國 立 交 通 大 學

資訊管理研究所

碩　士　論　文

線上遊戲之戰術規劃
Mission Planning in Computer Games

研究生：方癸棠

指導教授：林妙聰 博士

中華民國九十七年六月

線上遊戲之戰術規劃

Mission Planning in Computer Games

研　究　生：方癸棠　　　　Student: Kuei-Tang Fang

指導教授：林妙聰　　　　Advisor: B.M.T. Lin

國立交通大學

資訊管理研究所

碩士論文

A Thesis

Submitted to Institute of Information Management

College of Management

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Business Administration

in

Information Management

June 2008

Hsinchu, Taiwan, the Republic of China

中華民國九十七年六月

誌謝

兩年的研究所生涯，終於告一段落。回首剛回到校園進修時的期待，以及現在收成的喜悅，這一切都要感謝許多人對我的提攜與幫助。

這一篇論文能夠順利完成，首先要感謝我的指導老師—林妙聰教授。在老師辛勞叮嚀論文的進度之下，論文才能夠如期的完成，此外，這段期間內林教授也指導我們投稿方面的問題，讓學生的論文可以參加國外研討會，對外發表，此外也協助我解決許多論文的瓶頸，使得論文可以付梓，這都需感謝老師對於我的指導跟協助。

而我的同學們在這段時間給予了很多的協助，尤其是在我工作繁忙時，幫我解決了很多事務上的問題，讓我可以專注在論文的題目上。我要特別感謝田亞梅學姊、黃筱嵐學姐、盧彥廷學長、劉昱劭同學，由於大家的協助才得以順利完成兩年的學業。

另外要再次特別感謝黃筱嵐學姊，讓我在電子期刊以及文獻探討部份得到許多的資訊，沒有他們的協助，就無法完成這一篇論文，真的非常感謝他們對我的協助。

另外還有很多曾經幫助過我的朋友，因為有大家的幫助，我才能有今天的成果。』

Title of Thesis: Mission Planning in Computer Games

Name of Institutes: Institute of Information Management

Student Name: Kuei-Tang Fang Advisor Name: Professor B.M.T. Lin

Abstract

As the population of the Internet users increases explosively, the market value of the virtual world becomes remarkable. In this thesis, we propose a mission planning problem arising from computer games to optimize the time required to reach a target experience level. The problem is formulated as a resource-constrained scheduling problem in which each job (task) has a processing time, requires an amount of resources for its processing and returns another amount of resources. We give a formal mathematical formulation through a binary integer program. A pseudo-polynomial time dynamic programming algorithm is developed to produce optimal schedules. A result of non-approximability about constant performance ratios is established and a 2-approximate greedy algorithm is given for a special case. We also proposed solution algorithms for two special cases: (1) jobs have the same processing time, and (2) jobs belong to K different types.

Keywords: Computer games, relocation problem, resource-constrained scheduling, approximability

摘要

隨著網際網路使用者數量的快速累積，網路這個虛擬世界對現實世界的影響力隨之變得具體而引人注目。其中，線上遊戲這個產業在近幾年爆發性的市場成長更讓我們看到未來無限的潛力，在這麼多種的線上遊戲裡面又以角色扮演的線上遊戲佔最大的比例，同時也是眾多遊戲廠商的主要收入來源。然而，對於如何有效率的排程打怪練功過程，以提高經驗值的研究卻顯得稀少。因此，我們在這篇論文中，提出一線上遊戲的戰術規劃問題。這個問題主要建立在一個在角色扮演的線上遊戲，遊戲角色可以透過打怪、完成任務等等方式獲取經驗值，而我們的目標就是在最少的時間下，排程打怪練功的過程，以達到特定目標經驗值。在這篇論文裡，我們先把這個題目描述成一個資源限制下的排程問題，並設計一個動態規劃演算法來求出最佳解，分析題目近似解，以及在幾個特殊情況下的求解方法。

關鍵字：電腦遊戲、 重置問題、資源限制的排程、近似解

Table of Contents

Mission Planning in Computer Games

Kwei-Tang Fang
A Master Thesis Submitted to
the Institute of Information Management
National Chiao Tung University
Hsinchu 300, Taiwan

Abstract: As the population of the Internet users increases explosively, the market value of the virtual world becomes remarkable. In this thesis, we propose a mission planning problem arising from computer games to optimize the time required to reach a specified experience level. The problem is formulated as a resource-constrained scheduling problem in which each job (task) has a processing time, requires an amount of resources for its processing and returns another amount of resources at completion. We give a formal mathematical formulation through a binary integer program. A pseudo-polynomial time dynamic programming algorithm is developed to produce optimal schedules. A result of non-approximability about constant performance ratios is established and a 2-approximate greedy algorithm is given for a special case. We also proposed solution algorithms for two special cases: (1) jobs have the same processing time, and (2) jobs belong to K different types.

Keywords: Computer games, relocation problem, resource-constrained scheduling, approximability

Chapter 1

Introduction

Online games started in the 1980s with simple multi-player text-based games (MUD). It was usually played on a BBS with connections via modems. These games were based on fantasy settings, using rules similar to those in the role-playing game Dungeons & Dragons. Nowadays MUD becomes an abbreviation of Multi-User dungeon, Multi-User dungeon dimension or Multi-User Dialogue. It represents a text-based virtual reality that exists on the Internet and is available for participation of multiple users.

A real virtual world that sounds like a confliction is a goal that all online games endeavor to achieve. Because online games are a virtual society composed of many game players, every role on the monitor is controlled by a real person, and we can see kinds of players with different characteristics reflecting their personal preferences and behaviors.

In the whole Asia/Pacific area, except Japan, South Korea, China, Taiwan is the biggest online game market (see Figure 1) (IDC1, 2006). Based on a report from IDC Asia/Pacific Research, the online game market in Taiwan is US$289.8 million in 2007, a 15.2% increase to 2006 (see Figure 2) (IDC1, 2008). According to a new report from analyst firm DFC Intelligence, the increase in broadband households, higher PC penetration and more connected console video game systems. The worldwide online game market is forecasted to grow from

$4.5 billion in 2006 to over $13 billion in 2012, a 192% increase (see Figure 3) (DFC, 2007). The largest online game companies such as Blizzard Entertainment, NCsoft, Sony Online Entertainment and Shanda Entertainment, now generate hundreds of millions of dollars in annual revenue from high profile massively multiplayer online games (MMOGs).

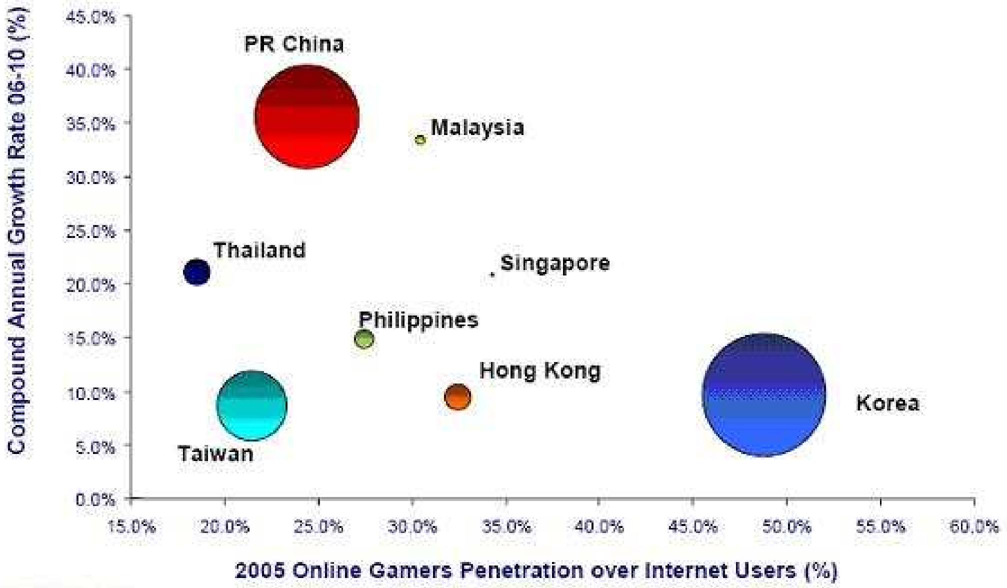

Figure 1

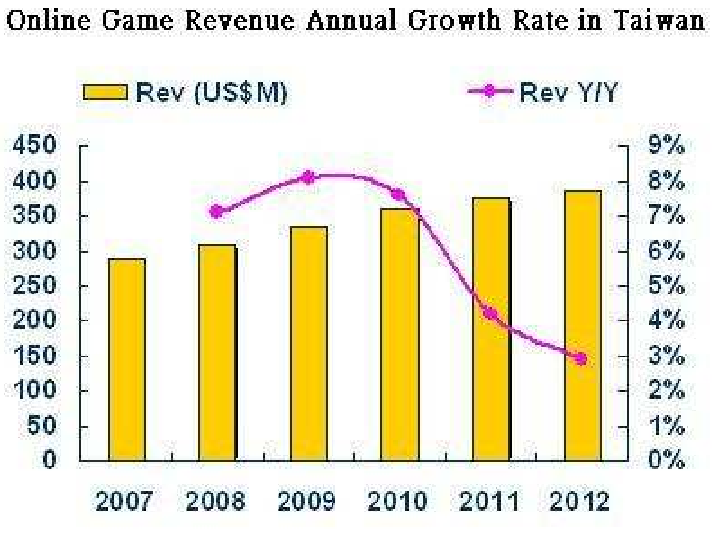

Figure 2

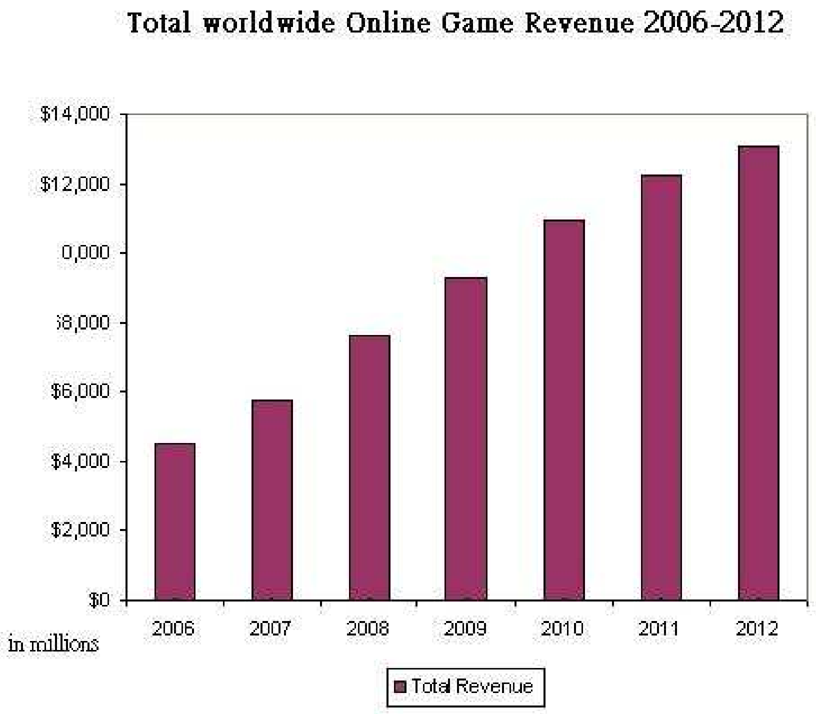

Figure 3

While computer games attract players from around the world and the industry is growing, we are interested in decision problems that can be observed from this area. In this thesis, we investigate a planning problem that could arise from online RPG games. In such games, players fight against the monsters or mobs they encounter so as to achieve the task of missions. In the process, players gain experiences if they successfully defeat the mobs. When a player accumulate his/her experience up to a certain level, he/she can acquire special weapons or upgrade to a higher level. Different types of mobs may come into scenes. The decision problem of the payer is to determine, subject to a personal experience level, a fighting plan to accumulate experience to so as meet a specific experience level in the shortest time. This thesis studies this decision problem from the aspect of resource-constrained scheduling.

The scheduling model we adopt is the relocation problem, which was first formulated by Kaplan (1986) from a public housing development project in Boston. We first introduce the generic relocation problem. There are n houses to redevelop. The town governor offers initial v_0 housing units for temporarily house the tenants of the buildings under development. Each building J_i has two parameters: α_i, the original capacity and β_i, new capacity after redevelopment. Building J_i is allowed to execute only if there are at least α_i housing units available. After building J_i is completed, it will return β_i units for common use. The goal is to find a feasible construction sequence subject to a fixed amount of initial resources. Mapping the relocation problem to the mission planning problem in the role-playing games, each player has initial experience v_0 and intend to accumulate sufficient experience through defeating monsters and completing missions as earlier as possible. Like the relocation problem, there are three parameters associated with a mission or monster: α_i, β_i, and p_i. Job J_i means the mission of fighting against a monster or complete a mission. Different mobs will require different experiences to fight. That is, if the player's experience is less than a mob's power α_i, then the player will be defeated and perish. On the other hand, if the player has sufficient experience, then his/her experience level will drop by α_i for the fight. After successfully defeating a mob, the player will pickup experiences β_i, which depends on the type of mob defeated. Given an initial experience level, the mission planning task of the player is to construct a sequence of mobs to fight with so as to accumulate experiences to a specific level in the shortest time.

The thesis is organized in five chapters. Chapter 2 provides formal statements of the problem and preliminary properties of the relocation problem. In Chapter 3, we will propose a binary integer program and design a pseudo-polynomial time dynamic program. Chapter 4 investigates the approximability of the mission planning problem. Chapter 5 is devoted to solution algorithms for two special cases: jobs have the same processing time and the scenario where exactly K types of jobs. In Chapter 6, we will give concluding remarks and suggest further research directions.

Chapter 2

Problem Statements and Preliminary Properties

In this chapter, we give a formal definition of the mission planning problem from the aspect of resource-constrained scheduling. An integer programming model will be given to describe the problem. Preliminary properties about the relocation problem follow.

In the mission planning problem, initial resource level v_0 of a common pool is given for processing a set of jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ on a single-machine. Each job J_j acquires and immediately consumes α_j units of resource and has a processing time p_i. A target resource level $\bar{v}$ is specified. At any time the machine can process at most one job, and no preemption is allowed. When job J_j is completed, it immediately returns β_i units to the common resource pool. It is assumed $\alpha_j \leq \beta_j$ for all $J_j \in \mathcal{J}$. We denote $\delta_j = \beta_j - \alpha_j$ and let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_{n'}), 0 \leq n' \leq n$ be a particular schedule. Schedule σ is *feasible* if at any moment the resource level can satisfy the resource requirement of each job in the schedule and the target resource level is met. The feasibility issues can be described as:

$$v_{[k]} = v_0 + \sum_{l=1}^{k-1} \delta_{\sigma_l} \geq \alpha_k, 1 \leq k \leq n' \text{ and} \tag{2.1}$$

$$\sum_{l=1}^{n'} \delta_{\sigma_l} \geq \bar{v}, \tag{2.2}$$

where $v_{[k]}$ is the resource level at the exact moment of the completion of the job in position k. The mission planning problem seeks to determine a feasible schedule such that the total processing time of the schedule is minimum, i.e. find a feasible schedule $\sigma^* = (\sigma_1^*, \sigma_2^*, \ldots, \sigma_{n'}^*)$ such that $\sum_{l=1}^{n'} p_{\sigma_l^*}$ is minimum among all feasible schedules.

In the basic relocation problem, constraint (2.2) is not required, i.e. given an initial resource level v_0 the decision is on the existence of a sequence of *all* jobs satisfying constraint (2.1). The optimization counterpart of the relocation problem becomes determining the minimum initial resource level that will guarantee the existence of a feasible sequence. In the following, we review some important properties of the basic relocation problem. Kaplan and Amir (1988) have shown that the relocation problem is equivalent to the classical two-machine flowshop scheduling (Johnson's) problem of makespan minimization. The equivalence is settled as follows. For each job J_j of the relocation problem, we create a corresponding job of the flowshop scheduling by letting α_j and β_j be the processing times required by machine-one and machine-two operations, respectively. For a given sequence of indices, the minimum amount of resources required to guarantee the feasibility of the job sequence in the relocation problem is equivalent to the sum of idle times on machine two in Johnson's problem. Therefore, the following lemma gives the polynomial solvability of the relocation problem.

Lemma 1 *(Kaplan and Amir 1988) To find the minimum resource provision for guaranteeing the existence of a feasible sequence in the relocation problem can be solved in $O(n \log n)$ time using Johnson's algorithm.*

As given in Cheng and Lin (2008), Johnson's algorithm can be described in the following forms:

1. Job J_i precedes job J_j if $\min\{\alpha_i, \beta_j\} \leq \min\{\beta_i, \alpha_j\}$.
2. Partition job set $\mathcal{J}$ by letting $\mathcal{J}^+ = \{J_i | \alpha_i \leq \beta_i, J_i \in \mathcal{J}\}$ and $\mathcal{J}^- = \{J_i | \alpha_i > \beta_i, J_i \in \mathcal{J}\}$. Schedule the jobs of $\mathcal{J}^+$ in non-decreasing order of α_i, and the jobs of $\mathcal{J}^-$ in non-increasing order of β_i. The sequence of $\mathcal{J}^+$ precedes the sequence of $\mathcal{J}^-$.
3. Select the smallest element from $\{\alpha_i\} \cup \{\beta_i\}$ of all unscheduled jobs J_i. If the element is α_i of some job J_i, then schedule job $J_i$at the earliest open position, else schedule job J_i at the last open position. Remove job J_i from the job set. Repeat the process until all jobs are scheduled.

Let $\sigma_{[i:j]} = (\sigma_i, \dots, \sigma_j), 1 \leq i \leq j \leq n$ be a subsequence of $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_n)$. We define the resource requirement $R(\sigma_{[i:j]})$ as the minimum resource provision to guarantee the successful processing of jobs $\sigma_i, \sigma_{i+1}, \dots, \sigma_j$ in the specified sequence. Kurisu (1976) introduced the notion of *composite jobs,* which has spurred many interesting applications (Cheng and Lin 2008). Composite job $J_{i:j}(\sigma)$ of subsequence $\sigma_{[i:j]}$ is defined by letting

$$\alpha_{i:j}(\sigma) = R(\sigma_{[i:j]}), \text{ and}$$

$$\beta_{i:j}(\sigma) = R(\sigma_{[i:j]}) + \sum_{l=1}^{j} (\beta_l - \alpha_l).$$

Lemma 2 *(Kurisu 1976, Lin 1994, Cheng and Lin 2008) For a particular sequence σ, the following equality about resource requirement holds:*

$$R(\sigma) = R((\sigma_1, \dots, \sigma_{i-1}, \sigma_{[i:j]}, \sigma_{j+1} \dots, \sigma_n)).$$

For all $i, j, 1 \leq i \leq j \leq n$, composite jobs $\sigma_{[i:j]}$ can be defined in a recursion. To define composite jobs $\sigma_{[i:j]}$, we need (composite) jobs composite jobs $\sigma_{[i:j-1]}$ and σ_j. The whole procedure is given as follows:

```
For $i = 1$ to $n$ do
  For $j = i + 1$ to $n$ do
      Define composite job $\sigma_{[1:j]}$ using $\sigma_{[1:j-1]}$ and $\sigma_j$.
```

Therefore, when input instance is given all composite jobs can be derived in $O(n^2)$ time. In the following section, we will apply the concept of composite jobs in the design of solution algorithms. This concept can reduce the time complexity by an order.

Chapter 3

Integer Program and Dynamic Program

In this chapter, we start to investigate the mission planning problem. We consider the case where $\alpha_j = 0$ for all jobs J_j. In comparison with the Knapsack problem, β_i and p_i will correspond to the value and weight of item x_i in the Knapsack problem. The special case of our problem can be interpreted as the *minimization* Knapsack problem: Minimizing the capacity required to achieving a specific target value. Note that the Knapsack problem is known for maximizing the total value subject to the knapsack capacity, however the minimization counterpart is to *minimize the total weight required to achieve a threshold on total value*. Therefore, the mission planning problem is also NP-hard. The major difference of the mission planning problem from the Minimization Knapsack problem is the feasibility requirement in the sequencing of selected jobs. This extra requirement may pose a higher complexity of the problem. By the preliminary properties cited in last chapter, we can give the optimality property to resolve part of the difficulties.

Lemma 3 *In an optimal solution to the mission planning problem, the jobs are sequenced* *in non-decreasing order of resource requirement* α_j*.*

Proof: Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_{n'})$ be an optimal schedule. If there are any two consecutive jobs σ_j and σ_{j+1} satisfying $\alpha_{\sigma_j} > \alpha_{\sigma_{j+1}}$, then we swap their positions. Feasibility of all jobs in $\{\sigma\} \setminus \{J_{\sigma_j}, J_{\sigma_{j+1}}\}$ will not be affected. Because $v_{[j-1]} \geq \alpha_{\sigma_j} > \alpha_{\sigma_{j+1}}$, job $J_{\sigma_{j+1}}$ can be processed with sufficient resources. Moreover, job $J_{\sigma j+1}$ adds extra $\delta_{\sigma j+1}$ units of resources to $v_{[j-1]}$ for job J_{σ_j}, and thus the processing of job J_{σ_j} is also successful. Therefore, the correctness follows. □

By Lemma 3, we can thus re-index the jobs in non-decreasing order of resource requirements. The mission planning problem can be considered as selection of jobs out from a list to discard such that the jobs remained in the list can be successfully process and the resource level at completion of the list can meet the target level. We can thus formulate the mission planning problem into a binary integer program. Binary variable $x_j, 1 \leq j \leq n$ indicates whether job J_j is included for processing or not.

$$\text{(MP)} \quad \text{Minimize} \quad \sum_{j=1}^{n} p_j x_j$$

subject to

$$\alpha_j x_j \leq v_0 + \sum_{i=1}^{j-1} \delta_i x_i, \qquad 1 \leq j \leq n; \tag{3.1}$$

$$v_0 + \sum_{j=1}^{n} \delta_j x_j \geq \bar{v}; \tag{3.2}$$

$$x_j \in \{0, 1\}, \qquad 1 \leq j \leq n. \tag{3.3}$$

In the above formulation, constraints (3.1) specify the feasibility issue of constraint (2.1), and constraints (3.2) reflect the target requirement of constraint (2.2).

We proceed to the development of a dynamic program for producing optimal solutions. From the preliminary observations mentioned above, the mission planning problem reduces to selecting a subset of jobs out from the ordered list of jobs to meet the target requirement. Let function $f(j, v)$ denote the shortest processing time required by a feasible subset of jobs from $\{J_1, J_2, \dots, J_i\}$ to meet the resource level v. Similar to the dynamic programs developed in the literature for the Knapsack problem, subject to optimal solutions of the previous $j-1$ jobs with various target levels we can include job J_j or discard it when it is considered. The only difference is that we need to abide by the feasibility constraint. The dynamic program is given in the following:

Initialization:

$$f(j, v) = \begin{cases} 0, & \text{if } j = v = 0; \\ \infty, & \text{otherwise.} \end{cases}$$

Recursion:

$$f(j, v) = \min\Big\{ f(j-1, v), f'(j-1, v-\delta_i) \Big\}, \text{ where}$$

$$f'(j-1, v-\delta_j) = \begin{cases} f(j-1, v-\delta_j) + p_j, & \text{if } v - \delta_j \geq \alpha_j; \\ \infty, & \text{otherwise.} \end{cases}$$

Goal:

Find $\min\Big\{ f(n, v) | \bar{v} \leq v \leq v_0 + \sum_{j=1}^{n} \delta_j) \Big\}$.

In the dynamic programming algorithm, there are $O(n\bar{v})$ states. A constant time step is sufficient to compute the optimal value of each state, Therefore, the overall time complexity $O(n\bar{v})$ is pseudo-polynomial in terms of the input length and the parameter $\bar{v}$. The existence of a pseudo-polynomial algorithm in the mean time indicates that the mission planning problem cannot be strongly NP-hard.

While the dynamic programming algorithm can optimally solve the mission planning problem, its run time can be prohibitively long if the values of parameter $\bar{v}$ is large. Development of approximation algorithms is an alternative for solving the problem. In the next chapter, we shall discuss the room for the development of approximation algorithms.

Chapter 4

Approximability

In this chapter, we study the approximability of the mission planning problem. In the study of theoretical approximation algorithms, the Knapsack problem provides a good source of classical results. Let Z^* denote the optimal solution value of problem $\mathcal{P}$ and $\mathcal{A}$ be an approximation algorithm with solution value $Z_{\mathcal{A}}$ for an NP-hard optimization problem. Algorithm $\mathcal{A}$ is an r-approximate algorithm if applied on any instance $\mathcal{I}$ of problem $\mathcal{P}$,

$$\max\{\frac{Z^*}{Z_{\mathcal{A}}}, \frac{Z_{\mathcal{A}}}{Z^*}\} \leq r$$

is guaranteed for constant rational $r > 0$. A *polynomial time approximation scheme* (PTAS) accepts any instance $\mathcal{I}$ and rational r can produce an approximation solution such that the ratio r is guaranteed with a computing time polynomial in terms of length of instance $\mathcal{I}$, but not of $(r-1)^{-1}$. If a PTAS has a computing time polynomial in terms of length of instance $\mathcal{I}$ and $(r-1)^{-1}$, then it is called a *fully polynomial time approximation scheme* (FPTAS). In the literature, a well-known modified greedy algorithm is shown to be a 2-approximate algorithm for the Knapsack problem (Kellerer et al. 2004). Due to the structure of pseudo-polynomial dynamic programs (Woeginger 2000), FPTAS exists for the Knapsack problem (Kellerer et al. 2004). PTAS also exists even if the Knapsack problem is generalized to allow acceptance of multiple copies of each individual item (Kellerer et al. 2004). While the Knapsack problems permits approximation algorithms with desired performance ratio, it is not that promising for us to design such approximation methods for the mission planning problem. The difficulty arises from the requirement of sequencing feasibility. In the following, we give a negative result of approximability.

Theorem 1 *Unless P=NP, there is no r-approximate algorithm for the mission planning algorithm for constant rational $r > 1$.*

Proof: To prove the result, we use the Equal-Size-Partition problem (Garey and Johnson 1979) for the reduction.

Equal-Size-Partition: The input instance consists of integer B and $2t$ positive integers $S = \{s_1, ..., s_{2t}\}$ such that $\sum_{s_i \in S} s_i = 2B$. The problem is to determine if there is a partition S_1 and S_2 of set S with $|S_1| = |S_2|$ and $\sum_{s_i \in S_1} s_i = \sum_{s_i \in S_2} s_i = B$.

For an instance of Equal-Size-Partition, we create an instance of $2t + 1$ jobs for the mission planning problem as follows:

Ordinary job J_j: $\alpha_j = 0, \beta_j = B + s_j, p_j = B - s_j, 1 \leq j \leq 2t$;

Large job J_{2t+1}: $\alpha_j = (t + 1)B, \beta_j = 3tB^2, p_j = 0$;

Initial resource level $v_0 = 0$;

Target resource level $\bar{v} = 3tB^2$;

Total processing time is no more than $(t - 1)B$.

Given the constructed instance, if the mission planning problem has a feasible schedule, then by the value of target resource level job J_{2t+1} must be selected and scheduled last (by Lemma 3). For a solution to have a total processing time no more than $(t-1)B$, the number of ordinary jobs selected for processing must be less than or equal to t. However, if less than t ordinary jobs are selected, then the resource level after the completion of all ordinary jobs is less than $(t+1)B$ and cannot support the processing of large job J_{2t+1}. Therefore, the number of ordinary selected jobs must be exactly t. Let $\mathcal{J}'$ denote the set of selected ordinary jobs. To support the processing of large job J_{2t+1}, $\sum_{J_j \in \mathcal{J}'} B + s_j \geq (t+1)B$ must hold. On the other hand, not to exceed the total processing length, we have $\sum_{J_j \in \mathcal{J}'} B - s_j \leq (t-1)B$. Combining the two inequalities, we have $\sum_{s_j \in S_1} s_j = B$, where S_1 contains the elements that define the jobs of $\mathcal{J}'$. Therefore, we have found a partition as specified in Equal-Size-Partition.

Assume now that there is a partition S_1 and S_2 of set S in Equal-Size-Partition. An optimal schedule of the mission planning problem is constructed by selecting job J_{2t+1} and the jobs defined by the elements of S_1 (or S_2), and the processing length is $(t-1)B$. If there is an r-approximate algorithm for the mission planning algorithm, then it will produce a solution value less than $r(t-1)B$. From the constructed instance, we know that if the approximation algorithm cannot report the optimal solution value, then it will not find a feasible schedule. That is, the performance ratio is unbounded, and the theorem follows. $\square$

The above theorem indicates that it is very unlikely to design r-approximate algorithms, PTAS or FPTAS for the mission planning problem. In the following, we investigate the approximability of a special case. We consider the case when the resource requirement α_j is uniform for all jobs. In this case, we can simply assume $v_0 = 0$ and $\alpha_j = 0$ for all jobs. Then, the problem can be formulated as the Minimization Knapsack problem.

The minimization knapsack problem (MinKP) is a transformation of the traditional knapsack problem (KP) into a minimization problem. From a finite number of items a subset shall be selected with total profit at least $\bar{p}$ such that the total weight of the selected items is minimized.

Because the initial resource level $v_0 = 0$, there is no feasible problem during executing each job. With a uniform resource requirement ($\alpha_i = \alpha$) among all jobs, the mission planning problem can thus be treated as the MinKP problem as in the following formulation:

$$\text{(MinKP)} \quad \text{Minimize} \quad \sum_{j=1}^{n} p_j x_j$$

subject to

$$v_0 + \sum_{j=1}^{n} \delta_j x_j \geq \bar{v}; \tag{4.1}$$

$$x_j \in \{0,1\}, \qquad 1 \leq j \leq n. \tag{4.2}$$

In the literature, a multiple-run 2-approximate algorithm was designed for MinKP. In this thesis, we design another greedy algorithm, called MinGreedy, and provide a 2-approximate proof. First, we re-index the jobs in non-decreasing order of the ratio of processing time and resource contribution, i.e. $\frac{p_1}{\delta_1} \leq \frac{p_2}{\delta_2} \leq \ldots \leq \frac{p_n}{\delta_n}$. The algorithm is given as follows.

```
ALGORITHM MINGREEDY :
begin
    Initializaion:
        $S_1 = \emptyset$. (Set of selected jobs)
        $S_2 = \{1, 2, \ldots, n\}$. (Set of untouched jobs)
        $v = 0$. (Current resource level)
        $p_{sum} = 0$. (Total processing time)
        $p_{min} = 0$. (Smallest $p_i$ for which $v + \delta_i \geq \bar{v}$)
    return $\textsc{Recursive}(v, p_{sum})$.
end

Function $\textsc{Recursive}(v, p_{sum})$
begin
    for $i = 1$ to $n$
    begin
        If$(v + \delta_i < \bar{v})$
            $v = v + \delta_i$;
            $p_{sum} = p_{sum} + p_i$;
            $S_2 = S_2 \setminus \{i\}$;
        else
            break; ($J_i$ is the first job in $S_2$ that satisfies the constraint.)
    end
    for $j = i$ to $n$
    begin
        If $(v + \delta_i \geq \bar{v})$
            If $(p_{min} < p_j)$ then $p_{min} = p_j$;
            $S_2 = S_2 \setminus \{J_j\}$;
    end
end

While $(S_2 \neq \emptyset)$
begin
    $\textsc{Recursive}(v, p_{sum})$;
```

```
        If $(p_{min} > \sum_{k=1}^{i-1} p_k)$ and $(\sum_{i \in S_2 - \{J_j\}} \delta_i > \bar{v} - \sum_{k=1}^{i-1} \delta_k)$
            $p_{sum} = \min\{\sum_{k=1}^{i-1} p_i + p_{min},$ RECURSIVE$(v, p_{sum})\}$;
        else
            return $(\sum_{k=1}^{i-1} p_i + p_{min})$;
end
```

Theorem 2 ALGORITHM MINGREEDY *is 2-approximate when all jobs have the same resource requirement.*

Proof: Let J_j be the job that first completes the solution $x_1, x_2, \ldots, x_n$ and p^* the optimal processing time. Because of the jobs have been sorted, we have

$$\sum_{i=1}^{j-1} p_i \leq p^*.$$

If $p_j \leq \sum_{i=1}^{j-1} p_i$, then the 2-approximation result is obvious. For the case with $p_j > \sum_{i=1}^{j-1} p_i$, we discuss two conditions, and get final optimal solution in two of them. The analysis is based on induction. For condition 1, we will continue our algorithm, eliminate job J_j without changing the optimal solution. Since $p_j > p^*$, job J_j will not be included in the optimal solution. We find another job J_k in one of the following for-loop. If this condition holds, that is, this condition will return smaller heuristic solution, we could get $p_k \leq \sum_{i \in S_2} p_i$, and complete the proof. For another condition, we will select job J_j into our solution. If condition 2 will get smaller heuristic solution, then we can say that p_j must be chosen to be included into the optimal solution. Thus, we get $p_j \leq p^*$, and this completes the proof. □

Chapter 5

Special Cases

In this Chapter, we consider two special cases that can be solved in polynomial time. In the first case, all jobs have the same processing time. Since processing time $p_i = p$, the problem reduces to minimizing the number of selected jobs to reach the target resource level. This special case can be solved by the following algorithm:

```
Algorithm Equal_Processing_Time
Step 1: Set $v = v_0$, $k = 1$.
Step 2: While (($k \leq n$) and ($v < \bar{v}$)) do
        begin
            Select job $J_j \in \mathcal{J}$ achieving $\delta_j = \max\{\delta_i : \alpha_i \leq v, J_i \in \mathcal{J}\}$.
            Schedule job $J_j$ in position $k$.
            Set $v = v + \delta_j$.
            Set $\mathcal{J} = \mathcal{J} \setminus \{J_j\}$.
            Set $k = k + 1$.
        end
Step 3: Report the schedule.
```

Theorem 3 ALGORITHM EQUAL_PROCESSING_TIME *provides an optimal schedule for the case with equal processing time in $O(n \log n)$ time.*

Proof: The correctness can be easily established by the standard job-interchange argument. As for the computing time, we maintain a a sorted of the jobs based on non-decreasing α_j and a max-heap based on δ_j. Whenever a job requires less than the updated resource level, it is inserted into the max-heap, from which the root is selected for processing. Each job can be inserted into and deleted from the max-heap exactly once. Adjustment operation of each insertion and deletion takes $O(\log n)$ time, thus the overall run time is $O(n \log n)$. $\square$

In the second case of interest, we assume the number of job types to be fixed. Assume the jobs are categorized into K different types, and mobs of a type have the same α_i and β_i. We will design a polynomial time algorithm for determining a feasible sequence whose makespan is minimum.

Let n_k denote the number of mobs in type $k, 1 \le k \le K$. Denote the experience requirement and experience returned of mobs in type $k, 1 \le k \le K$, by $\alpha_{(k)}$ and $\beta_{(k)}$, respectively. We assume the types of mobs are sorted by the values of required experience, i.e. $\alpha_{(k_1)} \le \alpha_{(k_2)}$ if $1 \le k_1 < k_2 \le K$. Moreover, mobs of the same type are sequenced in non-decreasing order of processing times. We re-index all jobs in the above order. If $\sum_{r=1}^{k-1} n_r < j \le \sum_{r=1}^{k} n_r$ and $\sum_{r=1}^{k-1} n_r + l = j$, then job J_j is the l-th job of type k.

Remark 1 *In some optimal schedule of an instance with K types of jobs, if n'_k type-k mobs, $0 \le k'_k \le n_k$, are selected, then they are the first n'_k jobs of type k.*

The property stems from the simple consideration: Within the same type, mobs with shorter processing times will be preferred than those with longer processing times.

Remark 2 *Given an instance with K types of jobs, there exists an optimal schedule $\sigma = \sigma^1 \oplus \sigma^2 \oplus \cdots \oplus \sigma^K$, where $\sigma^k, 1 \leq k \leq K$, is a subsequence of jobs of type k, and $\oplus$ is the sequence concatenation operation.*

This property highlights the structure of an optimal solution where blocks of mobs from the same type are constructed and the blocks are arranged in increasing order of type index. With Remark 2, we readily know that mobs of the same type will be in consecutive positions and mob types are arranged in ascending order of their experience requirements $\alpha_{(k)}$. A schedule can thus be represented in a canonical form $\sigma = (n'_1, n'_2, \ldots, n'_K)$ subject to $0 \leq n'_k \leq n_k$ for all $k = 1, 2, \ldots, K$. Note that for any type k, it is not that n'_k mobs are simultaneously processed but that they are processed in serial.

Remark 3 *Given an instance of an instance with K types of jobs, if schedule $\sigma^1 = (n'_1, \ldots, n'_K)$ is not feasible, then any schedule $\sigma^2 = (n''_1, \ldots, n''_K)$ with $n''_k \leq n'_k$ for all $k = 1, 2 \ldots, K$ is also infeasible.*

With the above optimality properties, we can enumerate and examine all feasible schedules and then select the best solution. The algorithm will start from a seed schedule $\sigma^0 = (n_1, n_2, \ldots, n_K)$ that selects all jobs. From the seed schedule, the algorithm generates all feasible schedules with $n - 1$ jobs selected. The process continues until all feasible schedules are examined.

Algorithm K_Types

Step 1: Set $\Sigma_n = \{\sigma^0\}$, where $\sigma^0 = (n_1, n_2, \ldots, n_K)$.

```
Step 2: Set $\Sigma_i = \emptyset$ for $1 \leq i \leq n$.
Step 3: Set $i = 1$.
Step 4: while $i \leq n$ and $\Sigma_{n-i+1} \neq \emptyset$ do
            begin
              for each element (schedule) $\sigma' = (n'_1, n'_2, \ldots, n'_K) \in \Sigma_{n-i+1}$ do
                begin
                  for $k = 1$ to $K$
                    if $n'_k \geq 1$ and schedule $\sigma'' = (n'_1, n'_2, \ldots, n'_k - 1, \ldots, n'_K)$ is feasible
                    then insert $\sigma''$ into $\Sigma_{n-i}$.
                  if for some $k$ schedule $\sigma''$ is generated, then delete $\sigma'$ from $\Sigma_{n-i+1}$.
                end
              Set $i = i + 1$.
            end
Step 5: Select the schedule from $\bigcup_{i=1}^{n} \Sigma_i$ whose makespan is smallest.
```

Lemma 4 *For any given instance with K types of jobs, all feasible sequences are be generated by Algorithm K_types.*

Proof. The proof is given by induction on index i of the main loop of Algorithm K_types. We show that any feasible schedules of $n - i + 1$ jobs must be included set σ_{n-i+1} during the course of the execution of iteration i, although some of which may be discarded later iteration. When $i = 1$, the only feasible schedule $\sigma^0 = (n_1, n_2, \ldots, n_K)$ is indeed in $\Sigma_{n-i+1} = \Sigma_n$. Assume the claim is true for some $i < n$. Suppose that when $i = k$, $\Sigma_{n-i+1} = \Sigma_{n-k+1}$ collected all feasible schedules σ' from Algorithm K_types. Now let $i = k+1$. Since we know that here are some feasible sequence generated when $i = k$. Thus, $\Sigma_{n-k} \neq \emptyset$, and $i = k + 1$

will execute step 4 in Algorithm K_types. Therefore, all feasible sequences in Σ_{n-k-1} will be generated by Algorithm K_types at step 4. Thus, in any variant sequence σ^k will be collected by Algorithm K_types into the pool Σ_{n-k}. The proof of Lemma 2 is thus complete.

Now let us define a schedule characteristic. Let σ be some feasible. If deletion of any job from $\{\sigma\}$ will result in infeasibility, schedule σ is *minimum.*

Lemma 5 *For any given instance with K types of jobs, the set $\bigcup_{i=1}^{n}\Sigma_i$ generated by AlgorithmK_types contains all minimal schedules.*

Proof. Assume there is a minimal schedule σ not collected into the set $\bigcup_{i=1}^{n}\Sigma_i$. Since σ is minimal, it is also a feasible sequence, too. From Lemma 4, we know that σ must be generated by Algorithm K_types. However, if schedule σ is minimal, it will not deleted by Algorithm K_types in Step 4. Then, σ must remain in the set $\bigcup_{i=1}^{n}\Sigma_i$. This completes the proof. $\square$

We conclude the above analysis into the following result.

Theorem 4 *Given any instance with K types of jobs, Algorithm K_types generates all minimal feasible schedule $\sigma^{n-r}(0 \leq r \leq n)$ and finds the optimal one.*

Since there are K types of job type, and each type has at most $n_i(i = 1, \ldots, K)$ jobs to select to execute. We can say that there are at most $n_1 n_2 \cdots n_K$ different jobs combinations. From Theorem 1, we know that for a given instance of problem G, Algorithm K_types will generate every feasible sequence $\sigma^{n-r}(1 \leq r \leq n)$ and find its optimal schedule. Thus, we come to the overall bound $O(n_1 n_2 \cdots n_K)$ on the number of elements in the pool. The time required to examine the feasibility of each schedule σ in $\bigcup_{i=1}^{n} \Sigma_i$ takes $O(n)$ time. However, we can improve the time to $O(1)$. Given the initial sequence σ^0, applying the concept of composite jobs addressed in Chapter 2 we can derive information on all composite jobs $\sigma^0_{[1:j]}$ and $\sigma^0_{[j:n]}$ for $1 \leq j \leq n$ in $O(n)$ time. The feasibility of each individual sequence generated from σ^0 by deleting one job can be checked in constant time. Therefore, we come up with the following theorem.

Theorem 5 *For the problem with K job types, Algorithm K_types produces the optimal schedule in $O(n'_1 n'_2 \cdots n'_K)$ time.*

As the computing time $O(n'_1 n'_2 \cdots n'_K)$ is polynomial when K is a constant, therefore the mission planning problem with K job types is polynomially solvable. We further restrict the jobs of the same type also have the same processing time. Then, the input will include

$$n_1, \alpha_1, \beta_1, p_1, \ldots, n_K, \alpha_K, \beta_K, p_K.$$

In such a case, even if K is fixed the run time $O(n'_1 n'_2 \cdots n'_K)$ becomes exponential in terms of input length, although it is polynomial in terms of the number of jobs $n = n_1 + n_2 + \cdots + n_K$.

Chapter 6

Concluding Remarks

In this thesis, we formulated the mission planning problem that arises from computer games. The problem is described and formulated using the concept of resource-constrained scheduling, the relocation problem. We designed a pseudo-polynomial time dynamic programming algorithm to produce optimal solutions. The development of the pseudo-polynomial time algorithm also indicates that the mission planning problem cannot be strongly NP-hard. We gave a negative result of approximability in the sense that there exists no r-approximate algorithm for constant rational r unless $P = NP$. For the case with uniform resource requirement, we presented a 2-approximate greedy algorithm. A polynomial time algorithm was designed for the case with uniform processing time. For the case where the jobs are categorized into K different resource types, we design an efficient algorithm that has a polynomial time complexity when K is fixed.

As the mission planning problem is new in the literature, there is considerable room for further extensions. For example, the mobs may come into scene dynamically rather than known in advance. There we can consider the scenario that each mob has its release date as assumed in the area of scheduling theory.

Acknowledgements

This research was partially supported by the National Science Council of Taiwan under grant NSC-93-2416-H-009-026.

Bibliography

[1] A. Amir, and E.H. Kaplan. 1988. Reloaction problems are hard. *Interational Journal of Computer Mathematics* 25(2): 101-126.

[2] J. Blazewicz, J.K. Lenstra, and A.H.G. Rinnooy Kan. 1983. Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics* 5: 11-24.

[3] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch. 1999. Resource-constrained project scheduling: Notation, classification, models and methods. *European Journal of Operational Research*, 112(1): 3-41.

[4] T.C.E. Cheng and B.M.T. Lin, Johnson's rule, composite jobs and the relocation problem, *European Journal of Operational Research*, in press, 2008.

[5] P.L. Hammer. 1986. *Scheduling under Resource Constraints - Deterministic Models. Annals of Operations Research*, 7, J.C. Baltzer AG, Switzerland.

[6] W. Herroelen. 2005. Project scheduling - theory and practices. *Production and Operations Management* 14(4): 413-432.

[7] W. Herroelen, B. De Reyck, and E. Demeulemeester. 1998. Resource-constrained project scheduling: A survey of recent developments. *Computers and Operations Research* 25(4): 279-302.

[8] E.H. Kaplan. 1986. Relocation models for public housing redevelopment programs. *Planning and Design* 13(1): 5-19.

[9] E.H. Kaplan and A. Amir. 1988. A fast feasibility test for relocation problems. *European Journal of Operational Research* 35: 201-205.

[10] E.H. Kaplan and O. Berman. 1988. Orient Heights housing projects. *Interfaces*, 18(6): 14-22.

[11] H. Kellerer, U.Pferschy, and D. Pisinger. 2004. *Knapsack Problems*, Springer-Verlag, Berlin Heidelberg.

[12] A.V. Kononov and B.M.T. Lin. 2006. On the relocation problems with multiple identical working crews. *Discrete Optimization* 21(4): 368-381.

[13] A.V. Kononov and B.M.T. Lin. 2008. Resource-constrined scheduling to minimizing weighted completion time on a single machine, in submission.

[14] B.M.T. Lin. 1994. On finding the most vital job in relocation problems, *Journal of the Operations Research Society of Japan* 37(3): 182-187.

[15] A. Mingozzi, A. Maniezzom, S. Ricciandelli, and L. Bianco. 1998. An exact algorithm for the resource-constrainted project scheduling problem based on a new mathematical formulation. *Management Science* 44(5): 714-729.

[16] M. Pinedo. 2002. *Scheduling: Theory, Algorithms and Systems*, Englewood Cliffs, NJ: Prentice Hall.

[17] M. Pinedo. 2005 *Pinning and Scheduling in Manufacturing and Services*, New York, NY: Springer.

[18] G.J. Woeginger. 2000. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme. *INFORMS Journal on Computing* 12(1) 57-74.

[18] DFC 2007. http://www.dfcint.com/game_article/may07article.html.

[18] IDC 2006. http://www.idc.com.tw/report/News/Taiwan/news_Taiwan_060622_2.htm.

[18] IDC 2008. http://www.idc.com.tw/report/News/Taiwan/news_Taiwan_080715.html.