Survey of Single Machine Scheduling with Application to Web Object Transmission

Ye Xia[*] David Tse

Department of Electrical Engineering and Computer Science
University of California, Berkeley

Abstract

The motivation of this paper is to optimize the downloading process of web pages when the communication capacity is limited. We see a web page as a collection of objects, each of which has certain utility to the user. The problem is to schedule the transmission of the objects so that the total utility is maximized. In this paper, we examine various formulations of the problem and utility assignment schemes. We survey of the the relevant results from the theory of single machine scheduling. The paper also contains some new results.

keywords optimization, single machine scheduling, utility functions, shortest processing time schedule, web

[*]This project is funded by NSF Award #9872764.

1 Introduction

The problem that has motivated this paper is derived from the work of [5] which studies how to optimize user's satisfaction in web browsing sessions via web object transmission scheduling. By web objects, we mean items on a web page, such as a piece of text, an image, an audio or video clip, etc. We do not limit the notion of web objects to those items with complete semantic boundaries, such as a complete JPEG image. A block of a JPEG image, or a low-resolution copy of a multi-resolution-encoded JPEG image can also be considered an object. In the same line of thoughts, an application data unit (ADU) can be considered an object. The defining property of an object is that as a whole it has some utility to the user and any part of it either has no utility or cannot be rendered by the application.

The scenario is the following. Considered a web browsing session in which a user wants to retrieve a particular web page with n objects. The rate of communication between the sever and the user is a constant μ, possibly limited at a slow link in the network or at the busy server. Suppose the user has certain preference over each web object. The question is how the server should schedule the transmission of these n objects in order to maximize the user's overall satisfaction. The notion of user's satisfaction will be made concrete later.

The problem stated here belongs to the general area of single machine scheduling. This paper surveys the results in that area which are relevant to our problem. Due to the online nature of our problem, we pay particular attention to the time complexity of scheduling algorithms. We discuss specific requirements and characteristics of web object transmission and make recommendations toward solving this problem.

The paper is organized as follows. In section 2, we introduce definitions in single machine scheduling and discuss necessary elementary results in this area. In section 3, we formulate the basic scheduling problem as an optimization problem whose objective is the sum of utility functions of objects, and consider some details in general terms. In section 4, we discusses a few families of utility functions and their associated schedules. In section 5, we introduce advanced results for linear utility functions. In section 6, 7 and 8, we deviate a little from the main thread of the paper and discuss three related problem formulations. We conclude in section 9.

2 Elementary Theory in Single Machine Scheduling

The results in this section can be found in [1] and [2]. Suppose that n objects, numbered from 1 to n, are to be transmitted by a single server with a constant capacity μ. We assume the server is non-preemptive. The choice between preemptive and non-preemptive server will be discussed in a separate section. We define the following variables.

r_i The release time, i.e., the time when object i is available for transmission. We assume $r_i = 0$, for all i.

s_i The size of object i.

p_i The transmission time of object i. $p_i = s_i/\mu$.

C_i The completion time of transmitting object i.

W_i The waiting time of object i, before it is selected for transmission.

d_i The deadline (or due date) of object i, i.e., the promised completion time for object i.

L_i The *lateness* of object i. $L_i = C_i - d_i$. Notice that L_i is negative when object i is completed ahead of schedule.

T_i The *tardiness* of object i. $T_i = \max\{C_i - d_i, 0\}$.

$N_u(t)$ The number of unfinished jobs at time t.

2.1 Some Simple Results - Polynomial Time Problems

The objective is to find a schedule to optimize certain performance measure. Following the tradition of machine scheduling theory, the performance measures throughout section 2 are cost functions that are to be minimized. We will list a few typical measures in the scheduling literature that might be relevant to the web object transmission problem, and present some known facts. For a sequence of numbers, $Y_1, Y_2, ..., Y_n$, we denote the mean or average by $\bar{Y} = \sum_{i=1}^{n} Y_i$, and $Y_{max} = \max\{Y_1, Y_2, ..., Y_n\}$. Also define

$$\bar{N}_u = \frac{1}{C_{max}} \int_0^{C_{max}} N_u(t)\ dt$$

mean completion time $\bar{C}$.

weighted completion time $\sum_{i=1}^{n} \alpha_i C_i$, where $\alpha_i > 0$ for all i.

mean tardiness $\bar{T}$.

maximum tardiness T_{max}.

number of tardy jobs $n_T = \sum_{i=1}^{n} \mathbf{1}\{C_i > d_i\}$.

We did not list all obvious performance measures due to the facts that some of the measures are equivalent.

Fact 2.1 *For the single server problem, the following formance measures are equivalent. (i)* $\bar{C}$, *(ii)* $\bar{W}$, *(iii)* $\bar{L}$, *(iv)* $\bar{N}_u$.

Fact 2.2 *A schedule which is optimal with respect to* L_{max} *is also optimal with respect to* T_{max}.

Definition 2.1 *A* ***regular measure*** R *is one that is a non-decreasing function of completion times. That is,* R *is a function of* $C_1, C_2, ..., C_n$ *such that* $C_1 \leq C_1', C_2 \leq C_2', ..., C_n \leq C_n'$ *together implies* $R(C_1, C_2, ..., C_n) \leq R(C_1', C_2', ..., C_n')$.

$\bar{C}, \bar{L}, L_{max}, \bar{T}$, and T_{max} are examples of regular measures.

A scheduler is called *work-conserving* if it may not stop when there are unfinished jobs in the system. A scheduler is called *non-preemptive* if it has the property that, once an object starts to be serviced, it will be completed before the scheduler services another object. When dealing with regular measures, we can restrict our attention to the class of work-conserving and non-preemptive schedulers due to the following facts.

Fact 2.3 *When the performance measure is regular, there exists an optimal work-conserving schedule.*

Fact 2.4 *When the performance measure is regular, no improvement may be gained in the optimal schedule by allowing preemption.*

Since the class of work-conserving and non-preemptive schedules corresponds to the set of all sequences of the n objects, the scheduling problem is equivalent to finding an optimal sequence of the n objects. We now state some results for some common schedules.

Fact 2.5 ***SPT****(Shortest Processing Time) schedule minimizes $\bar{C}$, $\bar{W}$, $\bar{L}$, and $\bar{N}_u$.*

Fact 2.6 *If all jobs have the same due dates,* ***SPT*** *schedule minimizes $\bar{T}$.*

Fact 2.7 *If it is impossible for any job to be on time in any sequence, then $\bar{T}$ is minimized by* ***SPT*** *sequencing; if* ***SPT*** *yields no jobs on time, then it minimizes $\bar{T}$.*

Fact 2.8 ***EDD****(Earliest Due Date) schedule minimizes L_{max} and T_{max}.*

Fact 2.9 *There exists a schedule without tardy jobs iff the* ***EDD*** *schedule yields no tardy jobs.*

Fact 2.10 *The* ***weighted completion time*** *measure $\sum_{i=1}^{n} \alpha_i C_i$, where $\alpha_i > 0$, is minimized by the* ***Weighted Shortest Processing Time*** *(WSPT) first schedule. That is, object i is transmitted before j if*

$$\frac{p_i}{\alpha_i} \leq \frac{p_j}{\alpha_j}$$

The number of tardy jobs, n_T, can be minimized using Moore and Hodgson algorithm [2].

Algorithm 2.1 Moore and Hodgson's
step 1. *Sequence the jobs according to the* ***EDD*** *rule to find the current sequence ($J_{i(1)}$, $J_{i(2)}$,..., $J_{i(n)}$) such that $d_{i(1)} \leq d_{i(2)} \leq ... \leq d_{i(n)}$.*
step 2. *Find the first tardy job $J_{i(l)}$ in the current sequence. If no such job is found, go to step 4.*
step 3. *Find the job in the sequence $(J_{i(1)}, J_{i(2)}, ..., J_{i(l)})$ with the largest processing time and reject this from the current sequence. Return to step 2 with a current sequence one shorter than before.*
step 4. *Form an optimal schedule by taking the current sequence and appending to it the rejected jobs, which may be sequenced in any order.*

Fact 2.11 *In the Moore and Hodgson algorithm, the rejected jobs are all tardy and will be the only tardy jobs.*

SPT, WSPT, EDD and MH schedules can all be suitable for web object transmission because their optimization objectives are reasonable and the scheduling algorithms have a complexity that is polynomial in n.

2.2 Some More Difficult Results - NP-Hard Problems

Notice we did not list a solution for minimizing the *mean tardiness*. Deriving such a schedule is more difficult than the cases above. For many objective functions, the scheduling problems are NP-hard, in which case we have three choices.

- Use some enumeration techniques to find the optimal schedule, such as dynamic programming, branch-and-bound, and curtailed enumeration. We can also formulate an integer programming problem from the scheduling problem and use the techniques in the general framework of integer programming. Or we can find pseudo-polynomial algorithms for problems that are NP-hard but not in the strong sense.
- Find polynomial time approximation algorithms that guarantee a bound for the worst case relative performance, such as the *fully polynomial time approximation scheme.*
- Find other polynomial time heuristic algorithms that either are locally optimal or have good average performance. The *neighborhood search* technique [1] is one example of the former.

Complete enumeration of all possible schedules takes $O(n!)$ operations. People have developed various enumeration strategies to cut down unnecessary enumeration. Dynamic programming takes advantage of the sub-optimal structure of the scheduling problem, i.e., the some sub-sequences of an optimal sequence are themselves optimal sequences for some sub-problem. A dynamic programming approach can reduce the number of enumeration if the solutions at level $k+1$ (with $k+1$ objects, for instance) can use the solutions at level k. Typically, dynamic programming needs to solve all sub-problems at level k before moving up to solve the sub-problems at level $k+1$. This parallelism can be demanding on the memory requirement, since the solutions to a potentially large number sub-problems need to be remembered before the whole problem is completely solved. Branch-and-bound also branches out sub-problems. But it quickly finds some trial sequence, typically through heuristic techniques. It then calculates optimistic bounds for each of the branches and eliminates those whose results are worse then the trial sequence.

For some performance measures, the candidate set of sequences for enumeration may be reduced based on specific properties of the problems. The systematic application of this idea is to look for *dominance conditions* that determine dominant sets. For any sequence outside a dominant set, one can always find another sequence inside the dominant set that is at least as good. Hence, it must be true the dominant set contains an optimal sequence.

2.3 Multiple Performance Measures - Efficient Schedule

Sometimes, it is possible to combine more than one performance measures in some fashion.

Definition 2.2 *A schedule, s, is* ***efficient*** *with respect to measures* γ_1, γ_2, ..., γ_k *if there does not exist a schedule* s' *such that* $\gamma_1(s) \leq \gamma_1(s')$, $\gamma_2(s) \leq \gamma_2(s')$, ..., $\gamma_k(s) \leq \gamma_k(s')$, *with strict inequality holding in at least one case.*

Suppose $g : \mathbb{R}^k \longrightarrow \mathbb{R}$ is an increasing function in each of its components. Define a performance measure $g(\gamma_1(s), \gamma_2(s), ..., \gamma_k(s))$. Then, it is sufficient to optimize this measure in the set of efficient schedules. An example of this is $g(T_{max}, \bar{C}) = 4T_{max} + 9\bar{C}$.

3 Considerations of Object Transmission Formulation with Regular Utility Function

In this section, we will take a more intrinsic approach of formulating the web object transmission problem.

3.1 Objective Functions

Before dealing with the question of how to optimize user's satisfaction, we need to consider how the user's satisfaction is expressed. We assume that each object on a web page has certain utility to the user and that the overall utility of the page is the sum of the utilities of individual objects. If an object cannot be displayed until it arrives completely, its utility should depend on the completion time only and can be denoted by $v_i(C_i)$. Since one would expect that the user is more satisfied if the transmission is completed sooner, the function $v_i(C_i)$ should be non-increasing. The total utility for the page becomes,

$$v(C_1, C_2, ..., C_n) = \sum_{i=1}^{n} v_i(C_i) \tag{1}$$

Note that, unlike the cost functions in section 2, the utility function reflects user's satisfaction and is to be maximized. The *mean completion time*, *weighted completion time*, and *mean tardiness* all have the same form as the right hand side of (1), but are cost functions. Another commonly used cost function of this sort is the tardiness penalty measure or weighted tardiness, $\sum_{i=1}^{n} w_i T_i$, where $w_i > 0$ for all i.

A utility function is said to be regular if the negative of it is regular. Since the utility function of the form as in (1) is regular, optimality can be achieved by non-preemptive schedules. The corollary is that processor sharing does not make further improvement.

On the other hand, if the object can be displayed progressively in very fine granularity, we can idealize the situation by assuming every infinitesimal piece of data is useful. Then, the utility of object i can be denoted by the function $v_i(t, x)$, which represents the value received by the user when x bits have arrived by time t. In this case, preemptive schedule may be required to achieve optimality.

3.2 Specifying Values of Utility Functions

In practice, a user faces two problems with specifying the utility functions. First, he needs to decide on a large number of function values for every C_i and for every object i. Second, since each value corresponds to an abstract notion of utility, he may find it impossible to determine the value. Paradoxically, the user might find it more natural to assign a ranking directly to each possible sequence of the objects, if he has the patience to enumerate the potentially large number of possible sequences. On the other hand, a particular permutation can be the optimal schedule for more than one utility functions. This suggests that we can approximate the true utility functions by simpler parametric functions and still derive optimal or near optimal permutation.

3.3 Specifying Deadlines

In traditional job scheduling problems, it is common for a job to have a due date so that costs can be associated with violation of the due date. The concept of deadline for a web object can be problematic. A deadline for the transmission of a web object may not have real significance. For example, a non-real time object has no hard deadline; however, it is always preferable for it to arrive sooner. It is plausible that the time requirement for interactivity (normally around 200 ms) can be used as the deadline. However, the significance of this kind of deadline depends crucially on the transmission speed. If the link speed is so low that the transmission time of the object far exceeds the its deadline, specifying such a deadline appears to be pointless. Similarly, if we choose the values for the deadlines so large that no deadline is violated by any permutation, then any permutation is an optimal schedule. It is, therefore, burdensome for the server or the user to make a decision on the deadline.

3.4 Uncertainty in Processing Capacity

In our context, the processing capacity is determined jointly by the web server's capacity and the transmission bandwidth in the network path from the web server to the user. The processing capacity can vary due to many factors, such as variation of cross-traffic load in the path, variation of server load, transport's congestion control algorithms, and retransmission of lost packets, etc. The processing capacity fluctuates on different time scales, depending on its cause. Sometimes, we need to model the processing capacity as a changing quantity rather than as a constant.

3.5 Precedence Constraints

Sometimes a web transfer is subject to precedence constraints. That is, it is required that some objects on the web page be transmitted before some other objects. An optimal schedule should respect the precedence constraints. Many factors may impose precedence constraints among objects. For instance, on-screen objects should be transmitted before off-screen objects. We will discuss a very important factor related to the notion of fine-grained encoding.

A peculiar characteristic of the current web documents is the large imbalance of the object sizes, which can differ by orders of magnitude. Text objects can be as small as a few kilobytes and image files can be as large as hundreds of kilobytes. Large objects often dominate the transmission delay. When large objects are present, an optimal schedule might be as obvious as transmitting small objects before large ones. However, the improvement perceived by the user can be very limited. If the encoding allows a large object to be further divided into smaller objects and each of the smaller objects can be displayed separately, then, the size imbalance of the web objects is reduced and further performance optimization is possible. For example, when multi-level encoding of images is possible, the user obtains certain utility for each coarse copy he receives. As additional copies are received, the image quality improves. For correct reconstruction of the large object, fine-grained encoding may subject those smaller objects to precedence constraints instead of allowing arbitrary orders.

3.6 Computation Complexity

Optimal object-sequencing problems are often NP-hard for even simple utility functions. Finding polynomial-time approximation to the optimal solution can also be difficult. Due to the on-line nature of web object transmission, we prefer utility functions that do not lead to NP-hard scheduling problems or that have simple nearly-optimal solutions. (see Appendix A for relevant results from complexity theory)

4 Several Regular Measures and Their Optimal Schedules

For most cases, the measures are interpreted as utility functions that are to be maximized. Occasionally, we relate them to their corresponding cost functions used in machine scheduling literature.

Formulation 4.1 *Let $v_i(t)$ be the value of object i if it is completed at time t. Find an ordering of the n objects so that $\sum_{i=1}^{n} v_i(C_i)$ is optimized, where C_i is the time when object i is completed.*

In the following, we will consider several functions for v_i, for a generic object i. These functions are summarized in Table 1.

Table 1: Objective Functions

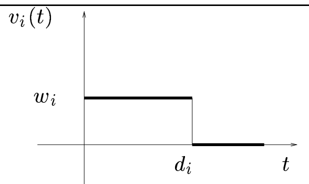
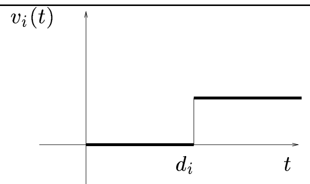
	Function Expression	Utility	Cost
I	$v_i(t) = \begin{cases} w_i & \text{if } 0 \le t \le d_i \\ 0 & \text{otherwise} \end{cases}$		

Table 1: Continued

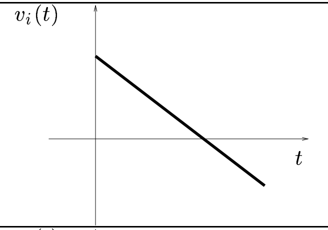
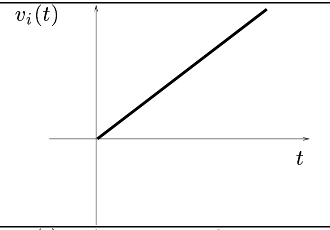
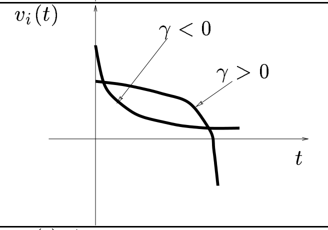
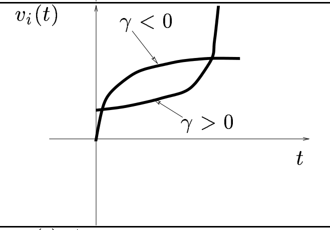
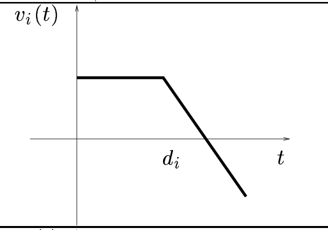
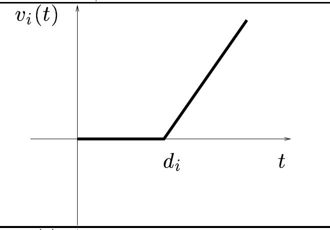
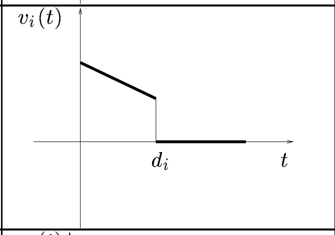
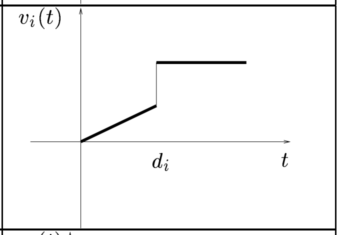
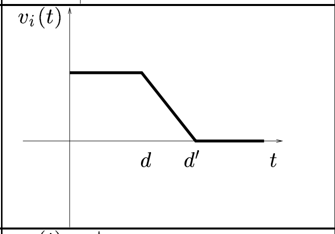
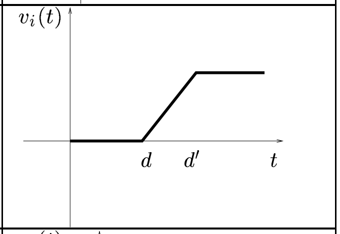
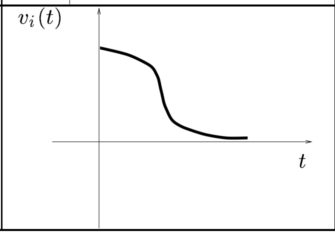
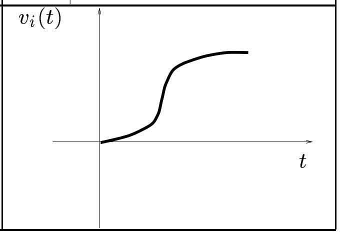
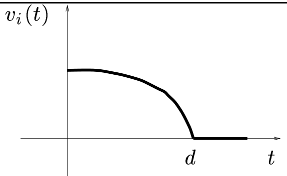
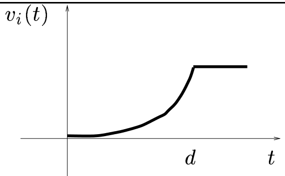
	Function Expression	Utility	Cost
II	$v_i(t) = \alpha_i t + \beta_i$		
III	$v_i(t) = \alpha_i e^{\gamma t} + \beta_i$		
IV	$v_i(t) = \begin{cases} w_i & \text{if } 0 \leq t \leq d_i \\ \alpha_i t + \beta_i & \text{otherwise} \end{cases}$		
V	$v_i(t) = \begin{cases} \alpha_i t + \beta_i & \text{if } 0 \leq t \leq d_i \\ 0 & \text{otherwise} \end{cases}$		
VI	$v_i(t) = \begin{cases} w_i & \text{if } 0 \leq t \leq d_i \\ \alpha_i t + \beta_i & \text{if } d_i \leq t \leq d_i' \\ 0 & \text{otherwise} \end{cases}$		
VII	$v_i(t) = \beta_i - \frac{\beta_i}{1+\alpha_i \exp(-\gamma_i t)}$		

Table 1: Continued

	Function Expression	Utility	Cost
VIII	$v_i(t) = \begin{cases} \alpha_i e^{\gamma t} + \beta_i & \text{if } 0 \le t \le d \\ 0 & \text{otherwise} \end{cases}$		

4.1 Function I - Step Functions

$$v_i(t) = \begin{cases} w_i & \text{if } 0 \le t \le d_i \\ 0 & \text{otherwise} \end{cases}$$

where w_i's are positive numbers, representing the values of the objects. In this model, a utility w_i is gained if the object is received before the due date d_i. Otherwise, the object becomes useless. A minimization version of the problem is as follows.

$$v_i(t) = \begin{cases} 0 & \text{if } 0 \le t \le d_i \\ w_i & \text{otherwise} \end{cases} \tag{2}$$

where w_i's are also positive numbers, representing a penalty when an object is completed after its due date. The corresponding scheduling problem is proved to be NP-hard in [7]. Pseudo-polynomial solution based-on dynamic programming was found for the minimization problem by Lawler and Moore [10]. It has a complexity $O(nT)$, where $T = \sum_{i=1}^{n} p_i$. The technique can be naturally extended to the maximization problem. Fully polynomial time approximation algorithms were found in [15] for the maximization version of the problem with time complexity $O(n^2/\epsilon)$, and in [4] for the minimization version of the problem with time complexity $O(n^2 \log n + n^2/\epsilon)$.

We will introduce the algorithm by Lawler and Moore [10] because the technique also serves as basis for some other more difficult problems. They start by considering the following general formulation. Suppose n jobs are to be processed in the *fixed* order, 1, 2, ..., n. Each job can be performed in either one of two different modes. In the first mode, the processing time of job j is a_j^1 and the cost function is $\rho_j^1(t)$. In the second mode, the processing time of job j is a_j^2 and the cost function is $\rho_j^2(t)$. The objective is to assign one mode to each object so that the total cost is minimized. Let $f(j,t)$ be the minimum total cost for the first j jobs, subject to the constraint that job j is completed no later than time t. The dynamic programming solution for this problem is as follows.

Algorithm 4.1 Lawer and Moore

$$
\begin{aligned}
&f(0,t) = 0 && (t \geq 0), \\
&f(j,t) = +\infty && (j = 0, 1, ..., n;\ t < 0), \\
&f(j,t) = \min \left\{ \begin{array}{l} f(j, t-1) \\ \rho_j^1(t) + f(j-1, t-a_j^1) \\ \rho_j^2(t) + f(j-1, t-a_j^2) \end{array} \right\} && (j = 1, 2, ..., n;\ t \geq 0)
\end{aligned}
$$

The assignment problem is solved by computing $f(n, T)$, where T is a sufficiently large number. For instance, we can choose $T = \sum_{i=1}^{n} \max\{a_i^1, a_i^2\}$ when the cost functions are regular. The computation requirement is $O(nT)$.

Going back to the object sequencing problem that minimizes the total cost $\sum_{i=1}^{n} v_i(C_i)$, where v_i is the step function as in (2), we only need to partition the objects into two groups: those that are completed before their deadlines and those that are tardy. In the actual schedule, all tardy objects simply follow those on-time objects in arbitrary order among themselves. Given an optimal sequence, we can always order those objects that are on time by the EDD schedule. These objects will still be completed before their deadlines in the resulting new order. Hence, the algorithm for finding an optimal schedule is as follows. First, order the n objects by the EDD schedule. Without loss of generality, we assume this order is $1, 2, ..., n$. For each object j, if it is in the first mode, let,

$$
\begin{aligned}
a_j^1 &= p_j \\
\rho_j^1(t) &= \left\{ \begin{array}{ll} 0 & \text{if } 0 \leq t \leq d_j \\ \infty & \text{otherwise} \end{array} \right.
\end{aligned}
$$

If it is in the second mode, let

$$
\begin{aligned}
a_j^2 &= 0 \\
\rho_j^2(t) &= w_j
\end{aligned}
$$

Then, apply Lawler and Moore's algorithm. The objects with the second mode assignment are tardy ones.

4.2 Function II - Linear Functions

$$v_i(t) = \alpha_i t + \beta_i$$

When v_i is interpreted as the utility function, we require $\alpha_i < 0$, and $\beta_i \geq 0$ for all i. That is, the value of the object decreases linearly with the completion time. The corresponding minimization version, where $\alpha_i > 0$ for all i, is recognized as minimizing weighted completion times. For both problems, the optimal schedule is to transmit the objects in increasing order of $p_i/|\alpha_i|$, which has a complexity $O(n \log n)$. Notice that β_i's play no role in determining the optimal sequencing. In fact, as long as all objects have to be transmitted, the β_i's contribute the same constant term for any permutation of the n objects. The proof of optimality of the schedule is a standard "adjacent pair interchange" argument.

Proof: Let us consider the maximization problem. Suppose the optimal sequence is $\pi = (1, 2, ..., n)$. Suppose we interchange the positions of object i and object $i+1$ in the optimal sequence. The loss of utility in object i is $|\alpha_i|p_{i+1}$, and the gain of utility in object $i+1$ is $|\alpha_{i+1}|p_i$. Since the sequence π is optimal, the total change of utility should be non-positive. Therefore, $-|\alpha_i|p_{i+1} + |\alpha_{i+1}|p_i \leq 0$. Or,

$$\frac{p_i}{|\alpha_i|} \leq \frac{p_{i+1}}{|\alpha_{i+1}|}$$

■

One possible drawback with function II is that it decreases indefinitely with t. Jobs that are completed late are penalized when v_i becomes negative. In the context of web object transmission, it may be more reasonable to assume that a late object has zero or small positive value to the user instead of negative value. For that purpose, we can either choose the functions v_i so that they become negative only for very large t, or choose alternative functions such as I, III (where $\gamma < 0$) , V, VI, VII or VIII.

4.3 Function III - Exponential Functions

$$v_i(t) = \alpha_i e^{\gamma t} + \beta_i$$

where $v_i(t)$ is interpreted as a utility function and is a decaying function of t. The optimal schedule is to sort the n objects in decreasing order of $(\alpha_i e^{\gamma p_i})/(1 - e^{\gamma p_i})$.

Proof: We again use adjacent pair interchange argument. Suppose the optimal sequence is $\pi = (1, 2, ..., n)$. Take object i and $i+1$ in the sequence. In the optimal schedule, object i starts to be transmitted at time C_{i-1}, where we assume $C_0 = 0$. Starting with the optimal sequence, switching the position of object i and $i+1$ decreases the utility of object i by

$$(\alpha_i e^{\gamma(C_{i-1}+p_i)} + \beta_i) - (\alpha_i e^{\gamma(C_{i-1}+p_{i+1}+p_i)} + \beta_i) = \alpha_i e^{\gamma C_{i-1}} e^{\gamma p_i}(1 - e^{\gamma p_{i+1}})$$

and increases the utility of object $i+1$ by

$$(\alpha_{i+1} e^{\gamma(C_{i-1}+p_{i+1})} + \beta_{i+1}) - (\alpha_{i+1} e^{\gamma(C_{i-1}+p_{i+1}+p_i)} + \beta_{i+1}) = \alpha_{i+1} e^{\gamma C_{i-1}} e^{\gamma p_{i+1}}(1 - e^{\gamma p_i})$$

Since the sequence π is optimal, the total change of utility should be non-positive. Therefore,

$$-\alpha_i e^{\gamma C_{i-1}} e^{\gamma p_i}(1 - e^{\gamma p_{i+1}}) + \alpha_{i+1} e^{\gamma C_{i-1}} e^{\gamma p_{i+1}}(1 - e^{\gamma p_i}) \leq 0$$

Equivalently,

$$\frac{\alpha_i e^{\gamma p_i}}{1 - e^{\gamma p_i}} \geq \frac{\alpha_{i+1} e^{\gamma p_{i+1}}}{1 - e^{\gamma p_{i+1}}} \tag{3}$$

■

Notice again that β_i's play no role in determining the optimal schedule.

We can have two types of decaying exponential functions, a convex function when $\gamma < 0$ and a concave function when $\gamma > 0$. In order for the function to represent the "value" of an object, we need to require $\alpha_i > 0$ and $\beta_i \geq 0$ when $\gamma < 0$. The convex function remains positive for all $t \geq 0$. Since we may expect the value of an object eventually decays to zero, we can assume $\beta_i = 0$ for all i. With this choice of β_i, α_i can be interpreted as the initial value of the object, and $1/|\gamma|$ can be interpreted as a "soft" deadline. Unlike the constant function with a hard deadline, when the deadline is violated by all permutations or when the deadline is satisfied by any permutation, the exponential function still matters in the sense that most schedules are not optimal.

When $\gamma > 0$ and the function is concave, we need to require $\beta_i > 0$, $\alpha_i < 0$ and $\beta_i + \alpha_i > 0$ in order to have an appropriate utility function. Caution should be taken with this function, since its value becomes negative and rapidly decreases when t becomes sufficiently large. The concave version of v_i may be appropriate when the deadline constraint needs to be rigorously enforced.

One potential drawback with the exponential function is that a single parameter γ is used for all objects. That is, all objects have the same deadline.

4.4 Function IV

$$v_i(t) = \begin{cases} w_i & \text{if } 0 \leq t \leq d_i \\ \alpha_i t + \beta_i & \text{otherwise} \end{cases}$$

In order to interpret v_i as the utility function for object i, we require $\alpha_i \leq 0$ and $\alpha_i d_i + \beta_i = w_i$. However, w_i's have no influence on the optimal schedule.

It is more convenient to discuss the minimization version of the problem, which minimizes total weighted tardiness, $\sum_{i=1}^{n} \alpha_i T_i$. In this case, the weights α_i are positive and can be interpreted as the prices to pay per unit of time for completing the object late. The problem is proved to be NP-hard in the strong sense in [12] and [8]. If all weights are equal, the problem is NP-hard in the ordinary sense [6] and can be solved in pseudo-polynomial time by the Lawler and Moore's algorithm (4.1) [8]. If all deadlines are equal with arbitrary weights, the problem is NP-hard in the ordinary sense [21], and can also be solved by the Lawler and Moore's algorithm (4.1). If precedence constraints are added, the problem is NP-hard even with equal deadlines and equal weights [11]. Researchers have developed branch-and-bound and integer programming techniques for finding an optimal sequence, as well as many heuristic techniques for finding approximate solutions. Many general textbooks on scheduling discuss this problem, e.g. [18] and [14].

4.5 Function V

$$v_i(t) = \begin{cases} \alpha_i t + \beta_i & \text{if } 0 \leq t \leq d_i \\ 0 & \text{otherwise} \end{cases}$$

When considered as an utility function, we require $\alpha_i \leq 0$, and $\beta_i > 0$ for all i. We also require $\alpha_i d_i + \beta_i \geq 0$ so that the function is non-increasing. This function can be viewed as a compromise of the step function I and the linear function II.

Lemma 4.1 *The scheduling problem is NP-hard in the strong sense.*

Two other problems that appear to be related to the current problem are proved to be NP-hard in the strong sense, the $n|1||\sum \alpha_i T_i$ and the $n|1|C_i \leq d_i|\sum \alpha_i C_i$, where α_i's are non-negative weights. The notation here is in the standard machine scheduling notation. The first problem stands for n jobs, single machine and minimizing *total weighted tardiness.* The second problem minimizes *total weighted completion time* subject to the constraint that all jobs are completed before their due dates. As mentioned in the case of function IV, the first problem is NP-hard in the strong sense. So is the second problem, as proved in [19] and suggested in [12]. Notice the second problem can be considered to have the following extended cost function v_i.

$$v_i(t) = \begin{cases} \alpha_i t & \text{if } 0 \leq t \leq d_i \\ \infty & \text{otherwise} \end{cases}$$

where $\alpha_i \geq 0$.

We now returns to our problem. When $\alpha_i \leq 0$ for all i and all objects are required to be completed before their due dates, the problem is NP-hard in the strong sense, by its equivalence to the second problem above. We also know when $\alpha_i \leq 0$, the problem is at least NP-hard in the ordinary sense. To see this, let $\alpha_i = 0$, and $\beta_i > 0$ for all i, then v_i becomes function I.

The proof of strong NP-hardness is accomplished by reducing the 3-partition problem to the current problem. The 3-partition problem is known to be NP-hard in the strong sense. We will adapt the proof for the $n|1||\sum \alpha_i T_i$ problem in [8]. We also work with the "decision" version of the problem rather than the "optimization" version of the problem. If the optimization problem has a polynomial-time solution, then the decision problem trivially has a polynomial-time solution. Therefore, to show the optimization problem is NP-hard, it is sufficient to show the decision problem is NP-complete.

We will work with the minimization version of the problem. Let us first redefine $v_i(t)$.

$$v_i(t) = \begin{cases} \alpha_i t & \text{if } 0 \leq t \leq d_i \\ w_i & \text{otherwise} \end{cases}$$

where $\alpha_i \geq 0$ and $w_i \geq \alpha_i d_i$.

3-Partition Problem: Given a set of $3n$ integers $a_1, a_2, ..., a_{3n}$ between 1 and $B-2$ such that $\sum a_i = nB$. Is there a partition of the a_i's into groups of 3, each summing to B?

scheduling problem:

"X"-jobs:	X_i, $1 \le i \le n$.
"A"-jobs:	A_i, $1 \le i \le 3n$.
Processing times:	$p(X_i) = L = (16B^2)\frac{n(n+1)}{2} + 1$, $1 \le i \le n$, $p(A_i) = B + a_i$, $1 \le i \le 3n$.
Weights	$\alpha(X_i) = 0$, $1 \le i \le n$, $\alpha(A_i) = p(A_i) = B + a_i$, $1 \le i \le 3n$.
Due dates	$d(X_i) = iL + (i-1)4B$, $1 \le i \le n$, $d(A_i) = \sum_{i=1}^{n} p(X_i) + \sum_{i=1}^{3n} p(A_i) + 1$, $1 \le i \le 3n$.
Constants	$w(X_i) = W = (L+4B)(4B)\frac{n(n+1)}{2} + 1$, $1 \le i \le n$, $w(A_i)$ can be any value greater than $w(A_i)d(A_i)$, $1 \le i \le 3n$.

Question: Is there a schedule π with total cost $R(\pi) \le W - 1$?

We need to establish that the 3-partition problem has a solution if and only if the scheduling problem above has a solution.

Proof: Suppose the desired partition exists. Without the loss of generality, suppose $\langle a_{3j-2}, a_{3j-1}, a_{3j} \rangle$ is a group, $i \le j \le n$. Consider the schedule

$$\pi = \langle X_1, A_1, A_2, A_3, X_2, A_4, A_5, A_6, X_3, ..., X_i, A_{3i-2}, A_{3i-1}, A_{3i}, ..., X_n, A_{3n-2}, A_{3n-1}, A_{3n} \rangle$$

Since $\sum_{i=-2}^{0} p(A_{3j+i}) = 4B$, for $1 \le j \le n$, X_i finishes at time $d(X_i) = iL + (i-1)4B$, for $1 \le i \le n$. That is, X-jobs all finish on time. Note that the common due date for A-jobs is chosen so that all jobs are completed before it in any work-conserving schedule. A_{3j-2}, A_{3j-1} and A_{3j} all finish by $j(L+4B)$ and their total weight is $4B$, $1 \le j \le n$. Their total cost is no greater than $j(L+4B)4B$. Therefore,

$$R(\pi) \le \sum_{j=1}^{n} j(L+4B)4B = (L+4B)(4B)\frac{n(n+1)}{2} = W - 1 \tag{4}$$

Conversely, suppose there is a schedule π such that $R(\pi) \le W - 1$. We need to show there is a 3-partition. First, notice that no X-job can be tardy, because the cost contributed by any tardy X-job is W. Next, define W_i to be the total weight of the A-jobs following X_i, with $W_{n+1} = 0$ by convention. The total cost due to the group of A-jobs between X_i and X_{i+1} is no less than $(W_i - W_{i+1})iL$. Hence,

$$R(\pi) \ge \sum_{i=1}^{n} (W_i - W_{i+1})iL = L\sum_{i=1}^{n} W_i$$

Note that W_i is also the total processing time for A-jobs following X_i. The total processing time for A-jobs proceeding X_i is $(4B)n - W_i$. Since all X-jobs meet their due dates, we must have

$$iL + (i-1)4B \ge (4B)n - W_i + iL \qquad 1 \le i \le n$$

Equivalently,

$$W_i \geq (n-i+1)4B \qquad 1 \leq i \leq n$$

Suppose for some i, $W_i \geq (n-i+1)4B+1$. Then,

$$\sum_{i=1}^{n} W_i \geq 1 + \sum_{i=1}^{n}(n-i+1)4B = \frac{n(n+1)}{2}4B+1$$

Then,

$$\begin{aligned}
R(\pi) &\geq L(\frac{n(n+1)}{2}4B+1) \\
&= L\frac{n(n+1)}{2}4B + (16B^2)\frac{n(n+1)}{2}+1 \\
&= (L+4B)(4B)\frac{n(n+1)}{2}+1 \\
&= W
\end{aligned}$$

This contradicts (4). Hence, it must be true that $W_i = (n-i+1)4B$, for $1 \leq i \leq n$. From this we conclude that the total weight for the group of A-jobs between X_i and X_{i+1} must be $4B$ in π, $1 \leq i \leq n-1$. Similarly, the total weight for the group of A-jobs following X_n must also be $4B$. Since all A-jobs have $B+1 \leq w(A) \leq 2B-2$, each such group must contain exactly 3 A-jobs. The n groups of 3 jobs correspond to the desired partition. ■

4.5.1 Case of Common Deadline

When all objects have a common deadline, d, the algorithm of Lawler and Moore (4.1) gives a pseudo-polynomial solution for finding an optimal sequence. Again, an optimal sequence divides the objects into two groups: those that are completed before the deadline and those that are tardy. For the objects that are on time, they must be ordered in increasing order of p_i/α_i, with the understanding that $p_i/0 = \infty$. Therefore, to find an optimal schedule, first sort the n objects in this order. Without loss of generality, we assume this order is $1, 2, ..., n$. For each object i, if it is in the first mode, let

$$\begin{aligned}
a_i^1 &= p_i \\
\rho_i^1(t) &= \begin{cases} \alpha_i t & \text{if } 0 \leq t \leq d \\ \infty & \text{otherwise} \end{cases}
\end{aligned}$$

If it is in the second mode, let

$$\begin{aligned}
a_i^2 &= 0 \\
\rho_i^2(t) &= w_i
\end{aligned}$$

Then apply Algorithm 4.1.

4.5.2 Case of Continuous Function

Suppose $\alpha_i < 0$ and $\alpha_i d_i + \beta_i = 0$ for all i. It turns out this continuity requirement makes the problem easier. Then, there is a pseudo-polynomial algorithm for finding an optimal sequence. Let π be an optimal sequence and A_π be the subset of all objects that are on time in π. Then,

Lemma 4.2 *In schedule π, objects in A_π must be ordered in increasing order of $p_i/|\alpha_i|$.*

Proof: First, notice that, in any optimal sequence, all objects in A_π must be contiguous and occupy the first $|A_\pi|$ positions, followed by tardy objects. Suppose the lemma is not true. There must exists two neighboring objects i and j, $i, j \in A_\pi$, with i proceeding j, such that $p_i/|\alpha_i| > p_j/|\alpha_j|$. By exchanging i and j, the value of j increases by $|\alpha_j|p_i$, and the value of i decreases by *no more than* $|\alpha_i|p_j$. But, because $|\alpha_j|p_i > |\alpha_i|p_j$, exchanging the positions of i and j improves the total utility, which contradicts the optimality of π. ■

The optimal sequencing problem can be solved by applying Lawler and Moore's algorithm. We first write the minimization version of the problem, where

$$v_i(t) = \begin{cases} \alpha_i t & \text{if } 0 \le t \le d_i \\ \beta_i & \text{otherwise} \end{cases}$$

where $\alpha_i > 0$ and $\alpha_i d_i = \beta_i$, for $1 \le i \le n$.

Without loss of generality, let us suppose the objects are numbered so that $p_1/|\alpha_1| \le p_2/|\alpha_2| \le ... \le p_n/|\alpha_n|$. Given that the objects are ordered from 1 to n, we need to decide, for each object j, whether it should be completed before d_j. Apply Lawler and Moore's algorithm with the following parameters for $1 \le j \le n$.

$$\begin{aligned} a_j^1 &= p_j \\ \rho_j^1(t) &= \begin{cases} \alpha_j t & \text{if } 0 \le t \le d_j \\ \infty & \text{otherwise} \end{cases} \\ a_j^2 &= 0 \\ \rho_j^2(t) &= \beta_j \end{aligned}$$

4.6 Function VI

$$v_i(t) = \begin{cases} w_i & \text{if } 0 \le t \le d_i \\ \alpha_i t + \beta_i & \text{if } d_i \le t \le d_i' \\ 0 & \text{otherwise} \end{cases}$$

where $w_i > 0$, $0 \le d_i \le d_i'$ and $\alpha_i < 0$. In order to have a non-increasing utility function, we also require $w_i \ge \alpha_i d_i + \beta_i$ and $\alpha_i d_i' + \beta_i \ge 0$. The value of the object starts at a fixed value and remains unchanged until time d_i. It decreases linearly from d_i to d_i'. After d_i', the value of the object becomes zero. This function is more versatile in approximating "real" utility functions than function V. Since this function becomes function V when all d_i's become zero, or when all d_i''s become very large, the scheduling problem is NP-hard in the strong sense. Even when

$d_i = d$ and $d'_i = d'$ for all i, we suspect that the problem is still NP-hard in the strong sense. In this case, it will be interesting to find good approximate algorithms.

Two other functions (VII and VIII) can approximate this function when their parameters are properly chosen, as can be seen from figure 1.

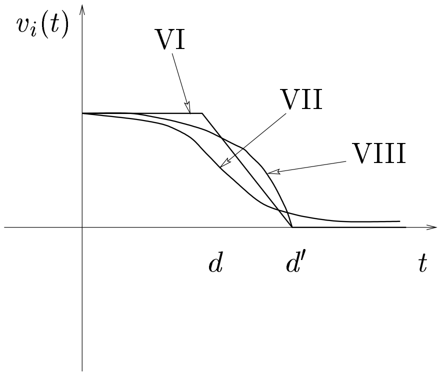


Figure 1: Function VI, VII and VIII

4.7 Function VII

$$v_i(t) = \beta_i - \frac{\beta_i}{1 + \alpha_i \exp(-\gamma t)}$$

where we assume $\gamma > 0$, $\alpha_i \gg 1$ and $\beta_i > 0$. With these constraints on the parameters, v_i is the logistic function reflected against the $t = 0$ and shifted. This function has three operating regions. It starts at $v_i(0) = \frac{\beta_i \alpha_i}{1+\alpha_i} \approx \beta_i$ and decreases gradually for small t. At around $t = \frac{\ln \alpha_i}{\gamma}$ is a transition region where the value of $v_i(t)$ decreases to nearly zero. After that, the function continues to decrease, tending to zero. This v_i can be interpreted as a fuzzy version of function I in the sense that the transition from a positive constant to zero occurs over a period of time rather than instantaneously. The time complexity of this problem is unknown. It will be interesting to find an efficient approximation for this problem.

When $\alpha_i = \alpha$ for all i, this function can also be viewed as a smoothed version for function VI. In this case, we can find a "locally" optimal solution easily. Suppose object i and j are adjacent in an optimal schedule, π, in that order. Let t be the starting time of object i. Denote

$$\phi(i,t) = \frac{\beta_i e^{-\gamma p_i}}{(1 + \alpha e^{-\gamma(t+p_i)})(1 - e^{-\gamma p_i})}$$

Using the neighborhood exchange argument, one can show that in π,

$$\phi(i,t) \geq \phi(j,t) \tag{5}$$

Our algorithm starts at time $t_o = 0$. We choose the object with the largest value of $\phi(i, t_o)$ among all n objects to be the first one. Suppose the k^{th} object in our schedule finishes at time t_k. The $(k+1)^{th}$ object in the schedule should have the largest value of $\phi(i, t_k)$ among the remaining objects. The resulting schedule cannot be improved by exchanging neighboring objects.

The condition in (5) only characterizes the relationship between two neighboring objects in an optimal sequence. The hope is that this necessary condition for optimality severely limits the number of "locally" optimal sequences in practice, and that the resulting sequence from our algorithm is often nearly optimal.

4.8 Function VIII

$$v_i(t) = \begin{cases} \alpha_i e^{\gamma t} + \beta_i & \text{if } 0 \le t \le d \\ 0 & \text{otherwise} \end{cases}$$

where we assume $\gamma > 0$, $\alpha_i < 0$, $\alpha_i + \beta_i > 0$, and $\alpha_i e^{\gamma d} + \beta_i \ge 0$. With these choices, the utility function first decreases as a concave exponential function until time d. After that, its value becomes zero. Notice that all objects have the same deadline in this case.

We again use Lawler and Moore's algorithm (4.1) to find a pseudo-polynomial solution for optimal sequencing. First, we construct a minimization version of the problem by defining $\beta_i + \alpha_i - v_i(t)$ as the cost function for each object. The cost function thus defined is non-negative. An optimal sequence divides the objects into two groups: those that are completed before the deadline and those that are tardy. For the objects that are on time, they must be ordered in decreasing order of $(\alpha_i e^{\gamma p_i})/(1 - e^{\gamma p_i})$. Therefore, to find an optimal schedule, first sort the n objects in this order. Without loss of generality, we assume this order is $1, 2, ..., n$. For each object i, if it is in the first mode, let

$$\begin{aligned} a_i^1 &= p_i \\ \rho_i^1(t) &= \begin{cases} \alpha_i - \alpha_i e^{\gamma t} & \text{if } 0 \le t \le d \\ \infty & \text{otherwise} \end{cases} \end{aligned}$$

If it is in the second mode, let

$$\begin{aligned} a_i^2 &= 0 \\ \rho_i^2(t) &= \alpha_i + \beta_i \end{aligned}$$

Then apply Algorithm 4.1.

Remark 1. In the cases of function II and III, the position of an object in the optimal schedule depends only on some function of its own parameters. This function is called priority-generating function in [19]. We will implicitly use this notion when we develop schedules for objects under precedence constraint.

Remark 2. It is interesting to find the general requirement on the functions v_i so that pseudo-polynomial time algorithms exist.

5 Advanced Results for Linear Utility Function

The results in this section applies to the linear utility function as in Formulation 4.1 (ii). We will show that the optimal schedules are still simple under more complicated situations for this utility function. We will occasionally compare the linear utility function with the exponential utility function, since these two functions have some common features as far as object scheduling is concerned.

5.1 Optimal Sequencing with Precedence Constraint

Suppose the objects to be scheduled have precedence constraints among themselves. We use the notation $i \to j$, if object i is required to be transmitted before object j. Precedence constraints can be represented on a directed acyclic graph (DAG) $G = (N, A)$, in which each node $i \in N$ represents an object and each arc $(i, j) \in A$ represents a precedence constraint that object i must be transmitted before object j. We will look at cases when the precedence constraints take the forms of parallel-chains, outtrees and intrees in the DAG representation (See figure 2). A type of DAG named series-parallel DAG covers a large class of precedence constraints including the above three types. Figure 3 shows one example of a series-parallel DAG. All these problems have polynomial time solutions. The simplest known solution for the case of series-parallel constraints was developed by Lawler [9], which has a time complexity $O(n \log n)$. The problem with general precedence constraint is NP-hard in the strong sense [9]. However, we can argue that restricting ourselves to series-parallel DAGs is quite adequate for the web object transmission problem, as we will see after the concept is defined.

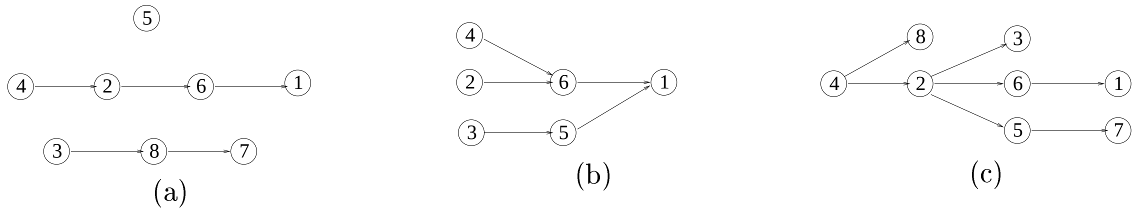


Figure 2: Precedence Constraints: (a) Parallel-Chains, (b) Intree, (c) Outtree

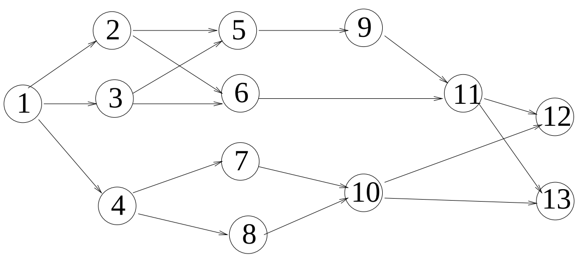


Figure 3: Series-Parallel Precedence Constraints

In the following we will first introduce a parallel chain algorithm, which was developed to solve the problem with parallel-chain constraints, and is then used as a sub-algorithm in the solution to the problem with series-parallel constraints. Even though this algorithm is not as efficient as Lawler's algorithm, it is simpler to explain and it motivates more abstract treatment for more general precedence constraints. We then turn to Lawler's algorithm.

5.1.1 Parallel-Chains

A chain of k jobs is a type of precedence constraint taking the form $i_1 \to i_2, \to ..., \to i_k$. Let $G = (N, A)$ be DAG representation of the precedence constraint. Then, $(i_l, i_{l+1}) \in A$, for $l = 1, 2, ..., k-1$. Suppose the precedence constraints take the form of chains in parallel, and the utility function for object i is $v_i(t) = \alpha_i t + \beta_i$, where $\alpha_i < 0$ and $\beta_i \geq 0$. Associate the chain $i_1 \to i_2, \to ..., \to i_k$ with a number $\xi(i_1, i_2, ...i_k)$, defined by,

$$\xi(i_1, i_2, ...i_k) = \max_{1 \leq l \leq k} \frac{\sum_{j=1}^{l} |\alpha_j|}{\sum_{j=1}^{l} p_j} \tag{6}$$

Let l^* be the index for which the above maximum is achieved. We say that job i_{l^*} *determines* the chain. Then the scheduling algorithm can be stated as the follows.

Algorithm 5.1 (Sidney's Parallel Chains)
step 1. *Calculate ξ and l^* for each chain. Transmit the chain with the smallest ξ factor without interruption upto and including the object that determines the chain.*
step 2. *Then repeat step 1. with the rest of the objects.*

One can observe that this algorithm is a generalization of the WSPT schedule.

5.1.2 Series-Parallel Constraints

Let $N = \{1, 2, ..., n\}$ be the set of objects. Let $G = (N, A)$ be the DAG that represents the precedence constraints. We will first define the class of transitive series-parallel directed graphs (digraph) recursively.

Definition 5.1 *[9]* **(Transitive Series Parallel)**
(i) A digraph consisting of a single node is a transitive series parallel.
(ii)If $G_1 = (N_1, A_1)$ and $G_2 = (N_2, A_2)$ where $N_1 \cap N_2 = \emptyset$, are transitive series parallel, then,

$$G = G_1 \times G_2 = (N_1 \cup N_2, A_1 \cup A_2 \cup N_1 \times N_2)$$

is also transitive series parallel. G is said to be formed by the series composition of G_1 and G_2.
(iii))If $G_1 = (N_1, A_1)$ and $G_2 = (N_2, A_2)$ where $N_1 \cap N_2 = \emptyset$, are transitive series parallel, then,

$$G = G_1 \cup G_2 = (N_1 \cup N_2, A_1 \cup A_2)$$

is also transitive series parallel. G is said to be formed by the parallel composition of G_1 and G_2.

A digraph $\bar{G} = (N, \bar{A})$ is the *transitive closure* of a digraph $G = (N, A)$ if $\bar{A} \supseteq A$, and for every $(i,j) \in \bar{A} \backslash A$, there is a path from node i to node j is G. A digraph is said to be *series parallel* if its transitive closure is transitive series and parallel (See figure 3). Every series-parallel digraph is acyclic. The class of series-parallel digraph is large enough to include chains, outtrees, where every node has no more than one predecessor, and intrees, where every node has no more than one successor. By definition, the series or parallel compositions of the above three types of DAGs are also series parallel.

We argue that the class of series-parallel DAG is adequate to represent precedence constraint at different levels of granularity for the web objects. We start with the coarsest level where each web object is a complete image, audio or video file, or text, etc. In other words, each web object has well defined semantic boundary. At this level, the number of objects on a typical web page is small, and the precedence relations among objects are typically simple, maybe taking the form of chains, outtrees and intrees in parallel. At a finer granularity, each of the above objects can be further divided into smaller objects. Let us call a set of objects originated from the same larger object an object group. If the precedence constraint among these finer objects in each object group can be represented by series-parallel DAGs, then the precedence constraints among all objects can be represented by a series-parallel DAG.

Any series-parallel DAG, G, can be decomposed into series and parallel components. The final decomposition representation of G can used to see how to recursively construct the transitive closure of G as in the definition 5.1. The decomposition takes the form of a binary tree in which each leaf node is a single node in G, and each non-leaf node is either labeled P or S. A P node indicates that its two children, each of which is a subtree rooted at the current P node, will form a parallel composition in the transitive closure of G. An S node indicates that its two children will form a series composition in the transitive closure of G. By convention, the left child of S corresponds to the subgraph G_1 and the right child corresponds to the subgraph G_2 in the definition. As an example, figure 4 shows the decomposition tree for the series-parallel DAG shown in figure 3. The decomposition tree not only shows that G is indeed series parallel, it is also the starting point for the scheduling algorithm. In the following, we assume that the decomposition tree is available for G. There exists an algorithm of a time complexity $O(n+m)$ to recognize and decompose a series-parallel DAG, where m is the number of arcs in G [20].

5.1.3 Sidney's Theory

The results of this section comes from Sidney's paper [16], and summarized by Lawler [9]. Given the DAG $G = (N, A)$,

Definition 5.2 *A nonempty subset $M \subseteq N$ is a (job) module if, for each job, $j \in N \backslash M$, exactly one of the following three conditions holds:*
(i) j must proceed every job in M;
(ii) j must follow every job in M;
(iii) j is not constrained with respect to any job in M.

Every singleton subset of N is a job module. A chain or a subsequence of consecutive elements in a chain are also examples of job modules. An important fact is that, given a series parallel

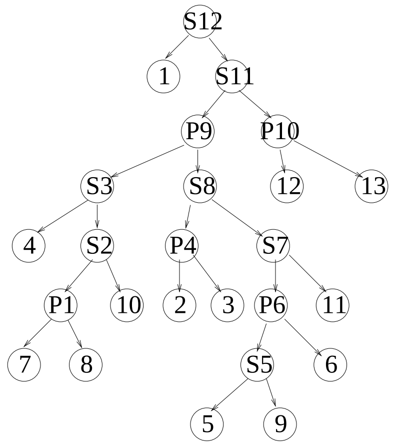


Figure 4: Decomposition Tree for Series-Parallel DAG in Figure 3

G, a subtree rooted at any node of its decomposition tree is identified with an module. The significance of the concept of job module is contained in the following theorem.

Theorem 5.1 *Let M be a module of $G = (N, A)$ and σ is an optimal sequence for M. Then there exists an optimal sequence for N consistent with σ in the sense that the jobs in M appear in the same order as in σ.*

The theorem and the fact that all subtrees in the decomposition tree for G are modules suggest a divide-and-conquer approach to find an optimal sequence for N. The theorem implies that the elements of module M can be considered to be subject to precedence constraints among themselves in the form of a chain σ. We can then work from the bottom of the decomposition tree for G upto the root, and combine modules into larger modules. For series composition of two modules, M_1 and M_2, rooted at the same node, each with precedence constraints in the form of chains σ_1 and σ_2, respectively, we can simply concatenate the σ_1 and σ_2 together. For parallel composition of the two modules, we can merge the chain σ_1 and σ_2 using the parallel-chain algorithm 5.1. In the following, we specify the algorithm for finding an optimal sequence for jobs subject to precedence constraints in the form of a series-parallel DAG, G. Assume that levels of the decomposition tree for G are numbered from top down and the root is at level 0.

Algorithm 5.2 (Sidney's Series Parallel)
step 1. *Let k be the second level from the bottom of the decomposition tree for G.*
step 2. *At level k, if a node is labeled S, find a subsequence by simply concatenating its left child with the right child in that order. If a node is labeled P, find a subsequence by running the parallel-chain algorithm 5.1 on its children. Record the newly found subsequence at the current node.*
step 3. *If $k > 0$, then let $k = k - 1$. Repeat step 2.*

5.1.4 Lawler's Algorithm for Series-Parallel Constraints

Lawler develops an algorithm with a time complexity $O(n \log n)$ once the decomposition tree for G is found [9]. We will summarize this result here. The following two definitions and theorem are by Sidney [16].

Definition 5.3 *Let M be a module. A subset $I \subseteq M$ is an **initial set** of M, if for each $j \in I$, all predecessors of j in M are also in I.*

For example, let the set of nodes of a chain to be module M. The subset consisting of the first k consecutive elements of the chain is an initial set of M, where k is no greater than the cardinality of M. For any subset $I \subseteq N$, let

$$\rho(I) = \frac{\sum_{j \in I} |\alpha_j|}{\sum_{j \in I} p_j} \tag{7}$$

Definition 5.4 *Let M be a module. An initial set I^* of M is said to be ρ-maximal if $\rho(I^*) \geq \rho(I)$, where I is any initial set of M.*

Theorem 5.2 *Let M be a module of $G = (N, A)$ and I be a ρ-maximal initial set of M. Then there exists an optimal sequence for N in which the jobs in I are a consecutive subsequence, preceding all other jobs in M.*

The theorem implies that all elements in a ρ-maximal initial set I for a module M can be treated as a 'single' composite job. The weight factor for the composite job is $\sum_{j \in I} |\alpha_j|$ and the processing time is $\sum_{j \in I} p_j$. In the case when module M is a chain $i_1 \rightarrow i_2, \rightarrow ..., \rightarrow i_k$ and i_{l^*} determines the chain, $\{i_1, i_2, ..., i_{l^*}\}$ is a ρ-maximal initial set for M. The theorem says that there exists an optimal sequence for N consistent with $(i_1, i_2, ..., i_{l^*})$, and hence has justified the use of parallel-chain algorithm 5.1 to merge parallel chains.

Sidney's algorithm for series-parallel DAGs can be summarized as joining chains of jobs for series composition and merging chains of jobs for parallel composition. Lawler's algorithm is based on the same theorems but operates slightly differently to achieve $O(n \log n)$ time complexity. An optimal sequence for a module is represented simply as a set of jobs, some of which may be composite. For any such set of jobs, an optimal sequence can be found by simply placing them in non-increasing order of the ratio $\rho(j) = |\alpha_j|/p_j$. The precedence constraints will have been dealt with by the formation of composite jobs so that such a sequence is necessarily feasible. Then for parallel composition of M_1 and M_2, all that is necessary is to form the union of the two sets M_1 and M_2. Non-increasing ratio order is feasible and optimal for $M = M_1 \cup M_2$ assuming this is true for M_1 and M_2 individually.

Series composition of M_1 and M_2 is more complicated since it is responsible for forming the composite jobs. First, find the minimum-ratio job i in M_1 and the maximum-ratio job j in M_2. If $\rho(i) > \rho(j)$, all that is needed is to form a union of the sets M_1 and M_2. Non-increasing ratio order is feasible and optimal for $M = M_1 \cup M_2$, assuming this is true for M_1 and M_2 individually.

If $\rho(i) \leq \rho(j)$, then $\{i,j\}$ can be treated as a module. Since $\rho(i) \leq \rho(j)$, $\{i,j\}$ is an initial ρ-maximal set of $\{i,j\}$. Hence, by Theorem 5.2, there exists an optimal sequence for N in which i and j are consecutive. We remove i from M_1 and j from M_2 and form a composite job $k=(i,j)$.

Next, we find the next minimal element i in M_1. If $\rho(i) \leq \rho(k)$, we remove i from M_1 and form a new composite job $k=(i,k)$. We continue in this way until either M_1 is empty or $\rho(i) > \rho(k)$. Then we find the next maximal element j in M_2. If $\rho(k) > \rho(j)$, we can let $M = M_1 \cup M_2 \cup \{k\}$. If $\rho(k) \leq \rho(j)$, we remove j from M_2 and form a new composite job $k=(k,j)$. Then, we repeat from the beginning of this paragraph.

The procedure for series composition is outlined below. In order to avoid tests for empty set, we assume M_1 contains a dummy element with ratio $+\infty$ and M_2 contains a dummy element with ratio $-\infty$.

step 1. Find a minimal element i in M_1 and a maximal element j in M_2. If $\rho(i) > \rho(j)$, let $M = M_1 \cup M_2$ and halt. Otherwise, remove i from M_1, j from M_2 and form a composite job $k=(i,j)$.

step 2.

2.1. Find a minimal element i in M_1. If $\rho(i) > \rho(k)$, go to Step 3.1.
2.2. Remove i from M_1 and form the composite job $k=(i,k)$. Return to Step 2.1.

step 3.

3.1. Find a maximal element j in M_2. If $\rho(k) > \rho(j)$, let $M = M_1 \cup M_2 \cup \{k\}$ and halt.
3.2. Remove j from M_2 and form the composite job $k=(k,j)$. Go to Step 2.1.

Remark 3. Module-based decomposition approach has been extended to the case of general precedence constraints. Refer to [17] and [13] for this development.

Remark 4. Suppose the utility functions are exponential functions with the form $v_i(t) = \alpha_i e^{\gamma t} + \beta_i$, for $i = 1, 2, ..., n$. It turns out there exist polynomial time solutions when the precedence constraints can be represented by series-parallel digraphs. A general framework treating both linear and exponential utility functions is based on the concept of priority-generating function, which is related to the concept of sequencing function in [13]. Refer to [19] for the development of this concept and its application to the exponential utility functions with series-parallel precedence constraints.

5.2 Optimal Schedule for Random Processing Times

Given the n objects to be transmitted, each with a fixed size, the transmission times for each object can be random due to a few factors. For instance, the bandwidth in the transmission pipe can be random, or the server's processing capability can vary due to the random load. If we consider the fact that utility is received only when the object arrives at the user's workstation, the random delay in the transmission route has to be added to the processing time. Another interesting situation is, when a packet is lost in the network, retransmission of the packet will take some additional time.

5.2.1 Some Results from Stochastic Scheduling

Let us denote the random processing time of object i by X_i, $i = 1, 2, ..., n$, which are not necessarily independent of each other. Since the object completion times are random, the objective is to schedule the n objects in order to optimize the expected utility. Unlike the deterministic scheduling case, we need to consider two classes of scheduling policies in the stochastic case.

Definition 5.5 *[14] In a* ***non-preemptive static policy****, the complete schedule is determined at time 0, and will not change for the entire processing session of the n objects. Non-preemption has the usual meaning.*

Definition 5.6 *In a* ***non-preemptive dynamic policy****, every time the server is free, the next object to be transmitted is determined using all the currently available information. Non-preemption has the usual meaning.*

The reason to make the above distinction is that the stochastic process $\{X_1, X_2, ..., X_n\}$ yields more information as they are observed, and therefore, the *dynamic policy* corresponds to a larger classes of scheduling algorithms than the *static policy*.

For the linear utility function in Formulation 4.1 (ii), let us use the same schedule as the deterministic case with the expected processing times replacing the deterministic processing times. That is, the objects are sequenced in the increasing order of $\mathbb{E}X_i/|\alpha_i|$. The following theorem is quoted from [14].

Theorem 5.3 *The sequencing rule that follows the increasing order of $\mathbb{E}X_i/|\alpha_i|$ maximizes the expected value of the linear utility function in the class of non-preemptive static policies and in the class of non-preemptive dynamic policies.*

The optimality of the above sequencing rule goes further. It can be shown that it is optimal also in the class of *preemptive dynamic policies* when all distributions of the processing times have *increasing completion rate* [14], where the *completion rate* of object i, $c_i(t)$, is defined by

$$c_i(t) = \frac{f_i(t)}{1 - F_i(t)}$$

Here, we assume that the processing time of object i, X_i, is a continuous random variable with distribution F_i and density f_i.

Remark 5. If the objects have precedence constraints in the form of parallel-chains, then Algorithm 5.1 still applies, with ρ to be defined using expected processing times, $\mathbb{E}X_i$, instead of processing times, p_i.

Remark 6. Suppose the objective is to optimize the expected value of the exponential utility function, as in Formulation 4.1 (iv). Then, the optimal schedule in both the class of *non-preemptive static policies* and *non-preemptive dynamic policies* is to sequence the objects in decreasing order of $\alpha_i \mathbb{E}(e^{\gamma X_i})/(1 - \mathbb{E}(e^{\gamma X_i}))$ (See page 187 in [14]).

5.2.2 A Simple Object Processing Time Model

In the deterministic model, the processing time of object i is simply $p_i = s_i/\mu$, where μ is the constant bandwidth. In the case of random transmission times, it is not a trivial task for the server to know the processing time distributions, which is essential for forming the optimal schedule. We hope to leverage on the only piece of information that is certain, the object sizes. Suppose the processing time distributions are such that the expected processing times are in agreement with the object sizes, i.e.,

$$\frac{\mathbf{E}X_p}{\mathbf{E}X_q} = \frac{s_p}{s_q} \tag{8}$$

for any pair of $p, q \in \{1, 2, ..., n\}$. Then, in the case of the linear utility function, the sequence that follow the increasing order of s_i/α_i is optimal for both the deterministic case and the stochastic case in the class of *non-preemptive static or dynamic policies.*

It is reasonable to believe that the condition in (8) can be satisfied in many realistic situations. We propose the following model to justify this. Suppose the basic transmission unit is a packet of a fixed size, and suppose the processing times for packets are independently and identically distributed. Consider two objects p and q. Let Z_p^k, Z_q^k be the processing times for the k^{th} packet for object p and q, respectively. Let s_p and s_q be the sizes of the objects in number of packets for object p and q, respectively. Then, the total processing times are $X_p = \sum_{k=1}^{s_p} Z_p^k$ and $X_q = \sum_{k=1}^{s_q} Z_q^k$, and condition in (8) is satisfied.

Remark 7. We have seen that the linear utility function has some nice properties when the processing times are random. The situation for the exponential utility function is different. In general, we cannot replace the expected quantities in $\alpha_i \mathbb{E}(e^{\gamma X_i})/(1 - \mathbb{E}(e^{\gamma X_i}))$ by deterministic functions of the object sizes except in specific cases. Even when the processing capacity μ is a deterministic constant, we cannot simply replace the processing times in (3) by the object sizes and hope the sequencing rule still derives an optimal schedule.

We will discuss a particular stochastic case with the exponential utility function where the object sizes determine the optimal schedule. Suppose the parameters for the exponential utility function $v_i(t) = \alpha_i e^{\gamma t} + \beta_i$ are such that $\gamma > 0$ and $\alpha_i < 0$ for $i = 1, 2, ..., n$. Let us look at two concepts of stochastic ordering between two random variables.

Definition 5.7 *A random variable X_p is said to be* ***stochastically larger*** *than a random variable X_q if*

$$P(X_p > t) \geq P(X_q > t) \quad \textit{for every } t > 0$$

We denote the relation by $X_p \geq_{st} X_q$.

Definition 5.8 *A random variable X_p with distribution F_p is said to be larger than a random variable X_q with distribution F_q* ***in the increasing convex sense*** *if*

$$\int_0^\infty h(t) dF_p(t) \geq \int_0^\infty h(t) dF_q(t)$$

for all increasing convex functions h.

If X_p is larger than X_q in the increasing convex sense, we denote this relation by $X_p \geq_{icx} X_q$. It can be shown that

$$X_p \geq_{st} X_q \text{ implies } X_p \geq_{icx} X_q \tag{9}$$

When $s_p > s_q$, the object processing model discussed above implies $X_p \geq_{st} X_q$. Since the function $e^{\gamma t}$ is increasing and convex when $\gamma > 0$, $X_p \geq_{st} X_q$ implies $\mathbb{E}e^{\gamma X_p} \geq \mathbb{E}e^{\gamma X_q}$. The function $g(t) = t/(1-t)$ is increasing for $t > 1$. Hence, $g(\mathbb{E}e^{\gamma X_p}) \geq g(\mathbb{E}e^{\gamma X_q})$. We see that when the utility function is $\alpha_i e^{\gamma t} + \beta_i$ with $\alpha_i < 0$ and $\gamma > 0$, for all $i = 1, 2, ..., n$, and when $s_p > s_q$ implies $\alpha_p \geq \alpha_q$, we have

$$\alpha_p g(\mathbb{E}e^{\gamma X_p}) \leq \alpha_q g(\mathbb{E}e^{\gamma X_q})$$

Since the optimal sequencing in this case is the decreasing order of $\alpha_i g(\mathbb{E}e^{\gamma X_i}) = \alpha_i \mathbb{E}e^{\gamma X_i}/(1 - \mathbb{E}e^{\gamma X_i})$, we see that sequencing the n objects in increasing order of their sizes actually yields the optimal schedule.

5.3 Linear Utility Function and Time-Varying Bandwidth

Suppose the bandwidth is deterministic but time varying and piece-wise continuous, denoted by $\mu(t)$, and suppose the size of the n objects are fixed. It is still true that, for a regular utility function, there exists an optimal schedule that is (i) work-conserving and (ii) non-preemptive. (iii) The optimal sequence in general depends on the bandwidth trajectory. For many utility functions, the scheduling problem becomes very hard. (iv) For linear utility functions with identical slope, an optimal schedule is to order the objects by their sizes in increasing order, which is independent of the bandwidth trajectory. We will give justifications for these claims.

Suppose that in an optimal schedule, the server has a period of inactivity from time t_1 to t_2. Then, eliminating the inactive gap by moving all objects scheduled after time t_2 ahead does not decrease the utility, since the utility function is non-increasing in the reception times. This proves (i).

Now, suppose π is a work-conserving optimal schedule, in which object i is preempted by other objects before its completion. Let t_f be the completion time of object i in π. Let s_i be the size of object i and define

$$t^* = \sup\{t : \int_t^{t_f} \mu(\tau) d\tau = s_i\}$$

Let π^* be the new schedule in which object i starts service at time t^* and ends at t^f. Complete or partial objects that preempt i in π are moved ahead without changing their relative orders, filling the void left by object i. The new schedule π^* is at least as good as π, and hence is also optimal. This procedure can be repeated for each object to get an optimal non-preemptive schedule, and hence, (ii).

To show (iii), consider a case with two objects. Let the size $s_1 = 5$ and $s_2 = 10$. The utility function for each object is defined as follows.

$$v_1(t) = \begin{cases} 10 & \text{if } 0 \leq t \leq 3 \\ 0 & \text{otherwise} \end{cases}$$

$$v_2(t) = 20 - 2t$$

Let the schedule $\pi^1 = \{1, 2\}$ and $\pi^2 = \{2, 1\}$. Consider a bandwidth trajectory $\mu^1(t) = 5$. The total utility gained from each schedule is: $g(\pi^1) = 24$ and $g(\pi^2) = 26$. Hence, π^2 is the optimal sequence. Consider another bandwidth trajectory $\mu^2(t) = 2.5$. In this case, $g(\pi^1) = 18$ and $g(\pi^2) = 12$. Hence, π^1 is the optimal sequence.

Now suppose all objects have a linear utility function of the form $v_i(t) = \beta_i - \alpha_i t$, where $\alpha_i > 0$ and $\beta_i \geq 0$. Given a fixed bandwidth trajectory $\mu(t)$ and an optimal sequence π, let us suppose object i and j are adjacent in π and i proceeds j. Let i starts service at time t_1, ends service at t_2 and let j ends service at t_3. Let π' be a sequence with object j and i interchanged. In π', object j starts service at t_1 and ends service at t_2', and object i ends service at t_3. Since π is optimal, we must have,

$$\alpha_i t_2 + \alpha_j t_3 \leq \alpha_j t_2' + \alpha_i t_3$$

which yields,

$$\frac{t_3 - t_2'}{\alpha_i} \leq \frac{t_3 - t_2}{\alpha_j} \tag{10}$$

Let us denote $p_i(t)$ as the service time for object i when it ends service at time t. We then have $p_i(t_3) = t_3 - t_2'$ and $p_j(t_3) = t_3 - t_2$. Suppose all α_i's are identical. Then, (10) becomes

$$p_i(t_3) \leq p_j(t_3) \tag{11}$$

Since at any time t, $s_i \leq s_j$ if and only if $p_i(t) \leq p_j(t)$, it suffices to sequence the n objects according to the increasing order of their sizes. This demonstrates (iv).

When α_i's are not identical, optimal sequencing seems to be a very difficult problem. The condition in (10) is only necessary but not sufficient for optimality. The following algorithm can find us a "locally" optimal sequence in the sense that the resulting sequence cannot be improved by exchanging positions of two neighboring objects.

Let t_n be the completion time for the entire transfer session. The last object to be transmitted, denoted by $\pi(n)$, should have the largest value of $p_i(t_n)/\alpha_i$ of all objects. Suppose we have determined the last k objects to be transmitted, $\pi(n-k+1), \pi(n-k+2), ..., \pi(n-1), \pi(n)$. Let t_{n-k+1} be the completion time of object $\pi(n-k+1)$. Then, $t_{n-k} = t_{n-k+1} - p_{\pi(n-k+1)}(t_{n-k+1})$ is the starting time of object $\pi(n-k+1)$. Then, the $(n-k)^{th}$ object, $\pi(n-k)$, should have the largest value $p_i(t_{n-k})/\alpha_i$ among the remaining objects yet to be scheduled. The total running time is $O(n^2)$ for this algorithm, where n is the number objects.

In reality, the complete bandwidth trajectory may not be known ahead of time. In that case, one can modify the above algorithm as follows. After finishing transmission of an object at time t, the object with the smallest value of $\frac{s_i/\mu(t)}{\alpha_i}$ among all remaining objects is chosen for the next transmission, where $\mu(t)$ is the bandwidth at time t. The modified algorithm also applies when the bandwidth is random but with unknown statistics.

6 Class-Based Utility Function Assignment

As we have seen that many optimal sequencing problems are very difficult. When the problem is strongly NP-hard, finding approximations that guarantee certain performance can also be difficult. In this section, we consider the situation where the n objects can be grouped into a small number of classes. This formulation may either faithfully reflected the reality, or may be regarded as a systematic approximation to reality. In either case, we hope for simpler solution coming out of this formulation. We will use function V as a canonical example, since it is both very versatile and very hard. We will show that, indeed, there exist algorithms which have polynomial complexity in n, the number of objects.

A group of objects with identical d_i, α_i, β_i and p_i are considered as one object class. Suppose there are total K classes, from 1 to K. From now on, let subscript indices denote classes. Let n_i be the number of objects for class i, $1 \leq i \leq K$. Without loss of generality, let $d_1 < d_2 < ... < d_K$. These K deadlines divide $[0, \infty)$ into $K+1$ intervals or semi-intervals. Let us index interval $(d_{j-1}, d_j]$ the j^{th} interval, $1 \leq j \leq K$, where $d_0 = 0$. Let us call (d_K, ∞) the $(K+1)^{th}$ interval (See figure 5 for an example).

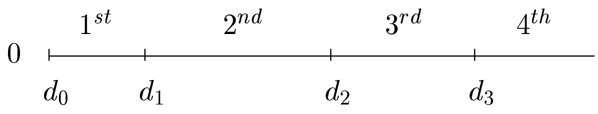


Figure 5: Half Real Line $[0, \infty)$, $K = 3$

Let the non-negative integer, n_i^j, be the number of class i objects that are completed in the j^{th} interval. It must be true that $\sum_{j=1}^{K+1} n_i^j = n_i$, $1 \leq i \leq K$. The problem is, therefore, to determine a feasible set of $\{n_i^j; 1 \leq i \leq K, 1 \leq j \leq K+1\}$ so that the resulting sequence is optimal. Feasibility means that the sequence indeed has the property that n_i^j class i objects are completed in the j^{th} interval, for all i and j.

After time d_k, the value of any class i object becomes zero, where $i < k$. In an optimal schedule, if any class i object, $i < k$, is serviced on the k^{th} interval, it can be moved after all class l objects on the interval, where $l \geq k$, without affecting the optimality. We also know that, on the k^{th} interval, objects from classes $l \geq k$ should be in increasing order of $p_l/|\alpha_l|$. Therefore, on each interval k, if we know the set of number n_i^k, $1 \leq i \leq K$, the order of these $\sum_{i=1}^{K} n_i^k$ objects can be made unambiguous. Therefore, given the set $\{n_i^j; 1 \leq i \leq K, 1 \leq j \leq K+1\}$, the order of the n objects can be made unambiguous. The feasibility of this set of numbers can be checked after the objects are ordered. Let $\underline{\underline{n}}$ denote the matrix $(n_i^j), 1 \leq i \leq K, 1 \leq j \leq K+1$, and let the resulting value from the schedule corresponding to a feasible $\underline{\underline{n}}$ be $v(\underline{\underline{n}})$. The problem is to find a feasible $\underline{\underline{n}}$ such that $v(\underline{\underline{n}})$ is maximized. This can be accomplished by enumerating all possible $\underline{\underline{n}}$'s.

We can compute the number of possible matrices $\underline{\underline{n}}$. Let us first focus on class i. When $\sum_{j=1}^{K+1} n_i^j = n_i$, and each n_i^j is a non-negative integer, the total number of distinct vectors $(n_i^1, n_i^2, ..., n_i^{K+1})$ is $M_K(i) = (n_i+1)^K/2 + (n_i+1)/2$. This can be shown inductively. First, When $K = 1$, it is clear that the vector has the form $(k, n_i - k)$, $0 \leq k \leq n_i$. Hence,

$M_1(i) = n_i + 1$. Suppose for $K = m$, $M_m(i) = (n_i+1)^m/2 + (n_i+1)/2$. Then, when $K = m+1$, n_i^{m+1} can take values from 0 to n_i. Therefore,

$$\begin{aligned}
M_{m+1}(i) &= (\frac{(n_i+1)^m}{2} + \frac{n_i+1}{2}) + (\frac{(n_i+1)^m}{2} + \frac{n_i+1}{2} - 1) \\
&\quad + ... + (\frac{(n_i+1)^m}{2} + \frac{n_i+1}{2} - n_i) \\
&= \frac{(n_i+1)^{m+1}}{2} + \frac{n_i+1}{2}
\end{aligned}$$

Hence, the total number of possible matrices $\underline{\underline{n}}$ is

$$\begin{aligned}
\Pi_{i=1}^K M_K(i) &= \Pi_{i=1}^K (\frac{(n_i+1)^K}{2} + \frac{n_i+1}{2}) \\
&\leq \Pi_{i=1}^K (n_i+1)^K \\
&\leq (\frac{n+K}{K})^{K^2} \\
&\sim \frac{n^{K^2}}{K^{K^2}} \qquad \text{as } n \longrightarrow \infty
\end{aligned}$$

Therefore, the algorithm for finding an optimal sequence is $O(n^{K^2}/K^{K^2})$, which is definitely polynomial in n. More careful counting shows that the power to n can be reduced further. Notice that, after time d_k, we do not need to consider class k objects anymore because they all have zero value. Hence, on the $(k+1)^{th}$ interval, i.e., $(d_k, d_{k+1}]$, we need only to determine n_l^{k+1} for $l > k$. Very crudely, the number of matrices, $\underline{\underline{n}}$, to be considered is $O(n^{(K-1)K/2})$. However, with this complexity, the corresponding algorithm is only practical for $K \leq 4$ when n is around 100.

6.1 Case of Continuous Function

Suppose $\alpha_i < 0$ and $\alpha_i d_i + \beta_i = 0$ for all i. In this case, an optimal schedule can be found using the algorithm from section 4.5.2. This algorithm does not take advantage of knowledge about object classes and has a running time $O(n \sum_{i=1}^K n_i p_i)$.

We will present another algorithm that does utilize the class information. Without loss of generality, suppose $p_1/|\alpha_1| \leq p_2/|\alpha_2| \leq ... \leq p_K/|\alpha_K|$. By lemma 4.2, an optimal schedule must be a sequence that starts with m_1 class 1 objects, followed by m_2 class 2 objects, ..., followed by m_K class K objects, all of which are on time, then followed by tardy objects in arbitrary order. We only need to determine the integers m_i's through enumeration, where $0 \leq m_i \leq n_i$, $1 \leq i \leq K$. Since $\sum_{i=1}^K n_i = n$, for $K > 1$, the total number of choices is

$$\begin{aligned}
\Pi_{i=1}^{K-1}(n_i+1) &\leq (\frac{n}{K-1} + 1)^{K-1} \\
&\sim \frac{n^{K-1}}{(K-1)^{K-1}} \qquad \text{as } n \longrightarrow \infty
\end{aligned}$$

where the upper bound above is obtained by standard maximization technique. The resulting algorithm is polynomial in n. For n around 100, the algorithm is practical for $K \leq 5$.

7 Version Selection

In this section, we consider a different model for web object transmission. Suppose each object can be encoded at different resolutions. When a user requests the page, we would like to choose a resolution for each object and a transmission schedule so that the total utility is maximized. To model this situation, let each object i have m versions, with processing time p_i^j and utility function $v_i^j(t)$, $1 \leq j \leq m$.

This problem seems to be very difficult even for linear utility functions (However, we were unable to show it is strongly NP-hard in this case). In the following, we'll look at step functions.

7.1 Step Functions

In this case, for each object $1 \leq i \leq n$ and version $1 \leq j \leq m$, let the utility function be

$$v_i^j(t) = \begin{cases} w_i^j & \text{if } 0 \leq t \leq d_i \\ 0 & \text{otherwise} \end{cases}$$

where $w_i^j > 0$ is the value of the object if it arrives before the deadline d_i. Note that, for each object, the deadline is common for all versions. The objective is to choose a version assignment for each object and the ordering of transmission so that $\sum_{i=1}^n v_i^j(C_i)$ is maximized. It is clear that there exists an optimal schedule in which the objects are ordered by the increasing order of their deadlines. We therefore arrange the objects in that order. Without loss of generality, we assume this order is $1, 2, ..., n$. Version assignment can be made by straightforward extension to Lawler and Moore's algorithm (4.1). Let us define the $(m+1)^{th}$ "version" for each object i, which corresponds to object i being unable to finish before its deadline.

$$v_i^{m+1}(t) = 0$$

$$p_i^{m+1} = 0$$

We will convert the problem into the minimization version so that we can reuse some notation from Algorithm (4.1). For each $1 \leq i \leq n$, without the loss of generality, let us suppose $w_i^m \geq w_i^j$ for all $1 \leq j \leq m$. For each $1 \leq i \leq n$ and each $1 \leq j \leq m$, define

$$\rho_i^j(t) = \begin{cases} w_i^m - w_i^j & \text{if } 0 \leq t \leq d_i \\ +\infty & \text{otherwise} \end{cases}$$

For each $1 \leq i \leq n$, define

$$\rho_i^{m+1}(t) = w_i^m$$

Then, the dynamic programming relation is captured by

$$f(j,t) = \min \left\{ \begin{array}{ll} f(j, t-1) & \\ \rho_j^k(t) + f(j-1, t-p_j^k) & \text{for } k = 1, 2, ..., m+1 \end{array} \right\} \qquad (j = 1, 2, ..., n;\ t \geq 0)$$

A different version of the problem is to maximize $\sum_{i=1}^n v_i^j(C_i)$, subject to the constraint that all objects will be completed before their deadlines. In this case, we simply remove the artificial $(m+1)^{th}$ version from the above algorithm.

8 Preemptive Server Formulations

Formulation 8.1 *Let $v_i(t,x)$ be the utility of completing x unit of object i by time t. Suppose for each t, $v_i(t,x)$ is non-decreasing in x; and for each x, it is non-increasing in t. Let $T = \sum_{i=1}^{n} s_i/\mu$, and $\mathbf{s} = (s_1, s_2, ..., s_n)^T$. Denote $\mathcal{E}$ the class of all continuous arcs $\{x : x_i(t) \quad (0 \le t \le T; i = 1,2,...,n)\}$ possessing piecewise continuous derivatives $\dot{x}_i(t)$ on $(0 \le t \le T)$. Find a schedule, i.e, $x \in \mathcal{E}$, by solving the following problem.*

$$\max_{x \in \mathcal{E}} \frac{1}{T} \int_0^T \sum_{i=1}^{n} v_i(t, x_i(t)) \ dt \tag{12}$$

subject to

$$x_i(t) = 0 \quad \textit{for } i = 1,2,...,n \tag{13}$$

$$x_i(T) = \mathbf{s} \quad \textit{for } i = 1,2,...,n \tag{14}$$

$$\sum_{i=1}^{n} \dot{x}_i(t) \le \mu, \quad \textit{for all } t \in [0,T] \tag{15}$$

$$\dot{x}_i(t) \ge 0 \quad \textit{for } i = 1,2,...,n \textit{ and all } t \in [0,T] \tag{16}$$

At each point of discontinuity, t_d, for $\dot{x}_i(t)$, we define $\dot{x}_i(t_d) = \lim_{t \to t_d+} \dot{x}_i(t)$, so that $\dot{x}_i$ is defined for all $t \ge 0$. We also assume necessary smoothness for the utility functions $v_i(t,x)$ so that the formulated problem here may be solved by the techniques from calculus of variations and optimal control theory.

9 Design Example of Web Object Transmission Schedule and Conclusion

9.1 Linear Utility Function for Web Object Transmission

The discussions in the paper lead to the conclusion that the linear utility function is among good choices of utility functions. Without precedence constraint, the optimal schedule is to sequence the n objects in the increasing order of p_i/α_i. Since p_i is not easily calculated due to the uncertainty in the transmission speed, we propose to sequences the objects in increasing order of s_i/α_i, where s_i is the size of object i. Assuming precedence constraint can be represented by series-parallel digraphs, we can use Lawler's algorithm [9] to find the optimal schedule, with the sizes s_i replacing the processing times p_i. The scheduling algorithm leads to an optimal or near-optimal schedule that maximizes the expected sum of utilities for a wide class of random bandwidth variations, and it is very likely that in reality the bandwidth distributions belong to this class. We summarize the advantages of linear utility function and its associated schedule as follows.

- Since the parameters β_i's are irrelevant for finding the optimal schedule, the algorithm requires only one parameter α_i to be specified for each object.

- Linear utility function is suitable in many situations when the notion of due dates and values of the objects are vague.
- Computational complexity for finding the optimal schedule is $O(n \log n)$ even in complex situations.

The optimal schedule follows *Weighted Shortest Processing Time* first rule, which is an extension of the *Shortest Processing Time* (SPT) scheduling rule. To some degree, WSPT schedule inherits some of the advantages of the *SPT* schedule. Given a fix time, SPT schedule completes more objects than any other scheduler. This is especially beneficial when a page consists of many small objects and one or a few large objects, because the large objects are pushed to the end of the transmission sequence and most objects will arrive during the early period of the transfer session. SPT schedule is also friendly to fine-grained encoding of document. For instance, in the case of multi-resolution encoding, the size of the lower resolution object is small, and can arrive early and be displayed quickly. It possible that, in many situations, most of the value is delivered to the user at this point. There can be a valuable trade-off in sending some smaller objects first at the expense of slight increase in the delay of larger objects. On the other hand, in the case when larger objects are much more valuable to the user and when the increase in the delay of larger objects becomes significant, WSPT schedule can move larger objects ahead of smaller objects.

9.2 General Conclusion

In this paper, we study a collection of scheduling problems motivated by the desire to arrange the downloading order of web objects. We formulated these problems as to optimize combined utilities received by the user. This formulation is somewhat different from traditional single machine scheduling formulation. As a result, new scheduling problems arise. We hope the collection of solutions we provide can have a wider range of applications of similar nature.

We stress that thinking about objects as having some utility to the user is very natural. In this paper, we do not address how the utility functions are obtained. Presumably, they can be either be measured, or directly assigned by the user. The user either can view utility functions from our paper as approximations to the true utility functions, or, a priori, he is restricted to choose from one of these families but can choose function parameters freely.

Finally, all our utility functions are non-increasing with time, and possibly have one or two discontinuities (or discontinuities in the first derivatives), which can be interpreted as deadlines. A salient thread in this paper is that the solutions are often based on the simple building blocks of linear or exponential functions with Lawler and Moore's algorithm (4.1) to handle the deadlines.

Appendix A: Relevant Results from Complexity Theory

The theory introduced in this section is concerned with the computational complexity of the optimal sequencing problem. Most of the results can be found in [3]. We begin with some definitions. Given a problem Π and one of its instances I, let $R^*(I)$ be the value of the optimal solution and $R(I)$ be the value of the solution generated by an approximation algorithm.

Definition 9.1 *[15] An algorithm will be said to be an* ***ϵ-approximation algorithm*** *for a problem Π if $|(R^*(I) - R(I))/R(I)| \leq \epsilon$ for all instances I, where $0 < \epsilon < 1$ if Π is a maximization problem and $\epsilon > 0$ if Π is a minimization problem.*

Definition 9.2 *A family of ϵ-approximation algorithm, one for each ϵ, is said to be a* ***fully polynomial time approximation scheme*** *if the time complexity is polynomial in both the problem size and in $1/\epsilon$.*

Loosely speaking, being a fully polynomial approximation scheme ensures the algorithm to be practical for very good approximations.

For an instance I of the problem Π, let $\text{length}(I)$ be the size of the instance and $\max(I)$ be an upper bound on the magnitude of each of the data.

Definition 9.3 *A* ***pseudo-polynomial algorithm*** *for Π is one whose time complexity is bounded above by a polynomial function in both length(I) and $\max(I)$.*

Hence, if data values of all instances of a problem Π is bounded, a pseudo-polynomial algorithm appears to have a time complexity polynomial in the size of the instance.

For any polynomial p over integers, let Π_p be the subproblem of Π obtained by restricting Π to only those instances I that satisfy $\max(I) \leq p(\text{length}(I))$. If Π can be solved by a pseudo-polynomial time algorithm, then Π_p can be solved by a polynomial time algorithm.

Definition 9.4 *An optimization problem is* ***NP-hard in the strong sense*** *if there exists a polynomial p over integers for which Π_p is NP-hard.*

When a problem is NP-hard, our first priority is to find either a pseudo-polynomial time algorithm or a fully polynomial time approximation scheme, where the latter is typically derived from the former. We typically want to find out first if the problem Π is NP-hard in the strong sense due to the following fact, which is a direct consequence of the definition of *NP-hard in the strong sense.*

Fact 9.1 *If $P \neq NP$, then Π cannot be solved in pseudo-polynomial time if it is NP-hard in the strong sense.*

The relationship between pseudo-polynomial algorithm and fully polynomial time approximation scheme is captured by the following theorem (See page 140 of [3]). Let us assume the problem Π has integer solution values.

Theorem 9.2 *If there exists a two-variable polynomial q such that for all instances of I, $R^*(I) < q(length(I), max(I))$, then the existence of a fully polynomial time approximation scheme for Π implies the existence of a pseudo-polynomial time optimization algorithm for Π.*

Because of Fact 9.1, we have the following result as a corollary of Theorem 9.2.

Corollary 9.3 *Let Π be an integer-valued optimization problem satisfying the hypothesis of Theorem 9.2. If Π is NP-hard in the strong sense, then Π cannot be solved by a fully polynomial approximation scheme unless $P = NP$.*

The requirement $R^*(I) < q(length(I), max(I))$ roughly says the value for the optimal solution is not very large, which is satisfied by many scheduling problems. A problem's being NP-hard in the strong sense rules out the existence of a pseudo-polynomial time solution, and very often, also rules out the existence of a fully polynomial approximation scheme.

Acknowledgement

The authors would like to thank David Starobinski for helpful suggestions and for raising the version selection problem.

References

[1] Kenneth R. Baker. *Introduction to Sequencing and Scheduling.* John Wiley & Sons, 1974.

[2] Simon French. *Sequencing and Scheduling - An Introduction to the Mathematics of the Job-Shop.* John Wiley & Sons, 1982.

[3] Michael R. Garey and David S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[4] G.V. Gens and E.V. Levner. Fast Approximation Algorithm for Job Sequencing with Deadlines. *Discrete Applied Mathematics, 3*, pages 313–318, 1981.

[5] Rajarshi Gupta. WebTP: A User-Centric Receiver-Driven Web Transport Protocol. Master's thesis, University of California, Berkeley, 1998.

[6] J.Du and J.Y.-T. Leung. Minimizing Total Tardiness on One Machine is NP-Hard. *Math. Oper. Res.*, 1989.

[7] Richard M. Karp. Reducibility among Combinatorial Problems. In *Complexity of Computer Computation*, pages 85–103. Plenum Press, 1972.

[8] E.L. Lawler. A Pseudo-Polynomial Algorithm for Sequencing Jobs to Minimize Total Tardiness. *Annals of Discrete Mathematics, 1*, pages 331–342, 1977.

[9] E.L. Lawler. Sequencing Jobs to Minimize Total Weighted Completion Time Subject to Precedence Constraints. *Annals of Discrete Mathematics, 2*, pages 75–90, 1978.

[10] E.L. Lawler and J. M. Moore. A Functional Equation and Its Application to Resource allocation and Sequencing Problems. *Management Science*, pages 77–85, 1969.

[11] J.K. Lenstra and A.H.G. Rinnooy Kan. Complexity of Scheduling under Precedence Constraints. *Annals of Discrete Mathematics*, pages 22–35, 1978.

[12] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of Machine Scheduling Problems. *Annals of Discrete Mathematics, 1*, pages 343–362, 1977.

[13] Clyde L. Monma and Jeffrey B. Sidney. Optimal Sequencing via Modular Decomposition: Characterization of Sequencing Functions. *Mathematics of Operations Research, Vol. 12*, pages 22–31, 1987.

[14] Michael Pinedo. *Scheduling - Theory, Algorithms and Systems.* Prentice-Hall, Inc., 1995.

[15] Sartaj K. Sahni. Algorithms for Scheduling Independent Tasks. *Journal of the Association for Computing Machinery, Vol. 23, No. 1*, pages 116–127, 1976.

[16] J.B. Sidney. Decomposition Algorithms for Single-Machine Sequencing with Precedence Relations and Deferral Costs. *Operations Research, 22*, pages 283–298, 1975.

[17] Jeffrey B. Sidney and George Steiner. Optimal Sequencing by Modular Decomposition: Polynomial Algorithms. *Operations Research, Vol. 34*, pages 606–612, 1985.

[18] Dileep R. Sule. *Industrial Scheduling.* PWS Pub. Co., 1997.

[19] V.S. Tanaev, Y.N. Scotskov, and V.A. Strusevich. *Scheduling Theory - Single-Stage Systems.* Kluwer Academic Publishers, 1989.

[20] Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The Recognition of Series Parallel Digraphs. *SIAM Journal on Computing*, pages 298–313, 1982.

[21] J. Yuan. NP Hardness of the Single Machine Common Due Date Weighted Tardiness Problem. *System Sci. Math. Studies*, 5:328–333, 1992.