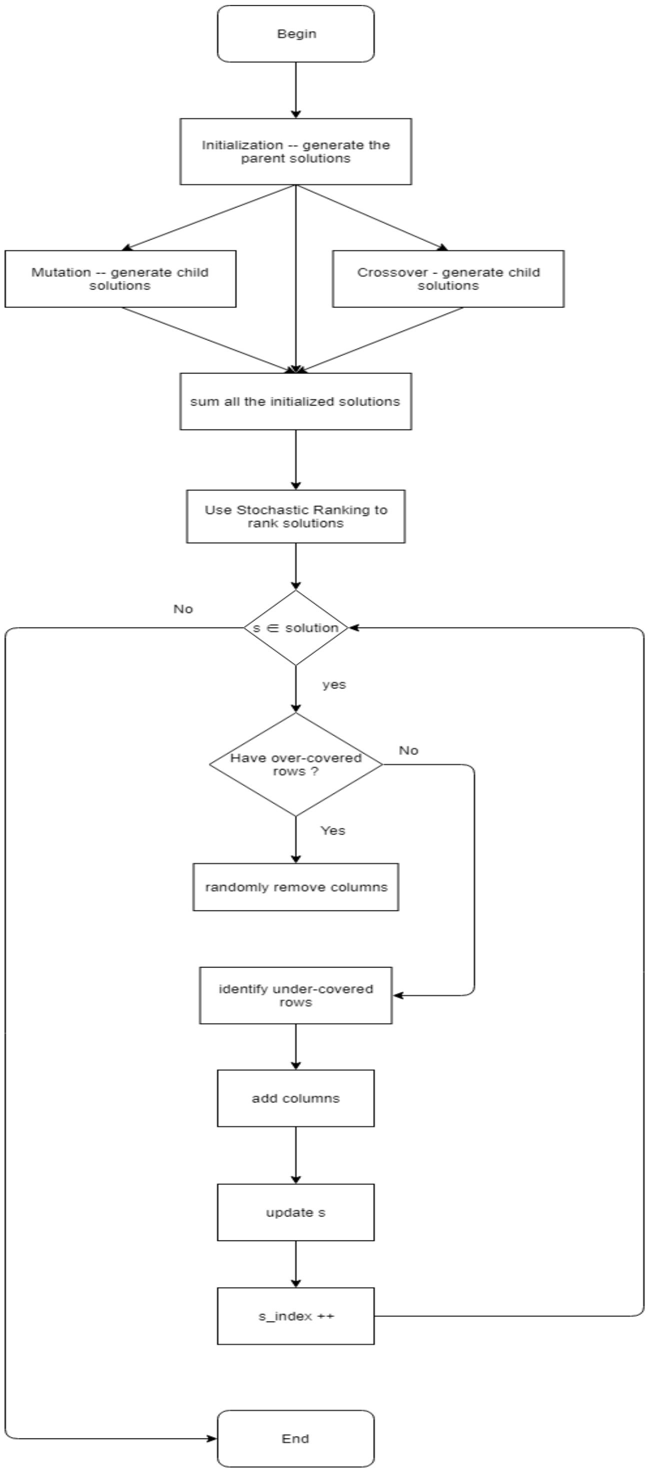
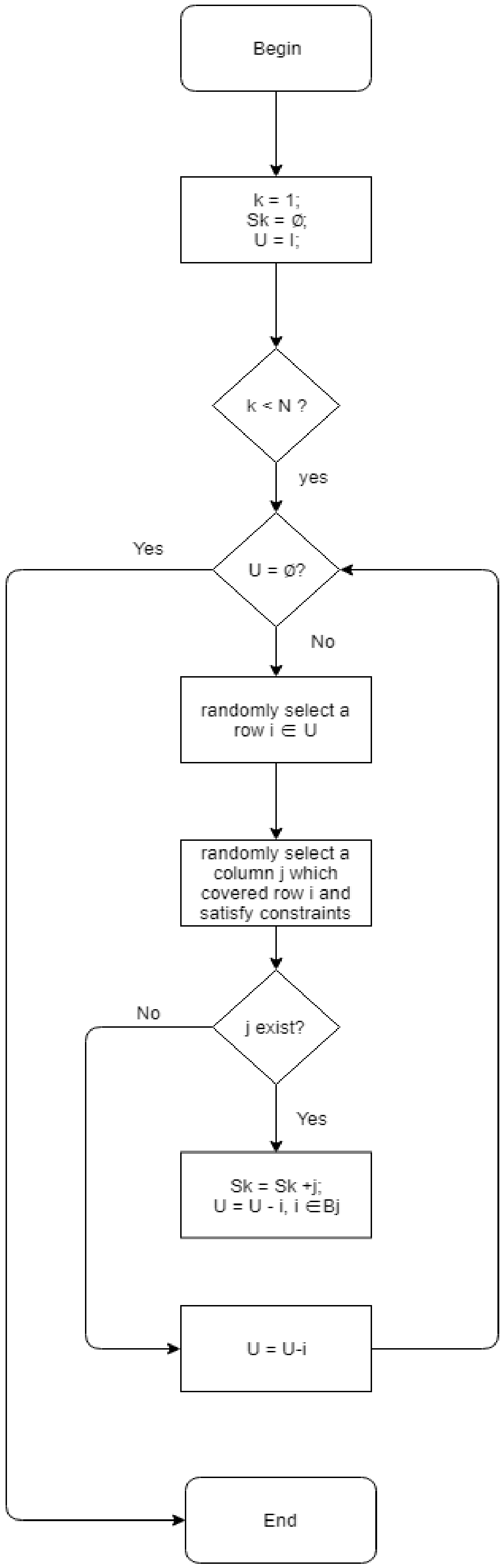
Genetic Algorithm

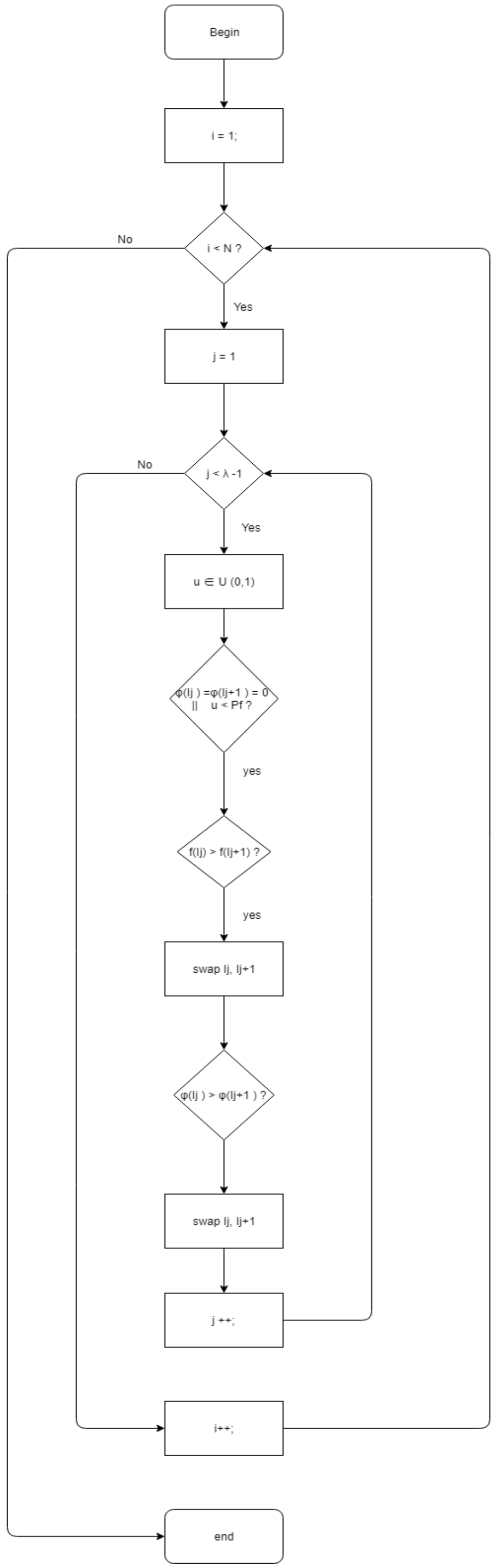
Flowchart of whole process



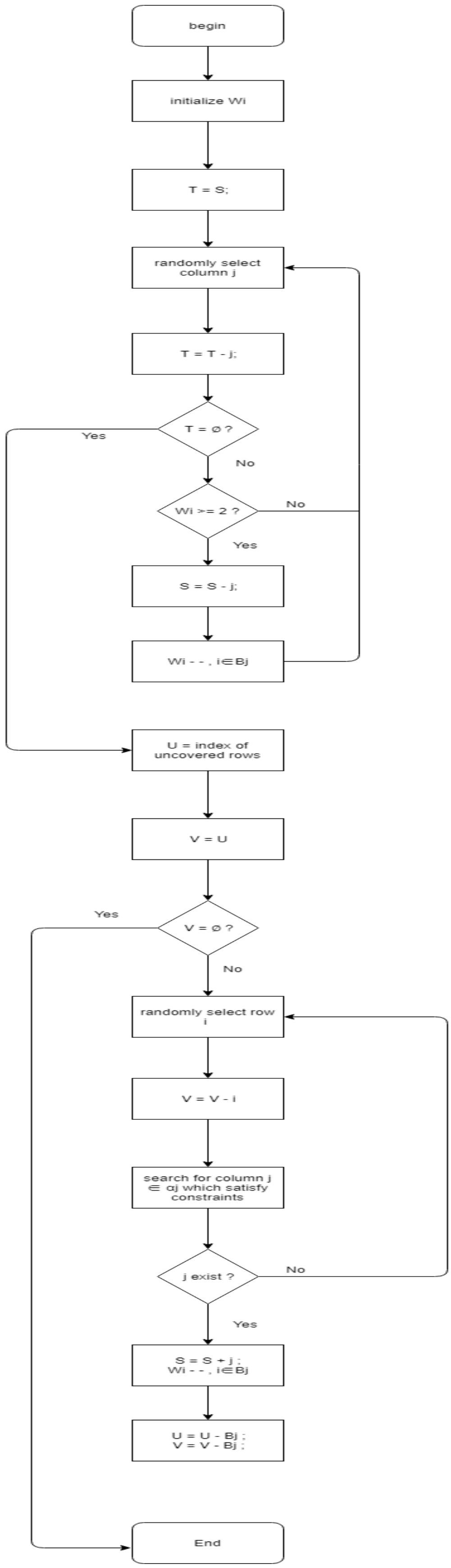
Flowchart of Initialization



Flowchart of Stochastic Ranking



Flowchart of Heuristic Improvement



Pseudo-code

- **Initialisation**

```
for k = 1:N
    Sk = zeros(1,n);
    U = I;
    % It terminates when all rows are covered, i.e., sum(I)=m
     while ~isempty(U)
        % randomly select an    row i in U
        i = randi(length(U));
        % alpha_i is the indices of columns that cover row i
        alpha_i = find(con_matrix(i,:)==1);
    %      randomly select a column j ∈ αi
         j = alpha_i(randi(length(alpha_i)));
    %       βj
        beta_j = find(con_matrix(:,j) == 1);
        K = setdiff(I, U);
        F = intersect(beta_j, K);
    %       not exist j
        num = length(beta_j);
        c = [];
        if isempty(F)
            for i = 1:num
                c = [c,i];
            end
            U(:,c) = [];
            Sk(1,j) = 1;
        else
           U(:,i) = [];
        end

     end
   result = [result;Sk];
end
```

- **StochasticRanking**

```
N = size(sum_result,1);
for j = 1:N
    for i = 2:N
        u = normrnd(0,1);
        if gx(sum_result, i-1) == gx(sum_result,i) == 0 || u <= 0.5
            if fx(sum_result,column_cost,i-1) < fx(sum_result,column_cost,i)
                % swap I
                swapi = sum_result(i,:);
```

```
                sum_result(i,:) = sum_result(i-1,:);
                sum_result(i-1,:) = swapi;
            end
        else
            if gx(sum_result, i-1) < gx(sum_result,i)
                % swap I
                swapi = sum_result(i,:);
                sum_result(i,:) = sum_result(i-1,:);
                sum_result(i-1,:) = swapi;
            end
        end
        i = i+1;
    end
    j = j +1;
end
```

- **Heuristic Improvement**

```
    w = con_matrix * sum_result(solutionIndex,:)';
    S = 1:numOfResultColumns;
    solution = sum_result(S);
    cost = solution * column_cost'
    I = 1:numOfResultRows;
    T = S;

    % DROP
    while ~isempty(T)
        j = randi(numOfResultColumns);
        T = setdiff(T,j);
        for i = 1:size(con_matrix,1)
            if w(i) >= 2
                S = setdiff(S,j);
                w(i) = w(i) - 1;
            end
        end
    end

    % initialize
    U = [];
    for i = 1:size(con_matrix,1)
        if w(i) == 0
            U = [U;i];
        end
    end
    U
    V = U;
```

```
        minValue = Inf;
        best_j = 0;

%       % ADD
        beta_j = [];
        best_beta_j = [];
        while ~isempty(V)
            i = V(randi(length(V)));
            V = setdiff(V,i);
            % the indices of columns that cover row i
            columnSet = find(con_matrix (i,:) == 1);
            beta_j = [];
            best_beta_j = [];
            for col = 1:length(columnSet)
                columnIndex = columnSet(col);
                temp = find(con_matrix(:,columnIndex) == 1);
                beta_j = temp;
                if all(ismember(beta_j, U))
                    value = column_cost(j) / (length(beta_j));
                    if value < minValue
                        minValue = value;
                        best_j = j;
                        best_beta_j = beta_j;
                    end
                end
            end
            if (best_j ~= 0)
                S = [S,best_j];
                for row = 1:length(best_beta_j)
%                       actualRow = con_matrix(row,:);
                    w(row) = w(row) + 1;
                end
                U = setdiff(U, best_beta_j);
                V = setdiff(V, best_beta_j);
            end
    end
```

Result

Run	Min_cost
1	11374
2	13803
3	12705

4	14127
5	12426
6	15810
7	15000
8	12784
9	13402
10	12044
11	11906
12	16797
13	14380
14	12775
15	11708
16	12405
17	13742
18	14738
19	12940
20	12036
21	11593
22	16700
23	14356
24	12539
25	14839
26	13489
27	12830
28	12349
29	13957
30	11374

Average: 13364
Standard deviations: 1479

Similarity & Difference

Similarity

1. These two methods both pursing a balance between fitness and unfitness (objective function and penalty).
2. In both methods, the initial solutions will move towards the optimal solution.
3. Both methods prevent from calculating a certain penalty parameter.
4. Although Ranking Replacement method divides the population into four subgroups, it selected the member with the worst unfitness by comparing with adjacent members in a

subspace, which is similar to the stochastic ranking method that ranked λ individuals by comparing adjacent individuals in at least λ sweeps.

Difference

1. In the process of replacement,
   For **Ranking replacement method**, offspring are divided into four subgroups, which is G1 (Higher cost & Higher violation), G2(Lower cost & Higher violation), G3(Higher cost & Lower violation) and G4(Lower cost & Lower violation). And a child solution will try to replace a solution in G1, then G2, G3, G4. In any subgroup, the member selected for replacement by the child is the member with the worst unfitness. For each generation of X offspring, it will try to replace the parents in four subsets, so this process will be performed X times.
   For **Stochastic ranking method**, offspring and parents are ranked together, then select some results to generate new generation.
2. In **Stochastic ranking method**, a probability P_f of using only the objective function for comparisons in ranking in the infeasible regions of the search space are introduced, which shows our preference between cost and violation.
   In **Ranking replacement method**, we don't have this kind of preference.