Journal of The Korea Society of Computer and Information
Vol. 21 No. 9, pp. 91-100, September 2016
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2016.21.9.091

A Dynamic Programming Approach for Emergency Vehicle Dispatching Problems

Jae Young Choi*, Heung-Kyu Kim**

Abstract

In this research, emergency vehicle dispatching problems faced with in the wake of massive natural disasters are considered. Here, the emergency vehicle dispatching problems can be regarded as a single machine stochastic scheduling problems, where the processing times are independently and identically distributed random variables, are considered. The objective of minimizing the expected number of tardy jobs, with distinct job due dates that are independently and arbitrarily distributed random variables, is dealt with. For these problems, optimal static-list policies can be found by solving corresponding assignment problems. However, for the special cases where due dates are exponentially distributed random variables, using a proposed dynamic programming approach is found to be relatively faster than solving the corresponding assignment problems. This so-called Pivot Dynamic Programming approach exploits necessary optimality conditions derived for ordering the jobs partially.

▸ Keyword : Emergency Vehicle Dispatching, Dynamic Programming, Stochastic Scheduling, Tardy Job

Ⅰ. Introduction

This research deals with emergency vehicle dispatching problems faced with in the wake of massive natural disasters. In massive disasters, the degree of injury varies depending on the emergency situation. Some injuries require immediate medical treatment, some need only minor treatment. Hence, First Come First Served (FCFS) or prioritized FCFS, which are commonly used rules, may not be useful because there will be many severe injuries in a short time period.

In emergency vehicle dispatching, both severity of injuries and response times to the emergency scene must be considered. A good emergency vehicle dispatching policy should have the following characteristics.

The policy should be optimal or near optimal in terms system such as the total waiting time.

Determining the order of dispatching should be fast since it is not acceptable to lose lives because of slow dispatching decisions, even though they are optimal.

The policy must be able to be adjusted according to the dynamic situation of emergency and the dynamics of situational information.

To simplify the problem, this research focuses on single-machine cases. We envision that the area of interest can be clustered into several clusters small enough for a single vehicle to serving a uniquely assigned cluster. The scheduling aspects of out research is used to determine the order of service by the unique vehicle to the casualties within such a cluster.

The approach to emergency vehicle dispatching in this

• First Author: Jae Young Choi, Corresponding Author: Heung-Kyu Kim
*Jae Young Choi (mobilechoi@ajou.ac.kr), Associate Professor, Department of Software, Ajou University.
**Heung-Kyu Kim (heungkyu@dankook.ac.kr), Professor, School of Business Administration, Dankook University.
• Received: 2016. 06. 30, Revised: 2016. 08. 10, Accepted: 2016. 08. 29.

research focuses on stochastic scheduling problems. Each injury constitutes a job and his unknown time until death unless treated is represented by a stochastic due date. The sum of the travel time to and from an injury and service time at the scene is expressed as a stochastic processing time. Injuries waiting for treatment are a set of jobs. To be more specific, this research considers scheduling problems with independent and identically distributed processing times. This case is useful when a large area of interest is clustered into several districts that are small enough such that travel times are not so much different.

Stochastic scheduling problems for minimizing the expected number of tardy jobs have been investigated in Balut [1], Boxma and Forst [3], Cai and Zhou [4], De et al. [6], Emmons and Pinedo [7], Jang [9], Pinedo [15], and Sarin et al. [17]. Two particular problems of stochastic scheduling with i.i.d. processing times and distinct due dates are considered in Boxma and Forst [3]. One problem has exponential processing times and deterministic due dates. The other problem has arbitrary processing times and exponential due dates as it is assumed here. Sufficient optimality conditions for both problems are provided therein. As noted in Boxma and Forst [3], there exist many optimal solutions that do not satisfy sufficient optimality conditions.

This research considers the cases where processing times have arbitrary i.i.d. distributions and due dates have arbitrary distributions. The optimal static list policies of these cases are found by solving corresponding assignment problems as shown in Emmons and Pinedo [7]. This paper focuses on the special cases where due dates are exponentially distributed. Then several necessary conditions of optimal solutions are derived and, using them, a dynamic programming algorithm is proposed. Using the proposed approach is found to be relatively faster than solving the corresponding assignment problems.

The organization of the paper is as follows. Section 2 describes the problem considered in this paper. Section 3 investigates several necessary conditions of optimal solutions. Section 4 uses these theoretical results to develop a dynamic programming solution procedure. Computational results are shown in Section 5. Final remarks are addressed in Section 6.

II. Probem Description

Let us consider a set of n jobs with arbitrary i.i.d. processing times, where processing time of each job i, $i=1,...,n$, is an independent nonnegative random variable $X_i(=X)$, and due date is an independent nonnegative random variable D_i. In addition, a weight w_i is associated with job i. Here the objective is to minimize the expected weighted number of tardy jobs. Given a sequence of jobs π, the expected weighted number of tardy jobs $ET(\pi)$ is defined by

$$ET(\pi)=\sum_{i=1}^{n} w_{\pi(i)}\Pr\{D_{\pi(i)} < C_{\pi(i)}\}$$

where $\pi(i)$ is the job at position i and $C_{\pi(i)}$ is the completion time of job $\pi(i)$ in sequence π, i.e., $C_{\pi(i)} = \sum_{k=1}^{i} X_{\pi(k)}$.

As long as processing times are i.i.d., the problem can be represented as an assignment problem even if due dates are arbitrarily distributed. For details, see Emmons and Pinedo [7]. Given a sequence of jobs λ, the kth job of the sequence gets tardy with probability $\Pr\left\{\sum_{i=1}^{k} X_{\lambda(i)} > D_{\lambda(k)}\right\}$. This probability depends only on the position of the job, but neither on its previous nor subsequent jobs.

Theorem 1

Suppose job i has an arbitrary i.i.d. processing time X_i and arbitrary independent due date D_i. Then the optimal static list policy is obtained by solving an assignment problem where the cost coefficients are defined as $c_{ij} = w_i\Pr\left\{\sum_{i=1}^{j} X_i > D_i\right\}$, where c_{ij} is the cost when job i is positioned at the jth position in a sequence.

To apply the above theorem, it is required to compute the probability that a job is tardy. If due dates are exponentially distributed, the objective function is simplified since the probability of being tardy has a product form. Let D_i be exponentially distributed with

mean value $\frac{1}{\delta_i}$. Most injuries can be treated rather quickly, but only occasional injuries will be treated long. An exponential distribution seems quite plausible for this type of service situation. The probability of the kth job in a sequence λ being early is $\Pr\left\{\sum_{i=1}^{k} X_i < D_{\lambda(k)}\right\}$, which is $\Pr\{X_1 < D_{\lambda(k)}\}^k$, which, in turn, is $E\left[e^{-\delta_{\lambda(k)}X_1}\right]^k$ for any mutually independent nonnegative random variables X_i, $i=1,...,n$. Therefore, the objective function can be expressed by

$$\begin{aligned} ET(\lambda) &= \sum_{i=1}^{n} w_{\lambda(i)} \Pr\{C_{\lambda(i)} > D_{\lambda(i)}\} \qquad (1)\\ &= \sum_{i=1}^{n} w_{\lambda(i)}\left(1-\Pr\left\{\sum_{k=1}^{i} X_i < D_{\lambda(i)}\right\}\right)\\ &= \sum_{i=1}^{n} w_{\lambda(i)}\left(1-E\left[e^{-\delta_{\lambda(i)}X_1}\right]^i\right) \end{aligned}$$

Note that $E\left[e^{-\delta_{\lambda(i)}X_1}\right]$ is the value of the Laplace-Stieltjes Transform (LST) of the random variable X_1 evaluated at $\delta_{\lambda(i)}$. For Laplace-Stieltjes Transform, see Widder[19]. Let $f_i = E\left[e^{-\delta_i X_1}\right]$. Then, eq. (1) becomes

$$\sum_{i=1}^{n} w_{\lambda(i)}\left(1-f_{\lambda(i)}^i\right) \qquad (2)$$

Clearly, minimizing $ET(\lambda)$ is maximizing the expected weighted number of early jobs $EE(\lambda)$

$$EE(\lambda) = \sum_{i=1}^{n} w_{\lambda(i)} f_{\lambda(i)}^i \qquad (3)$$

Now the problem becomes to find a sequence that maximizes eq. (3).

III. Theoretical Development

In Emmons and Pinedo [7], it is shown that EDD(Earliest Due Date) schedule is optimal to the class of preemptive dynamic policy when processing times are exponential i.i.d. and due dates are stochastically ordered as $D_1 \le_f D_2 \le_f \cdots \le_f D_n$, where $\le_f$ means 'stochastically less than' in failure rate sense. However, when a static policy is considered, EDD schedule is not always optimal.

Suppose there is a single machine with two jobs each of which has an i.i.d. exponential processing time with mean $\frac{1}{\lambda}$ and exponential due date D_i with mean $\frac{1}{\delta_i}$. In addition, suppose $\delta_1 \ge \delta_2$ and $w_1 = w_2 = 1$. Obviously $D_1 \le_f D_2$. There are only two possible sequences, (1, 2) and (2, 1). Here the sequence (1, 2) is an EDD sequence.

Let us compare $EE((1,2))$ and $EE((2,1))$.

$$\begin{aligned} &EE((1,2)) - EE((2,1))\\ &= \frac{\lambda}{\lambda+\delta_1} + \left[\frac{\lambda}{\lambda+\delta_2}\right]^2 - \frac{\lambda}{\lambda+\delta_2} - \left[\frac{\lambda}{\lambda+\delta_1}\right]^2\\ &= \left[\frac{1}{(\lambda+\delta_1)(\lambda+\delta_2)}\right]^2 \lambda(\delta_1-\delta_2)\left(\lambda^2-\delta_1\delta_2\right) \end{aligned}$$

Clearly from the above equation, the optimality of EDD schedule is guaranteed only when $\lambda^2 > \delta_1\delta_2$.

If there are two arbitrary i.i.d. jobs to be scheduled, there exist only two possible sequences, (1, 2) and (2, 1). The difference between EEs of sequence (1, 2) and sequence (2, 1) is

$$EE((1,2)) - EE((2,1)) = f_1(1-f_1) - f_2(1-f_2)$$

where $f_i = \Pr\{X < D_i\} = E\left[e^{-\delta_i X}\right]$.

Suppose $|f_1 - 0.5| \le |f_2 - 0.5|$. Then this easily leads us to $f_1(1-f_1) \ge f_2(1-f_2)$. Therefore the sequence (1, 2) is optimal.

DOMINANCE RULES

It is very useful to know whether a job precedes another job in optimal solutions. This knowledge provides dominance rules that identify a set of solutions as non-promising, i.e., non-optimal solutions. Those rules generate partial sequences which can significantly reduce the computational efforts. Let us consider the gain coming from interchanging two jobs in a sequence. Suppose sth job and tth job in a given sequence λ, where $s < t$, is interchanged. Let us denote the resulting sequence by

λ^*. Then the gain of the interchange can be expressed as follows when all the weights are assumed to be equal to 1.

$$
\begin{aligned}
&EE(\lambda)-EE(\lambda^*)\\
&=f_{\lambda(1)}+\ldots+f^s_{\lambda(s)}+\ldots+f^t_{\lambda(t)}+\ldots+f^n_{\lambda(n)}\\
&\quad-\left(f_{\lambda(1)}+\ldots+f^s_{\lambda(t)}+\ldots+f^t_{\lambda(s)}+\ldots+f^n_{\lambda(n)}\right)\\
&=f^s_{\lambda(s)}+f^t_{\lambda(t)}-f^s_{\lambda(t)}-f^t_{\lambda(s)}\\
&=f^s_{\lambda(s)}\left(1-f^{t-s}_{\lambda(s)}\right)-f^s_{\lambda(t)}\left(1-f^{t-s}_{\lambda(t)}\right)\\
&=f^s_{\lambda(s)}(1-f_{\lambda(s)})f^{s-1}_{\lambda(s)}\sum_{k=0}^{t-s-1}f^k_{\lambda(s)}\\
&\quad-f^s_{\lambda(t)}(1-f_{\lambda(t)})f^{s-1}_{\lambda(s)}\sum_{k=0}^{t-s-1}f^k_{\lambda(t)}\\
&=f_{\lambda(s)}(1-f_{\lambda(s)})f^{s-1}_{\lambda(s)}\sum_{k=0}^{t-s-1}f^k_{\lambda(s)}\\
&\quad-f_{\lambda(s)}(1-f_{\lambda(t)})f^{s-1}_{\lambda(t)}\sum_{k=0}^{t-s-1}f^k_{\lambda(t)}
\end{aligned}
$$

If the gain is negative, the jobs need to be interchanged. From the last equality, it is clear that if $f_{\lambda(s)} \geq f_{\lambda(t)}$, then $f^{s-1}_{\lambda(s)}\sum_{k=0}^{t-s-1}f^k_{\lambda(s)} \geq f^{s-1}_{\lambda(t)}\sum_{k=0}^{t-s-1}f^k_{\lambda(t)}$. It is time to show important optimal partial sequencing rules for jobs with certain characteristics.

Lemma 1

In optimal sequences, job i precedes job j if $f_i \geq f_j$ and $|f_i-0.5| \leq |f_j-0.5|$.

Proof

Suppose there is a sequence λ in which job i precedes job j. Also job i and j are at the sth and tth position, respectively. That is, $\lambda(s)=i$ and $\lambda(t)=j$. Since $f_i \geq f_j$, it follows $f_i^{s-1}\sum_{k=0}^{t-s-1}f_i^k \geq f_j^{s-1}\sum_{k=0}^{t-s-1}f_j^k$. And also, since $|f_i-0.5| \leq |f_j-0.5|$, it follows $f_i(1-f_i) \geq f_j(1-f_j)$. Therefore, interchanging job i and j yields no benefit. Consequently, for any sequence, one can improve the expected number of early jobs by interchanging jobs that satisfy the condition but violate Lemma 1. This completes proof.

Lemma 1 is immediately followed by the next two Corollaries.

Corollary 1

If $f_i \leq 0.5$ for all $i=1,\ldots,n$, then it is optimal to process jobs in non-increasing order of f_i.

Corollary 2

Let $A=\{i\in\{1,\ldots,n\}: f_i > 0.5\}$ and $B=\{i\in\{1,\ldots,n\}: f_i \leq 0.5\}$. If $\left|\max_{i\in A} f_i-0.5\right| \leq \left|\max_{i\in B} f_i-0.5\right|$, then all the jobs in A precede all the jobs in B in optimal solutions.

Note that in Corollary 2, it is known that the exact sequence of jobs in set B, which is non-increasing order of f_i's. It is not known, however, the exact sequence of jobs in set A although it is known all the jobs in A precede all the jobs in B.

MONOTONIC PROPERTIES

Lemma 1 provides precedence relationships that need to be preserved regardless of job positions. It is possible to obtain position-dependent precedence relationship of two jobs: depending on the position of one job, the other job may precede or succeed the job.

The gain of interchanging job i at position s and job j at position t is $w_if_i^s+w_jf_j^t-w_if_i^t-w_jf_j^s$. Let $D_{ij}(s)=w_if_i^s-w_jf_j^s$. Then the gain can be redefined as $D_{ij}(s)-D_{ij}(t)$. When $D_{ij}(s)<D_{ij}(t)$, job i and job j should be interchanged. The function $D_{ij}(s)$ has very useful monotonic properties. These properties serve as a basis for a powerful partial ordering scheme, Pivot Rule. The Pivot Rule governs the optimal partial orderings in such a way that all the jobs whose f_i is less than that of the first job (pivot) should be processed in non-increasing order.

The following lemmas jointly show that $D_{ij}(s)$ is monotonically increasing (decreasing) with respect to s up to some $k \geq 1$, and thereafter is monotonically decreasing (increasing) when $f_i > f_j$ ($f_i < f_j$).

Lemma 2

For some $n \geq 2$, if $f_i > f_j$ and $D_{ij}(n-1) > D_{ij}(n)$, then $D_{ij}(m-1) > D_{ij}(m)$ for all $m > n$.

Proof

It suffices to show that if $D_{ij}(n-1) > D_{ij}(n)$, then $D_{ij}(n) > D_{ij}(n+1)$. Suppose $D_{ij}(n-1) > D_{ij}(n)$. Then it follows

$$\Rightarrow w_i f_i^{n-1} - w_j f_j^{n-1} > w_i f_i^n - w_j f_j^n$$
$$\Rightarrow w_i\left(f_i^{n-1} - f_i^n\right) > w_j\left(f_j^{n-1} - f_j^n\right)$$
$$\Rightarrow w_i f_i^{n-1}(1-f_i) > w_j f_j^{n-1}(1-f_j)$$
$$\Rightarrow w_i f_i^n(1-f_i) > w_j f_j^n(1-f_j)$$
$$\Rightarrow w_i f_i^n - w_j f_j^n > w_i f_i^{n+1} - w_j f_j^{n+1}$$
$$\Rightarrow D_{ij}(n) > D_{ij}(n+1)$$

Lemma 3

For some $n \geq 2$, if $f_i > f_j$ and $D_{ij}(m) < D_{ij}(m+1)$, then $D_{ij}(m-1) > D_{ij}(m)$ for all $1 \leq m < n$.

Proof

It also suffices to show that if $D_{ij}(n) < D_{ij}(n+1)$, then $D_{ij}(n-1) < D_{ij}(n)$. Suppose $D_{ij}(n) < D_{ij}(n+1)$. Then it follows

$$\Rightarrow w_i f_i^n - w_j f_j^n < w_i f_i^{n+1} - w_j f_j^{n+1}$$
$$\Rightarrow w_i\left(f_i^n - f_i^{n+1}\right) < w_j\left(f_j^n - f_j^{n+1}\right)$$
$$\Rightarrow w_i f_i^n(1-f_i) < w_j f_j^n(1-f_j)$$
$$\Rightarrow w_i f_i^{n-1}(1-f_i) < w_j f_j^{n-1}(1-f_j)$$
$$\Rightarrow w_i f_i^{n-1} - w_j f_j^{n-1} < w_i f_i^n - w_j f_j^n$$
$$\Rightarrow D_{ij}(n-1) > D_{ij}(n)$$

Lemma 2 states that once $D_{ij}(\cdot)$ decreases, it will decrease thereafter, while Lemma 3 states that once it increases, it has been increasing when $f_i > f_j$.

The Figure 1 shows the typical shape of $D_{ij}(\cdot)$ when $f_i > f_j$. If $f_i < f_j$ and $w_i = w_j$, $D_{ij}(\cdot)$ takes negative values and has a unique minimum point.

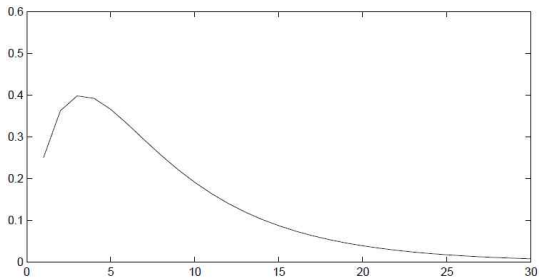

Figure 1. Typical Shape of $D_{ij}(\cdot)$ (when $f_i > f_j$ and $w_i = w_j$)

From monotonic properties of $D_{ij}(\cdot)$'s, it is clear that the function has a unique extreme point. Let us denote the unique position that maximizes(or minimizes) $D_{ij}(\cdot)$ by e_{ij} if $f_i > f_j$ ($f_i < f_j$). Then

$$e_{ij} = \begin{cases} \arg\max_{k=\{1,\ldots,n\}} & \text{if } f_i > f_j \\ \arg\max_{k=\{1,\ldots,n\}} & \text{if } f_i < f_j \end{cases}$$

The position-dependent precedence relationships for a pair of jobs are easily determined by comparing the desired position of a job and the extreme point. The following lemma illustrates such idea.

Lemma 4

Suppose $f_i < f_j$ and job i is to be located at position s. In an optimal solution, the following rules should be satisfied.

1. if $e_{ij} > s$, job i should precede job j.
2. if $e_{ij} < s$, job j should precede job i.
3. if $e_{ij} = s$, job i never be positioned at s.

Proof

It is clear from Figure 2 that, when $e_{ij} > s$, the gain of interchange $D_{ij}(s) - D_{ij}(t)$ is positive only if $t > s$, i.e., job i should precede job j. The similar logic is applied to the case where $e_{ij} < s$. Job i can't be located at e_{ij} since there are no other positions that make the gain positive.

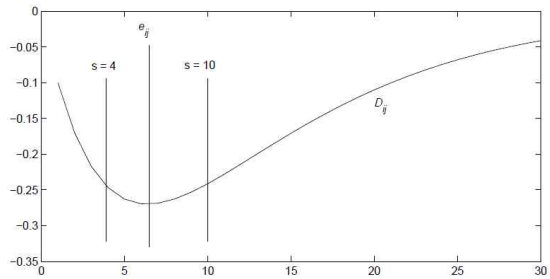

Figure 2. Pictorial Explanation of Lemma 4

Lemma 4 establishes a procedure for obtaining position-dependent precedence relationships for a pair of jobs. The procedure comprises of obtaining the precedence relationships for finding e_{ij}'s and comparing integers. Utilization of this procedure reduces computational burden significantly since it allows

eliminating a huge amount of expensive floating-point operations required for obtaining the position-dependent precedence relationships.

PIVOT RULE

If all the weights are equal, the extreme point of D_{ij} has an additional property: given the position of the extreme point of D_{ij}, one can predict where the extreme points of D_{ik} and D_{kj} are with $f_k \neq f_i$ and $f_k \neq f_j$.

Lemma 5

Suppose $f_k < f_j < f_i$ and $D_{ij}(\cdot\ )$ has its maximum at e_{ij}. Then $D_{ij}(\cdot\ )$ has its maximum at $e_{ik} \leq e_{ij}$.

Proof

Let $e_{ij} = e$ for simplicity. We prove the lemma by contradiction. Suppose $e_{ik} > e$. It implies that

$$f_i^e - f_k^e < f_i^{e+1} - f_k^{e+1} \Rightarrow f_i^e(1-f_i) < f_k^e(1-f_k)$$

By assumption,

$D_{ij}(e) > D_{ij}(e+1) \Rightarrow f_i^e(1-f_i) > f_j^e(1-f_j)$. Hence it follows $f_j^e(1-f_j) < f_i^e(1-f_i) < f_k^e(1-f_k)$.

When $e = 1$, it is impossible to satisfy the above inequality for any value of f_i, f_j, and f_k such that $f_k < f_j < f_i$ since the inequality implies that when $e = 1$, f_k is the closest to 0.5 and f_j is the farthest to 0.5. This contradiction results from assuming $e_{ik} > e$.

The Lemma 5 provides a basis for Lemma 6 which establishes partial ordering scheme that is far more efficient than partial ordering established in Lemma 1. Lemma 6 defines Pivot Rule: all the jobs whose f_i's are less than that of the first job are sequenced in non-increasing order in optimal solutions.

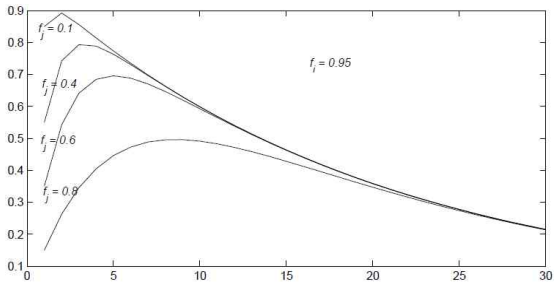

Figure 3. Pictorial Explanation of Lemma 5

Lemma 6

In an optimal solution, the jobs whose $f_{\cdot} \leq f_{\lambda(1)}$ are sequenced in non-increasing order of $f_{\cdot}$.

Proof

Without loss of generality, let $f_1 \leq f_{\lambda(1)}$ for simplicity. Again the lemma is proved by contradiction. Suppose an optimal sequence λ is obtained such that $\lambda(s) = i$ and $\lambda(t) = j, s < t$, and $f_i < f_j < f_1$. That is, a job with smaller f value is scheduled earlier than a job with larger f value.

By optimality assumption,

$$f_i^s - f_j^s > f_i^t - f_j^t$$
$$f_1 - f_i > f_1^s - f_i^s$$
$$f_1 - f_j > f_1^t - f_j^t$$

The first inequality implies $s < e_{ij} = e_{ji} \leq e_{1i}$, where $e_{ji} \leq e_{1i}$ is from Lemma 5. However, the second inequality implies $e_{1i} < s$, which is a contradiction.

Lemma 6 suggests that after obtaining the right pivot, it remains only to sequence jobs with larger f_i value than that of the pivot.

Ⅳ. Pivot Dynamic Programming

This section introduces a new solution method: Pivot Dynamic Programming. An approach shown in Held and Karp [8] is often used for obtaining an optimal sequence in a single machine scheduling problem. The dynamic program in Held and Karp [8] is as follows.

$$v(S) = \max_{i \in S} \{g(i, S) + v(S - \{i\})\}$$

where S is the set of unscheduled jobs, and $g(i, S)$ is the cost associated with scheduling job i. The drawback of this dynamic program is that the solution space grows exponentially with the number of jobs since it requires examining all the possible subsets of the given set of jobs, i.e., the power set.

However, since the dominance rules described in Section 2 reduce the solution space, the optimal value

function for the dynamic program can be formulated as follows.

$$v(S)=\max_{i\in S}\{g(i,S)+v(S-\{i\})\}$$

$$v(\varnothing)=0$$

$$g(i,S)=\begin{cases}-\infty \text{ if } f_i \le 0.5 \text{ and } \\ \qquad f_i > \min\limits_{k\in A_S(i)} f_k \\ -\infty \text{ if } f_i \le 0.5 \text{ and } \\ \qquad \exists\ k\in A_S(i)\ s.t. \\ \qquad |f_i-0.5|<|f_k-0.5| \\ -\infty \text{ if } \exists\ k\in S-\{i\}\ s.t. \\ \qquad D_{ki}(t)-D_{ki}(|S|)<0 \\ \qquad \text{for } \forall\ t, 1\le t<|S| \\ f_i^{|S|} \quad otherwise\end{cases}$$

where $A_S(i)=\{k: k\in S-\{i\} \text{ and } f_k \le 0.5\}$ and $D_{ji}(t)=f_j^t-f_i^t$. Note that, if $D_{ji}(t)-D_{ji}(s)<0$, then the objective value can be improved by swapping the job j at position t and the job i at position s.

Above dynamic program recursively selects the last job in S which should satisfy Lemma 1 and can be preceded by the other jobs.

The third condition of the dynamic programming prohibits job i from being located at the last position since there is a job k which should be scheduled later than job i since swapping job i and k improves the objective value.

It seems computationally expensive to check the third condition. However, the monotonic properties of $D_{ij}(\cdot)$ make this procedure faster.

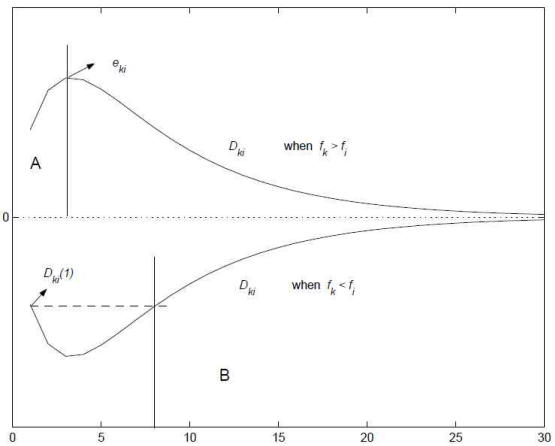

Figure 4: Pictorial Explanation
(third condition of Dynamic Programming)

In the case where $f_k > f_i$ for some $k\in S-\{i\}$, if $e_{ki} > |S|$, then job i should not be the last job in S (See region A in Figure 4). On the other hand, if $f_k < f_i$ and $D_{ki}(1) > D_{ki}(|S|)$ for some $k\in S-\{i\}$, again job i should not be the last job in S (See region B in Figure 4).

Since the dynamic programming formulation in Held and Karp [9] requires enumeration of all the subsets of jobs, it cannot avoid exponentially growing number of computations. To avoid this drawback, a new dynamic programming approach to our problem is proposed.

In this new approach, instead of selecting a job for the last position in a subset as proposed in Held and Karp [8], a position for an unscheduled job with the smallest f_i value in a sequence is recursively selected. So, the state variable of the dynamic programming is a mapping of position to a job $\lambda: k\rightarrow j$ rather than a subset as in Held and Karp [8].

Let $B(\lambda,i,s)$ and $A(\lambda,i,s)$ denote the set of jobs that should precede job i and the set of jobs that should be preceded by job i respectively, given that job i is located at position s in sequence λ. The set $B(\lambda,i,s)$ and $A(\lambda,i,s)$ is determined by applying Lemma 4 to all the unscheduled jobs. Of course, jobs that are already scheduled in λ should be included. For a set of unscheduled jobs $U(\lambda)$, $\lambda^{-1}(i)$ is undefined if $i\in U(\lambda)$. This indicates job i has not been scheduled yet. Let $P(\lambda)=\{s\in\{1,\ldots,n\} \text{ and } \lambda(s)=0\}$ denote a set of positions that are not currently assigned to a job.

It is now ready to present the dynamic programming formulation as follows.

$$v(\lambda)=\begin{cases}\max\limits_{s\in P(\lambda)}\{g(s,\lambda)+v(\lambda\oplus s)\} & \text{if } U(\lambda)\neq\varnothing \\ 0 & otherwise\end{cases}$$

where the binary operation $\lambda\oplus s$ is defined as it gives us a new mapping λ' such that $\lambda(t)=\lambda'(t)$ for $t\neq s$ and $\lambda'(s)=\arg\min\limits_{i\in U(\lambda)} f_i$.

$g(s,\lambda)$ is defined as follows.

$$g(s,\lambda)=\begin{cases}-\infty \text{ if } \lambda(s)\neq 0 \\ -\infty \text{ if } \exists\ k\in U(\lambda)\ s.t. e_{i^*k}=s \\ -\infty \text{ if } \left|B(\lambda,i^*,s)\right|+1\neq s \\ -\infty \text{ if } \left|A(\lambda,i^*,s)\right|+s\neq n \\ f_{i^*}^s \quad otherwise\end{cases}$$

where $i^*=\arg\min\limits_{i\in U(\lambda)} f_i$.

In the above formulation, a position for job i^* is recursively selected. The first condition of $g(s,\lambda)$ is that position s is not selected if it is already occupied by a job. The second condition means that job i^* should not take the position s because it violates the third rule in Lemma 4. The third and fourth condition avoid conflicts with the other jobs when we schedule job i^* at position s.

Choices of the Pivot

A naive approach of the above dynamic programming will recursively select a position for a job with the smallest f_i value. However, Lemma 6 and the Pivot Rule allow avoiding substantial amount of computations. The main problem here is how to make a right choice of the first job, the pivot. To make choices of pivot, the following matrix is computed.

$$k_{ij}=\begin{cases}\min\{k: k\in\{1,\ldots,n\}\text{ and }D_{ij}(k)<D_{ij}(1)\} \\ \quad \text{if } f_i>f_j \\ \max\{k: k\in\{1,\ldots,n\}\text{ and }D_{ij}(k)<D_{ij}(1)\} \\ \quad \text{if } f_i<f_j \\ 0 \quad \text{if for } \forall\ k\in\{1,\ldots,n\}, D_{ij}(k)>D_{ij}(1) \\ \quad \text{or } i=j\end{cases}$$

where k_{ij} is the possible starting (or ending) position of job j when job i takes the first position in a sequence when $f_i>f_j$ $(f_i<f_j)$.

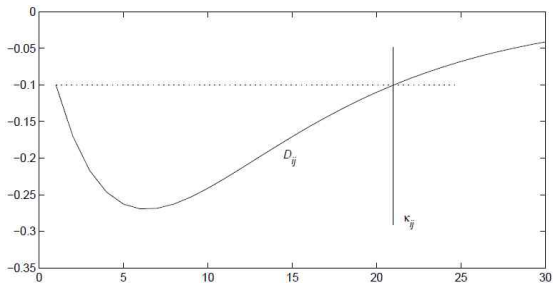

Figure 5. An Example of k_{ij}

The matrix, whose elements are k_{ij}, is denoted by K. Let N_i^- be the set of jobs whose $f_{\cdot}<f_i$ and K_{ij}^+ be a set $\{k: f_k>f_i \text{ and } k_{ik}\le k_{ij}\}$. Here K_{ij}^+ is the set of jobs that should be located on or before k_{ij}.

Proposition 1

Job i can be chosen as a possible pivot candidate if the ith row of K conforms the following rules.

1. $k_{ij}\neq 0$ for all $j\neq i$
2. $n-k_{ij}>\left|N_j^-\right|$ for all j s.t. $f_j<f_i$
3. $k_{ij}>\left|K_{ij}^+\right|$ for all j s.t. $f_j>f_i$

The rule 1 rules out job i if there's a job j such that $D_{ij}(k)>D_{ij}(1)$ for all $k>1$. This means that job j should precede job i at any position of job i, therefore, job i will never be the first job in a sequence. The rule 2 is the application of Pivot Rule. Since all the jobs whose $f_{\cdot}<f_{\pi vot}=f_i$ should be processed in non-increasing order, if $\left|N_j^-\right|>n-k_{ij}$, for a job j, jobs with $f_{\cdot}<f_j$ cannot be placed after job j because the number of jobs with $f_{\cdot}<f_j$ overflows the possible number of positions, $n-k_{ij}$. The rule 3 checks if there's a subset of jobs with $f_{\cdot}>f_i$ which overflows the number of the possible number of positions.

Partitioning Jobs

Once candidates of pivots are determined, the optimal values for each pivot needs to be examined. The number of candidates of pivots is at least 1.

Given a candidate pivot i, a partial sequence which results from the Pivot Rule λ_i' such that $i=\lambda_i'(1)$ and $f_{\lambda_i'(1)}\ge f_{\lambda_i'(2)}\ge\ldots\ge f_{\lambda_i'(n')}$ is obtained where $\lambda_i'(k)$ is the kth job in the partial sequence given that job i is the pivot and $\lambda_i'(n')$ is the job with the smallest $f_{\cdot}$. Let N_i^+ denote the set of jobs with $f_{\cdot}>f_i$. The possible positions of $\lambda_i'(2)$ is from $\max\{2,k_{i\lambda_i'(2)}\}$ to $n-\left|N_{\lambda_i'(2)}^-\right|$. For each possible position of $\lambda_i'(2)$, N_i^+ can be partitioned into two sets, the front set and the rear set. This partitioning can be done by applying Lemma 4. Let $F(N_i^+,\lambda_i'(2),p)$ denote the front set for a possible position p of $\lambda_i'(2)$. If the partition has no conflicts, i.e., $2+\left|F(N_i^+,\lambda_i'(2),p)\right|=p$, then the new dynamic programming is applied to the set $F(N_i^+,\lambda_i'(2),p)$. For the rear set, the partitioning is done again with $\lambda_i'(3)$, and if it has no conflict, the

dynamic programming is applied, and so on.

Ⅴ. Computational Results

Table 1 illustrates the performance of two dynamic programming formulations. The programs for the dynamic programming are written in Microsoft Visual Basic and run on a PC with Intel Pentium IV 2.0 GHz and 512MB of main memory. Each set of experiments consists of 100 random samples of values for up to 20-jobs case, 30 random samples for the bigger jobs case.

Table 1. Comparison of CPU Time (Sec.)

Algo.	Held and Karp's DP		Pivot DP	
# of Jobs	Avg.	Max.	Avg.	Max.
5	0.0001	0.0100	0.0000	0.00
10	0.0002	0.0100	0.0001	0.01
15	0.0017	0.0400	0.0002	0.01
20	0.0237	0.4000	0.0003	0.01
25	0.2650	4.8570	0.0003	0.01
30	7.2094	80.3960	0.0007	0.01
35	94.5322	1923.8360	0.0010	0.01
40	613.5409	11448.4020	0.0007	0.01
45			0.0013	0.01
50			0.0017	0.01
...			...	...
90			0.0080	0.03
95			0.0130	0.09
100			0.0110	0.02

While the CPU time for the formulation in Held and Karp [8] grows exponentially as the number of jobs increases, the Pivot Dynamic Programming performs very well even in 100-jobs case. However, the formulation in Held and Karp [8] cannot generate the optimal solutions for more than 40-jobs case in reasonable time.

Table 2 shows the performances of Pivot Dynamic Programming and assignment problem (Linear Programming) method.

Table 2. Comparison of CPU Times (Sec.)

Algo.	Pivot DP		LP		Perf. Ratio
# of Jobs	Avg.	Max.	Avg.	Max.	
5	0.0001	0.01	0.0015	0.01	15.000
10	0.0002	0.01	0.0040	0.01	20.000
15	0.0007	0.01	0.0079	0.02	11.286
20	0.0016	0.01	0.0126	0.02	7.875
25	0.0020	0.01	0.0187	0.02	9.350
30	0.0037	0.01	0.0257	0.03	6.946
35	0.0053	0.01	0.0387	0.05	7.302
40	0.0063	0.01	0.0450	0.05	7.143
45	0.0080	0.02	0.0613	0.07	7.663
50	0.0127	0.03	0.0687	0.08	5.409
55	0.0157	0.09	0.0920	0.10	5.860
60	0.0180	0.05	0.1070	0.12	5.944
65	0.0193	0.03	0.1353	0.15	7.010
70	0.0350	0.26	0.1590	0.18	4.543
75	0.0280	0.09	0.1760	0.20	6.286
80	0.0410	0.16	0.2137	0.25	5.212
85	0.0800	0.77	0.2720	0.31	3.400
90	0.0530	0.23	0.2577	0.30	4.862
95	0.0990	0.79	0.2977	0.33	3.007
100	0.0830	0.25	0.4080	0.47	4.916

Pivot Dynamic Programming outperforms Linear Programming on average and is up to 20 times faster than Linear Programming while the worst case performance of Pivot Dynamic Programming is not always better than Linear Programming.

Ⅵ. Final Remarks

This paper addresses two types of stochastic scheduling problems for minimizing the expected number of tardy jobs. First, the cases, where the processing times are arbitrary i.i.d. and due dates are arbitrarily distributed, have been considered.

It is known that optimal static list policy on a single machine can be found by solving an assignment problem as long as the processing times are i.i.d. If the due dates are exponentially distributed, the probability of being tardy for a job has a product form. Therefore, the objective function becomes simpler since it has a form of sum of products.

Several dominance rules, including Pivot Rule that generates very efficient partial sequences, and the position-dependent precedence relationships, are derived to reject non-promising solutions that are guaranteed not be optimal. Utilization of these dominance rules provides a foundation for a dynamic programming algorithm proposed in this paper.

In computational experiments, the new algorithm outperforms the classic method in Held and Karp [9]. The classic method in Held and Karp [9] fails to solve the problems with more than 40 jobs. Computational

experiments show our algorithm is up to 20 times faster than solving assignment problem using Linear Programming.

REFERENCES

[1] S. J. Balut. Scheduling to Minimize the Number of Late Jobs When Set-Up and Processing Times are Uncertain. Management Science, 19(11):1283-1288, 1973.

[2] R. A. Blau. N-Job, One Machine Sequencing Problems under Uncertainty. Management Science, 20(1):101-109, 1973.

[3] O. J. Boxma and F. G. Forst. Minimizing the Expected Weighted Number of Tardy Jobs in Stochastic Flow Shops. Operations Research Letters, 5:119-126, 1986.

[4] X. Cai and S. Zhou. Stochastic Scheduling on Parallel Machines subject to Random Breakdowns to Minimize Expected Costs for Earliness and Tardy Jobs. Operations Research, 47(3):422-437, 1999.

[5] C-S. Chang, J. G. Shanthikumar, and D. D. Yao. "Stochastic Convexity and Stochastic Majorization," in Stochatic Modeling and Analysis of Manufacturing Systems (D. D. Yao, editor). Springer-Verlag, New York, 1994.

[6] P. De, J. B. Ghosh, and C. E. Wells. On the Minimization of the Weighted Number of Tardy Jobs with Random Processing Times and Deadline. Computers and Operations Research, 18:457-463, 1991.

[7] H. Emmons and M. Pinedo. Scheduling Stochastic Jobs with Due Dates on Parallel Machines. European Journal of Operational Research, 47:49-55, 1990.

[8] M. Held and R. M. Karp. A Dynamic Programming Approach to Sequencing Problems. J. SIAM, 10:196-210, 1962.

[9] W. Jang. Dynamic Scheduling of Stochastic Jobs on a Single Machine. European Journal of Operational Research, 138:518-530, 2002.

[10] R. M. Karp. "Reducibility Among Combinatorial Problems," in Complexity of Computer Computations (R. E. Miller and J. W. Thatcher, eds). Plenum Press, New York, 1972.

[11] H. Kise and T. Ibaraki. On Balut's Algorithm and NP-Completeness for a Chance-Constrained Scheduling Problem. Management Science, 29(3):384-388, 1983.

[12] W. Li and K. D. Galzebrook. On Stochastic Machine Scheduling with General Distributional Assumptions. European Journal of Operational Research, 105:525 - 536, 1998.

[13] M. J. Moore. An N Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs. Management Science, 15(1):102 - 109, 1968.

[14] G. R. Parker and R. L. Rardin. Discrete Optimization. Academic Press Inc., 1988.

[15] M. Pinedo. Stochastic Scheduling with Release Dates and Due Dates. Operations Research, 31(3):559-572, 1983.

[16] M. Pinedo. Scheduling : Theory, Algorithms, and Systems. Prentice-Hall Inc., 2002.

[17] S. C. Sarin, E. Erel, and G. Steiner. Sequencing Jobs on a Single Machine with a Common Due Date and Stochastic Processing Times. European Journal of Operational Research, 51:188-198, 1991.

[18] H. M. Soroush and L. D. Fredendall. The Stochastic Single Machine Scheduling Problem with Earliness and Tardiness Costs. European Journal of Operational Research, 77:287-302, 1994.

[19] D. V. Widder. The Laplace Transform. Princeton University Press, 1941.

Authors

Jae Young Choi received the B.S.,degree in Management Science from KAIST in 1990, M.S. degree in Management Science from KAIST in 1993, and Ph.D. degree in Industrial Engineering from State University of New York at Buffalo in 2002. Dr. Choi joined the faculty of College of Information Technology at Ajou University in 2016. He is currently an associate Professor in the Department of Software. His significant areas of interest are Business Analytics, Bigdata and IoT.

Heung-Kyu Kim received the B.S.,degree in Management Science from KAIST in 1991, M.S. degree in Industrial Engineering from KAIST in 1993, and Ph.D. degree in Industrial Engineering from Purdue University in 2002. Dr. Kim joined the faculty of School of Business Administration at Dankook University in 2003. He is currently a Professor in School of Business Administration, Dankook University. He is interested in operations research, economic analysis, and R&D performance evaluation.