% Function to calculate the Robust Ranking for Octagonal Fuzzy Numbers

function R = robust\_ranking\_octagonal(A)

if length(A) ~= 9

error('Input must be a vector of 8 elements for octagonal fuzzy number.');

end

% Robust ranking formula

R = (1/25) \* (A(1) + 2\*A(2) + 3\*A(3) + 4\*A(4) + 5\*A(5) + 4\*A(6) + 3\*A(7) + 2\*A(8) + A(9));

end

% Define the octagonal fuzzy numbers for each item in the Cutting (M1), Sewing (M2), and Pressing (M3)

M1 = [

-2, -1, 0, 1, 4, 5, 6, 7, 8; % A1

1, 2, 3, 4, 5, 6, 9, 10, 11; % A2

1, 2, 4, 5, 8, 10, 12, 14, 17; % A3

2, 3, 6, 7, 8, 10, 11, 13, 15; % A4

3, 5, 6, 8, 10, 12, 13, 15 ,17 % A5

];

M2 = [

-3, -2, -1, 0, 1, 2, 3, 4, 5; % A1

-3, -2, -1, 0, 4, 5, 6, 7, 8; % A2

-4, -3, 0, 1, 2, 3, 4, 5, 6; % A3

-2, -1, 0, 1, 2, 3, 4, 5, 6; % A4

-1, 0, 1, 2, 3, 4, 5, 6, 7 % A5

];

M3 = [

7, 8, 10, 11, 14, 15, 17, 18, 19; % A1

6, 8, 9, 10, 12, 13, 14, 16, 18; % A2

-1, 0, 1, 2, 4, 5, 6, 7, 8; % A3

5, 6, 7, 8, 12, 13, 14, 15, 16; % A4

1, 2, 3, 4, 5, 6, 7, 8, 9 % A5

];

% Calculate the robust ranking for each fuzzy number in M1, M2, M3

num\_items = size(M1, 1);

rankings\_M1 = zeros(num\_items, 1);

rankings\_M2 = zeros(num\_items, 1);

rankings\_M3 = zeros(num\_items, 1);

for i = 1:num\_items

rankings\_M1(i) = robust\_ranking\_octagonal(M1(i, :));

rankings\_M2(i) = robust\_ranking\_octagonal(M2(i, :));

rankings\_M3(i) = robust\_ranking\_octagonal(M3(i, :));

end

% Aggregate the rankings by summing the ranks from M1, M2, and M3 for each item

total\_rankings = rankings\_M1 + rankings\_M2 + rankings\_M3;

% Display the rankings for each process

disp('Rankings for Cutting (M1):');

disp(rankings\_M1);

disp('Rankings for Sewing (M2):');

disp(rankings\_M2);

disp('Rankings for Pressing (M3):');

disp(rankings\_M3);

disp('Total Rankings for all processes:');

disp(total\_rankings);

% Step 1: Create artificial machines

Machine\_A = rankings\_M1 + rankings\_M2; % M1 + M2

Machine\_B = rankings\_M2 + rankings\_M3; % M2 + M3

% Number of jobs

num\_jobs = length(rankings\_M1);

;

% Step 2: Initialize job sequence and flags

job\_sequence = zeros(1, num\_jobs); % To store the final job sequence

scheduled\_jobs = false(1, num\_jobs); % To keep track of scheduled jobs

% Initialize pointers for sequence (front and back)

front\_pointer = 1;

back\_pointer = num\_jobs;

% Step 3: Apply Johnson's Rule

for i = 1:num\_jobs

% Find unscheduled jobs and their processing times

unscheduled\_A = Machine\_A(~scheduled\_jobs);

unscheduled\_B = Machine\_B(~scheduled\_jobs);

% Find minimum processing times in Machine A and Machine B

[min\_A, idx\_A] = min(unscheduled\_A);

[min\_B, idx\_B] = min(unscheduled\_B);

% Find the job index for the minimum time

unscheduled\_job\_indices = find(~scheduled\_jobs);

if min\_A <= min\_B

job\_index = unscheduled\_job\_indices(idx\_A); % Minimum on Machine A

job\_sequence(front\_pointer) = job\_index; % Schedule at front

front\_pointer = front\_pointer + 1; % Move front pointer

else

job\_index = unscheduled\_job\_indices(idx\_B); % Minimum on Machine B

job\_sequence(back\_pointer) = job\_index; % Schedule at back

back\_pointer = back\_pointer - 1; % Move back pointer

end

% Mark the job as scheduled

scheduled\_jobs(job\_index) = true;

end

% Display the optimal job sequence

fprintf('Optimal Job Sequence: ');

disp(job\_sequence);