concern about the processor allocation during the scheduling. All we need to take care of is the allocation of the additional resource R. Figure 1.3 (b) and (c) show that this scheduling problem is equivalent to the problem described earlier in this section, which wants to minimize the number of processors. So it is also equivalent to the bin packing problem.

When we study the bin packing, for simplicity we assume that all the numbers in the list are in $(0, 1]$ and each bin has the capacity of one.

1.3 Computational Complexity and Performance Bounds

In the field of scheduling theory, as in many other areas, some problems are much easier to solve than the others. For example, the optimal scheduling on two processors of n unit-processing-time tasks can be done in $O(n^2)$ [7], while the same problem with more than two processors appears to require time exponential in n [5]. The theory of NP-completness, pioneered

by S. A. Cook [8] and R. M. Karp [18], provides a framework for studying such problems. Let us classify problems as "easy" and "hard", where easy problems are those having polynomial-time solutions while hard problems are those with no polynomial-time solutions. There is a large class of problems, called "NP-complete" problems, for which either all or none of them are easy. It is widely believed that problems in this class are all hard. For computational problems in general, and in particular the various scheduling problems, it is of interest to determine whether they are NP-complete. For problems that are NP-complete, the study of approximation algorithms is important from a practical point of view. To assess the quality of an approximation algorithm, we use the criterion of *worst-case performance bound.*

Definition

Suppose we are considering an optimization problem which needs to minimize a function f, and A is an approximation algorithm for the problem. Let f^* be the optimal value of the function, which is also the value of f achieved by the optimal algorithm, and let $A(f)$ be the value of the function when algorithm A is used to solve the problem. Then we define $R[A]$, the worst-case performance bound of A, to be $\max\{A(f)/f^*\}$ over all possible problem instances. $\Diamond$

For an optimization problem, there might be many approximation algorithms for it. For each of them, we wish to find out its worst-case performance bound so that we could have a better look of how good the approximation algorithm is.

Example

Let us consider the general scheduling problem with m identical processors and n tasks. Suppose that there are no precedence constraints among the tasks, the number of processors m is greater than 1, and no additional resources are involved. The performance measure is to minimize C_{max} with no preemption allowed. It has been proved that this problem is NP-complete, so it is of practical interest to study its approximation algorithms and their performance bounds.

As introduced earlier, list scheduling furnishes a quick strategy to schedule the tasks on the processors. Given a priority list $L = (T_{i1}, T_{i2}, \ldots, T_{in})$, the first unexecuted executable task on the list will be assigned to the idle processor. Let C^*_{max} be the total elapsed time for the execution of all the tasks in the optimal schedule and $LS(C_{max})$ be the total elapsed time for the execution of all the tasks in the list schedule.

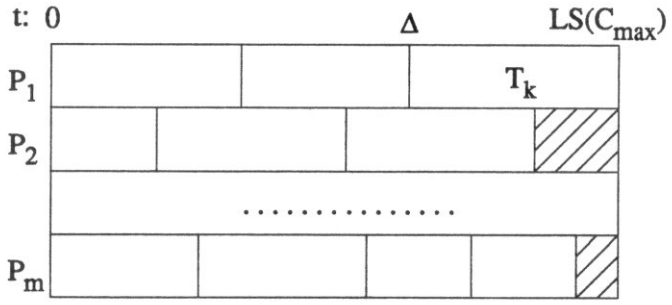


Figure 1. 4 The list scheduling of independent tasks.

In any schedule, it is clear that a lower bound to the completion time is

provided by assuming that all processors are busy all the time. So $C^*_{max} \geq \frac{1}{m}\sum_{j=1}^{n} p_j = \frac{1}{m}X$. On the other hand, $C^*_{max} \geq \max_{1\leq j\leq n}\{p_j\}$.

In the list schedule, let T_k be the task that finishes last, i.e., $LS(C_{max}) = C_k = \Delta + p_k$, where Δ is the time when T_k starts. Figure 1.4 shows the list scheduling.

Because of the nature of list scheduling, and because there is no precedence constraint, no processor is idle from time 0 to time Δ. Thus, $\Delta \leq \frac{1}{m}(X - p_k)$.

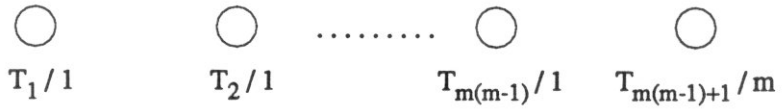
$$\begin{aligned} LS(C_{max}) &= \Delta + p_k \\ &\leq \frac{1}{m}(X - p_k) + p_k \\ &= \frac{1}{m}X + (1 - \frac{1}{m})p_k \\ &\leq C^*_{max} + (1 - \frac{1}{m})C^*_{max} \\ &= (2 - \frac{1}{m})C^*_{max} \end{aligned}$$

Since the bound $2 - \frac{1}{m}$ can actually be achieved by the problem instance in Figure 1.5, we conclude that the worst-case performance bound for list scheduling $R[LS]$ is equal to $2 - \frac{1}{m}$. ◊

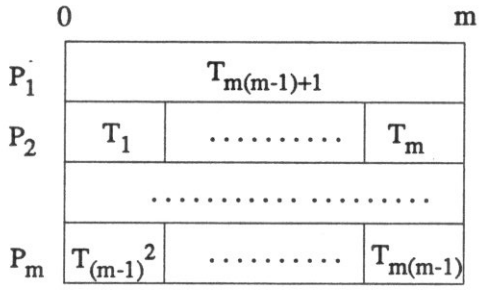
1.4 The Outline of the Thesis

In the following chapters, we will study both scheduling and bin packing problems from the point of view of worst-case performance bounds.

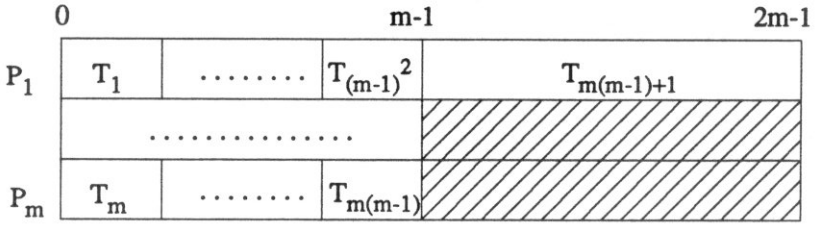
In Chapter 2, the performance bound of optimal non-preemptive scheduling vs. optimal preemptive scheduling is studied based on the previous work done by C. L. Liu [24] in early seventies. A new upper bound is given.



(a)



(b)



(c)

Figure 1. 5 (a)Task system with n=m(m-1)+1 independent tasks; (b) The optimal schedule : $C_{max}^{*}=m$; (c) The list schedule for L $=(T_1, T_2, \ldots, T_{m(m-1)+1})$: $LS(C_{max})=2m-1$.

In chapter 3 and 4, we will study the worst-case performance bounds of some approximation algorithms for bin packing, such as Next-k-Fit and Best-k-Fit. Tight bounds are proved, which resolves a longstanding open problem.

2

Optimal Preemptive and Non-preemptive Schedulings

Optimal Preemptive and Non-preemptive Schedulings

2.1 C. L. Liu's Conjecture

In a multi-processor system, scheduling the tasks on the processors can be done with two different strategies. A scheduling strategy is said to be non-preemptive if once the execution of a task begins on a processor, it must continue until its total completion. A scheduling strategy is said to be preemptive if the execution of a task can be interrupted at any time and be resumed later on. Obviously, preemption generates a faster scheduling than non-preemption. On the other hand, to determine when to have preemption during the execution of a task may require extra time. Suppose that we want to minimize the total elapsed time for the execution of all tasks. Our questions are: How good is preemption vs. non-preemption? Is it worthwhile to waste a lot of time to have a preemptive scheduling?

Let us consider the following scheduling problem. A task system containing n tasks $T = \{T_1, T_2, \ldots, T_n\}$ is to be scheduled on an m-processor system. The processing time of task T_j is p_j for $j = 1, 2, \ldots, n$. Moreover, precedence constraints $\prec$ among the tasks are specified. By $T_i \prec T_j$, we mean that the execution of T_j will not begin until the completion of the execution of T_i. Let ω denote the total elapsed time for the execution of all the tasks in set T, i.e., $C_{\max}$, when the optimal non-preemptive scheduling is used. Let ω' denote the total elapsed time of the schedule when the optimal preemptive

scheduling is used. Figure 2.1 shows both optimal preemptive and optimal non-preemptive schedulings of a task system on two processors.

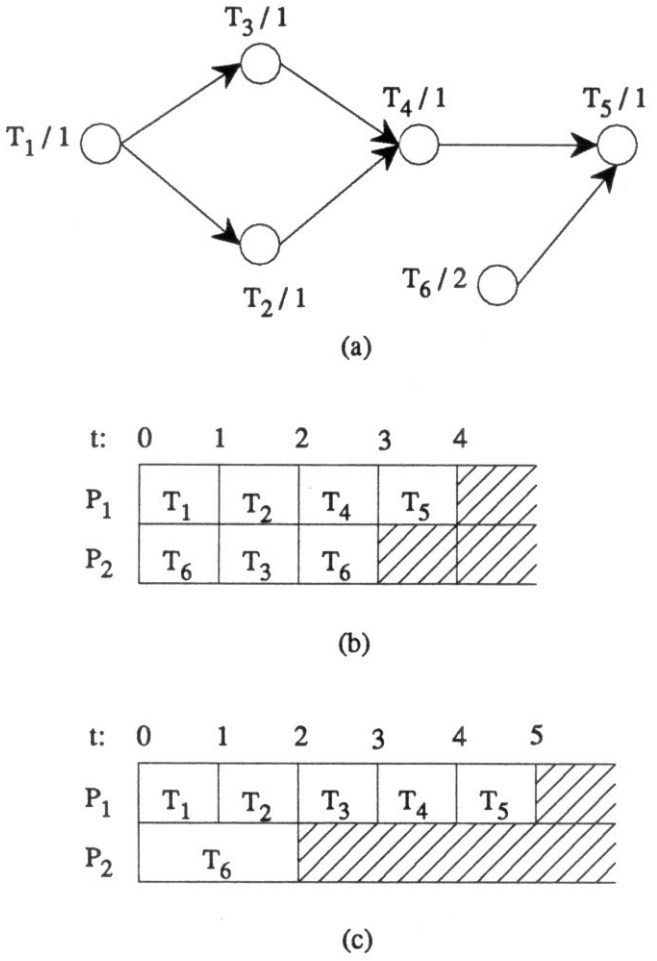


Figure 2. 1 (a)A task system; (b) The optimal preemptive scheduling in a two-processor system; (c) The optimal non-preemptive scheduling.

To compare preemption and non-preemption, we are interested in deter-

mining the maximum value of $\frac{\omega}{\omega'}$ over all schedules. In the early seventies, C. L. Liu [24] studied this problem and made the following conjecture.

Conjecture (C. L. Liu)

$$\frac{\omega}{\omega'} \leq \frac{2m}{m+1}$$

and this bound is the best possible. ◊

In the next two subsections, we will study this conjecture first for the case when the precedence constraint $\prec$ is empty, and then for the case with a general precedence constraint.

Before doing that, we first review some results, which were derived prior to Liu's conjecture.

Theorem 2.1 (R. L. Graham [14])

Let ϕ be the total idle time in the optimal non-preemptive schedule, and m be the number of processors in the system. Also, let C be the longest path or chain in the partial order DAG, and $|C|$ be the length of the chain. Therefore we have,

$$\phi \leq (m-1)|C|$$

Proof See [14]. ◊

Using Theorem 2.1, R. R. Muntz [25] studied the ratio $\frac{\omega}{\omega'}$ and proved the following theorem.

Theorem 2.2 (R. R. Muntz)

If m is the number of processors, then

$$\frac{\omega}{\omega'} \leq \frac{2m-1}{m}$$

Proof Assume X is the total processing time of all tasks, and ϕ is the total idle time in the optimal non-preemptive schedule, then it is easy to show that $\omega = \frac{1}{m}(X+\phi)$. Thus by Theorem 2.1, we have $\omega = \frac{1}{m}(X+\phi) \leq \frac{1}{m}X + \frac{m-1}{m}|C|$. On the other hand, the total elapsed time in the optimal preemptive scheduel ω' must be no less than $\frac{1}{m}X$ since the best possibility is to let all processors busy all the time. ω' must also be no less than the length of the chain $|C|$ since all the tasks in the chain have to be executed sequentially. Thus, we have $\omega' \geq \max\{\frac{1}{m}X, |C|\}$.

$$\begin{aligned}
\frac{\omega}{\omega'} &\leq \frac{\frac{1}{m}X + \frac{m-1}{m}|C|}{\max\{\frac{1}{m}X, |C|\}} \\
&= \frac{\frac{1}{m}X}{\max\{\frac{1}{m}X, |C|\}} + \frac{m-1}{m} \cdot \frac{|C|}{\max\{\frac{1}{m}X, |C|\}} \\
&\leq 1 + \frac{m-1}{m} \cdot 1 \\
&= \frac{2m-1}{m}
\end{aligned}$$

So the ratio $\frac{\omega}{\omega'}$ is no greater than $\frac{2m-1}{m}$ over all possible schedules. ◇

The bound which is proved in Theorem 2.2 may not be the tightest because no schedule that achieves bound $\frac{2m-1}{m}$ has ever been found. On the contrary, the largest achievable bound found so far is $\frac{2m}{m+1}$. So the following

question arises: Is there a possibility that the tight bound of the ratio is exactly $\frac{2m}{m+1}$? We will prove in subsection 2.2 that this is indeed the case when the precedence constraint is empty.

2.2 Independent Tasks

Theorem 2.3

When the precedence constraint $\prec$ is empty,

$$\frac{\omega}{\omega'} \leq \frac{2m}{m+1}$$

and this bound is the best possible.

Proof First we look at the optimal preemptive scheduling. It is easy to show that $\omega' \geq \max_{1 \leq j \leq n}\{p_j\}$ and $\omega' \geq \frac{1}{m}X$.

Then let us consider the optimal non-preemptive scheduling. Let LPT stand for the longest-processing-time-first list scheduling, in which we order the tasks by p_j decreasingly; whenever a processor is idle assign the next task on the list to the idle processor. LPT is actually list scheduling. Since there are no precedence constraints among the tasks, there won't be any idle time during the entire execution of all tasks. See Figure 2.2.

Let T_k be the task that finishes last, i.e., $C_k = LPT(C_{max})$. Let Δ be the starting time of T_k. Because LPT scheduling may not be the optimal one, $\omega \leq LPT(C_{max}) = \Delta + p_k \leq \frac{1}{m}(X - p_k) + p_k = \frac{1}{m}X + \frac{m-1}{m}p_k \leq \omega' + \frac{m-1}{m}p_k$. Let us consider the following two cases.

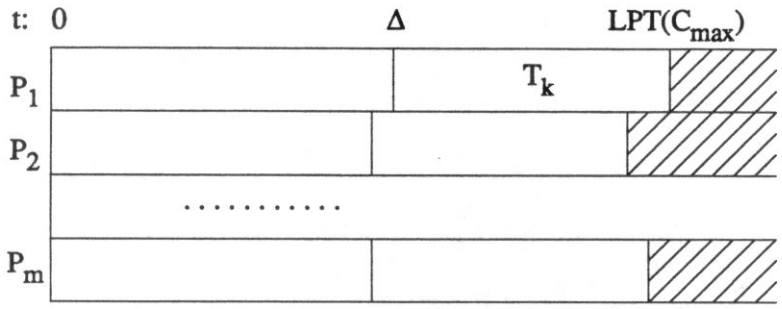


Figure 2. 2 The longest-processing-time-first list scheduling.

case 1. $\Delta > 0$

Every task T_j starting at time $t = 0$ has $p_j \geq p_k$ since it is LPT. And there are at least m such tasks. Let Y be the sum of the processing time of these tasks. So,

$$p_k \leq \frac{1}{m}Y \leq \frac{1}{m}(X - p_k)$$

$$p_k + \frac{1}{m}p_k \leq \frac{1}{m}X \leq \omega'$$

$$\frac{m+1}{m}p_k \leq \omega'$$

$$\frac{m-1}{m}p_k \leq \frac{m-1}{m+1}\omega'$$

Then we have,

$$\omega \leq \omega' + \frac{m-1}{m}p_k \leq \omega' + \frac{m-1}{m+1}\omega'$$

$$\frac{\omega}{\omega'} \leq \frac{2m}{m+1}$$

case 2. $\Delta = 0$

In this case, LPT is in fact the optimal non-preemptive schedule. Thus $\omega = p_k$ and $\omega' = p_k$.

$$\frac{\omega}{\omega'} = 1 < \frac{2m}{m+1}$$

The bound $\frac{2m}{m+1}$ is the best possible because it can be exactly achieved by the example in Figure 2.3. $\Diamond$

2.3 Tasks With Precedence Constraints

In subsection 2.2, we have shown that Liu's conjecture is true when the precedence constraint $\prec$ is empty. In this section, we will look at the situation when $\prec$ is non-empty. Let X be the total processing time of all tasks, and ϕ and ϕ' be the total idle time of optimal non-preemptive and optimal preemptive schedules respectively. It is easy to prove that if m is the number of processors in the system, $\omega = \frac{1}{m}(X + \phi)$ and $\omega' = \frac{1}{m}(X + \phi')$. We will consider the following three possible cases: $X \geq \frac{m+1}{2}\omega$, $X \leq \frac{m^2+1}{m+1}\omega'$, and $\frac{m^2+1}{m+1}\omega' < X < \frac{m+1}{2}\omega$.

Lemma 2.1

If $X \geq \frac{m+1}{2}\omega$, then

$$\frac{\omega}{\omega'} \leq \frac{2m}{m+1}$$

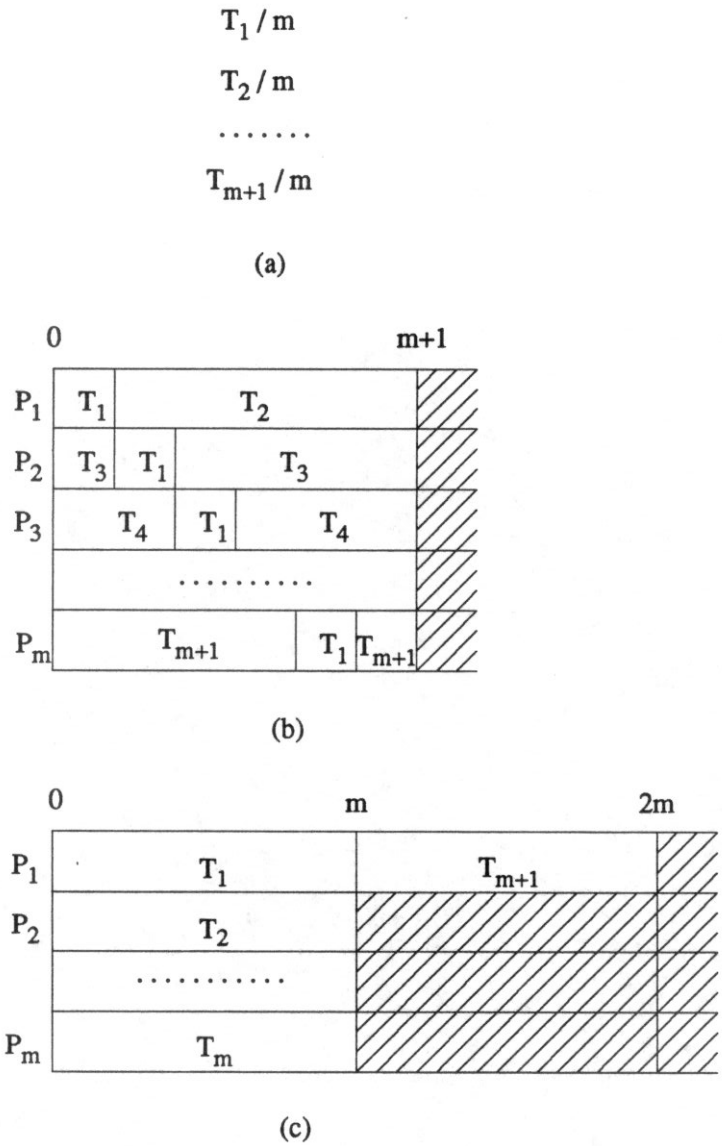


Figure 2. 3 (a) An independent task system; (b) Optimal preemptive scheduling: $\omega'=m+1$; (c) Optimal non-preemptive scheduling: $\omega=2m$.

Proof Since the best we can do in any schedule (non-preemptive as well as preemptive) is never to leave any processor idle,

$$\omega' = \frac{1}{m}(X + \phi') \geq \frac{1}{m}X \geq \frac{m+1}{2m}\omega$$

Thus, the ratio $\frac{\omega}{\omega'}$ is no greater than $\frac{2m}{m+1}$. ◊

Lemma 2.2

If $X \leq \frac{m^2+1}{m+1}\omega'$, then

$$\frac{\omega}{\omega'} \leq \frac{2m}{m+1}$$

Proof Because $X \leq \frac{m^2+1}{m+1}\omega'$, then $\phi' = m\omega' - X \geq m\omega' - \frac{m^2+1}{m+1}\omega' = \frac{m-1}{m+1}\omega'$.

Theorem 2.1 also tells us that $\phi \leq (m-1)|C|$.

$$\phi' = (m-1)|C| - (m-1)|C| + \phi' \geq \phi - (m-1)\omega' + \frac{m-1}{m+1}\omega'$$

$$\phi - \phi' \leq \frac{m(m-1)}{m+1}\omega'$$

$$m\omega - X - m\omega' + X \leq \frac{m(m-1)}{m+1}\omega'$$

$$\frac{\omega}{\omega'} \leq \frac{2m}{m+1}$$

Again, C. L. Liu's conjecture is true in this case. ◊

Lemma 2.3

If $\frac{m^2+1}{m+1}\omega' < X < \frac{m+1}{2}\omega$, and let $\alpha = \frac{\phi'}{\omega'}$, then

$$\frac{\omega}{\omega'} \leq \frac{2m-1}{m} - \frac{1}{m}\alpha$$

Proof Since $\frac{m^2+1}{m+1}\omega' < X < \frac{m+1}{2}\omega$, then $0 \leq \phi' < \frac{m-1}{m^2+1}X$ and $\phi > \frac{m-1}{m+1}X$. The approach we use is simple. We want to increase the idle time of the optimal preemptive schedule by adding a new task so that this case will become the case in Lemma 2.2, and we can use the result in Lemma 2.2.

Suppose that the original task system is $T = \{T_1, T_2, \ldots, T_n\}$. Now let T_{n+1} with processing time $p_{n+1} = \frac{1}{m^2}X - \frac{m^2+1}{m^2(m-1)}\phi'$ be added to the task system T. And also assume that for any task T_j, $1 \leq j \leq n$, we have $T_j \prec T_{n+1}$. It is not too hard to prove that the new optimal scheduling with T_{n+1} added is in fact the old optimal scheduling with T_{n+1} appended at the end. See Figure 2.4.

Since $\phi'_{new} = \phi' + (m-1)p_{n+1} = \phi' + \frac{m-1}{m^2}X - \frac{m^2+1}{m^2}\phi' = \frac{m-1}{m^2}X - \frac{1}{m^2}\phi' = \frac{m-1}{m^2+1}(X + p_{n+1}) = \frac{m-1}{m^2+1}X_{new}$, then the new scheduling actually falls into the case in Lemma 2.2. Hence we have

$$\frac{\omega_{new}}{\omega'_{new}} = \frac{\omega + p_{n+1}}{\omega' + p_{n+1}} \leq \frac{2m}{m+1}$$

$$(m+1)\omega \leq 2m\omega' + (m-1)p_{n+1} = 2m\omega' + (m-1)(\frac{1}{m^2}X - \frac{m^2+1}{m^2(m-1)}\phi')$$

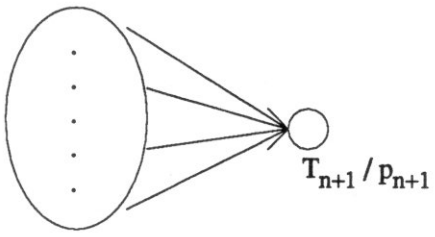
$$(m+1)\omega \leq \frac{(m+1)(2m-1)}{m}\omega' - \frac{m+1}{m}\phi'$$

Assume that $\phi' = \alpha\omega'$, where $0 \leq \alpha < \frac{m-1}{m+1}$, then

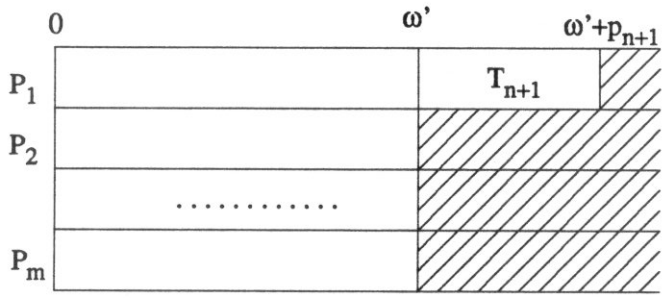
$$(m+1)\omega \leq \frac{(m+1)(2m-1)}{m}\omega' - \frac{m+1}{m}\alpha\omega'$$

$$\frac{\omega}{\omega'} \leq \frac{2m-1}{m} - \frac{1}{m}\alpha$$

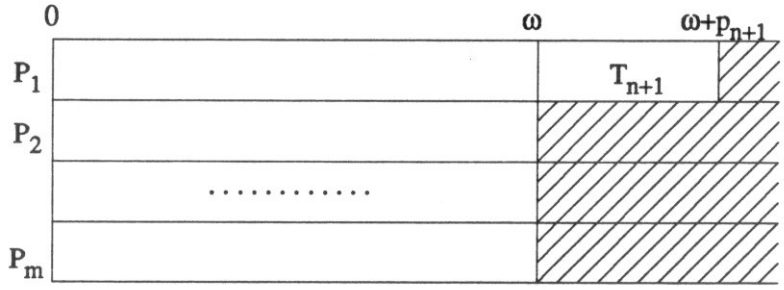
This proves Lemma 2.3. ◊



(a)



(b)



(c)

Figure 2. 4 (a)New task T_{n+1} is added to the original task system; (b) The new optimal preemptive scheduling : $\omega'_{new}=\omega'+p_{n+1}$; (c) The new optimal non-preemptive scheduling : $\omega_{new}=\omega+p_{n+1}$.

In Lemma 2.3, the bound on the ratio $\frac{\omega}{\omega'}$ depends on the value of α, as can be seen from Figure 2.5. The larger α is, the closer the ratio is to the one in Liu's conjecture. If α is 0, the ratio becomes the one in Theorem 2.2 given by Muntz in his Ph.D. thesis [25]. However, we don't know whether the bound proved in Lemma 2.3 is the best possible or not. It is very likely that this bound can be improved to that in Liu's conjecture since no schedule with the ratio greater than $\frac{2m}{m+1}$ has been found.

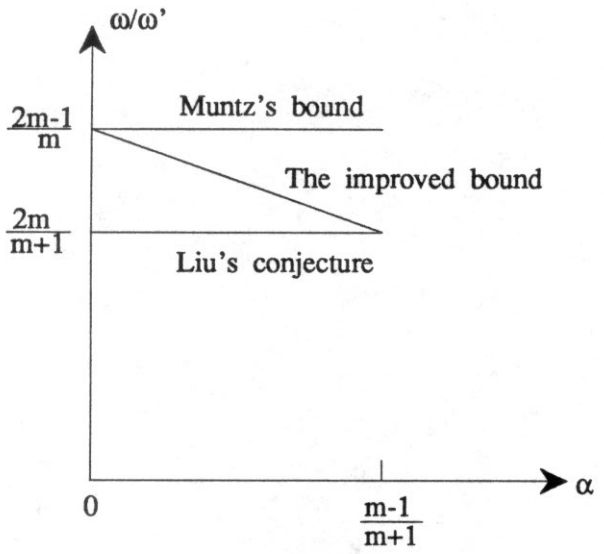


Figure 2. 5 The improved bound of the ratio when $((m^2+1)/(m+1))\omega' < X < ((m+1)/2)\omega$.

Combining the results in the above three lemmas, we have the following theorem.

Theorem 2.4

If m is the number of processors, and let $\alpha = \frac{\phi'}{\omega'}$, then,

$$\frac{\omega}{\omega'} \leq \begin{cases} \frac{2m}{m+1} & \text{if } X \geq \frac{m+1}{2}\omega \text{ or } X \leq \frac{m^2+1}{m+1}\omega'; \\ \frac{2m-1}{m} - \frac{1}{m}\alpha & \text{otherwise.} \end{cases}$$

3

Next-k-Fit Bin Packing

Next-k-Fit Bin Packing

3.1 Problem Definition

Given a finite list $L = (a_1, a_2, \ldots, a_m)$ of reals in (0, 1], and a sequence of unit-capacity bins, $B_1, B_2, \ldots$, the bin packing problem is to pack the numbers in the list into the bins such that no bin contains a total exceeding 1 and that the number of bins used is minimized.

Example

Suppose $L = (0.5, 0.2, 1, 0.9, 0.25)$, Figure 3.1 shows the optimal packing of L, which uses three bins. ◇

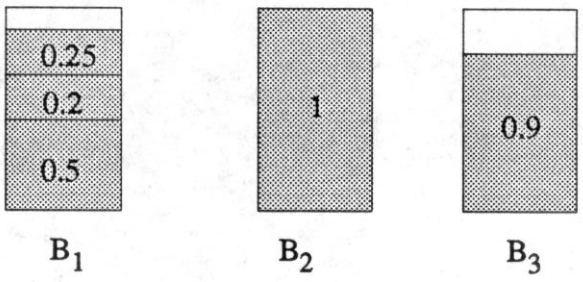


Figure 3. 1 The optimal packing of L=(0.5, 0.2, 1, 0.9, 0.25).

The bin packing problem is proved to be NP-complete by R. M. Karp [18], and no polynomial-time complexity algorithm has ever been found so far. A lot of effort has been made to find good approximation algorithms for the problem.

As we pointed out in Chapter 1, in order to evaluate and compare

the quality of different approximation algorithms, we need to have a rigorous mathematical analysis of the worst-case behavior of these algorithms. Given an approximation algorithm A, and for any list L, let $A(L)$ be the number of bins used in the packing resulting when A is applied to L and L^* be the minimum number of bins needed to pack L. Therefore, the worst-case performance bound of the approximation algorithm A is $R[A] = \limsup \max\{A(L)/L^*\}$ as $L^* \to \infty$.

Besides those well-studied approximation algorithms as First-Fit (FF), Best-Fit (BF), First-Fit-Decreasing (FFD), Best-Fit-Decreasing (BFD), and Next-Fit (NF) [15][16][17], whose worst-case performance bounds are shown in Table 3.1, there is another important class of algorithms called Next-k-Fit (NkF), where k is an integer greater than 1. In NkF, we process the numbers in L in turn, starting with a_1, which is placed at the bottom of first bin B_1. Suppose that a_i is now to be packed. We look at the last k non-empty bins. If a_i does not fit into any of them, a new bin is created; otherwise, a_i will go to the lowest indexed one of these k non-empty bins into which it fits. D. S. Johnson [16] proved that $1.7 + \frac{3}{10k} \leq R[NkF] \leq 2$. In this chapter, we study the worst-case performance bound for the Next-k-Fit algorithm. Our result is the following theorem.

Theorem 3.1

$$R[NkF] = 1.7 + \frac{3}{10(k-1)}, \quad k \geq 2$$

In subsection 3.2 we will prove the theorem for the special case $k = 2$,

since in this case the claimed limit equals the previously known upper bound, all we need to do is to provide a lower bound example. In subsection 3.3, we look at the general case when $k \geq 3$.

A	R[A]
FF	1.7
BF	1.7
FFD	11 / 9
BFD	11 / 9
NF	2.0

Table 3. 1 Some approximation algorithms for bin packing and their worst-case performance bounds.

3.2 Next-2-Fit Algorithm

It was known that $1.85 \leq R[N2F] \leq 2$ [16]. Here we show that the precise value equals the old upper bound.

Theorem 3.2

$$R[N2F] = 2$$

Proof Consider the following list L, where $0 < \epsilon \ll 1$,

$\frac{1}{2} - \frac{\epsilon}{2}, \frac{1}{2} + \epsilon, \epsilon,$

$\frac{1}{2} - \frac{\epsilon}{4}, \frac{1}{2} + \frac{\epsilon}{2}, \frac{\epsilon}{2},$

.....

$\frac{1}{2} - \frac{\epsilon}{2^n}, \frac{1}{2} + \frac{\epsilon}{2^{n-1}}, \frac{\epsilon}{2^{n-1}}.$

Since we can pack the list in a way using only $n+1$ bins (Figure 3.2(a)), $L^* \leq n+1$. On the other hand, if we apply $N2F$ to L, we will get the packing shown in Figure 3.2(b). Thus, $N2F(L) = 2n$. By definition, $R[N2F] = \limsup \max\{N2F(L)/L^*\} \geq \lim_{n\to\infty} \frac{2n}{n+1} = 2$. Together with $R[N2F] \leq 2$, we have $R[N2F] = 2$. ◇

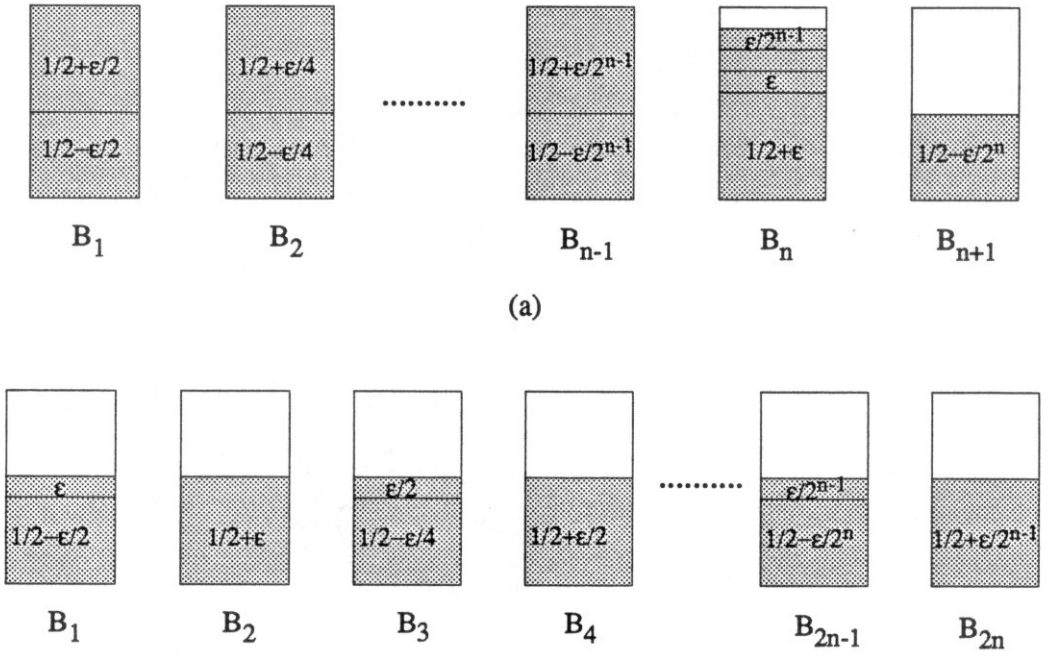


Figure 3. 2 (a) Optimal packing for L; (b) N2F packing for L.

3.3 Next-k-Fit Algorithm ($k \geq 3$)

We already know from [16] that for $k \geq 3$, $R[NkF]$ is between $1.7 + \frac{3}{10k}$ and 2. In this subsection, we will concentrate on finding the exact value for $R[NkF]$.

3.3.1 Lower Bound

Let us start from the easier part. As usual, we can prove the lower bound result by giving an instance of a list which yields a high performance bound.

Theorem 3.3

$$R[NkF] \geq 1.7 + \frac{3}{10(k-1)}, \quad k \geq 3$$

Proof Suppose $(k-2)|10n$, $0 < \epsilon \ll 18^{-n}$, $\epsilon_i = \epsilon \times 18^{n-i}$, for $i = 1, 2, \ldots, n$. Let us look at the list L which contains the following three groups of numbers in the order of their appearance.

The first group consists of:

$\frac{1}{6}+33\epsilon_1, \frac{1}{6}-3\epsilon_1, \frac{1}{6}-7\epsilon_1, \frac{1}{6}-7\epsilon_1, \frac{1}{6}-13\epsilon_1, \frac{1}{6}+9\epsilon_1, \frac{1}{6}-2\epsilon_1, \frac{1}{6}-2\epsilon_1, \frac{1}{6}-2\epsilon_1, \frac{1}{6}-2\epsilon_1,$

$\frac{1}{6}+33\epsilon_2, \frac{1}{6}-3\epsilon_2, \frac{1}{6}-7\epsilon_2, \frac{1}{6}-7\epsilon_2, \frac{1}{6}-13\epsilon_2, \frac{1}{6}+9\epsilon_2, \frac{1}{6}-2\epsilon_2, \frac{1}{6}-2\epsilon_2, \frac{1}{6}-2\epsilon_2, \frac{1}{6}-2\epsilon_2,$

.....

$\frac{1}{6}+33\epsilon_n, \frac{1}{6}-3\epsilon_n, \frac{1}{6}-7\epsilon_n, \frac{1}{6}-7\epsilon_n, \frac{1}{6}-13\epsilon_n, \frac{1}{6}+9\epsilon_n, \frac{1}{6}-2\epsilon_n, \frac{1}{6}-2\epsilon_n, \frac{1}{6}-2\epsilon_n, \frac{1}{6}-2\epsilon_n.$

The second group contains:

$\frac{1}{3}+46\epsilon_1, \frac{1}{3}-34\epsilon_1, \frac{1}{3}+6\epsilon_1, \frac{1}{3}+6\epsilon_1, \frac{1}{3}+12\epsilon_1, \frac{1}{3}-10\epsilon_1, \frac{1}{3}+\epsilon_1, \frac{1}{3}+\epsilon_1, \frac{1}{3}+\epsilon_1, \frac{1}{3}+\epsilon_1,$

$\frac{1}{3}+46\epsilon_2, \frac{1}{3}-34\epsilon_2, \frac{1}{3}+6\epsilon_2, \frac{1}{3}+6\epsilon_2, \frac{1}{3}+12\epsilon_2, \frac{1}{3}-10\epsilon_2, \frac{1}{3}+\epsilon_2, \frac{1}{3}+\epsilon_2, \frac{1}{3}+\epsilon_2, \frac{1}{3}+\epsilon_2,$

.....

$\frac{1}{3}+46\epsilon_n, \frac{1}{3}-34\epsilon_n, \frac{1}{3}+6\epsilon_n, \frac{1}{3}+6\epsilon_n, \frac{1}{3}+12\epsilon_n, \frac{1}{3}-10\epsilon_n, \frac{1}{3}+\epsilon_n, \frac{1}{3}+\epsilon_n, \frac{1}{3}+\epsilon_n, \frac{1}{3}+\epsilon_n.$

The last group has the following numbers, where in each row there are $k+1$ numbers, and h is an integer equal to $\frac{10n}{k-2}$.

$\frac{1}{2} - \frac{\epsilon}{2}$, $\frac{1}{2} + \epsilon$, $\frac{1}{2} + \epsilon$, ..., $\frac{1}{2} + \epsilon$, ϵ,

$\frac{1}{2}-\frac{\epsilon}{4}, \frac{1}{2}+\frac{\epsilon}{2}, \frac{1}{2}+\frac{\epsilon}{2}, \ldots, \frac{1}{2}+\frac{\epsilon}{2}, \frac{\epsilon}{2},$

.....

$\frac{1}{2}-\frac{\epsilon}{2^h}, \frac{1}{2}+\frac{\epsilon}{2^{h-1}}, \frac{1}{2}+\frac{\epsilon}{2^{h-1}}, \ldots, \frac{1}{2}+\frac{\epsilon}{2^{h-1}}, \frac{\epsilon}{2^{h-1}}.$

As we see, the first group, which has $10n$ numbers, contains numbers slightly greater or smaller than $\frac{1}{6}$. The second group, also having $10n$ numbers, contains those numbers slightly greater or smaller than $\frac{1}{3}$. And the third group, which has $\frac{(k+1)\times 10n}{k-2}$ numbers, contains some numbers around $\frac{1}{2}$ and some very small numbers.

When we apply NkF algorithm to L, we find that in the first group every 5 consecutive numbers will be packed into one bin, and in the second group every 2 consecutive numbers will be packed into one bin, and in the third group every number except for the last small one in each row will be packed into one bin. So altogether we need $\frac{10n}{5}+\frac{10n}{2}+\frac{k\times 10n}{k-2}=7n+\frac{10nk}{k-2}$ bins. Hence, $NkF(L)=7n+\frac{10nk}{k-2}$.

How about the optimal packing? Clearly, we can arrange the packing in the following way. In the third group, we pick out the first, second, and the last numbers in each row, which in fact form the list in the proof of Theorem 3.2. We can use $h+1$ bins to pack them. As to the remaining numbers in our list, they are in fact the same list used in proving the lower bound result for $R[FF]$ by D. S. Johnson *et al* in [17]. We can employ the same method, using $10n+1$ bins, to pack them. Then $L^* \leq h+1+10n+1 = \frac{10n}{k-2}+10n+2=\frac{10n(k-1)}{k-2}+2$.

$$R[NkF] = \limsup \max\{NkF(L)/L^*\}$$

$$\geq \lim_{n\to\infty}(7n + \frac{10nk}{k-2})/(\frac{10n(k-1)}{k-2} + 2)$$

$$= 1.7 + \frac{3}{10(k-1)}$$

This proves the lower bound result for Next-k-Fit. ◇

3.3.2 Upper Bound

Theorem 3.4

$$R[NkF] \leq 1.7 + \frac{3}{10(k-1)}, \quad k \geq 3$$

This is the more difficult part. We need some preliminary results. Given any list L, after applying NkF to L, we get $NkF(L)$ non-empty bins, $B_1, B_2, \ldots, B_{NkF(L)}$. For each bin B_i, its content can be divided into k areas, $A_{i,1}, A_{i,2}, \ldots, A_{i,k}$, where $A_{i,1}$ contains all the numbers coming to B_i when B_i is the rightmost, or in other words the most recently created non-empty bin in the packing, and $A_{i,2}$ contains all the numbers coming to B_i when B_i becomes the second rightmost non-empty bin, etc. Finally, $A_{i,k}$ contains all the numbers coming to B_i when B_i becomes the oldest among the k active bins and is about to be thrown away. As we will see, this kind of analysis gives us a deeper insight of the packing. Figure 3.3 shows the division for $N3F$.

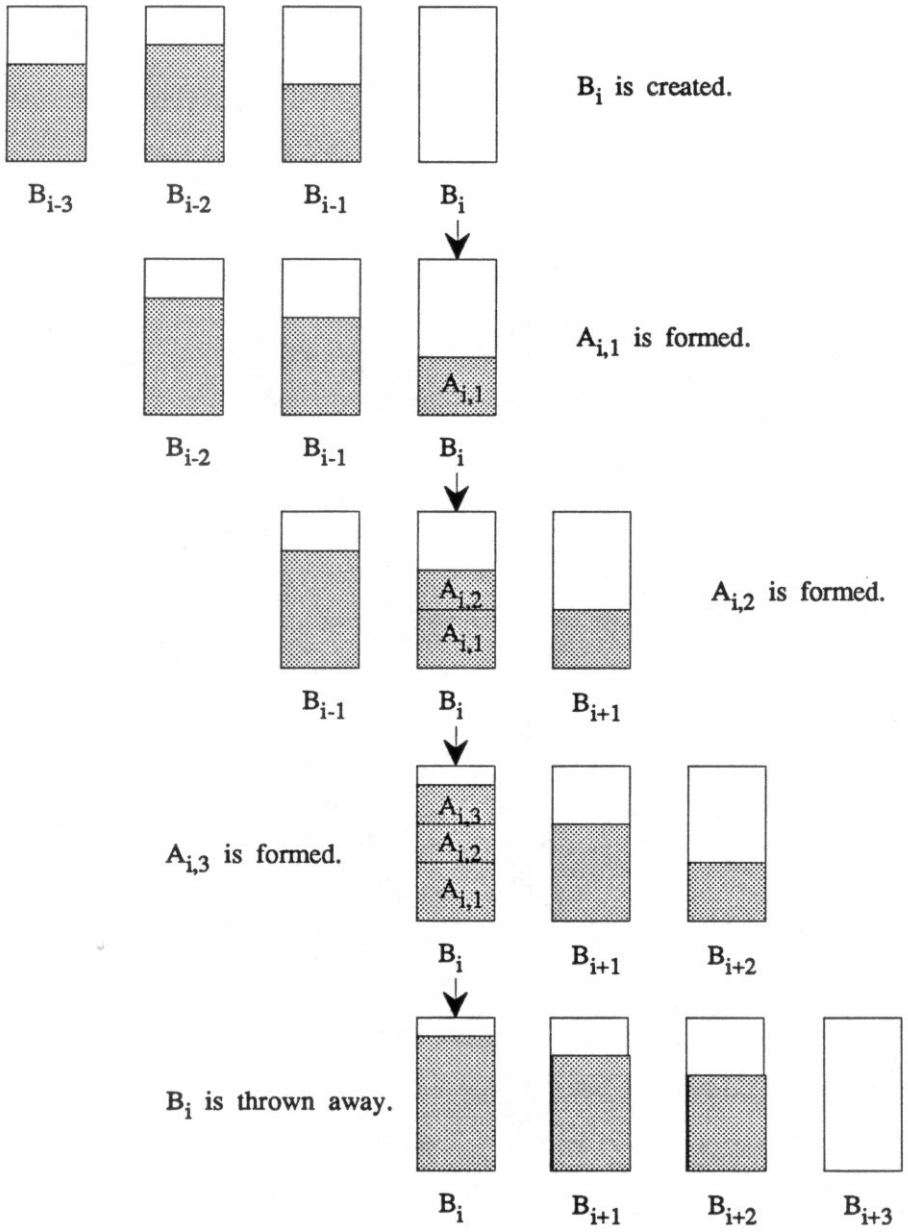


Figure 3. 3 How the three areas of B_i in N3F packing are formed.

Weighting function is a classified method for proving upper bounds, though it is not always easy to find a suitable weighting function. To prove

Theorem 3.4, we wish to show that $NkF(L) \leq (1.7 + \frac{3}{10(k-1)})L^* + c$ for all L, where c is a constant. With the help of the following weighting function $W : (0,\ 1] \rightarrow R^+$, we will find the relation between $NkF(L)$ and L^*.

$$W(\alpha) = \begin{cases} \frac{6}{5}\alpha & \text{if } \alpha \in (0,\ \frac{1}{6}]; \\ \frac{9}{5}\alpha - \frac{1}{10} & \text{if } \alpha \in (\frac{1}{6},\ \frac{1}{3}]; \\ \frac{6}{5}\alpha + \frac{1}{10} & \text{if } \alpha \in (\frac{1}{3},\ \frac{1}{2}]; \\ \frac{6}{5}\alpha + \frac{2}{5} + \frac{3}{10(k-1)} & \text{if } \alpha \in (\frac{1}{2},\ 1]. \end{cases}$$

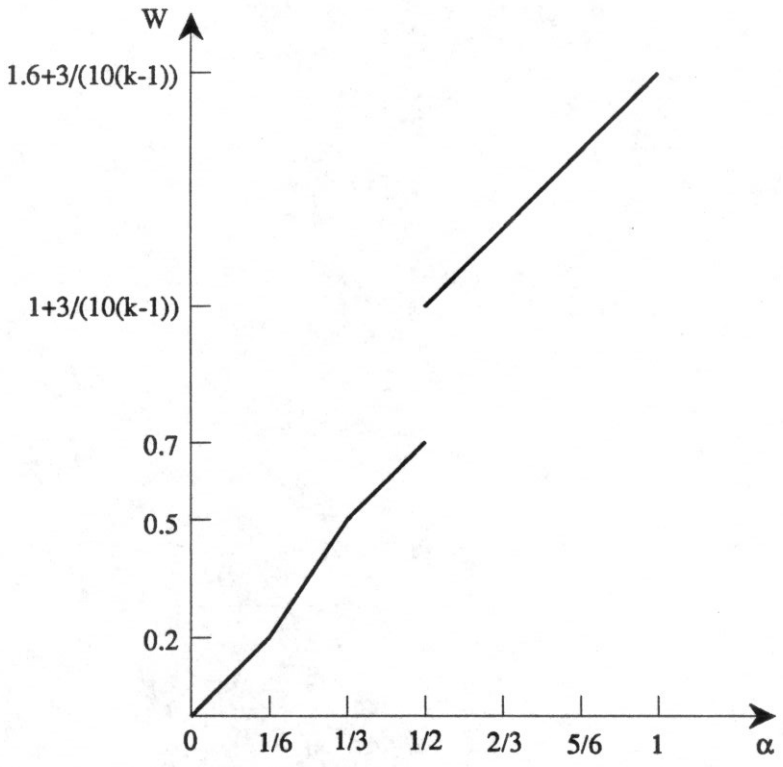


Figure 3. 4 The weighting function W(α).

For any number a_i in L, $W(a_i)$ is called the weight of a_i. We define $W(B_i)$, the weight of the bin B_i, as the sum of the weight of all numbers in B_i, i.e. $W(B_i) = \sum_{\forall a_j \in B_i} W(a_j)$. And W(L), the weight of the list L, is defined to be the sum of the weight of all numbers in L. i.e. $W(L) = \sum_{\forall a_j \in L} W(a_j)$. When there is no possibility of confusion, we also use B_i to denote the sum of the numbers in bin B_i, $A_{i,h}$ the sum of the numbers in area $A_{i,h}$, and b_i the bottommost item in bin B_i.

Claim 3.1

For any bin B of items of total size 1 or less,

$$W(B) \leq 1.7 + \frac{3}{10(k-1)}$$

Proof See the proof of Lemma 1 in the paper "Resource constrained scheduling as generalized bin packing", by Garey, Graham, Johnson and Yao [10]. We note that our weighting function differs from that in the reference only by the addition of $\frac{3}{10(k-1)}$ for the items of size exceeding $\frac{1}{2}$, and there can be only one such item in B. So the bound in the claim exceeds the bound 1.7 in the reference by precisely this amount. ◇

Claim 3.2

For any list L,

$$W(L) \leq (1.7 + \frac{3}{10(k-1)})L^*$$

Proof Apply the optimal algorithm to L. We get L^* non-empty bins. Thus,

$$W(L) = \sum_{i=1}^{L^*} W(B_i)$$

$$\leq \sum_{i=1}^{L^*} (1.7 + \frac{3}{10(k-1)})$$

$$= (1.7 + \frac{3}{10(k-1)})L^*$$

Notice that we used the result in Claim 3.1. ◊

Claim 3.3

There exists a constant c such that for any list L,

$$NkF(L) \leq W(L) + c$$

Proof Given a list L, after applying NkF to L, we get a sequence of non-empty bins $B_1, B_2, \ldots, B_{NkF(L)}$. We prove the claim by induction on the value of $NkF(L)$.

Inductive basis: If $NkF(L) \leq k$, then $NkF(L) = FF(L)$, where $FF(L)$ is the number of bins in the First-Fit bin packing. In the paper "Worst-case performance bounds for simple one-dimensional packing algorithms"[17], D. S. Johnson *et al* showed that $FF(L) \leq W'(L)+2$, in which W' is a weighting function similar to W except that for any item $b > \frac{1}{2}, W'(b) = 1$, while $W(b) > 1$. So $W'(L) \leq W(L)$. Thus, we have $NkF(L) = FF(L) \leq W'(L)+2 \leq W(L)+2$.

Inductive hypothesis: If $NkF(L) = i-1$, then $NkF(L) \leq W(L)+2$.

Inductive step: Assume $NkF(L) \geq i$. By inductive hypothesis, we know that $i-1 \leq \sum_{h=1}^{i-1} W(B_h) + 2$. If $W(B_i) \geq 1$, then $i \leq \sum_{h=1}^{i} W(B_h) + 2$. We are done by induction. Otherwise, when $W(B_i) < 1$, we will show that there exists a j such that $W(B_i) + \cdots + W(B_{i+j-1}) + W(B_{i+j}) \geq j + \frac{6}{5} B_{i+j}$. This means we can borrow weight $\Delta = W(B_{i+j}) - \frac{6}{5} B_{i+j}$, from B_{i+j} to make up the shortfall of $B_i, \ldots, B_{i+j-1}$, and reconstruct B_{i+j}, so it has the same size but has its weight reduced by the amount loaned to $B_i, \ldots, B_{i+j-1}$. Thus, $i+j-1 \leq \sum_{h=1}^{i+j-1} W(B_h) + \Delta + 2$. Since the weight left in B_{i+j} is $\frac{6}{5} B_{i+j}$, we can equivalently assume B_{i+j} has all its numbers no larger than $\frac{1}{6}$ with the total unchanged. Since our inductive step will never care about the content of B_i except its size, i.e., we will only use $W(B_i) = \frac{6}{5} B_i$, the inductive step can again be applied to B_{i+j}. (Figure 3.5)

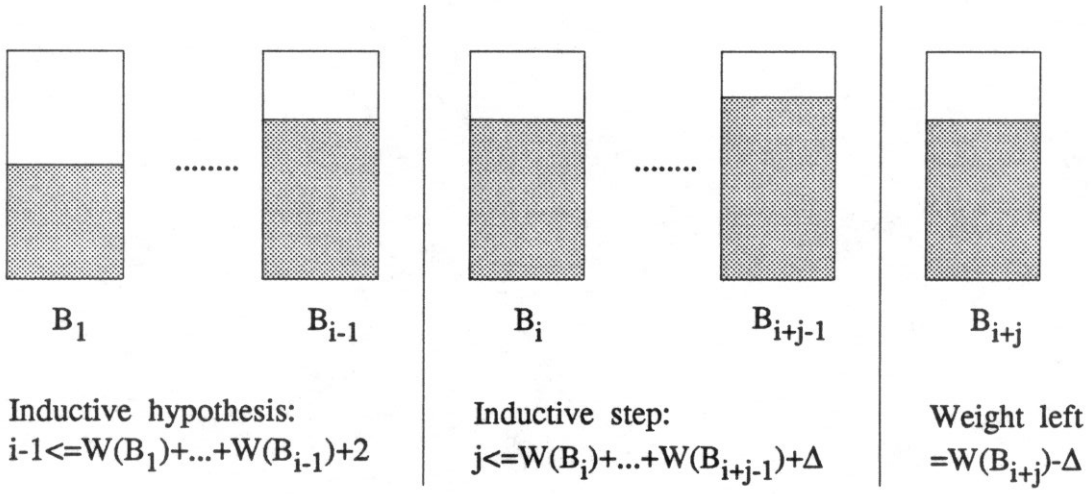


Figure 3. 5 Inductive proof of Claim 3. 3.

Because $W(B_i) < 1$, B_i must be less than $\frac{5}{6}$, and items in $A_{i+1,1}$ and $A_{i+2,1}$ must be greater than $\frac{1}{6}$. For notational simplicity, we assume $i = 1$. Let us consider the following cases.

Case I. If $B_1 \leq \frac{1}{2}$, then B_1 must be followed by k bins with their bottommost items greater than $\frac{1}{2}$, i.e. $b_2, \ldots, b_{k+1} > \frac{1}{2}$. (Figure 3.6)

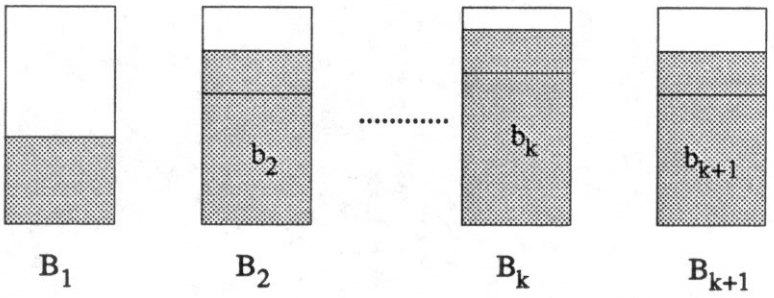


Figure 3. 6 The possible packing when B_1<=1/2.

$W(B_1) + \cdots + W(B_k) + W(B_{k+1})$

$\geq \frac{6}{5}A_{1,1} + (\frac{6}{5}b_2 + \frac{2}{5} + \frac{3}{10(k-1)}) + \cdots + (\frac{6}{5}b_k + \frac{2}{5} + \frac{3}{10(k-1)}) + \frac{6}{5}B_{k+1} + \frac{2}{5} + \frac{3}{10(k-1)}$

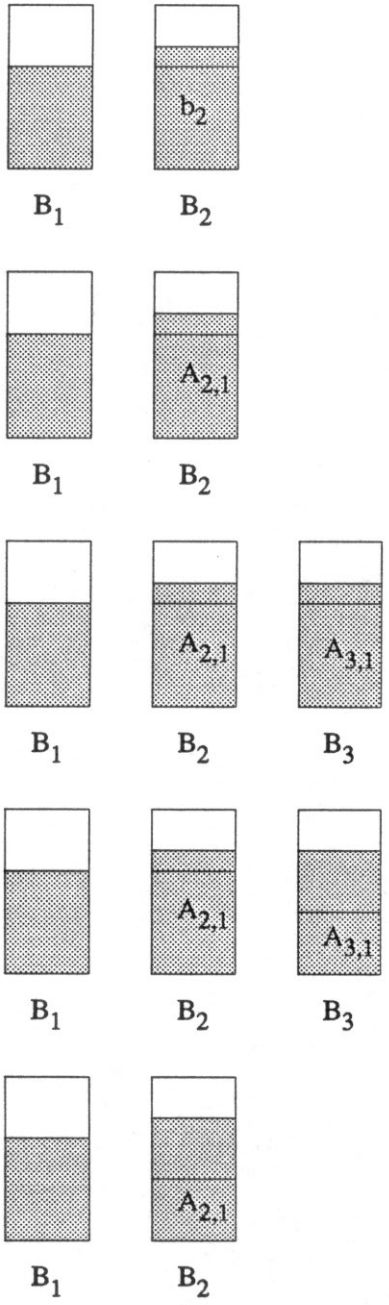
$\geq \frac{6}{5}(A_{1,1} + b_2) + \frac{6}{5}(b_3 + \cdots + b_k) + (\frac{2}{5} + \frac{3}{10(k-1)})k + \frac{6}{5}B_{k+1}$

$\geq \frac{6}{5} \times 1 + \frac{6}{5} \times \frac{1}{2} \times (k-2) + (\frac{2}{5} + \frac{3}{10(k-1)})k + \frac{6}{5}B_{k+1}$

$\geq k + \frac{3}{10} + \frac{3}{10(k-1)} + \frac{6}{5}B_{k+1}$

$\geq k + \frac{6}{5}B_{k+1}$

Case II. If $\frac{1}{2} < B_1 < \frac{5}{6}$, then we consider the following subcases. (Figure 3.7)



case 1: B_2 has one item >1/2

case 2: $A_{2,1}$>1/2, with at least one of two bottommost items in (1/6, 1/3]

case 3: $A_{2,1}$>1/2, with its two bottommost items in (1/3,1/2] and $A_{3,1}$>1/2

case 4: $A_{2,1}$>1/2, with its two bottommost items in (1/3,1/2]and $A_{3,1}$<=1/2

case 5: $A_{2,1}$ <=1/2.

Figure 3. 7 The possible packings for 1/2<B_1<5/6.

subcase II.1. B_2 has one item greater than $\frac{1}{2}$, then,

$W(B_1) + W(B_2)$

$\geq \frac{6}{5}B_1 + \frac{6}{5}B_2 + \frac{2}{5} + \frac{3}{10(k-1)}$

$\geq \frac{6}{5} \times \frac{1}{2} + \frac{2}{5} + \frac{6}{5}B_2$

$\geq 1 + \frac{6}{5}B_2$

Starting from the following case, we assume that all the items in B_2 are no greater than $\frac{1}{2}$.

subcase II.2. $A_{2,1} > \frac{1}{2}$. Since $A_{2,1}$ has at least two items, assume at least one of its two bottommost items is in $(\frac{1}{6}, \frac{1}{3}]$. B_1 is greater than $\frac{2}{3}$.

If the other item is also in $(\frac{1}{6}, \frac{1}{3}]$, then,

$W(B_1) + W(B_2)$

$\geq \frac{6}{5}B_1 + \frac{6}{5}B_2 + \frac{3}{5}(1 - B_1) - \frac{1}{10} + \frac{3}{5}(1 - B_1) - \frac{1}{10}$

$\geq 1 + \frac{6}{5}B_2$

If the other item is in $(\frac{1}{3}, \frac{1}{2}]$, then,

$W(B_1) + W(B_2)$

$\geq \frac{6}{5}B_1 + \frac{6}{5}B_2 + \frac{3}{5}(1 - B_1) - \frac{1}{10} + \frac{1}{10}$

$\geq \frac{3}{5}B_1 + \frac{3}{5} + \frac{6}{5}B_2$

$\geq \frac{3}{5} \times \frac{2}{3} + \frac{3}{5} + \frac{6}{5}B_2$

$\geq 1 + \frac{6}{5}B_2$

subcase II.3. $A_{2,1} > \frac{1}{2}$, with its two bottommost items in $(\frac{1}{3}, \frac{1}{2}]$, and $A_{3,1} > \frac{1}{2}$. It is clear that $B_2 > \frac{2}{3}$.

If $A_{3,1}$ has one item greater than $\frac{1}{2}$, then,

$W(B_1) + W(B_2) + W(B_3)$

$\geq \frac{6}{5}B_1 + \frac{6}{5}B_2 + \frac{1}{10} + \frac{1}{10} + \frac{6}{5}B_3 + \frac{2}{5} + \frac{3}{10(k-1)}$

$\geq \frac{6}{5} \times \frac{1}{2} + \frac{6}{5} \times \frac{2}{3} + \frac{3}{5} + \frac{6}{5}B_3$

$\geq 2 + \frac{6}{5}B_3$

If the two bottommost items of $A_{3,1}$ are in $(\frac{1}{6}, \frac{1}{3}]$, then,

$W(B_1) + W(B_2) + W(B_3)$

$\geq \frac{6}{5}B_1 + \frac{6}{5}B_2 + \frac{1}{10} + \frac{1}{10} + \frac{6}{5}B_3 + \frac{3}{5}(1 - B_1) - \frac{1}{10} + \frac{3}{5}(1 - B_2) - \frac{1}{10}$

$\geq \frac{6}{5} + \frac{3}{5}B_1 + \frac{3}{5}B_2 + \frac{6}{5}B_3$

$\geq \frac{6}{5} + \frac{3}{5}B_1 + \frac{3}{5}(1 - B_1 + \frac{1}{3}) + \frac{6}{5}B_3$

$\geq 2 + \frac{6}{5}B_3$

If one of the two bottommost items is in $(\frac{1}{6}, \frac{1}{3}]$, and the other is in $(\frac{1}{3}, \frac{1}{2}]$, then,

$W(B_1) + W(B_2) + W(B_3)$

$\geq \frac{6}{5}B_1 + \frac{6}{5}B_2 + \frac{1}{10} + \frac{1}{10} + \frac{6}{5}B_3 + \frac{3}{5}(1 - B_2) - \frac{1}{10} + \frac{1}{10}$

$\geq \frac{6}{5}B_1 + \frac{3}{5}B_2 + \frac{4}{5} + \frac{6}{5}B_3$

$\geq \frac{6}{5}B_1 + \frac{3}{5}(1 - B_1 + 1 - B_1) + \frac{4}{5} + \frac{6}{5}B_3$

$\geq 2 + \frac{6}{5}B_3$

If $A_{3,1}$ has two bottommost items in $(\frac{1}{3}, \frac{1}{2}]$, then,

$W(B_1) + W(B_2) + W(B_3)$

$\geq \frac{6}{5}B_1 + \frac{6}{5}B_2 + \frac{1}{10} + \frac{1}{10} + \frac{6}{5}B_3 + \frac{1}{10} + \frac{1}{10}$

$\geq \frac{6}{5}B_1 + \frac{6}{5}(1 - B_1 + \frac{1}{3}) + \frac{2}{5} + \frac{6}{5}B_3$

$\geq 2 + \frac{6}{5}B_3$

subcase II.4. $A_{2,1} > \frac{1}{2}$, with its two bottommost items in $(\frac{1}{3}, \frac{1}{2}]$, and $A_{3,1} \leq \frac{1}{2}$. We need to consider several possibilities according to the area distribution of B_3. In Figure 3.8, on the right side of the vertical line are the three such possible packings that may follow the bins B_1 and B_2.

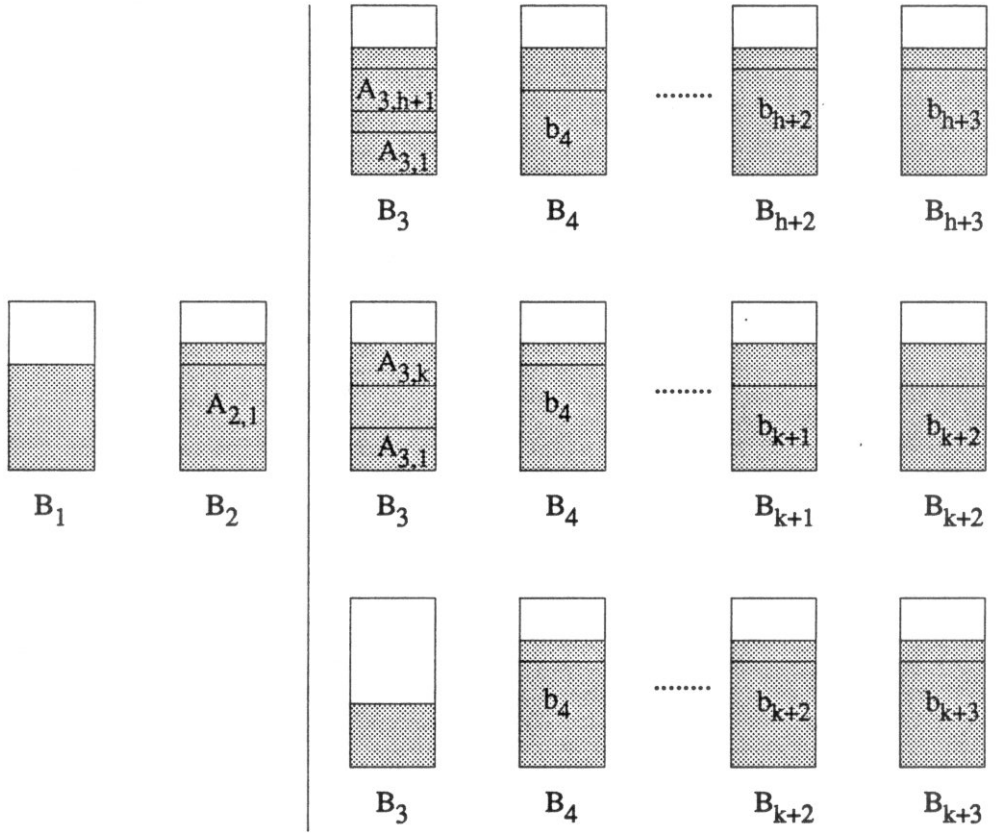


Figure 3. 8 The possible packing when $A_{2,1}$>1/2, with its two bottommost items in (1/3, 1/2], and $A_{3,1}$<=1/2.

If $A_{3,1}+\cdots+A_{3,h} \le \frac{1}{2}$, but $A_{3,1}+\cdots+A_{3,h+1} > \frac{1}{2}$, for $1 \le h \le k-2$, then,

$W(B_1)+\cdots+W(B_{h+3})$

$\ge \frac{6}{5}B_1+\frac{6}{5}B_2+\frac{1}{10}+\frac{1}{10}+\frac{6}{5}B_3+\frac{6}{5}(b_4+\cdots+b_{h+2})+(\frac{2}{5}+\frac{3}{10(k-1)})h+\frac{6}{5}B_{h+3}$

$\ge \frac{6}{5}B_1+\frac{6}{5}B_2+\frac{1}{5}+\frac{6}{5}(A_{3,1}+A_{3,h+1})+\frac{6}{5}\times\frac{1}{2}\times(h-1)+\frac{2}{5}h+\frac{6}{5}B_{h+3}$

$\ge \frac{6}{5}B_1+\frac{6}{5}B_2+\frac{6}{5}(1-B_1+1-B_2)+h-\frac{2}{5}+\frac{6}{5}B_{h+3}$

$\ge h+2+\frac{6}{5}B_{h+3}$

If $A_{3,1}+\cdots+A_{3,k-1} \le \frac{1}{2}$, but $A_{3,1}+\cdots+A_{3,k} > \frac{1}{2}$, then,

$W(B_1)+\cdots+W(B_{k+2})$

$\ge \frac{6}{5}B_1+\frac{6}{5}B_2+\frac{1}{10}+\frac{1}{10}+\frac{6}{5}B_3+\frac{6}{5}(b_4+\cdots+b_{k+1})+(\frac{2}{5}+\frac{3}{10(k-1)})(k-1)+\frac{6}{5}B_{k+2}$

$\ge \frac{6}{5}B_1+\frac{6}{5}(1-B_1+\frac{1}{3})+\frac{1}{5}+\frac{6}{5}\times\frac{1}{2}+\frac{6}{5}\times\frac{1}{2}\times(k-2)+\frac{2}{5}(k-1)+\frac{3}{10}+\frac{6}{5}B_{k+2}$

$\ge k+1+\frac{6}{5}B_{k+2}$

If $A_{3,1}+\cdots+A_{3,k} \le \frac{1}{2}$, then,

$W(B_1)+\cdots+W(B_{k+3})$

$\ge \frac{6}{5}B_1+\frac{6}{5}B_2+\frac{1}{10}+\frac{1}{10}+\frac{6}{5}B_3+\frac{6}{5}(b_4+\cdots+b_{k+2})+(\frac{2}{5}+\frac{3}{10(k-1)})k+\frac{6}{5}B_{k+3}$

$\ge \frac{6}{5}B_1+\frac{6}{5}(1-B_1+\frac{1}{3})+\frac{1}{5}+\frac{6}{5}A_{3,1}+\frac{6}{5}b_4+\frac{6}{5}\times\frac{1}{2}\times(k-2)+\frac{2}{5}k+\frac{3}{10}+\frac{6}{5}B_{k+3}$

$\ge \frac{6}{5}(A_{3,1}+b_4)+k+\frac{9}{10}+\frac{6}{5}B_{k+3}$

$\ge \frac{6}{5}\times 1+k+\frac{9}{10}+\frac{6}{5}B_{k+3}$

$\ge k+2+\frac{6}{5}B_{k+3}$

subcase II.5. $A_{2,1} \le \frac{1}{2}$. Check the subcases in Figure 3.9.

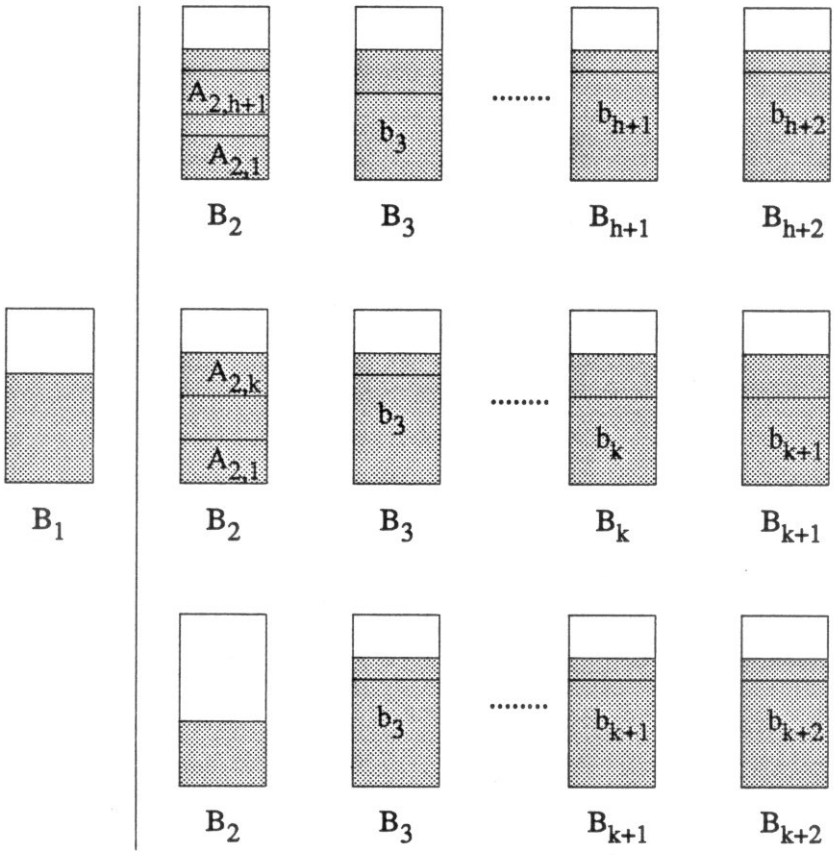


Figure 3. 9 The possible packing when $A_{2,1}$<=1/2.

If $A_{2,1}+\cdots+A_{2,h} \leq \frac{1}{2}$, but $A_{2,1}+\cdots+A_{2,h+1} > \frac{1}{2}$, where $1 \leq h \leq k-2$, then it is easy to prove that $W(B_2) \geq \frac{6}{5}a+\frac{2}{5}$, where a is the smallest item among $A_{2,1},\ldots,A_{2,h+1}$. We know that $A_{2,1}$ and $A_{2,h+1}$ are both non-zero, so there are at least two items in these areas. Since $A_{2,1}+\cdots+A_{2,h+1} > \frac{1}{2}$, then $(A_{2,1}+\cdots+A_{2,h+1})-a > \frac{1}{4}$. If there is at least one item in $A_{2,1},\ldots,A_{2,h+1}$ in $(\frac{1}{3},\frac{1}{2}]$, then $W(B_2) \geq \frac{6}{5}a+\frac{6}{5}((A_{2,1}+\cdots+A_{2,h+1})-a)+\frac{1}{10} > \frac{6}{5}a+\frac{2}{5}$. If all numbers in $A_{2,1},\ldots,A_{2,h+1}$ are less than $\frac{1}{3}$ and there are only two numbers

in $(\frac{1}{6}, \frac{1}{3}]$, then $W(B_2) \geq \frac{6}{5}a + \frac{6}{5}((A_{2,1} + \cdots + A_{2,h+1}) - a) + \frac{3}{5}(A_{2,1} + \cdots + A_{2,h+1}) - \frac{2}{10} > \frac{6}{5}a + \frac{6}{5} \times \frac{1}{4} + \frac{3}{5} \times \frac{1}{2} - \frac{2}{10} = \frac{6}{5}a + \frac{2}{5}$. If $A_{2,1}, \ldots, A_{2,h+1}$ have at least three items in $(\frac{1}{6}, \frac{1}{3}]$, then $W(B_2) \geq \frac{6}{5}a + \frac{6}{5} \times (\frac{1}{6} + \frac{1}{6}) = \frac{6}{5}a + \frac{2}{5}$. So we have proved that $W(B_2) \geq \frac{6}{5}a + \frac{2}{5}$. Therefore,

$W(B_1) + \cdots + W(B_{h+2})$

$\geq \frac{6}{5}B_1 + \frac{6}{5}a + \frac{2}{5} + \frac{6}{5}(b_3 + \cdots + b_{h+1}) + (\frac{2}{5} + \frac{3}{10(k-1)})h + \frac{6}{5}B_{h+2}$

$\geq \frac{6}{5}(B_1 + a) + \frac{2}{5} + \frac{6}{5} \times \frac{1}{2} \times (h-1) + \frac{2}{5}h + \frac{6}{5}B_{h+2}$

$\geq \frac{6}{5} \times 1 + \frac{2}{5} + \frac{3}{5}(h-1) + \frac{2}{5}h + \frac{6}{5}B_{h+2}$

$\geq h + 1 + \frac{6}{5}B_{h+2}$

If $A_{2,1} + \cdots + A_{2,k-1} \leq \frac{1}{2}$, but $A_{2,1} + \cdots + A_{2,k} > \frac{1}{2}$, then it is easy to prove that $W(B_2) \geq \frac{6}{5}A_{2,1} + \frac{1}{10}$. Because if $A_{2,1}$ has at least one item in $(\frac{1}{3}, \frac{1}{2}]$, then the inequality is obvious. If all the items in $A_{2,1}$ are in $(\frac{1}{6}, \frac{1}{3}]$, then there are at most two such items in $A_{2,1}$ since $A_{2,1}$ is less than $\frac{1}{2}$. So $W(B_2) \geq \frac{9}{5}A_{2,1} - \frac{1}{10} \times 2 + \frac{6}{5}(A_{2,2} + \cdots + A_{2,k}) > \frac{6}{5}A_{2,1} - \frac{1}{5} + \frac{3}{5}B_2 > \frac{6}{5}A_{2,1} - \frac{1}{5} + \frac{3}{5} \times \frac{1}{2} = \frac{6}{5}A_{2,1} + \frac{1}{10}$. Therefore,

$W(B_1) + \cdots + W(B_{k+1})$

$\geq \frac{6}{5}B_1 + \frac{6}{5}A_{2,1} + \frac{1}{10} + \frac{6}{5}(b_3 + \cdots + b_k) + (\frac{2}{5} + \frac{3}{10(k-1)})(k-1) + \frac{6}{5}B_{k+1}$

$\geq \frac{6}{5} \times \frac{1}{2} + \frac{6}{5}(A_{2,1} + b_3) + \frac{1}{10} + \frac{6}{5} \times \frac{1}{2} \times (k-3) + \frac{2}{5}(k-1) + \frac{3}{10} + \frac{6}{5}B_{k+1}$

$\geq \frac{3}{5} + \frac{6}{5} \times 1 + \frac{1}{10} + \frac{3}{5}(k-3) + \frac{2}{5}(k-1) + \frac{3}{10} + \frac{6}{5}B_{k+1}$

$\geq k + \frac{6}{5}B_{k+1}$

If $A_{2,1}+\cdots+A_{2,k} \le \frac{1}{2}$, then it is easy to prove that $W(B_2) \ge \frac{6}{5}B_2+\frac{3}{5}A_{2,1}-\frac{1}{5}$. Because if $A_{2,1}$ has at least one item in $(\frac{1}{3},\frac{1}{2}]$, then $W(B_2) \ge \frac{6}{5}B_2+\frac{1}{10} = \frac{6}{5}B_2+\frac{3}{5}\times\frac{1}{2}-\frac{1}{5} \ge \frac{6}{5}B_2+\frac{3}{5}A_{2,1}-\frac{1}{5}$. If all the numbers in $A_{2,1}$ are in $(\frac{1}{6},\frac{1}{3}]$, then $W(B_2) \ge \frac{9}{5}A_{2,1}-\frac{1}{10}\times 2+\frac{6}{5}(A_{2,2}+\cdots+A_{2,k}) = \frac{6}{5}B_2+\frac{3}{5}A_{2,1}-\frac{1}{5}$. Therefore,

$W(B_1)+\cdots+W(B_{k+2})$

$\ge \frac{6}{5}B_1+\frac{6}{5}B_2+\frac{3}{5}A_{2,1}-\frac{1}{5}+\frac{6}{5}(b_3+\cdots+b_{k+1})+(\frac{2}{5}+\frac{3}{10(k-1)})k+\frac{6}{5}B_{k+2}$

$\ge \frac{3}{5}B_1+\frac{3}{5}(B_1+A_{2,1})-\frac{1}{5}+\frac{6}{5}(A_{2,1}+b_3)+\frac{6}{5}\times\frac{1}{2}\times(k-2)+\frac{2}{5}k+\frac{3}{10}+\frac{6}{5}B_{k+2}$

$\ge \frac{3}{5}\times\frac{1}{2}+\frac{3}{5}\times 1-\frac{1}{5}+\frac{6}{5}\times 1+\frac{3}{5}(k-2)+\frac{2}{5}k+\frac{3}{10}+\frac{6}{5}B_{k+2}$

$\ge k+1+\frac{6}{5}B_{k+2}$

So far, we have checked all possible cases. If the packing ends without matching the above cases, then it is not hard to show that we need no more than weight 1. So we can assume the constant c in Claim 3 to be $2+1$, i.e. $c=3$. ◇

Now we are prepared to prove Theorem 3.4.

Proof of Theorem 3.4

$R[NkF] = \limsup \max\{NkF(L)/L^*\}$

$\le \lim_{L^*\to\infty}(W(L)+c)/L^*$ (by Claim 3.3)

$\le \lim_{L^*\to\infty}((1.7+\frac{3}{10(k-1)})L^*+c)/L^*$ (by Claim 3.2)

$= 1.7+\frac{3}{10(k-1)}$ ◇

By Theorem 3.2, Thoerem 3.3 and Theorem 3.4, we have our Theorem 3.1, which solves a longstanding open problem.

4
Best-k-Fit Bin Packing

Best-k-Fit Bin Packing

4.1 Some Observations

At the beginning of Chapter 3, we mentioned some approximation algorithms for bin packing problem. First-Fit(FF) is an algorithm in which we pack each number in the list into the first bin that can hold it; while in First-Fit-Decreasing(FFD), we first sort the list decreasingly and then apply First-Fit to the sorted list. Similarly, Best-Fit(BF) is an algorithm in which we pack each number into the fullest bin that can hold it; while in Best-Fit-Decreasing(BFD), we first sort the list decreasingly and then apply Best-Fit to the sorted list. And finally, Next-Fit(NF) is actually Next-1-Fit, which does not need any explanation. Figure 4.1 shows the FF, FFD, BF, BFD, and NF packings of the list $L = (0.3, 0.8, 0.2, 0.7, 0.5)$.

Also in Chapter 3, we have shown that the worst-case performance bound of Next-k-Fit algorithm for bin packing is $1.7 + \frac{3}{10(k-1)}$. Let us compare this result with the worst-case performance bound of First-Fit. It is not difficult to notice that the fraction part of $R[NkF]$ becomes 0 when k goes to infinity, and therefore $R[NkF]$ becomes 1.7, which is exactly the same as $R[FF]$. This fact is no coincidence since we know that the Next-k-Fit algorithm is actually the restricted First-Fit in the sense that Next-k-Fit only keeps the last, or youngest, k non-empty bins created in the packing instead of all of them as in First-Fit. So Next-k-Fit behaves like First-Fit when k is large.

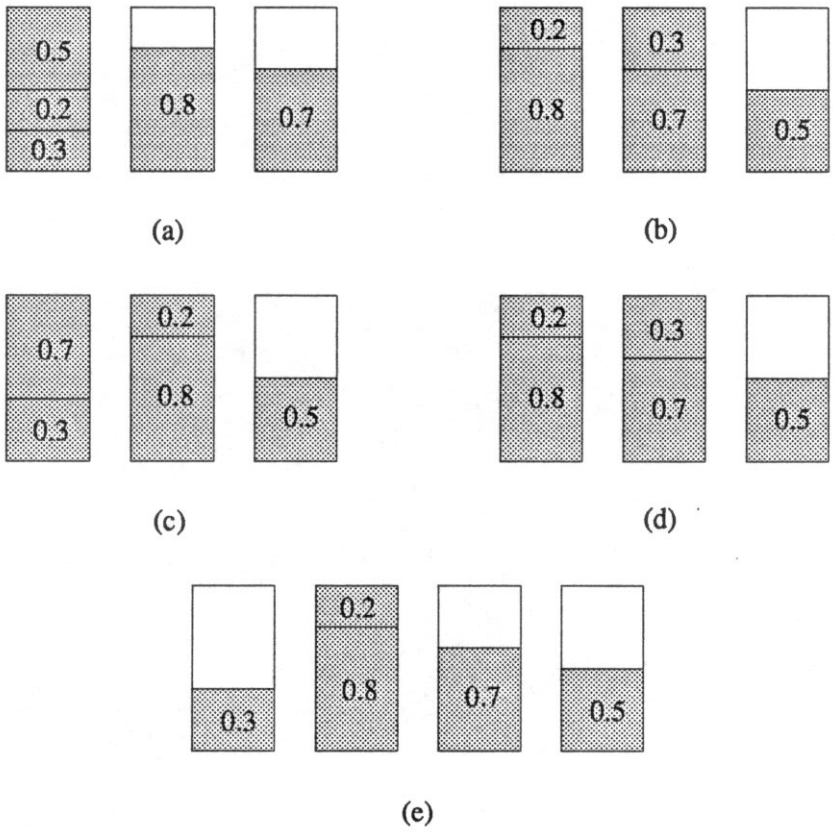


Figure 4. 1 (a) First-Fit ; (b) First-Fit-Decreasing ; (c) Best-Fit ; (d) Best-Fit-Decreasing ; and (e) Next-Fit packings for list L=(0.3, 0.8, 0.2, 0.7, 0.5).

We can also look at this fact from the point of view of complexity. Assume that our list L contains n real numbers. Then FF, FFD, BF, and BFD can all be done in $O(n \log n)$ since we can always keep a sorted sequence of bins created so far according to their current sizes, and when a number comes, a binary search is necessary and sufficient to determine which bin that number will go to. As for NkF, as well as NF, since only k bins are involved in the packing at any time, we only need $O(n \log k)$, which is linear

for constant k. When k is very large, the number of active bins which are involved in the packing will never be greater than n. Thus the time complexity of NkF also converges to that of FF.

In the same way that Next-k-Fit is a restricted form of First-Fit, we can consider a restricted form of Best-Fit, to be called Best-k-Fit, in which we also keep the last k non-empty bins in the packing active and a number will be packed into the fullest bin among these k bins. As proved by D. S. Johnson *et al* [17], First-Fit and Best-Fit happen to have the same worst-case performance bound 1.7. A natural conjecture is that perhaps Next-k-Fit and Best-k-Fit also have the same worst-case performance bound $1.7 + \frac{3}{10(k-1)}$. We will prove, surprisingly, that this turns out not to be the case.

4.2 Best-k-Fit Algorithm

When $k = 1$, Best-1-Fit, also called Best-Fit, is in fact the same as Next-Fit, thus $R[BF] = 2.0$. From now on, we assume that k is an integer greater than 1.

4.2.1 Lower Bound

Surprisingly, when we try to use Best-k-Fit algorithm to pack the long and complicated list in the proof of Theorem 3.3, the lower bound we get turns out to be smaller than $1.7 + \frac{3}{10(k-1)}$. This means either we have to find another list in order to get the $1.7 + \frac{3}{10(k-1)}$ lower bound or the lower bound of $R[BkF]$ is indeed smaller than that of $R[NkF]$. After careful study

of both possibilities, we are inclined toward the latter, i.e., Best-k-Fit might have a better (smaller) performance bound than Next-k-Fit. Interesting!

Theorem 4.1

$$R[BkF] \geq 1.7 + \frac{3}{10k}$$

Proof Let us consider a list similar to the one in the proof of the lower bound for Next-k-Fit in Theorem 3.3. Everything is the same except that last group in the list. In this case, we assume $(k-1)|10n$ and the third group in our list contains the following real numbers, where in each row there are $k+1$ numbers, and h is an integer equal to $\frac{10n}{k-1}$.

$\frac{1}{2} - \frac{\epsilon}{2}, \frac{1}{2} + \epsilon, \frac{1}{2} + \epsilon, \ldots, \frac{1}{2} + \epsilon,$

$\frac{1}{2} - \frac{\epsilon}{4}, \frac{1}{2} + \frac{\epsilon}{2}, \frac{1}{2} + \frac{\epsilon}{2}, \ldots, \frac{1}{2} + \frac{\epsilon}{2},$

.....

$\frac{1}{2} - \frac{\epsilon}{2^h}, \frac{1}{2} + \frac{\epsilon}{2^{h-1}}, \frac{1}{2} + \frac{\epsilon}{2^{h-1}}, \ldots, \frac{1}{2} + \frac{\epsilon}{2^{h-1}}.$

After we apply Best-k-Fit algorithm to this special list, we find that similar to Next-k-Fit, the first group and second group will consume $\frac{10n}{5}$ bins and $\frac{10n}{2}$ bins respectively. As to the third group, each number has to occupy one bin, which requires $h(k+1)$ bins. Thus $BkF(L) = 7n + h(k+1)$.

In the optimal packing, the first two columns in the third group can be packed diagonally, using $h+1$ bins. All the remaining numbers in the list actually form the same list in the proof of $R[FF]$ given by D. S. Johnson *et al* [17], which will use $10n+1$ bins. So altogether, $L^* = 10n + h + 2$.

$$R[BkF] = \limsup \max\{\frac{BkF(L)}{L^*}\}$$

$$\geq \lim_{n\to\infty} \frac{7n + \frac{10n(k+1)}{k-1}}{10n + \frac{10n}{k-1} + 2}$$

$$= 1.7 + \frac{3}{10k}$$

Hence, the largest lower bound of $R[BkF]$ found so far is $1.7 + \frac{3}{10k}$. ◊

4.2.2 Upper Bound

We can employ the same method we used in Chapter 3 to prove the upper bound result for $R[BkF]$. Let the weighting function W be the following.

$$W(\alpha) = \begin{cases} \frac{6}{5}\alpha & \text{if } \alpha \in (0, \frac{1}{6}]; \\ \frac{9}{5}\alpha - \frac{1}{10} & \text{if } \alpha \in (\frac{1}{6}, \frac{1}{3}]; \\ \frac{6}{5}\alpha + \frac{1}{10} & \text{if } \alpha \in (\frac{1}{3}, \frac{1}{2}]; \\ \frac{6}{5}\alpha + \frac{2}{5} + \frac{3}{10k} & \text{if } \alpha \in (\frac{1}{2}, 1]. \end{cases}$$

Claim 4.1

For any bin B of items of total size 1 or less,

$$W(B) \leq 1.7 + \frac{3}{10k}$$

Proof See the proof of Claim 3.1. ◊

Claim 4.2

For any list L,

$$W(L) \leq (1.7 + \frac{3}{10k})L^*$$

Proof See the proof of Claim 3.2. ◊

Claim 4.3

For any list L,

$$BkF(L) \leq W(L) + c$$

Proof The method we will use is exactly the same as in the proof of Claim 3.3 except for a few details in the case analysis.

Given a list L, after applying BkF to L, we get a sequence of non-empty bins $B_1, B_2, \ldots, B_{BkF(L)}$, where $BkF(L)$ is the number of bins generated in BkF packing. As what we did in the proof of Claim 3.3, we can prove this claim by induction on the value of $BkF(L)$.

Inductive basis: It is easy to show that if $BkF(L) \leq k$, then $BkF(L) = BF(L) \leq W(L) + 2$.

Inductive hypothesis: Assume that if $BkF(L) = i - 1$, then $BkF(L) \leq W(L) + 2$.

Inductive step: Assume $BkF(L) \geq i$. By inductive hypothesis, we have $i-1 \leq \sum_{h=1}^{i-1} W(B_h) + 2$. Therefore if $W(B_i) \geq 1$, $i \leq \sum_{h=1}^{i} W(B_h) + 2$. We are done by induction. However if $W(B_i) < 1$, all we need to do is to prove that there exists a j such that $W(B_i) + \cdots + W(B_{i+j-1}) + W(B_{i+j}) \geq j + \frac{6}{5}B_{i+j}$. This inequality means that by borrowing weight $\Delta = W(B_{i+j}) - \frac{6}{5}B_{i+j}$ from

B_{i+j}, bins $B_i, \ldots, B_{i+j-1}$ have weight greater than the number of bins j. So $i+j-1 \leq \sum_{h=1}^{i+j-1} W(B_h) + \Delta + 2$. The unused weight left in B_{i+j} is $\frac{6}{5}B_{i+j}$. So the inductive step can again be applied to B_{i+j}.

The following comes the case analysis, which is almost the same as the one we went through in Claim 3.3. The only difference is caused by the properties of NkF and BkF. In NkF, an item is packed into a certain bin but not any previous bins is because none of the previous active bins can hold the item. However in BkF, it is because of the two reasons: either none of the previous active bins can hold it or there are some previous bins that can hold the item but the current bin it chooses has the fullest content.

For simplicity we assume that $i = 1$, and we wish to prove that if $W(B_1) < 1$, there exists j such that $W(B_1) + \cdots + W(B_j) + W(B_{j+1}) \geq j + \frac{6}{5}B_{j+1}$. We will list all the possible cases and will only do those that are different from those in Claim 3.3.

Case I. $B_1 \leq \frac{1}{2}$.

Case II. $\frac{1}{2} < B_1 < \frac{5}{6}$. We have the following subcases.

subcase II.1. B_2 has one item greater than $\frac{1}{2}$.

subcase II.2. $A_{2,1} > \frac{1}{2}$. The two bottommost items in $A_{2,1}$ are obviously greater than $\frac{1}{6}$. Suppose that at least one of them is in $(\frac{1}{6}, \frac{1}{3}]$. The case analysis of case 2 in Claim 3.3 can be applied to this case.

subcase II.3. $A_{2,1} > \frac{1}{2}$ with its two bottommost items in $(\frac{1}{3}, \frac{1}{2}]$ and $A_{3,1} > \frac{1}{2}$. The old case analysis still holds when the two bottommost items

in $A_{3,1}$ are greater than $\frac{1}{6}$. However right now this might not be the case since when $k = 2$ only the bottommost item in $A_{3,1}$ is guaranteed to be greater than $\frac{1}{6}$. Suppose the other item is in $(0, \frac{1}{6}]$. Then $B_2 > \frac{5}{6}$.

If the bottommost item b_3 of $A_{3,1}$ is in $(\frac{1}{6}, \frac{1}{3}]$, then,

$W(B_1) + W(B_2) + W(B_3)$

$\geq \frac{6}{5}B_1 + \frac{6}{5}B_2 + \frac{1}{10} + \frac{1}{10} + \frac{6}{5}B_3 + \frac{3}{5}b_3 - \frac{1}{10}$

$\geq \frac{3}{5}(B_1 + b_3) + \frac{3}{5}B_1 + \frac{6}{5}B_2 + \frac{1}{10} + \frac{6}{5}B_3$

$\geq \frac{3}{5} \times 1 + \frac{3}{5} \times \frac{1}{2} + \frac{6}{5} \times \frac{5}{6} + \frac{1}{10} + \frac{6}{5}B_3$

$\geq 2 + \frac{6}{5}B_3$

If the bottommost item b_3 of $A_{3,1}$ is in $(\frac{1}{3}, \frac{1}{2}]$, and assume the smaller of the two bottommost items in $A_{2,1}$ to be a, then,

$W(B_1) + W(B_2) + W(B_3)$

$\geq \frac{6}{5}B_1 + \frac{6}{5}B_2 + \frac{1}{10} + \frac{1}{10} + \frac{6}{5}B_3 + \frac{1}{10}$

$\geq \frac{6}{5}(B_1 + a) + \frac{6}{5}(B_2 - a) + \frac{3}{10} + \frac{6}{5}B_3$

$\geq \frac{6}{5} \times 1 + \frac{6}{5} \times \frac{5}{12} + \frac{3}{10} + \frac{6}{5}B_3$

$\geq 2 + \frac{6}{5}B_3$

subcase II.4. $A_{2,1} > \frac{1}{2}$, with its two bottommost items in $(\frac{1}{3}, \frac{1}{2}]$, and $A_{3,1} \leq \frac{1}{2}$. The case analysis is done by checking the following subcases: (i) $A_{3,1} + \cdots + A_{3,h} \leq \frac{1}{2}$, but $A_{3,1} + \cdots + A_{3,h+1} > \frac{1}{2}$, where $1 \leq h \leq k-2$; (ii) $A_{3,1} + \cdots + A_{3,k-1} \leq \frac{1}{2}$, but $A_{3,1} + \cdots + A_{3,k} > \frac{1}{2}$; and (iii) $A_{3,1} + \cdots + A_{3,k} \leq \frac{1}{2}$. Notice for $k = 2$, only (ii) and (iii) can happen. Subcases (i) and (iii) are

exactly the same as those in Claim 3.3. Now let us look at subcase (ii).

If $A_{3,1} + \cdots + A_{3,k-1} \leq \frac{1}{2}$, but $A_{3,1} + \cdots + A_{3,k} > \frac{1}{2}$, then we need to consider the following two possibilities. If $k \geq 3$, then $\frac{3(k-1)}{10k} \geq \frac{1}{5}$. Therefore, the previous case analysis for the proof of Claim 3.3 can be used. On the other hand, let us assume $k = 2$. Because it is Best-k-Fit, then $A_{3,k} + B_4 > 1$.

If the bottommost item b_3 in $A_{3,1}$ is in $(\frac{1}{3}, \frac{1}{2}]$, then,

$W(B_1) + W(B_2) + W(B_3) + W(B_4)$

$\geq \frac{6}{5}B_1 + \frac{6}{5}B_2 + \frac{1}{10} + \frac{1}{10} + \frac{6}{5}B_3 + \frac{1}{10} + \frac{6}{5}B_4 + \frac{2}{5} + \frac{3}{20}$

$\geq \frac{6}{5}(B_1 + A_{3,1}) + \frac{6}{5}B_2 + \frac{3}{10} + \frac{6}{5}(A_{3,k} + B_4) + \frac{2}{5} + \frac{3}{20}$

$\geq \frac{6}{5} \times 1 + \frac{6}{5} \times \frac{2}{3} + \frac{3}{10} + \frac{6}{5} \times 1 + \frac{2}{5} + \frac{3}{20}$

≥ 4

If the bottommost item b_3 in $A_{3,1}$ is in $(\frac{1}{6}, \frac{1}{3}]$, then,

$W(B_1) + W(B_2) + W(B_3) + W(B_4)$

$\geq \frac{6}{5}B_1 + \frac{6}{5}B_2 + \frac{1}{10} + \frac{1}{10} + \frac{6}{5}B_3 + \frac{3}{5}b_3 - \frac{1}{10} + \frac{6}{5}B_4 + \frac{2}{5} + \frac{3}{20}$

$\geq \frac{6}{5}(B_1 + A_{3,1}) + \frac{6}{5}B_2 + \frac{1}{10} + \frac{6}{5}(A_{3,k} + B_4) + \frac{3}{5}(1 - B_2) + \frac{2}{5} + \frac{3}{20}$

$\geq \frac{6}{5} \times 1 + \frac{3}{5}B_2 + \frac{1}{10} + \frac{6}{5} \times 1 + \frac{3}{5} + \frac{2}{5} + \frac{3}{20}$

$\geq \frac{6}{5} + \frac{3}{5} \times \frac{2}{3} + \frac{1}{10} + \frac{6}{5} + \frac{3}{5} + \frac{2}{5} + \frac{3}{20}$

≥ 4

subcase II.5. $A_{2,1} \leq \frac{1}{2}$. Similar to case 4, we need to look at the following subcases: (i) $A_{2,1} + \cdots + A_{2,h} \leq \frac{1}{2}$, but $A_{2,1} + \cdots + A_{2,h+1} > \frac{1}{2}$, where $1 \leq h \leq k-2$; (ii) $A_{2,1} + \cdots + A_{2,k-1} \leq \frac{1}{2}$, but $A_{2,1} + \cdots + A_{2,k} > \frac{1}{2}$; and

(iii) $A_{2,1} + \cdots + A_{2,k} \leq \frac{1}{2}$. Again for $k = 2$, only (ii) and (iii) can happen. It turns out that the case analyses for (i) and (iii) are exactly the same as those in Claim 3.3, so we only have to look at (ii), which eventually makes $R[BkF]$ to be smaller than $R[NkF]$.

If $A_{2,1} + \cdots + A_{2,k-1} \leq \frac{1}{2}$, but $A_{2,1} + \cdots + A_{2,k} > \frac{1}{2}$, then we need to look at the two simple possibilities. Because it is Best-k-Fit, $A_{2,k}$ is put into B_2 because the fuller B_3 cannot hold it, thus $A_{2,k} + B_3 > 1$. If $k \geq 3$, then,

$W(B_1) + \cdots + W(B_{k+1})$

$\geq \frac{6}{5}B_1 + \frac{6}{5}B_2 + \frac{6}{5}B_3 + \frac{6}{5}(b_4 + \cdots + b_k) + (\frac{2}{5} + \frac{3}{10k})(k-1) + \frac{6}{5}B_{k+1}$

$\geq \frac{6}{5}(B_1 + A_{2,1}) + \frac{6}{5}(A_{2,k} + B_3) + \frac{6}{5} \times \frac{1}{2} \times (k-3) + \frac{2}{5}(k-1) + \frac{3(k-1)}{10k} + \frac{6}{5}B_{k+1}$

$\geq \frac{6}{5} \times 1 + \frac{6}{5} \times 1 + \frac{3}{5}(k-3) + \frac{2}{5}(k-1) + \frac{6}{5}B_{k+1}$

$\geq k + \frac{6}{5}B_{k+1}$

On the other hand, let us assume $k = 2$. Let b_2 be the bottommost item in $A_{2,1}$ and a be the smallest item in $A_{2,k}$ which has the property that right before a is packed into B_2, the size of B_2 is no greater than $\frac{1}{2}$. We claim that $W(B_2) > \frac{6}{5}(b_2 + a) + \frac{1}{20}$. If B_2 has at least one item in $(\frac{1}{3}, \frac{1}{2}]$, then $W(B_2) > \frac{6}{5}(b_2 + a) + \frac{1}{10} > \frac{6}{5}(b_2 + a) + \frac{1}{20}$. If $A_{2,1}$ has at least two items, then $W(B_2) > \frac{6}{5}(b_2 + a) + \frac{6}{5} \times \frac{1}{6} > \frac{6}{5}(b_2 + a) + \frac{1}{20}$. Now we assume that all items in B_2 are less than $\frac{1}{3}$ and $A_{2,1}$ only has b_2. If there is no item between b_2 and a, and $b_2 + a > \frac{1}{2}$, then at least one of them is greater than $\frac{1}{4}$. Therefore $W(B_2) > \frac{6}{5}(b_2 + a) + \frac{3}{5} \times \frac{1}{4} - \frac{1}{10} > \frac{6}{5}(b_2 + a) + \frac{1}{20}$. If $A_{2,k}$ has at least two items,

then $A_{2,k} - a$ must be greater than $\frac{1}{12}$ otherwise $B_2 = b_2 + (A_{2,k} - a) + a < \frac{1}{3} + \frac{1}{12} + \frac{1}{12} = \frac{1}{2}$. Therefore $W(B_2) > \frac{6}{5}(b_2 + a) + \frac{6}{5}(A_{2,k} - a) > \frac{6}{5}(b_2 + a) + \frac{1}{20}$. So,

$W(B_1) + W(B_2) + W(B_3)$

$\geq \frac{6}{5}B_1 + \frac{6}{5}(b_2 + a) + \frac{1}{20} + \frac{6}{5}B_3 + \frac{2}{5} + \frac{3}{20}$

$\geq \frac{6}{5}(B_1 + b_2) + \frac{6}{5}(a + B_3) + \frac{1}{20} + \frac{2}{5} + \frac{3}{20}$

$\geq \frac{6}{5} \times 1 + \frac{6}{5} \times 1 + \frac{3}{5}$

≥ 3

So far, we have checked all the possible cases. Therefore we can say that for any list L, $BkF(L) \leq W(L) + c$, where c is a constant less than 3, as proved in Claim 3.3. ◊

Theorem 4.2

$$R[BkF] \leq 1.7 + \frac{3}{10k}$$

Proof The proof is straightforward using Claim 4.1, Claim 4.2, and Claim 4.3. ◊

Combining Theorem 4.1 and Theorem 4.2, we have the tight worst-case performance bound of Best-k-Fit bin packing.

Theorem 4.3

$$R[BkF] = 1.7 + \frac{3}{10k}, \quad k \geq 1$$

Why does Best-k-Fit algorithm have a better (smaller) worst-case performance bound than Next-k-Fit? To get an intuitive idea, let us look at the

following example.

Example

Let the weighting function used in proving the upper bound of $R[NkF]$ be W_1, which results in the $1.7 + \frac{3}{10(k-1)}$ bound, and let the weighting function used in the Best-k-Fit proof be W_2, which results in the $1.7 + \frac{3}{10k}$ bound. After checking the case analysis of Next-k-Fit, we find that the only case that prevents us from getting the $1.7 + \frac{3}{10k}$ is subcase II.5(ii). To understand this point, consider the list given in the proof of Theorem 3.3.

The list L consists of three groups L_1, L_2 and L_3. After we use NkF to pack the list, we get $NkF(L)$ bins as discussed in Theorem 3.3. The case analysis can be used to classify different sections of the packing. Subcase II.5(ii) does happen in the third list L_3. If weighting function W_1 is used, $W_1(L) = W_1(L_1) + W_1(L_2) + W_1(L_3) = 2n + c_1\epsilon + 5n + c_2\epsilon + \frac{10nk}{k-2} + c_3\epsilon \geq NkF(L)$, which means that list L satisfies Claim 3.3 and the $1.7 + \frac{3}{10(k-1)}$ bound can be verified.

However, if we use W_2 to calculate the weight of the list, $W_2(L) = W_2(L_1) + W_2(L_2) + W_2(L_3) = 2n + c_1\epsilon + 5n + c_2\epsilon + \frac{10nk}{k-2} - \frac{3n}{k(k-2)} + c_3\epsilon = NkF(L) - \frac{3n}{k(k-2)} + (c_1 + c_2 + c_3)\epsilon$, which can never satisfies Claim 3.3 for the unbounded n. This is why the $1.7 + \frac{3}{10k}$ bound is not good for NkF. On the other hand, BkF can prevent this because the small items $\epsilon, \frac{\epsilon}{2}, \ldots, \frac{\epsilon}{2^{k-1}}$ in sublist L_3 go to the fuller bins with size greater than $\frac{1}{2}$ instead of those with

size less than $\frac{1}{2}$, which saves space for the upcoming items less than $\frac{1}{2}$. $\diamondsuit$

5
Conclusion

Conclusion

5.1 Summary

The theory of NP-completeness has provided possibilities to study different kinds of difficult problems. The worst-case performance bound study has become an efficient method to analyse the quality of approximation algorithms for NP-complete problems.

The research in this thesis was first inspired by the study of the Next-2-Fit algorithm for bin packing problem three years ago. The complete solution of the problem gives a promising start for the Next-k-Fit algorithm study, which has remained open since the early seventies. Best-k-Fit algorithm is a natural extension of Best-Fit, as Next-k-Fit is of First-Fit. The following table shows what we have achieved on this problem.

k	R[NkF]		R[BkF]	
	Old results	New results	Old results	New results
1	2	2	2	2
2	[1.85, 2]	2	[1.85, 2]	1.85
3	[1.8, 2]	1.85	[1.8, 2]	1.8
....	.....	.....	.....	.....
k	[1.7+3/(10k), 2]	1.7+3/(10(k-1))	[1.7+3/(10k), 2]	1.7+3/(10k)
....	.....	.....	.....	.....

Table 5. 1 The comparison of new results with old results for NkF and BkF.

One thing I should mention is that at the time I just finished writing this thesis, I found that the results $R[N2F] = 2$ and $R[NkF] \geq 1.7 + \frac{3}{10(k-1)}$ were proved by J. Csirik and B. Imreh [9] in 1989. As part of my thesis which shows the continuous work I did from 1986 to 1990, I still present my version of these results in this thesis. However I do think that Csirik and Imreh should get the credit for the work.

We have also studied scheduling problems, which are closely related to bin packing. We have focused on the relation between preemptive and non-preemptive schedulings, and the results we obtain are summarized in the following table.

ω / ω'	
Old results	New results
[2m/(m+1), (2m-1)/m]	[2m/(m+1), (2m-1)/m-α/m] , where $\alpha=\phi'/\omega'$ if $(m^2+1)\omega'/(m+1)<X<(m+1)\omega/2$ 2m/(m+1) otherwise

Table 5. 2 The comparison of new results with old results of ratio ω/ω'.

5.2 Future Work

The field of scheduling and bin packing problems is rich and colorful for researchers. There are many versions of the problems that are unknown to be NP-complete or not. For those we already know their classification in NP theory, there are still many algorithms that need to be designed and

analysed. For the problem we studied in Chapter 2, it is very possible that the bound can be improved to the one conjectured by C. L. Liu.

References

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

[2] S. Baase, *Computer Algorithms: Introduction to Design and Analysis*, 2nd edition, Addison-Wesley Publishing Company, 1988.

[3] K. R. Baker, E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, *Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints*, Oper. Res. 31(1983), pp.381–386.

[4] W. Clark, *The Gantt Chart*, 3rd edition, Pitman and Son, London, 1952.

[5] E. G. Coffman, Jr. (editor), *Computer and Job-Shop Scheduling Theory*, John Wiley & Sons, New York, London, Sydney, Toronto, 1976.

[6] E. G. Coffman, Jr., M. R. Garey and D. S. Johnson, *Approximation algorithms for bin packing—An updated survey*, in Algorithm Design for Computer System Design, G. Ausiello, M. Lucertini, and P. Serafini, eds., Springer–Verlag, Berlin, New York, 1984, pp.49–106.

[7] E. G. Coffman, Jr. and R. L. Graham, *Optimal scheduling for two processor systems*, Acta Informat. 1(1972), pp.200–213.

[8] S. A. Cook, *The complexity of theorem proving procedures*, Proceedings, 3rd ACM Symposium on Theory of Computing, 1971, pp.151–158.

[9] J. Csirik and B. Imreh, *On the worst-case performance of the NkF bin-*

packing heuristic, Acta Cybernetica 9(1989), pp.89–105.

[10] M. R. Garey, R. L. Graham, D. S. Johnson and A. C. Yao, *Resource constrained scheduling as generalized bin packing*, J. Comb. Theory, 21 (1976), pp.257–298.

[11] M. R. Garey and D. S. Johnson, *Approximation algorithms for combinatorial problems: An annotated bibliography*, in Algorithms and Complexity: New Directions and Recent Results, J. F. Traub, eds., Academic Press, Inc., New York, San Francisco, London, 1976, pp.41–51.

[12] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[13] T. Gonzalez and D. B. Johnson, *A new algorithm for preemptive scheduling of trees*, J. ACM 27(1980), pp.287–312.

[14] R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell System Tech. J. 45(1966), pp.1563–1581.

[15] D. S. Johnson, *Near-optimal bin packing algorithms*, Ph.D. thesis, Tech. Report Mac TR-109(1973), MIT.

[16] D. S. Johnson, *Fast algorithms for bin packing*, J. of Comput. System Sci., 8 (1974), pp.274–314.

[17] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey and R. L. Graham, *Worst-case performance bounds for simple one-dimensional packing algorithms*, SIAM J. Comput., 3 (1974), pp.229–325.

[18] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds.,

Plenum Press, New York, 1972, pp.85–103.

[19] B. J. Lageweg, E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, *Computer aided complexity classification of deterministic scheduling problems*, Report BW 138, Centre for Mathematics and Computer Science, Amsterdam, 1981.

[20] B. J. Lageweg, J. K. Lenstra, E. L. Lawler and A. H. G. Rinnooy Kan, *Computer-aided complexity classification of combinatorial problems*, Comm. ACM, 25(1982), pp.817–822.

[21] E. L. Lawler, *Optimal sequencing of a single machine subject to precedence constraints*, Management Sci. 19(1973), pp.544–546.

[22] E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, *Recent developments in deterministic sequencing and scheduling: A survey*, in Deterministic and Stochastic Scheduling, M. A. H. Dempster, J. K. Lenstra, and A. H. G. Rinnooy Kan, eds., D. Reidel Publishing Company, 1982, pp.35–73.

[23] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys, *Sequencing and scheduling: Algorithms and complexity*, will appear in the Handbooks in Operation Research and Management Science, Vol. 4: Logistic of Production and Inventory, S. C. Graves, A. H. G. Rinnooy, and P. Zipkin, eds., North-Holland.

[24] C. L. Liu, *Optimal scheduling on multi-processor computing systems*, in Proceeding of the 13th Annual Symposium on Switching and Automata Theory, 1972, pp.155–160.

[25] R. R. Muntz, *Scheduling of computations on multiprocessor systems: The preemptive assignment discipline*, Ph.D. thesis, Electrical Engineering Department, Princeton University, April 1969.

[26] R. R. Muntz and E. G. Coffman, Jr., *Preemptive scheduling of real time tasks on multiprocessor systems*, J. ACM 17(1969), pp.324–338.

[27] J. D. Ullman, *NP-complete scheduling problems*, J. of Comput. System Sci. 10(1975), pp.384–393.

[28] A. C. Yao, *New algorithms for bin packing*, J. Assoc. Comput. Mach., 27 (1980), pp.207–227.

[29] A. C. Yao, *Note:* $R[NkF] \leq \frac{1.7k}{k-2}$, personal communication (1987).