Fuzzy Gupta Scheduling for Flow Shops with More Than Two Machines

Tzung-Pei Hong*
Department of Electrical Engineering
National University of Kaohsiung
Kaohsiung, 811, Taiwan, R.O.C.
e-mail: tphong@nuk.edu.tw

Tzung-Nan Chuang
Department of Merchant Marine
National Taiwan Ocean University
Keelung 202, Taiwan, R.O.C.
e-mail: chu@mail.ntou.edu.tw

Abstract

In the past, we demonstrated how fuzzy concepts can easily be used in the Johnson algorithm for managing uncertain scheduling on two-machine flow shops. This paper extends these concepts to fuzzy flow shops with more than two machines. A new fuzzy heuristic flow-shop scheduling algorithm (the fuzzy Gupta algorithm) is then designed since optimal solutions seem unnecessary for uncertain environments. Also, the conventional Gupta algorithm is shown as a special case of the fuzzy Gupta algorithm with special membership functions being assigned.

Keywords: Gupta algorithm, completion time, flow shop, fuzzy task, scheduling.

* Corresponding author
1. Introduction

One kind of scheduling problem that frequently occurs in real-world application environments is the flow-shop problem [5][15][17]. In a flow shop, jobs are processed by a series of machines in a predefined order. For jobs scheduled in a two-machine flow shop, Johnson proposed an algorithm to achieve a minimum makespan [11]. For jobs scheduled in a flow shop of more than two machines, no optimal solution except exhaustive search has ever been proposed. Gupta proposed a heuristic method to achieve a nearly minimum makespan [7]. The Gupta scheduling algorithm first partitions the jobs into two groups by comparing the processing times of the first task and the last task in each job. For each group, it then finds the minimum sum of processing times of any two adjacent tasks in a job, and then schedules the jobs according to their minimum summed processing times using the Johnson algorithm.

In the past, the processing time for each job has usually been assigned or estimated as a fixed value. In many real-world applications, however, the processing time for each job may vary dynamically with the situation. Several theories such as fuzzy set theory [21][22][23], probability theory, D-S theory [18], and approaches based on certainty factors [1], have been developed to manage uncertainty. Among them, fuzzy set theory is becoming more frequently used in intelligent control, because of its simplicity and similarity to human reasoning. Although fuzzy set concepts are mainly used in linguistic domains, they are also used in numerical domains, where each number is assigned a membership value. Examples are Chanas and Kolodziejczyk's fuzzy network flow capacity [2][3], Gazdik's fuzzy network planning [6], Han et al.'s fuzzy duedate scheduling [8], Hong et al.'s Fuzzy LPT scheduling [9], Klein's fuzzy shortest path [12], Nasution's fuzzy critical path [16], McCahon and Lee's fuzzy project network analysis [13] and fuzzy job sequencing [14], Chang et al.'s fuzzy project planning [4], and so on.
In [10], we used fuzzy concepts in Johnson's algorithm for two-machine fuzzy scheduling. In this paper, we attempt to generalize the idea to the Gupta algorithm for flow shops with more than two machines. Given a set of jobs, each having \( m \) (\( m \geq 2 \)) tasks executed respectively on \( m \) machines, and their processing time membership functions, the fuzzy Gupta algorithm yields a scheduling result with a final completion time (or makespan) membership function. Since the processing time for each task may vary dynamically and be full of uncertainty in some real-world applications, optimal solutions seem unnecessary, and the Gupta algorithm is thus sufficient for uncertain environments. Also, the conventional Gupta algorithm is shown as a special case of the fuzzy Gupta algorithm with special membership functions assigned. The fuzzy Gupta algorithm is then a feasible solution for both deterministic and uncertain scheduling in flow shops with more than two machines.

The remainder of this paper is organized as follows. In Section 2, the conventional Gupta algorithm is reviewed. In Section 3, the assumptions and notation used in this paper are stated. In Section 4, a fuzzy Gupta algorithm is proposed for scheduling uncertain jobs in an \( m \)-machine (\( m \geq 2 \)) flow shop. An example is also given to illustrate the new scheduling algorithm. In Section 5, the conventional Gupta scheduling algorithm is proven to be a special case of the fuzzy Gupta scheduling algorithm. Finally, conclusions are given in Section 6.

2. Review of the Gupta Heuristic Algorithm

We now state an \( m \)-machine (\( m \geq 2 \)) flow-shop problem with makespan criterion. Given a set of \( n \) independent jobs, each having \( m \) tasks \((T_{11}, T_{21}, ..., T_{m1}, T_{12}, T_{22}, ..., T_{(m-1)n}, T_{mn})\) that must be executed in the same sequence on \( m \) machines \((P_1, P_2, ..., P_m)\), scheduling seeks the minimum completion time for the last job. Since this problem is NP-hard, Gupta proposed the following heuristic algorithm to solve it in polynomial time [7]:

\[
\text{Algorithm}\;\text{(Gupta)}:
\]

1. Sort the jobs in non-decreasing order of their total processing time on the first machine.
2. Assign the first job to the first machine.
3. For each subsequent job, assign it to the machine that results in the minimum completion time.
4. Repeat steps 2 and 3 until all jobs are scheduled.

The algorithm effectively minimizes the makespan, which is the total time required to complete all jobs in the flow shop.
The Gupta heuristic algorithm:

Input: A set of \( n \) jobs, each having \( m \) (\( m > 2 \)) tasks executed respectively on each of \( m \) machines.

Output: A schedule with a nearly minimum completion time of the last job.

Step 1: Form the group of jobs \( U \) that take less time on the first machine than on the last such that \( U = \{ i \mid t_{1i} < t_{mi} \} \).

Step 2: Form the group of jobs \( V \) that take less time on the last machine than on the first such that \( V = \{ j \mid t_{mj} \leq t_{1j} \} \).

Step 3: For each job \( J_i \) in \( U \), find the minimum of \( (t_{ki} + t_{(k+1)i}) \) for \( k = 1 \) to \( m-1 \); restated set:

\[
\pi_i^{(m-1)} = \min_{k=1}^{(m-1)} (t_{ki} + t_{(k+1)i}).
\]

Step 4: For each job \( J_j \) in \( V \), find the minimum of \( (t_{kj} + t_{(k+1)j}) \) for \( k = 1 \) to \( m-1 \); restated set:

\[
\pi_j^{(m-1)} = \min_{k=1}^{(m-1)} (t_{kj} + t_{(k+1)j}).
\]

Step 5: Sort the jobs in \( U \) in ascending order of \( \pi_i \)’s; if two or more jobs have the same value of \( \pi_i \), sort them in an arbitrary order.

Step 6: Sort the jobs in \( V \) in descending order of \( \pi_j \)’s; if two or more jobs have the same value of \( \pi_j \), sort them in an arbitrary order.

Step 7: Schedule the jobs on the machines in the sorted order of \( U \), then in the sorted order of \( V \).

An example is given below to illustrate the Gupta scheduling algorithm.
Example 1: Assume five jobs $J_1$ to $J_5$ are to be scheduled. Each job ($J_i$) has three tasks ($T_{1i}$, $T_{2i}$, $T_{3i}$) executed on each of three machines ($P_1$, $P_2$, $P_3$). Assume the execution time for each task is as shown in Table 1.

<table>
<thead>
<tr>
<th>Job</th>
<th>$t_{1j}$</th>
<th>$t_{2j}$</th>
<th>$t_{3j}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>4</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>$J_2$</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>$J_3$</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$J_4$</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>$J_5$</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Using the Gupta scheduling algorithm, the execution process is as follows.

Step 1: Form the group of $U$ as $\{J_2, J_4, J_5\}$.

Step 2: Form the group of $V$ as $\{J_1, J_3\}$.

Step 3: Set $\pi_2 = \min\{1+5, 5+2\} = \min\{6, 7\} = 6$,

$\pi_4 = \min\{2+5, 5+3\} = \min\{7, 8\} = 7$,

$\pi_5 = \min\{5+5, 5+6\} = \min\{10, 11\} = 10$.

Step 4: Set $\pi_1 = \min\{4+7, 7+3\} = \min\{11, 10\} = 10$,

$\pi_3 = \min\{5+2, 2+4\} = \min\{7, 6\} = 6$.

Step 5: Sort the jobs in $U$ as $\{J_2, J_4, J_5\}$.

Step 6: Sort the jobs in $V$ as $\{J_1, J_3\}$.

Step 7: Schedule the tasks on the machines. The complete sequence is then $\{J_2, J_4, J_5, J_1, J_3\}$. The final scheduling result is shown in Figure 1, and the final completion time is $f = 30$. Note that $T_{ij}$ denotes the $i$-th task in the $j$-th job.
3. Assumptions and Notation

Assumptions and notation used in this paper are listed below.

Assumptions:
- Jobs are not preemptive.
- Each job has \( m \) tasks to be executed in sequence on \( m \) machines \((m>2)\).
- The execution-time membership function of each task is known.

Notation:
- \( n \): the number of jobs;
- \( m \): the number of machines;
- \( T_{ij} \): the \( i \)-th task for the \( j \)-th job, \( i=1, 2, \ldots, m \) and \( j = 1, 2, \ldots, n \);
- \( t_{ij} \): the fuzzy execution time (a fuzzy set) of \( T_{ij} \);
- \( t_{ijk} \): the \( k \)-th possible execution time of \( t_{ij} \), \( 1 \leq k \leq |\text{supp}(t_{ij})| \), where \(|\text{supp}(t_{ij})|\) is the number of elements in the fuzzy set \( t_{ij} \);
- \( \mu(t_{ijk}) \): the membership value of \( t_{ijk} \);
- \( P_k \): the \( k \)-th machine, \( k=1, 2, \ldots, m \);
- \( p_k \): the fuzzy execution time (a fuzzy set) of \( P_k \);
• \( p_{kj} \): the \( j \)-th possible execution time of \( p_k \), \( 1 \leq j \leq |\text{supp}(p_k)| \);

• \( \mu(p_{kj}) \): the membership value of \( p_{kj} \);

• \( f \): the final completion time (a fuzzy set) of the whole schedule.

Fuzzy addition and fuzzy ranking operations are used in the Gupta scheduling algorithm to schedule fuzzy tasks and to find fuzzy completion times. Let \( A \) and \( B \) be two fuzzy sets, in which each element is a number. \( A \) and \( B \) can then be represented as follows:

\[
A = [\frac{\mu_{a1}}{x_{a1}}, \frac{\mu_{a2}}{x_{a2}}, \ldots, \frac{\mu_{an}}{x_{an}}],
\]

\[
B = [\frac{\mu_{b1}}{x_{b1}}, \frac{\mu_{b2}}{x_{b2}}, \ldots, \frac{\mu_{bm}}{x_{bm}}],
\]

where \( x_{ai} \) is the \( i \)-th element in fuzzy set \( A \), \( \mu_{ai} \) is the membership value of \( x_{ai} \), \( x_{bj} \) is the \( j \)-th element in fuzzy set \( B \), and \( \mu_{bj} \) is the membership value of \( x_{bj} \).

The addition of fuzzy sets \( A \) and \( B \) is as follows:

\[
A + B = [\min(\frac{\mu_{a1}}{x_{a1}}, \frac{\mu_{b1}}{x_{b1}})/(x_{a1}+x_{b1}), \min(\frac{\mu_{a1}}{x_{a1}}, \frac{\mu_{b2}}{x_{b2}})/(x_{a1}+x_{b2}), \ldots, \min(\frac{\mu_{an}}{x_{an}}, \frac{\mu_{bm}}{x_{bm}})/(x_{an}+x_{bm})].
\]

If the sums of two or more \((x_{ai}+x_{bj})\)'s are the same, then only the one with the highest membership value is kept.

Ranking using the averaging method is defined below. Let

\[
x_{a}^{\text{ave}} = \frac{\sum_{i=1}^{[\text{supp}(A)]} (\mu_{ai} \times x_{ai})}{\sum_{i=1}^{[\text{supp}(A)]} (\mu_{ai})}, \text{ and}
\]

\[\text{7}\]
Using the averaging method, we say $A > B$ if $x_a^{\text{ave}} > x_b^{\text{ave}}$.

4. The Fuzzy Gupta Heuristic Algorithm

In the fuzzy Gupta algorithm, each task processed is a fuzzy set. Fuzzy operations are then used to schedule uncertain jobs and to find the fuzzy completion time. The fuzzy Gupta scheduling algorithm using the fuzzy averaging ranking method is shown below. The average ranking method has the merit of simplicity in easily understanding its concepts. Note that other ranking methods can also be used in our fuzzy scheduling algorithm as long as they can determine a crisp ascending or descending order.

**The fuzzy Gupta heuristic algorithm:**

Input: A set of $n$ jobs, each with $m$ tasks to be executed respectively on each of $m$ machines; each task has a processing time membership function.

Output: A fuzzy schedule with a completion time membership function $f$.

Step 1: For each job $T_j$, find the average processing time of $T_{1j}$ and $T_{2j}$ using the following formula:

$$t_{ij}^{\text{ave}} = \frac{\sum_{k=1}^{\text{supp}(t_{ij})} (\mu(t_{ijk}) \times t_{ijk})}{\sum_{k=1}^{\text{supp}(t_{ij})} \mu(t_{ijk})}.$$ 

Step 2: Form the group of jobs $U$ that take fuzzily less time on the first machine than on the last such that $U = \{ i \mid t_{i1}^{\text{ave}} < t_{mi}^{\text{ave}} \}$. 
Step 3: Form the group of jobs $V$ that take fuzzily less time on the last machine than on the first such that $V = \{ j \mid t^{\text{ave}}_{mj} \leq t^{\text{ave}}_{ij} \}$.

Step 4: For each job $r$, set $t^{\prime}_{kr} = t_{kr} + t_{(k+1)r}$, $k = 1$ to $(m-1)$, using the fuzzy addition operation.

Step 5: For each $t^{\prime}_{kr}$, find the average processing time $t^{\text{ave}}_{kr}$ using the formula in Step 1.

Step 6: For each job $J_i$ in $U$, find the minimum of $t^{\text{ave}}_{ki}$ for $k = 1$ to $m$; restated, set:

$$
\pi_i = \min_{k=1}^{m-1} t^{\text{ave}}_{ki}.
$$

Step 7: For each job $J_j$ in $V$, find the minimum of $t^{\text{ave}}_{kj}$ for $k = 1$ to $m$; restated, set:

$$
\pi_j = \min_{k=1}^{m-1} t^{\text{ave}}_{kj}.
$$

Step 8: Sort the jobs in $U$ in ascending order of $\pi_i$'s; if two or more jobs have the same value of $\pi_i$, sort them in an arbitrary order.

Step 9: Sort the jobs in $V$ in descending order of $\pi_j$'s; if two or more jobs have the same value of $\pi_j$, sort them in an arbitrary order.

Step 10: Set the initial completion time $p_1, p_2, \ldots, p_m$ for machines $P_1, P_2, \ldots, P_m$ to zero with a fuzzy value of 1.

Step 11: Schedule the first job $J_j$ in $U$ to the machines such that $T_{1j}$ is assigned to $P_1$, $T_{2j}$ is assigned to $P_2$, ..., and $T_{mj}$ is assigned to $P_m$.

Step 12: Set $p_1 = p_1 + t_{1j}$ using the fuzzy addition operation.

Step 13: Set $p(i+1) = \text{find-longer-time} (p_i, p(i+1)) + t(i+1)_j$, $i = 1, 2, \ldots, (m-1)$, where the find-longer-time procedure is used to find the fuzzy start time for machine $P_{j+1}$ (details can be found below).

Step 14: Remove task $J_j$ from $U$.

Step 15: Repeat Steps 11 to 14 until $U$ is empty.

Step 16: Schedule jobs in $V$ in a similar way (Steps 11 to 14).
Step 17: Set the final completion time \( f = p_m \).

After Step 17, scheduling is finished and a fuzzy completion time with a membership function \( f \) has been found. The *find-longer-time* Procedure, which is similar to that [10], is stated as follows.

For an \( m \)-machine flow shop problem, tasks on machine \( r + 1 \) are executed only after the corresponding tasks on machine \( r \) are finished. In Gupta's original algorithm for scheduling crisp tasks on machine \( r+1 \), start time on machine \( r+1 \) is the longer time between the completion time \( p_{r+1} \) of the previous job on machine \( r+1 \) and the completion time \( p_r \) of the current job on machine \( r \). That is, \( p_{r+1} \) is the start time for the next job on machine \( r+1 \) if and only if \( p_r \) is smaller than \( p_{r+1} \); similarly, \( p_r \) is the start time for the next job on machine \( r+1 \) if and only if \( p_{r+1} \) is smaller than \( p_r \). This idea has been generalized to fuzzy sets to find the longer start time, which is also a fuzzy set. Assume after Step 12 of the fuzzy Gupta's scheduling algorithm, \( \mu_{p_r}(x)/x \) is in the set of completion time \( p_r \) (a fuzzy set) of the current job for machine \( Pr \). As noted for crisp sets, \( x \) in \( p_r \) is the start time for machine \( r+1 \) if no completion time in \( p_{r+1} \) is later than \( x \). Therefore, the membership value \( \mu_{sp_r}(x) \) of \( x \) in \( p_r \) as the start time for machine \( r+1 \) is:

\[
\mu_{sp_r}(x) = \mu_{\text{in } p_r(x)} \land \mu_{\text{not } > \text{ in } p_{r+1}(x)}
= \text{Min } [\mu_{\text{in } p_r(x)}, \mu_{\text{not } > \text{ in } p_{r+1}(x)}], \quad (1)
\]

where \( \mu_{\text{not } > \text{ in } p_{r+1}(x)} \) denotes the membership value for all the elements in \( p_{r+1} \) not bigger than \( x \). For a different \( x \), there may be a different membership value.

Besides,

\[
\mu_{\text{not } > \text{ in } p_{r+1}(x)} = \mu_{\text{in } p_{r+1}(x+ \delta_1)} \land \mu_{\text{in } p_{r+1}(x+ \delta_2)} \land \ldots \land \mu_{\text{in } p_{r+1}(x+ \delta_l)}
= \text{Min } \mu_{\text{in } p_{r+1}(x+ \delta_1)} \ldots \mu_{\text{in } p_{r+1}(x+ \delta_l)}
\]
\[ \mu_{sp_{r+1}}(x) = \min \left[ \mu_{in_{p_r+1}}(x) \cdot 1 - \max_{i=1}^{l} \mu_{= in_{p_r+1}}(x+ \delta_S) \right], \]

where \( \mu_{= in_{p_r+1}}(x+ \delta_S) \) denotes the membership value for the elements in \( p_{r+1} \) not equal to \( x+ \delta_S \), and \( \mu_{= in_{p_r+1}}(x) \) denotes the membership value for the elements in \( p_{r+1} \) equal to \( x+ \delta_S \), and \( \delta_S \) is the possible increment of \( x \) in the fuzzy set \( p_{r+1} \). Inserting Equation (2) into Equation (1), we then have:

\[ \mu_{sp_{r+1}}(x) = \min \left[ \mu_{in_{p_r}}(x), 1 - \max_{i=1}^{l} \mu_{= in_{p_r+1}}(x+ \delta_S) \right]. \]  

Similarly, we can get \( \mu_{sp_{r+1}}(x) \) in \( p_{r+1} \) as the start time for machine \( r+1 \) as follows:

\[ \mu_{sp_{r+1}}(x) = \min \left[ \mu_{in_{p_{r+1}}}(x), \mu_{not > in_{p_r}}(x) \right] = \min \left[ \mu_{in_{p_{r+1}}}(x), 1 - \max_{i=1}^{l} \mu_{= in_{p_r}}(x+ \delta_S) \right]. \]  

Membership value \( \mu_S(x) \) of \( x \) as the start time for machine \( r+1 \) is then:

\[ \mu_S(x) = \mu_{sp_r}(x) \vee \mu_{sp_{r+1}}(x) = \max \left[ \mu_{sp_r}(x), \mu_{sp_{r+1}}(x) \right]. \]  

Inserting Equations (3) and (4) into Equation (5), we then have:

\[ \mu_S(x) = \max_{i=1}^{r+1} \min \left[ \mu_{in_{p_i}}(x), 1 - \max_{i=1}^{l} \mu_{= in_{p_{j-t}}}(x+ \delta_{(j-t)S}) \right]. \]  

According to the above derivation, the find-longer-time Procedure, which scans the elements in \( p_k \)'s only several times, can be designed as follows.
The find-longer-time Procedure:

Input: Two fuzzy sets with completion time, \( p_r \) and \( p_{r+1} \), for each machine.

Output: The start time \( S \) (a fuzzy set) for the next job to be executed on machine \( r+1 \).

Procedure:

(a1) FOR each fuzzy set \( p_k \), \( k = r \) and \( r+1 \), DO the following:

(b1) Sort \( p_k \) in descending order of \( p_k[j] \)’s, where \( 1 \leq j \leq |\text{supp}(p_k)| \).

(b2) Build a new fuzzy set \( l_k \) with a term \( \mu(l_{k0})/l_{k0} \) added to the front of it, and set \( l_{k0} = \infty \) and \( \mu(l_{k0}) = 0 \) as a starting point.

(b3) Set each other element \( l_{kj} = p_{kj} \) and \( \mu(l_{kj}) = \max \{\mu(p_{kj}), \mu(l_{k(j-1)})\} \).

END FOR

(a2) Merge the \( p_k \)'s (\( k = r \) and \( r+1 \)) into a fuzzy list \( S \) in a descending order of \( p_k[j] \)’s; the label \( P_k \) is also attached to each element to represent its source set, that is,

\[
S = \{(\mu(s_1)/s_1, P_{h(s_1)}), (\mu(s_2)/s_2, P_{h(s_2)}), \ldots, (\mu(s_{|\text{supp}(p_r)|+|\text{supp}(p_{r+1})|}/s_{|\text{supp}(p_r)|+|\text{supp}(p_{r+1})|}, P_{h(s_{|\text{supp}(p_r)|+|\text{supp}(p_{r+1})|})}\}
\]

where \( P_{h(s_i)} \) denotes the source set of \( s_i \).

(a3) Attach a pointer to the first element \( \mu(l_{k0})/l_{k0} \) for each \( l_k \), \( k = r \) and \( r+1 \).

(a4) FOR each element \( (\mu(s_i)/s_i, P_{h(s_i)}) \) in \( S \), \( i = 1 \) to \( |\text{supp}(p_r)| + |\text{supp}(p_{r+1})| \), DO the following:

(c1) Process \( l_k \) such that \( k \neq h \ (s_i) \) is satisfied as following:

(d1) get the element which the current pointer points to;

(d2) move the pointer to the element \( \mu(l_{kj})/l_{kj} \), where \( l_{kj} > s_i \) and \( l_{k(j+1)} \leq s_i \).

(c2) Set \( \mu(s_i) = \min\{\mu(s_i), 1-\mu(l_{kj})\} \);

(c3) IF \( s_i = s_{i-1} \), THEN set \( \mu(s_i) = \max\{\mu(s_i), \mu(s_{i-1})/l_{i-1}\} \) and remove the term \( (\mu(s_{i-1})/s_{i-1}, P_{h(s_{i-1})}) \) from \( S \).

END FOR
(a5) Normalize and output the final $S$ in ascending order of $s_i$ (without the $P_{hl(s_i)}$ subterm).

After Step (a5), $S$, the start time for the next job to be executed on machine $r+1$, has been found. Details can be found in [10].

The following example shows how the fuzzy Gupta algorithm works.

**Example 2:** Assume a group of five jobs are to be processed in a three-step operation. The first step is degreasing, the second is painting, and the third is drying. Assume also the fuzzy execution times of these jobs are as listed in Table 2.

<table>
<thead>
<tr>
<th>Table 2: The processing time of the five jobs for Example 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job</td>
</tr>
<tr>
<td>$J_1$</td>
</tr>
<tr>
<td>$J_2$</td>
</tr>
<tr>
<td>$J_3$</td>
</tr>
<tr>
<td>$J_4$</td>
</tr>
<tr>
<td>$J_5$</td>
</tr>
</tbody>
</table>

Using the fuzzy Gupta scheduling algorithm, the execution process is as follows.

**Step 1:** For each job $J_j$, find the average processing time of $t_{1j}^{\text{ave}}$ and $t_{3j}^{\text{ave}}$ as shown in Table 3.

<table>
<thead>
<tr>
<th>Table 3: The average processing time $t_{1j}^{\text{ave}}$ and $t_{3j}^{\text{ave}}$ for Example 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job $j$</td>
</tr>
<tr>
<td>$J_1$</td>
</tr>
</tbody>
</table>
Step 2: Form the group of $U$ as \{J₂, J₄, J₅\}.

Step 3: Form the group of $V$ as \{J₁, J₃\}.

Step 4: For each job $r$, set $t'_{1r} = t_{1r} + t_{2r}$, and $t'_{2r} = t_{2r} + t_{3r}$, using the fuzzy addition operation. The results are shown in Table 4.

Table 4: The results of $t'_{1r}$ and $t'_{2r}$ for Example 2

<table>
<thead>
<tr>
<th>Job j</th>
<th>$t'_{1r}$</th>
<th>$t'_{2r}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>J₁</td>
<td>{1.0/11}</td>
<td>{1.0/10, 0.9/11}</td>
</tr>
<tr>
<td>J₂</td>
<td>{0.9/5, 1.0/6}</td>
<td>{0.9/6, 1.0/7, 0.9/8}</td>
</tr>
<tr>
<td>J₃</td>
<td>{1.0/7, 0.9/8, 0.8/9}</td>
<td>{1.0/6, 0.8/7}</td>
</tr>
<tr>
<td>J₄</td>
<td>{1.0/7, 0.2/9}</td>
<td>{0.7/7, 1.0/8}</td>
</tr>
<tr>
<td>J₅</td>
<td>{0.5/9, 1.0/10}</td>
<td>{1.0/11}</td>
</tr>
</tbody>
</table>

Step 5: For each $t'_{kr}$, find the average processing time $t'_{kr}^{\text{ave}}$ as shown in Table 5.

Table 5: The results of $t'_{kr}^{\text{ave}}$ for Example 2

<table>
<thead>
<tr>
<th>Job j</th>
<th>$t'<em>{kr}^{\text{ave}}</em>{1r}$</th>
<th>$t'<em>{kr}^{\text{ave}}</em>{2r}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>J₁</td>
<td>11</td>
<td>10.47</td>
</tr>
<tr>
<td>J₂</td>
<td>5.33</td>
<td>7</td>
</tr>
<tr>
<td>J₃</td>
<td>7.93</td>
<td>6.44</td>
</tr>
<tr>
<td>J₄</td>
<td>7.33</td>
<td>7.59</td>
</tr>
</tbody>
</table>
Step 6: In the set $U$, set $\pi_2 = \min\{5.53, 7\} = 5.53,$
$\pi_4 = \min\{7.33, 7.59\} = 7.33,$
$\pi_5 = \min\{9.67, 11\} = 9.67.$

Step 7: In the set $V$, set $\pi_1 = \min\{11, 10.47\} = 10.47,$
$\pi_3 = \min\{7.93, 6.44\} = 6.44.$

Step 8: Sort the jobs in $U$ as $\{J_2, J_4, J_5\}$.

Step 9: Sort the jobs in $V$ as $\{J_1, J_3\}$.

Step 10: Set $p_1 = p_2 = p_3 = \{1.0/0\}$.

Step 11: Assign $J_2$ (the first job in $U$) to machines for execution.

Step 12: Set $p_1 = p_1 + t_{12} = \{1.0/0\} + \{1.0/1\} = \{1.0/1\}.$

Step 13: Set $p_2 = \text{find-longer-time}(p_1, p_2) + t_{22}$

$\quad = \{1.0/1\} + \{0.9/4, 1.0/5\}$

$\quad = \{0.9/5, 1.0/6\}.$

Set $p_3 = \text{find-longer-time}(p_2, p_3) + t_{32}$

$\quad = \{0.9/5, 1.0/6\} + \{1.0/2, 0.9/3\}$

$\quad = \{0.9/7, 1.0/8, 0.9/9\}.$

Step 14: Remove $J_2$.

Steps 15 and 16: Repeat the above steps for the other jobs. The final results are shown in Table 6 and Figure 2. Note that $T_{ij}$ denotes the $i$-th task in the $j$-th job and the length for each task in Figure 2 is proportional to its average time.

<table>
<thead>
<tr>
<th>Job</th>
<th>$P_1$ (after $T_{1j}$ is added)</th>
<th>$P_2$ (after $T_{2j}$ is added)</th>
<th>$P_3$ (after $T_{3j}$ is added)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_2$</td>
<td>${1.0/1}$</td>
<td>${0.9/5,1.0/6}$</td>
<td>${0.9/7,1.0/8,0.9/9}$</td>
</tr>
<tr>
<td>$J_4$</td>
<td>${1.0/3,0.2/5}$</td>
<td>${0.9/10,1.0/11}$</td>
<td>${0.7/12,0.9/13,1.0/14}$</td>
</tr>
</tbody>
</table>
Figure 2: The final schedule results for Example 2

Step 17: The final completion time $f = p_3 = \{0.2/29, 1.0/30, 0.9/31\}$.

The execution of the find-longer-time procedure is shown in the next example.

**Example 3:** We take the processing of $J_1$ in Table 6 as an example. After Step 12 is finished for $J_1$, $p_1$ (after $T_{11}$ has been added) and $p_2$ (before $T_{21}$ has been added) are as follows:

- $p_1 = \{0.5/11, 1.0/12, 0.2/13, 0.2/14\}$, and
- $p_2 = \{0.9/15, 1.0/16\}$.

The procedure proceeds as follow.

(b1): We first sort $p_1$ and $p_2$ in descending order of execution time as follows:
\[ p_1 = \{0.2/14, 0.2/13, 1.0/12, 0.5/11\}, \]
\[ p_2 = \{1.0/16, 0.9/15\}. \]

(b2) and (b3): \( l_1 \) and \( l_2 \) are built as follows:

\[ l_1 = \{0/\infty, 0.2/14, 0.2/13, 1.0/12, 1.0/11\}, \]
\[ l_2 = \{0/\infty, 1.0/16, 1.0/15\}. \]

(a2): List \( S \) merged from \( p_1 \) and \( p_2 \) is as follows:

\[ S = \{(1.0/16, P_2), (0.9/15, P_2), (0.2/14, P_1), (0.2/13, P_1), (1.0/12, P_1), (0.5/11, P_1)\}. \]

(a3): Attach pointers to the first elements of \( l_1 \) and \( l_2 \).

(a4): The first element \((1.0/16, P_2)\) in \( S \) is processed.

(c1): We then search \( l_1 \) for \( l_{1j} \) satisfying \( l_{1j} > 16 \) and \( l_{1(j+1)} \leq 16 \). \( l_{10} (0/\infty) \) is then found.

(c2): \( \mu(s_1) = \mu(16) = \text{Min} \left[ \mu(s_1), 1-\mu(l_{10}) \right] = \text{Min} \left[ 1.0, 1 - 0 \right] = 1.0. \)

In a similar way, we can find:

\[ \mu(s_2) = \mu(15) = 0.9, \]
\[ \mu(s_3) = \mu(14) = 0, \]
\[ \mu(s_4) = \mu(13) = 0, \]
\[ \mu(s_5) = \mu(12) = 0, \]
\[ \mu(s_6) = \mu(11) = 0. \]

We can then derive the final \( S \) as:

\[ S = \{1.0/16, 0.9/15\}. \]
S is then the start time for $t_{21}$ to be executed on machine 2.

5. Reducing to the Original Gupta Scheduling Algorithm

In this section, the fuzzy Gupta scheduling algorithm is shown to be equivalent to the original Gupta scheduling algorithm in environments where the latter is applied. For the original Gupta scheduling algorithm to work, the task time must be known and definite. That is, each task must have a time with a single membership value = 1, and all other time membership values = 0. In this environment, the two algorithms can be proven to be equivalent as follows.

**Theorem 1:**

If Fuzzy set $A = \{1/a\}$ and Fuzzy set $B = \{1/b\}$, then find-longer-time $(A, B) = \{1/\max(a, b)\}$.

**Proof:**

This has been proven in [10].

**Theorem 2:**

The execution sequences using the fuzzy Gupta algorithm and using the original Gupta algorithm are the same for definite tasks if when there is a tie, the rules that allocate jobs to the processors are the same.

**Proof:**

Let $o_{ij}$ and $f_{ij}$ denote, respectively, the definite execution time in the original Gupta algorithm and the fuzzy execution time in the fuzzy Gupta algorithm for task $T_{ij}$. Since the time for each task $T_{ij}$ is definite, the fuzzy execution time $f_{ij} = \{1/o_{ij}\}$. This implies that:
\[ ft_{kj}^{ave} = ot_{kj}. \]

Also,
\[
ft'_{kj} = ft_{kj} + ft_{(k+1)j}
= \{1/ot_{kj}\} + \{1/ot_{(k+1)j}\}
= \{1/(ot_{kj} + ot_{(k+1)j})\}
= \{1/ot'_{kj}\}.
\]

The sorted job list resulting from Steps 1 to 9 of the fuzzy Gupta algorithm is then the same as that yielded by the original Gupta scheduling algorithm. Since the tasks are scheduled according to the sorted list in both algorithms, the proof is complete.

**Theorem 3:**

*If the schedule time from the original Gupta scheduling algorithm is \( of \), then the fuzzy schedule time \( ff \) from the fuzzy Gupta scheduling algorithm is \( \{1/of\} \).*

**Proof:**

Without loss of generality, we may assume that the final scheduling order is \( J_1 \) to \( J_n \) using both algorithms (from Theorem 2). We prove by induction that the intermediate scheduling results yielded by the two scheduling algorithms are the same. First, the correctness of the theorem for \( J_1 \) is proven. In this case, the original Gupta scheduling algorithm initially sets the execution time \( (op_1 \ to \ op_m) \) for \( P_1 \ to \ P_m \) to 0, and the fuzzy Gupta scheduling algorithm initially sets the fuzzy execution time \( \bar{fp}_k = \{1/0\} = \{1/op_k\} \), for \( k = 1 \) to \( m \). The first job \( J_1 \) is then put into the processors by the two algorithms. The execution time \( op_1 \) yielded by the original Gupta scheduling algorithm becomes:

\[ new \ op_1 = old \ op_1 + ot_{11} \]
The execution time \( op_2 \) yielded by the original Gupta scheduling algorithm becomes:

\[
new \, op_2 = \max (new \, op_1, \, old \, op_2) + \, ot_{21}
\]

\[
= \max (ot_{11}, \, 0) + \, ot_{21}
\]

\[
= ot_{11} + \, ot_{21}.
\]

In general, we can get:

\[
New \, \, op_k = \max (new \, \, op_{k-1}, \, old \, \, op_k) + \, ot_{k1}
\]

\[
= \sum_{i=1}^{k} ot_{i1}.
\]

The execution time, \( fp_1 \), yielded by the fuzzy Gupta scheduling algorithm becomes:

\[
new \, fp_1 = old \, fp_1 + \, ft_{11}
\]

\[
= \{1/0\} + \{1/ \, ot_{11}\}
\]

\[
= \{1/ \, ot_{11}\}
\]

\[
= \{1/\, new \, op_1\}.
\]

The execution time, \( fp_2 \), yielded by the fuzzy Gupta scheduling algorithm becomes:

\[
new \, fp_2 = \text{find-longer-time}(new \, fp_1, \, old \, fp_2) + \, ft_{21}
\]

\[
= \text{find-longer-time}(1/\, new \, op_1, \, 1/0) + \, ft_{21}
\]

\[
= \{1/\, \max (new \, op_1, \, 0)\} + \{1/ \, ot_{21}\} \quad (\text{From Theorem 1})
\]

\[
= \{1/(\, new \, op_1 + \, ot_{21})\}
\]
\[ \frac{1}{(\alpha_{1} + \alpha_{2})} = \frac{1}{\text{new } \alpha_{2}}. \]

In general, we can get:

\[ \text{new } \alpha_{k} = \text{find-longer-time(new } \alpha_{k-1}, \text{ old } \alpha_{k}) + \alpha_{k1} \]
\[ = \frac{1}{\sum_{i=1}^{k} \alpha_{i1}} \]
\[ = \frac{1}{\text{new } \alpha_{k}}. \]

Next, by induction, we assume \( \alpha_{k} = \frac{1}{\alpha_{k}} \) for \( k = 1 \) to \( m \) after \( J_{S} \) is scheduled, and prove that the results are also valid for \( J_{S+1} \). The execution time \( \alpha_{1} \) yielded by the original Gupta scheduling algorithm becomes:

\[ \text{new } \alpha_{1} = \text{old } \alpha_{1} + \alpha_{1(s+1)}. \]

The execution time \( \alpha_{2} \) yielded by the original Gupta scheduling algorithm becomes:

\[ \text{new } \alpha_{2} = \text{Max}(\text{new } \alpha_{1}, \text{ old } \alpha_{2}) + \alpha_{2(s+1)}. \]

In general, we can get:

\[ \text{new } \alpha_{k} = \text{Max}(\text{new } \alpha_{k-1}, \text{ old } \alpha_{k}) + \alpha_{k(s+1)}. \]

The execution time, \( \alpha_{1} \), yielded by the fuzzy Gupta scheduling algorithm becomes:

\[ \text{new } \alpha_{1} = \text{old } \alpha_{1} + \alpha_{1(s+1)} \]
\[ = \frac{1}{\text{old } \alpha_{1}} + \frac{1}{\alpha_{1(s+1)}} \]
\[ = \frac{1}{(\text{old } \text{op} + \text{ot}(s+1))} \]
\[ = \frac{1}{\text{new } \text{op}}. \]

The execution time, \( f_{p_2} \), yielded by the fuzzy Gupta scheduling algorithm becomes:

\[ \text{new } f_{p_2} = \text{find-longer-time(new } f_{p_1}, \text{ old } f_{p_2}) + f_{t_2(s+1)} \]
\[ = \text{find-longer-time}(\{1/\text{new } \text{op}\}, \{1/\text{old } \text{op}\}) + f_{t_2(s+1)} \]
\[ = \frac{1}{\text{Max(new } \text{op}, \text{ old } \text{op})} + \frac{1}{\text{ot}_2(s+1)} \quad \text{(from Theorem 1)} \]
\[ = \frac{1}{(\text{Max(new } \text{op}, \text{ old } \text{op}) + \text{ot}_2(s+1))} \]
\[ = \frac{1}{\text{new } \text{op}}. \]

In general, we can get:

\[ \text{new } f_{p_k} = \text{find-longer-time(new } f_{p_{k-1}}, \text{ old } f_{p_k}) + f_{t_{k(s+1)}} \]
\[ = \text{find-longer-time}(\{1/\text{new } \text{op}_{k-1}\}, \{1/\text{old } \text{op}_k\}) + f_{t_{k(s+1)}} \]
\[ = \frac{1}{\text{Max(new } \text{op}_{k-1}, \text{ old } \text{op}_k)} + \frac{1}{\text{ot}_{k(s+1)}} \]
\[ = \frac{1}{(\text{Max(new } \text{op}_{k-1}, \text{ old } \text{op}_k) + \text{ot}_{k(s+1)})} \]
\[ = \frac{1}{\text{new } \text{op}_k}. \]

Therefore, after all jobs are scheduled, \( f_{p_m} = \{1/\text{op}_m\} \). According to Step 17 of the fuzzy Gupta scheduling algorithm, the completion time \( \text{ff} \) is \( f_{p_m} \). According to the original Gupta scheduling algorithm, the completion time \( \text{of} \) is \( \text{op}_m \). So, \( \text{ff} = f_{p_m} = \{1/\text{op}_m\} = \{1/\text{of}\} \). This completes the proof. \( \square \)

6. Conclusion

In this paper, we have proposed the fuzzy Gupta algorithm for scheduling uncertain jobs in a flow shop with more than two machines. The fuzzy Gupta algorithm can yield a scheduling result with a membership function for the final completion time.
The results can then help managers gain a broader overall view of scheduling. Also, the conventional Gupta algorithm is shown as a special case of the fuzzy Gupta algorithm with special membership functions assigned. The fuzzy Gupta algorithm is then a feasible solution for both deterministic and uncertain flow shops with more than two machines.

The Gupta algorithm basically transforms the flow-shop scheduling problems of more than two machines into those of two machines and then uses the Johnson algorithm to solve them. In the transformation, the minimum sum of the execution times of any two adjacent tasks in a job is used to estimate the execution order of this job. It is heuristic and in general not optimal. The schedule determined by the Gupta algorithm may be further adjusted by some local search approaches, such as 2-opt [20]. The 2-opt local search approach was originally proposed for the traveling salesrep problem. It used two positions in a feasible solution and reversed the substring between the two positions. The 2-opt approach may be used to swap any two jobs in the schedule determined by the Gupta algorithm, with the best swap chosen. Of course, it needs some additional time complexity and may get better final results. In the adjustment, the fuzzy concepts may also be adopted to process fuzzy execution times. We will invest this problem in the future. Besides, in the past, many kinds of inference rules and defuzzifiers were proposed for the control problems [19]. They are mainly used with existing fuzzy rule bases. They are not used here since the Gupta scheduling algorithm works by considering the execution times of tasks, instead of using a fuzzy rule base. Using a fuzzy rule base to help good scheduling is, however, an interesting problem while considering complicated environment variables. More research should be done in this area in the future.

Acknowledgement
The authors would like to thank the anonymous referees for their very constructive comments. This research was supported by the National Science Council of the Republic of China under contract NSC92-2213-E-390-001.
References


