Effective GA Operator for Product Assembly Sequence Planning

Nima Geran Malek[1], Armin Dadras Eslamlou[2], Qingjin Peng[3], and Shiping Huang[2]

[1]University of Manitoba, Geranman@myumanitoba.ca
[2]South China University of Technology, Armindadraseslamlou@gmail.com
[3] University of Manitoba, Qingjin.Peng@umanitoba.ca
[2]South China University of Technology, ctasihuang@scut.edu.cn

Corresponding author: Qingjin Peng, Qingjin.Peng@umanitoba.ca

Abstract. Assembly operations of products can be determined by assembly sequence planning to meet assembly criteria. Although the genetic algorithm (GA) is one of the common algorithms used to find the optimal sequence of components in the product assembly, the optimal sequence is selected after the elimination of sequences that do not meet constraints. There is a lack of research on the effect of different types of crossover operators on GA performance. This paper introduces applications of different GA operators in the search for the optimal product assembly sequence. Four versions of GA are evaluated for their performances by employing different crossover operators in the algorithm. The roulette wheel is used as the selection mechanism. The solutions are examined by the statistical analysis in three case studies. This investigation obtains the pros and cons of each method to select the most suitable GA crossover operator for solving this specific optimization problem.

Keywords: Assembly sequence planning (ASP), Genetic algorithm (GA), Crossover operators
DOI: https://doi.org/10.14733/cadaps.2024.713-728

1 INTRODUCTION

Assembly sequence planning (ASP) develops a comprehensive plan for product assembly operations [2], which requires the identification of relevant constraints and requirements that must be satisfied for the optimal sequence of assembly operations. ASP ensures that the final product is safe and reliable in operations and reduces the production cost and time to market. As the assembly process accounts for approximately 20-30% of the manufacturing cost and total production time, it is essential to reduce the assembly time and cost through ASP [17][29].

ASP determines operation orders of product components in the assembling process. Assembly sequences of a product are searched to meet constraints. The optimal sequence takes the minimum time or cost for assembly operations. An ASP problem can exhibit a search space explosion when the number of components increases, which is an NP-hard problem. For instance, if a product consists of 9 components being assembled in any order, the total number of possible assembly sequences is 9! which equals 362880 solutions [22].

Metaheuristic optimization algorithms are widely used in finding optimal or near-optimal solutions to explore a large solution space [11]. Genetic algorithm (GA) is a type of metaheuristic optimization technique that imitates the process of natural selection to solve complex problems, which provides an effective way to solve ASP problems by using a population of candidate solutions that evolve over multiple generations. Using GA, each candidate solution is represented as a chromosome, and the algorithm evaluates the fitness of each chromosome based on its satisfaction with the objective function, such as minimizing assembly time. The GA uses techniques such as selection, crossover, and mutation to generate new populations of candidate solutions.

GA can be applied in ASP to handle different variables and constraints for specific problem requirements. For example, different types of selection, crossover, and mutation operators can be used to balance exploration and exploitation of the search space. Crossover is a crucial genetic operator in GA to combine the genetic information of two or more individuals to create new individuals [7]. Different types of crossover operators have been used for ASP.

Although GA is commonly used to find the optimal sequence of components in the product assembly, the optimal sequence is selected after the elimination of sequences that do not meet constraints. There is a lack of research on the effect of different crossover operators on GA performance. In this paper, four different versions of GA are compared for the ASP problem by employing different crossover operators. In the following section, the literature is reviewed for the existing research on ASP and GA. Section 3 describes the research definition for the objective function and constraints of the problem. In Section 4, different types of GA operators are described for applications. Section 5 discusses three case studies by implementing and executing the different GA operators. Finally, the conclusions and future work are discussed in Section 6.

2 LITERATURE REVIEW

Different approaches have been proposed for ASP including knowledge-based, graph-based, and artificial intelligence (AI)-based methods. The knowledge-based approaches use expert knowledge and experience to form a set of rules and heuristics to generate feasible and optimal assembly sequences. In these approaches, assembly sequences are determined by predefined rules [27]. A set of rules is typically used for ASP searching. Multiple yes-no questions can be applied to identify these rules. For example, Niu et al. [18] developed rules for automatically extracting precedence graphs to form assembly sequences. Dong et al. [5] explored solutions to the ASP problem by using connection-semantic trees to represent non-geometric and geometric rules. Using the knowledge-based strategy, Hsu et al. [8] predicted a near-optimal assembly sequence.

Graph-based approaches represent ASP problems as a graph while using heuristics to search the graph for solutions. Graph-based methods can be used to find the shortest path through the graph for the optimal assembly sequences. There are three types of graph-based approaches: connector, AND/OR, and directed graphs. Directed graphs form product assembly plans at the intermediate level using nodes for a set of elements such as components or sub-assemblies. AND/OR graphs [2] are used to represent component relationships of the assembly. Connector-based graphs have limited functions for joined parts [1]. The graph-based approach requires considerable computational time and resources when the product consists of many parts. An explosion graph of assembly representations can be formed for the transforming rules. The hierarchical graph and the associated part configuration graph were used by Pan et al. [20] for representing a furniture model.

AI-based methods for ASP have shown promise in optimizing the assembly process and improving the efficiency and accuracy of assembly operations. The AI-based methods for minimizing the computation time of ASP problems has been applied extensively in recent years. AI-based methods have been used to develop efficient algorithms and heuristics for solving ASP problems. These methods are used to reduce time to solve complex problems in real-world applications. According to Deepak et al. [4], various AI techniques have been used to optimize ASP using various approaches, including GA, ant colony optimization (ASO), simulated annealing algorithm (SA), particle swarm optimization (PSO), artificial neural networks (ANNs), and artificial immune systems (AIS) [25]. Recent developments in machine learning approaches have also contributed to solving of ASP problems using AI [12].

Applications of GA in ASP include optimizing assembly sequences with multiple objectives, satisfying constraints, handling uncertainties, incorporating disassembly operations, and optimizing assembly sequences for different manufacturing processes [15]. Marian et al. [16] used GA to generate and evaluate different assembly sequences for the optimal sequence in Assembly Sequence Planning Problem (ASPP). The algorithm has a classic structure, but the genetic operators are modified and adapted for the specific set of tasks to fulfill. Kaya et al. [14] introduced an approach to assembly planning by using a multi-objective GA. The approach incorporates tolerance and clearance factors as constraints in assembly planning to consider their effect on different assembly sequences. To enhance the search for feasible solutions, a multi-objective GA is proposed that utilizes a fuzzy weight distribution algorithm to establish diverse fitness functions. Wang et al. [26] presented a multi-objective optimization mathematical model to integrate ASP and Assembly Line Balancing (ALB) for the product plan selection. An improved GA-based approach for ASP optimization was proposed by Peng et al. [21] in a mechanical assembly system with multiple objectives.

Crossover operators are a key component of GA for generating solutions by combining information from parent solutions. Using different operators can result in different search behaviors to affect the convergence rate and quality of solutions [10]. The advantage of comparing different crossover operators is to identify the best operator for a given problem. Several approaches can be used to select the crossover operator with a better performance in GA. One common approach is to compare the fitness values of solutions using different crossover operators. Another approach is to evaluate the convergence rate of GA using different crossover operators. Additionally, statistical tests can be used to compare the performance of different crossover operators.

Although GA has been extensively applied in ASP, there is a lack of research on the effect of different crossover operators in GA on the ASP problem. Umbarkar and Sheth [24] reviewed different types of crossover operators that could be employed in GA. It was found that researchers often use GA crossover operators that have been successful in similar applications for new problems. Crossover operators typically are built upon existing ones with additional changes to increase their effectiveness. To find the best crossover operator for a new problem, it is suggested to first examine similar problems that have been solved by GA and various crossover operators. It is also important to gain an understanding of the problem search space and its modality extremes. Hussain et al. [9] applied GA to solve a traveling salesman problem using three crossover operators, including Partially-Mapped Crossover (PMX), Order Crossover (OX), and a proposed operator called CX2. These operators were tested in an experiment, it was found that CX2 outperformed the other two operators. The operators were also applied in twelve benchmark instances to evaluate their global performance. Xin et al. coded an acyclic serial-parallel sequence with the original and mutation operators for the product assembly of a satellite [28].

In summary, GA is an important tool for solving ASP problems to handle complex problems and explore the search space to meet specific requirements. However, one of the critical factors that influence the performance of GA in ASP is the choice of crossover operators. The crossover operator is a genetic operator to combine parent chromosomes to produce one or more offspring chromosomes. There are several types of crossover operators, including Order Crossover (OX), Cycle Crossover (CX), Partially-Mapped Crossover (PMX), and Position-Based Crossover (PBX). Each type of crossover has its strengths and weaknesses and can be customized according to the specific requirements of the problem. Therefore, this research aims to compare and evaluate the performance of different crossover operators for GA to determine the best solution for ASP. This comparison study can gain insights into strengths and weaknesses of each approach to identify the best algorithm for this particular optimization problem.

3 MODELING THE ASSEMBLY SEQUENCE PLANNING PROBLEM

Objective function and constraints are defined for the optimal search of product assembly sequences.

3.1 Constraints

3.1.1 Liaison/Contact Constraint

Liaison or contact constraint represents a part contact with other parts in the product. The contact matrix of an n-parts product is an *n x n* matrix defined in Equation (3.1), where if part P_i has contact with part P_j, $C_{P_iP_j}=1$*, otherwise* $C_{P_iP_j}=0$*.*

$$Contact\ Matrix = \begin{bmatrix} c_{p_1p_1} & c_{p_1p_2} & \cdots & c_{p_1p_{n-1}} & c_{p_1p_n} \\ c_{p_2p_1} & c_{p_2p_2} & & c_{p_2p_{n-1}} & c_{p_2p_n} \\ \vdots & & \ddots & \vdots & \\ c_{p_{n-1}p_1} & c_{p_{n-1}p_2} & \cdots & c_{p_{n-1}p_{n-1}} & c_{p_{n-1}p_n} \\ c_{p_np_1} & c_{p_np_2} & & c_{p_np_{n-1}} & c_{p_np_n} \end{bmatrix} \tag{3.1}$$

3.1.2 Geometrical constraint

Geometrical constraint requires a collision-free path in the product assembly, represented by an interference matrix in Equation (3.2). Six principal directions axes $\{\pm x, \pm y, \pm z\}$ are used in this study to represent part movements, where $I_{P_iP_j}$ represents the interference of parts P_j and P_i, $P_{ij}=1$ ($j \in [1,n]$), otherwise $P_{ij}=0$.

$$IM_k = \begin{bmatrix} I_{p_1p_1} & I_{p_1p_2} & \cdots & I_{p_1p_{n-1}} & I_{p_1p_n} \\ I_{p_2p_1} & I_{p_2p_2} & & I_{p_2p_{n-1}} & I_{p_2p_n} \\ \vdots & & \ddots & \vdots & \\ I_{p_{n-1}p_1} & I_{p_{n-1}p_2} & \cdots & I_{p_{n-1}p_{n-1}} & I_{p_{n-1}p_n} \\ I_{p_np_1} & I_{p_np_2} & & I_{p_np_{n-1}} & I_{p_np_n} \end{bmatrix} \tag{3.2}$$

$D_d(P_i \to S_h)$ is defined for moving parts P_1 to P_h as follows.

$$D_d(P_i \to S_h) = \sum_{j=1}^{i-1} I_{p_ip_j}^d \tag{3.3}$$

Equation (3.3) requires that there is at least one direction $D_d(P_i \to S_h) = 0$, otherwise, part P_i cannot be moved from its current position.

3.1.3 Precedence constraint

The precedence constraint refers to the order of components being assembled to ensure parts to be assembled before certain components, represented by Equation (3.4). For instance, a nut must be assembled after the screw.

$$Precedence\ Matrix = \begin{bmatrix} P_{p_1p_1} & P_{p_1p_2} & \cdots & P_{p_1p_{n-1}} & P_{p_1p_n} \\ P_{p_2p_1} & P_{p_2p_2} & & P_{p_2p_{n-1}} & P_{p_2p_n} \\ \vdots & & \ddots & \vdots & \\ P_{p_{n-1}p_1} & P_{p_{n-1}p_2} & \cdots & P_{p_{n-1}p_{n-1}} & P_{p_{n-1}p_n} \\ P_{p_np_1} & P_{p_np_2} & & P_{p_np_{n-1}} & P_{p_np_n} \end{bmatrix} \tag{3.4}$$

where $P_{ij}=1$ if part *j* is assembled before part *i*, otherwise, $P_{ij}=0$.

The precedence matrix plays a crucial role in assembly sequence planning, particularly when certain parts need to be assembled before others. The precedence matrix indicates dependencies of parts that must be assembled before others. By utilizing the precedence matrix, assembly planning can effectively form the correct assembly order.

3.2 Objective Function

Different measures can be used in searching for optimal sequences to reduce product assembly time and cost. In this research, the minimum change of assembly orientations and tools is applied in achieving the objective. The objective function is as follows.

$$F = ((2 \times S) - (w1 \times N_{or}) - (w2 \times N_t))/Denominator;$$
$$Denominator = ((2 \times m) + (L_{coeff} \times liaisonIndex)); \tag{3.5}$$

where S is the total number of parts in a product, N_{or} is the number of orientation changes, W_1=0.5, W_2=0.5, and N_t is the number of tool changes. In other words, N_{or} refers to the change of operation directions required for assembling components. Moreover, N_t represents switches between tools used in the assembly process. A tool change is required when two consecutive assembled parts are different.

A penalty is employed in the denominator, where "*m*" represents the total number of interferences, and "*liaisonIndex*" is the total number of violations of the liaison/contact constraint. The "*liaisonIndex*" represents constraints related to the contact between parts in the product. The liaison/contact constraint ensures that parts are properly connected or in contact with each other. The "*m*" in the denominator represents the total number of interferences, which refers to any collisions or overlaps between parts during the assembly process. These two factors play a role in the penalty employed in the denominator of the objective function (Equation 3.5). The penalty considers the number of interferences and violations of the liaison/contact constraint in the assembly sequence. Overall, the "*liaisonIndex*" and "*m*" in the objective function reflect the importance of ensuring proper part contact and minimizing interferences during the assembly process. By incorporating the penalty, the objective function seeks to minimize the number of interferences and violations, leading to an optimized assembly sequence that reduces assembly time and cost with the minimum change of orientations and tools.

A framework of the ASP approach is shown in Figure 1.

4 PROPOSED METHOD

4.1 Genetic Algorithm (GA)-based Method for ASP

The pseudo-code of GA is shown in Figure 2. The GA algorithm begins at initialization to generate random chromosomes or sequences, *npop*. The fitness values of these sequences are then calculated. Populations are sorted accordingly, as shown in Pseudo-code 2 in Figure 3. The optimal population is selected from the sorted populations in the main loop of the algorithm. The loop continues until the number of function evaluations reaches a certain value. In the crossover step, the roulette wheel selection mechanism chooses parents. One of the four crossover types is applied to generate new populations, *popc*. The mutation operation is then applied to generate other sets of populations called *popm*, whose fitness values are evaluated. All three Populations, i.e., *npop*, *popc*, and *popm*, are finally combined and sorted to decide the optimal sequence.

The objective function is searched based on the constraint matrices of the product. The geometric feasibility constraint determines the geometric feasibility of a sequence. The liaison feasibility constraint is used to determine the liaison index, base component, and a flag indicating whether the sequence is feasible. Furthermore, the precedence feasibility constraint is used to determine the number of precedence feasibility violations.

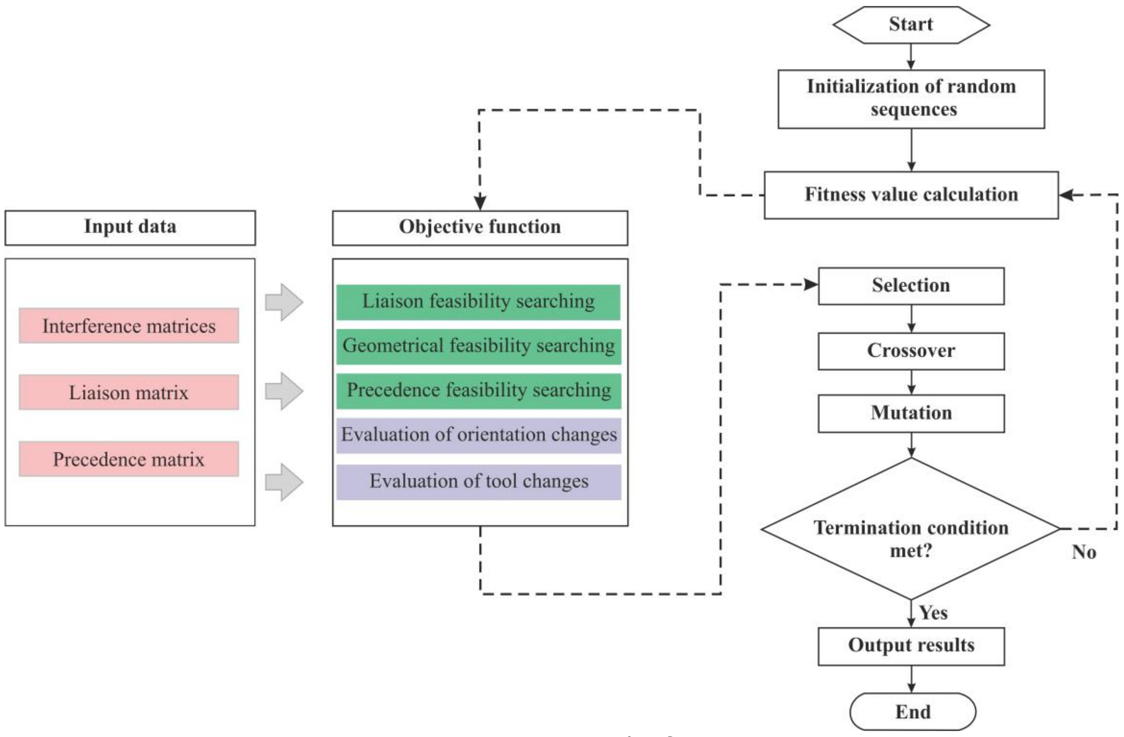
Start
Initialization of random sequences
Fitness value calculation
Input data
Interference matrices
Liaison matrix
Precedence matrix
Objective function
Liaison feasibility searching
Geometrical feasibility searching
Precedence feasibility searching
Evaluation of orientation changes
Evaluation of tool changes
Selection
Crossover
Mutation
Termination condition met?
No
Yes
Output results
End

Figure 1: Framework of ASP.

```
% Defining GA Parameters
Selection of the number of populations (npop) and iterations
% Initialization
For i=1: number of runs
    Randomly initializing the chromosomes (npop)
    Fitness value calculation
    Sorting the initialized populations based on their fitness values (pseudo-code 2)
    % Main loop
    while $NFE \leq MaxNFE$:
        it = it + 1;
        % Crossover
        for $Number\ of\ chromosomes/2$:
            Choosing parents using a selection mechanism
            Applying one of the four crossover types and generating new populations (popc)
            Fitness value calculation of popc by calling the objective function (pseudo-code 2)
        end for
        % Mutation
        for $Number\ of\ chromosomes/2$:
            Applying mutation operator and generating new populations (popm)
            Fitness value calculation of popm by calling the objective function (pseudo-code 2)
        end for
        $NFE \leftarrow NFE + 2$
        Combining the lately generated chromosomes with the previous ones
        Sort chromosomes from the best to the worst based on their fitness values
        Save the best half and discard the remaining.
    end while
end for
```

Figure 2: The main loop of GA (Pseudo-code 1)

```
% Objective function
Fitness value ← ObjFunction(Sequence)
Input: sequence for GA generated randomly
Output: fitness value of the sequence
% Input information
Importing matrices of the product representation

% Checking the feasibility of the constraints for the sequence and finding m as well as liaisonIndex
[m, gFlag] ←GeomFConstraint(Sequence);
[liaisonIndex, bese component, lFlag] ←LiaisonFConstraint(Sequence);
[nPV,pFlag]←PrecFConstraint(Sequence)

% Evaluation of the solution
$N_{or}$ ← nOr(Sequence, IMs)
$N_t$ ← nTool(Sequence, tool table)
% objective function
Denominator = ((2*m)+(2*liaisonIndex));
If Denominator ==0
    Denominator=1
end if
F=-((2*S)-($w_1$*$N_{or}$)-($w_2$*$N_t$))/Denominator;
end function
```

Figure 3: The objective function search (Pseudo-code 2)

4.2 GA Crossover Operators

The crossover represents one of the three main operations in GA, i.e., mutation, crossover, and selection mechanism. Four GA crossover operators are examined, namely Cycle crossover (CX), Position-based crossover (PBX), Order crossover (OX), and Partially-mapped crossover (PMX) [7],[14], to evaluate the performance of GA for ASP. There are two common GA mechanisms to select parents and survivors: the roulette wheel and tournament selection. This study uses the roulette wheel selection method as it is simple to implement, computationally efficient, and has smooth selection pressure to avoid the premature convergence of solutions.

4.1.1 Cycle Crossover (CX)

The cycle crossover conserves as much information as possible for the absolute positions of elements [19],[23]. CX operators divide elements into cycles. When parents of entities are in alignment with one another, elements will form a cycle. To generate the offspring, alternative cycles are chosen from each parent's permutation as shown in Figure 4. To construct cycles, it is necessary to identify cyclical elements of the offspring and copy each one into the offspring [13].

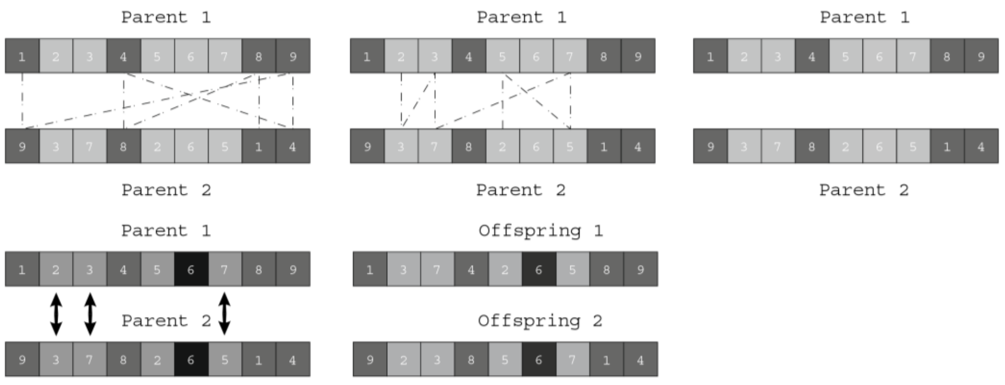

Figure 4: Operations of CX crossover.

4.1.2 Position-based crossover (PBX)

Position information is maintained throughout recombination of the PBX operator. In this operator, a sequence is constructed by selecting several random locations and one parent as shown in Figure 5. Those elements have the same parent as those in the positions. The remaining elements are inherited in the order they appear in the second parent after removing elements of the second parent in those random locations of the first parent. Accordingly, its elements are chosen at random, not based on their locations within a parent.

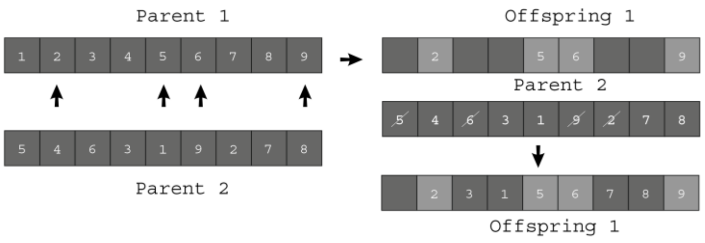

Figure 5: PBX crossover procedure.

4.1.3 Order crossover (OX)

This operator is for solving order-based permutation problems [3]. It involves copying the first portion of the first parent into an empty offspring at random. From the first element of the second parent, the remaining numbers are copied to the new child, and unused numbers are removed from the subsequent offspring as shown in Figure 6. A second offspring can be created by switching the functions of the parents.

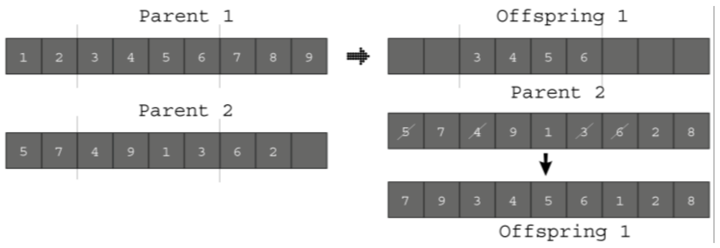

Figure 6: Operations of OX crossover.

4.1.4 Partially Mapped Crossover (PMX)

In this crossover operator, two points are selected [6]. From one parent to the other, elements are replaced between these two points as shown in Figure 7. It is necessary to find and replace the corresponding element from the other parent if it is already present between two crossover points of the offspring. The second parent must also contain a corresponding element if it is also present between the crossover points of the first chromosome. This process is continued until there are no more corresponding numbers between crossover points.

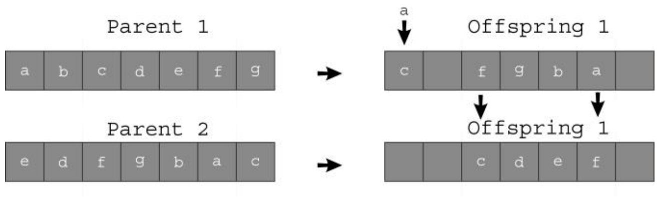

Figure 7: Representation of PMX crossover

5 RESULTS AND DISCUSSIONS

5.1 Case Study 1

The proposed method is applied in ASP for a 22-component product as shown in Figure 8 [15]. The optimization search is executed using four different GA operators, each for 30 independent runs. Statistical data of the 30 runs for each algorithm are extracted. The proposed algorithms are coded including the four crossover types. This gives us a full control over the implementation process and allows us to customize the process to match the research objectives.

Descriptive statistical data such as best, worst, mean, and standard deviation are employed to gain insights into the behavior and quality of solutions obtained by the algorithms. The 'best' values represent the highest-quality solutions obtained among the 30 runs, while the 'worst' measure indicates the poorest solution achieved. The 'mean' provides an average solution quality, serving as an overall performance measure, and the 'standard deviation' quantifies the variability in the solution quality for the robustness of the algorithms. Although such characteristics are not standardized in the literature, they are widely accepted and utilized for evaluating optimization algorithms. In this research, they enable objective comparisons among different crossover operators and offer a quantitative assessment of the algorithm performance. Running each algorithm 30 times allows for capturing the stability, convergence, and consistency of the algorithm across multiple runs. By incorporating these statistical measures, a rigorous evaluation framework is established to assess the effectiveness and efficiency of each genetic algorithm in optimizing the ASP problem.

The best values for the objective function, mean values, standard deviation (STD), and worst values of each of the runs are shown in Table 1. It shows that OX-GA has the optimum best, mean, and worst values. An algorithm is considered more robust when it shows larger values for best, mean, and worst fitness values while having the least standard deviation. Fewer values for STD show more chance to obtain similar results when reusing the algorithm. For the best fitness values, after PMX-GA, PBX-GA and OX-GA reach values of 8.723 and 9.264, respectively.

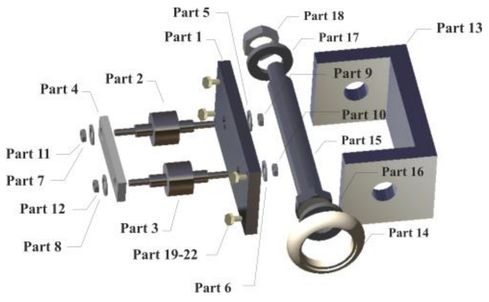

Figure 8: Product of Case Study 1.

Algorithm	*Best*	*Mean*	*STD*	*Worst*	*Feasibility*
OX-GA	**33.3**	**24.015**	9.264	**8.6**	✓
PBX-GA	**33.3**	13.379	8.723	4.287	✓
PMX-GA	32.5	10.361	7.423	3.48	✓
CX-GA	16.75	5.543	**3.577**	2.175	x

Table 1: Statistical data and feasibility status of results obtained by GAs for case study 1 (Number of iterations:300, Number of populations: 200, Number of runs: 30).

The mean convergence curves of the algorithms are shown in Figure 9. It shows that OX-GA has the fastest convergence curve. Although PMX-GA outperforms the PBX-GA at first iterations, it converges to a lower fitness value. The CX-GA could not form any feasible sequences to meet the constraints. The best sequences obtained by the algorithms are listed in Table 2. All the sequences start from part 1 as it is the part with the maximum number of connections with adjacent parts. This result indicates that among four GA crossover operators, OX is the best candidate for ASP.

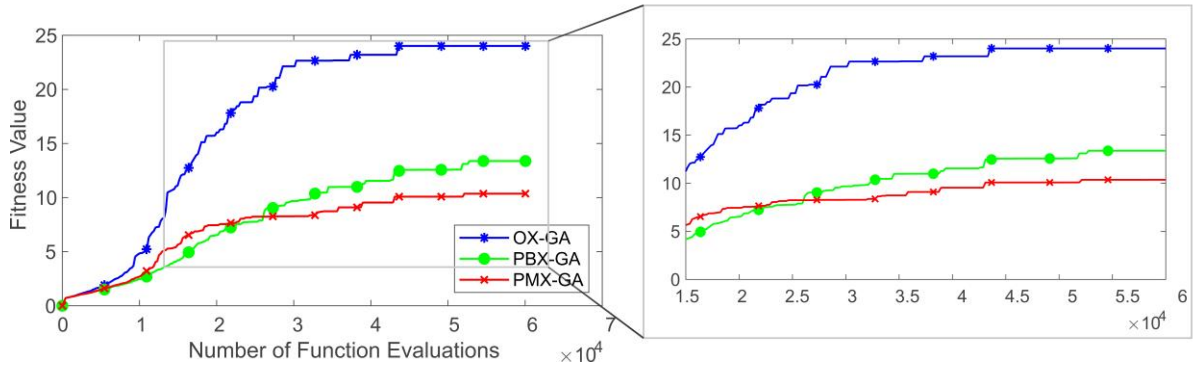

Figure 9: Average convergence curves of GAs with different crossover operators.

Algorithm	*Obtained sequences from the optimization*	*Feasibility*
OX-GA	**1,2,5,9,3,6,10,13,22,19,21,20,4,8,12,7,11,14,16,15,17,18**	✓
PBX-GA	1,2,5,9,3,6,10,13,21,19,22,20,4,8,12,7,11,14,16,15,17,18	✓
PMX-GA	1,2,5,9,3,6,10,4,7,11,13,21,20,19,22,8,12,14,16,15,17,18	✓
CX-GA	1,2,5,9,3,6,10,13,4,7,11,14,16,15,17,18,8,12,22,21,20,19	x

Table 2: Obtained sequences with the lowest fitness values by different algorithms for case study 1 (Number of iterations: 300, Number of populations: 200).

As shown in Table 2, despite conducting 30 runs with 300 iterations and a population size of 200, CX-GA is unable to find any feasible assembly sequences. The specific characteristics of this crossover operator may limit its ability to explore the search space thoroughly and effectively, failing to discover feasible assembly sequences. The lack of feasibility in CX-GA could also be attributed to the specific characteristics of this crossover operator. Likely, CX-GA combined data from parent individuals results in incompatible or invalid assembly sequences. As a result, the algorithm is unable to generate feasible solutions that satisfy assembly constraints and requirements. This limitation highlights the importance of carefully selecting and evaluating crossover operators in the GA for assembly sequence planning. While other crossover types demonstrate better performance in finding feasible assembly sequences, the inability of CX-GA to achieve feasibility emphasizes the need for further investigation and the development of alternative approaches to address this challenge.

5.2 Case Study 2

The product of case study 2 is depicted in Figure 10 [15]. The ASP optimization search is conducted using four GA crossover operators, each with 30 independent runs. Statistical data from the 30 runs for each algorithm are extracted and presented in Table 3, which includes the best values for the objective function, mean values, STD, and worst values. The results indicate that the OX-GA algorithm achieves the maximum best, mean, and worst fitness values, while having the least STD value. Hence, the OX-GA algorithm demonstrates its robustness in finding the optimal solution for the ASP problem.

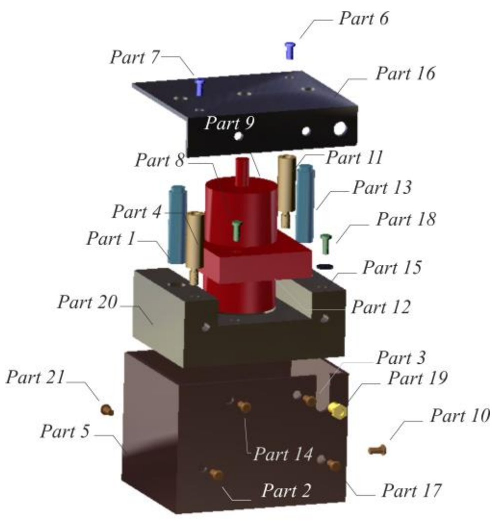

Figure 10: Product of Case Study 2.

Figure 11 displays the mean convergence curves of the various algorithms, revealing that the OX-GA algorithm has the quickest convergence. Although the CX-GA algorithm initially outperforms PBX-GA and PMX-GA, it eventually converges to a lower fitness value. Table 4 presents the best sequences generated by each algorithm, all of them start at part 5 due to their maximal number of connections with adjacent parts. The results suggest that OX-GA is the optimal choice for ASP among the four GA crossover operators as it has the maximum mean, maximum worst, and least STD while having the same best value obtained by the PBX-GA and PMX-GA algorithms.

Algorithm	*Best*	*Mean*	*STD*	*Worst*	*Feasibility*
OX-GA	**37.2**	**37.093**	**0.211**	**36.5**	✓
PBX-GA	**37.2**	33.218	7.446	18.35	✓
PMX-GA	**37.2**	34.358	6.412	18.2	✓
CX-GA	36.9	30.59	8.336	17.9	✓

Table 3: Statistical data and feasibility status of results for case study 2 (Number of iterations: 300, Number of populations: 200, Number of runs: 30).

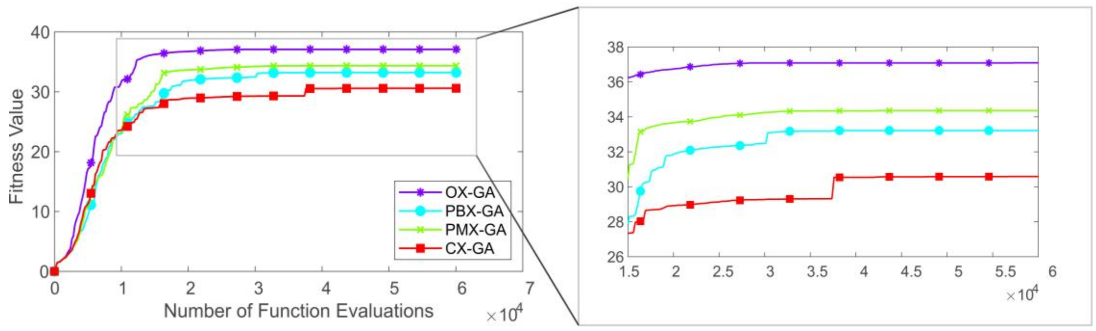

Figure 11: Average convergence curves of GAs with different crossover operators.

Algorithm	*Obtained sequences from the optimization*	*Feasibility*
OX-GA	**5,20,12,1,13,4,11,15,18,9,8,16,7,6,21,10,2,17,3,14,19**	✓
PBX-GA	5,20,12,1,13,4,11,15,18,9,8,16,6,7,10,21,3,14,17,2,19	✓
PMX-GA	5,20,12,1,13,11,4,15,18,8,9,16,7,6,10,21,17,2,3,14,19	✓
CX-GA	5,20,12,1,13,4,11,15,18,8,9,16,7,6,2,17,19,14,3,10,21	✓

Table 4: Sequences with the lowest fitness values by different algorithms for case study 2 (Number of iterations: 300, Number of populations: 200).

5.3 Case Study 3

Figure 10 [4] shows the product of case study 3. Like the previous cases, to optimize the ASP, four different GA crossover operators are employed, each runs independently for a total of 30 times. Outcomes of these runs are analyzed and summarized in Table 5, which provides the best objective function values, mean values, STD, and worst values. From the results, it is observed that the OX-GA algorithm achieves the highest fitness values for the objective functions, in terms of the best, mean, and worst values.

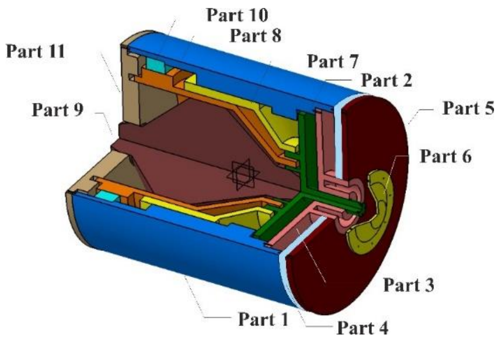

Figure 12: Product of Case Study 3.

The mean convergence curves of the different algorithms are depicted in Figure 13, illustrating that, once again, the OX-GA algorithm exhibits the fastest convergence rate. While the CX-GA algorithm initially shows superior performance compared to PBX-GA and PMX-GA, it eventually reaches a lower fitness value. Table 5 shows the best sequences generated by each algorithm. The results suggest that among the four GA crossover operators, OX-GA is the optimal choice for ASP. This conclusion is supported by its maximum best, maximum mean, and maximum worst fitness values obtained by this crossover operator.

Algorithm	*Best*	*Mean*	*STD*	*Worst*	*Feasibility*
OX-GA	**22**	**21.653**	0.138	**21.6**	✓
PBX-GA	**22**	21.613	**0.073**	**21.6**	✓
PMX-GA	**22**	21.613	**0.073**	**21.6**	✓
CX-GA	21.6	21.520	0.163	21.2	✓

Table 5: Statistical data and feasibility status of results for case study 3 (Number of iterations: 300, Number of populations: 200, Number of runs: 30).

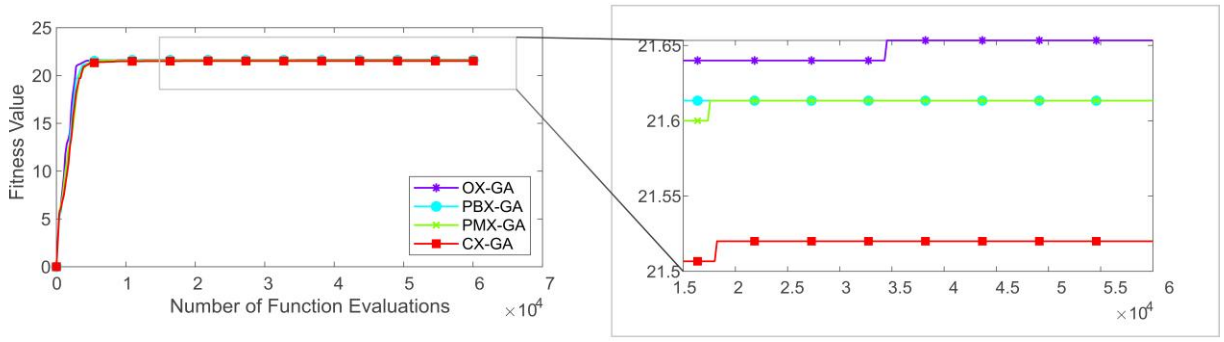

Figure 13: Average convergence curves of GAs with different crossover operators.

Algorithm	*Obtained sequences from the optimization*	*Feasibility*
OX-GA	**11,9,8,7,10,1,2,3,4,5,6**	✓
PBX-GA	11,9,8,10,7,1,2,3,4,5,6	✓
PMX-GA	11,9,8,10,7,1,2,3,4,5,6	✓
CX-GA	9,8,10,7,1,2,3,4,5,6,11	✓

Table 6: Sequences with the lowest fitness values by different algorithms for case study 3 (Number of iterations: 300, Number of populations: 200).

In summary, the OX-GA algorithm performs well for all three cases in achieving the best statistical outcomes, obtaining the highest best, highest mean, and highest worst fitness values for all three cases.

One limitation of our investigation is the challenge of determining the most suitable type of crossover in the genetic algorithm. Although the OX crossover outperforms other types in our experiments, different types of assembly problems or variations in problem instances may require different crossover operators for better performance. To address this limitation, the future research will focus on developing robust methodologies for automatically selecting or adapting crossover operators based on problem characteristics or problem-specific knowledge. Such approaches could involve incorporating machine learning techniques to learn relationships between the problem instances and the performance of different crossover operators. By leveraging this approach, the genetic algorithm can dynamically adapt to the specific assembly sequence planning problem for enhancing its effectiveness and efficiency.

This research holds practical implications for assembly line planning, offering the potential for cost savings and increasing efficiency for manufacturers. By implementing the optimal assembly sequence planning obtained by the proposed approach, manufacturers can achieve cost savings by minimizing unnecessary movements and assembly time, reducing waste and rework, and optimizing resource allocation.

6 CONCLUSIONS

This paper conducts a comparison study on different GA crossover operators to examine their effect on the performance of ASP. An objective function is proposed for the minimum changes of the assembly orientations and tools in searching for the optimal assembly solution. The feasibility of the sequences is considered based on the product constraints. For the two case studies, four variants of GA are executed independently over 30 times. The reliability of each GA variant is assessed by statistical data of the results, including the best, mean, worst, and STD.

The comparison of four different approaches for assembly sequence planning highlights distinct performance differences. The PMX-GA and CX-GA approaches yield suboptimal results in the first case study, with CX-GA even failing to generate feasible assembly sequences. In the second case study, PBX-GA, PMX-GA, and CX-GA struggle to match the good performance of OX-GA in terms of mean, standard deviation, and worst fitness values. Additionally, the CX-GA approach is outperformed by other algorithms in the third case study, while OX-GA obtains the best results in terms of best, mean, and worst measures. Overall, OX-GA consistently demonstrates the superior performance, making it the preferred choice in GA for optimizing ASP due to its ability to find optimal or near-optimal assembly sequences.

As shown in this research, the genetic algorithm must be tailored and tuned for the problem of ASP, and results highly depend on the selection of crossover operators. Furthermore, the performance of the genetic algorithm is dependent on the choice of the selection mechanism. This is because selection mechanisms influence the diversity of the population, which is necessary to maintain genetic variability and enable the genetic algorithm to explore the search space more effectively. Therefore, in the future work, we will explore machine learning approaches, such as Reinforcement Learning (RL) for a more efficient and adaptable approach to solving the ASP problem. By using RL to dynamically select the type of crossovers, the algorithm can adapt to changes in the problem or search space, leading to better performance and faster convergence to optimal solutions. Additionally, it will enable us to take advantage of both RL and GA strengths, leading to improved performance and robustness of ASP.

ACKNOWLEDGMENTS

The first and third authors would like to acknowledge that this research is supported by the Discovery Grants from the Natural Sciences and Engineering Research Council (NSERC) of Canada, Graduate Enhancement of Tri-Council Stipends (GETS) program from the University of Manitoba, University of Manitoba Graduate Fellowships (UMGF) and Mitacs Lab2Market program.

Nima Geran Malek, https://orcid.org/0000-0003-3725-738X
Armin Dadras Eslamlou, https://orcid.org/0000-0003-3460-8367
Qingjin Peng, https://orcid.org/0000-0002-9664-5326
Shiping Huang, https://orcid.org/0000-0002-0092-1753

REFERENCES

[1] Abdullah, M.A.; Ab Rashid, M.F.F.; and Ghazalli, Z.: Optimization of assembly sequence planning using soft computing approaches: a review, Archives of Computational Methods in Engineering, 26, 2019, 461-474. https://doi.org/10.1007/s11831-018-9250-y
[2] Bahubalendruni, M.R.; and Biswal, B.B.: A review on assembly sequence generation and its automation, Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, 230(5), 2016, 824-838. https://doi.org/10.1177/0954406215584633
[3] Davis, L.; Handbook of genetic algorithms, 1991.
[4] Deepak, B.B.V.L.; Bala Murali, G.; Bahubalendruni, M.R.; and Biswal, B.B.: Assembly sequence planning using soft computing methods: a review, Proceedings of the Institution of Mechanical Engineers, Part E: Journal of Process Mechanical Engineering, 233(3), 2018, 653-683. https://doi.org/10.1177/0954408918764459
[5] Dong, T.; Tong, R., Zhang, L. and Dong, J.: A knowledge-based approach to assembly sequence planning, The International Journal of Advanced Manufacturing Technology, 32(11-12), 2007, 1232-1244. https://doi.org/10.1007/s00170-006-0438-1.

[6] Eiben, A. E.; and J. E. Smith: Introduction to evolutionary computing. Springer, 2003. https://doi.org/10.1007/978-3-662-44874-8
[7] Holsapple, C.W.; Jacob, V.S.; Pakath, R.; and Zaveri, J.S.: A genetics-based hybrid scheduler for generating static schedules in flexible manufacturing context, IEEE Trans. Systems Man Cyber., 23(4), 1993, 953—972. https://doi.org/10.1109/21.247881
[8] Hsu, Y.Y.; Tai, P.H.; Wang, M.W.; and Chen, W.C.: A knowledge-based engineering system for assembly sequence planning, The International Journal of Advanced Manufacturing Technology, 55, 2011, 763-782. https://doi.org/10.1007/s00170-010-3093-5
[9] Hussain, A.; Muhammad, Y.S.; Nauman Sajid, M.; Hussain, I.; Mohamd Shoukry, A.; and Gani, S.: Genetic algorithm for traveling salesman problem with modified cycle crossover operator, Computational Intelligence and Neuroscience, 2017, 1-7. https://doi.org/10.1155/2017/7430125
[10] Kaveh, A.; Dadras, A.; and Malek, N.G.: Buckling load of laminated composite plates using three variants of the biogeography-based optimization algorithm, Acta Mechanica, 229, 2018, 1551-1566. https://doi.org/10.1007/s00707-017-2068-0
[11] Kaveh, A.; Dadras Eslamlou, A.; Geran Malek, N.; and Ansari, R.: An open-source computational framework for optimization of laminated composite plates, Acta Mechanica, 231, 2020, 2629-2650. https://doi.org/10.1007/s00707-020-02648-0
[12] Kaveh, A.; Dadras Eslamlou, A.; Javadi, S.M.; and Geran Malek, N.: Machine learning regression approaches for predicting the ultimate buckling load of variable-stiffness composite cylinders, Acta Mechanica, 232, 2021, 921-931. https://doi.org/10.1007/s00707-020-02878-2
[13] Kaveh, A.; Rahmani, P.; and Dadras Eslamlou, A.: An efficient hybrid approach based on Harris Hawks optimization and imperialist competitive algorithm for structural optimization, Engineering with Computers, 38, 2022, 2, 1555-1583, https://doi.org/10.1007/s00366-020-01258-7
[14] Kaya, M.: The effects of two new crossover operators on genetic algorithm performance, Applied Soft Computing, 11(1), 2011, 881-890. https://doi.org/10.1016/j.asoc.2010.01.008
[15] Lu, C.; Wong, Y.S.; and Fuh, J.Y.H.: An enhanced assembly planning approach using a multi-objective genetic algorithm. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 220(2), 2006, 255-272. https://doi.org/10.1243/09544054JEM359
[16] Marian, R.M.; Luong, L.H.; and Abhary, K.: A genetic algorithm for the optimization of assembly sequences, Computers & Industrial Engineering, 50(4), 2006, 503-527. https://doi.org/10.1016/j.cie.2005.07.007
[17] Masehian, E.; and Ghandi, S.: Assembly sequence and path planning for monotone and nonmonotone assemblies with rigid and flexible parts, Robotics and Computer-Integrated Manufacturing, 72, 2021, 102180. https://doi.org/10.1016/j.rcim.2021.102180
[18] Niu, X.; Ding, H.; and Xiong, Y.: A hierarchical approach to generating precedence graphs for assembly planning, International Journal of Machine Tools and Manufacture, 43(14), 2003, 1473-1486. https://doi.org/10.1016/S0890-6955(03)00168-8
[19] Oliver, I.; Smith, D.; and J. R. Holland: Study of permutation crossover operators on the traveling salesman problem in Genetic algorithms and their applications, Proceedings of the Second International Conference on Genetic Algorithms, 1987, 28-31.
[20] Pan, W.; Wang, Y.; and Chen, X.D.: Domain knowledge based non-linear assembly sequence planning for furniture products, Journal of Manufacturing Systems, 49, 2018, 226-244. https://doi.org/10.1016/j.jmsy.2018.10.003
[21] Peng, Y.A.N.G.; Ji-hong, L.I.U. and Qiang, G.U.A.N.: An improved genetic algorithm for assembly sequence optimization, Computer Integrated Manufacturing System, 8(6), 2002, 0
[22] Prasad, V.S.S.; Gulivindala, A.K.; Uppada, S.; Matta, V.R.; Raju Bahubalendruni, M.V.A.; and Biswal, B.B.: A Design for Assembly Framework Based on Subassembly Detection Method, In Recent Advances in Manufacturing Modelling and Optimization: Select Proceedings of RAM, 2022, 511-519. https://doi.org/10.1007/978-981-16-9952-8_43

[23] Starkweather, T.; McDaniel, S.; Mathias, K. E.; Whitley, L. D.; and Whitley, C.: A Comparison of Genetic Sequencing Operators. in ICGA, 1991, 69-76.
[24] Umbarkar, A.J.; and Sheth, P.D.: Crossover operators in genetic algorithms: a review. ICTACT Journal on Soft Computing, 06, 2015, 1083-1092. https://doi.org/10.21917/ijsc.2015.0150
[25] Wang, D.; Shao, X.; and Liu, S.: Assembly sequence planning for reflector panels based on genetic algorithm and ant colony optimization, The International Journal of Advanced Manufacturing Technology, 91, 2017, 987-997. https://doi.org/10.1007/s00170-016-9822-7
[26] Wang, H.S.; Che, Z.H.; and Chiang, C.J.: A hybrid genetic algorithm for multi-objective product plan selection problem with ASP and ALB, Expert systems with applications, 39(5), 2012, 5440-5450. https://doi.org/10.1016/j.eswa.2011.11.041
[27] Wu, M.; Prabhu, V.; and Li, X.: Knowledge-based approach to assembly sequence planning, Journal of Algorithms & Computational Technology, 5(1), 2011, 57-70. https://doi.org/10.1155/2013/908316
[28] Xin, L.; Jianzhong, S.; and Yujun, C.: An efficient method of automatic assembly sequence planning for aerospace industry based on genetic algorithm, The International Journal of Advanced Manufacturing Technology, 90, 2017, 1307-1315. https://doi.org/10.1007/s00170-016-9449-8
[29] Xing, Y.; and Wang, Y.: Assembly sequence planning based on a hybrid particle swarm optimization and genetic algorithm, International Journal of Production Research, 50(24), 2012, 7303-7312. https://doi.org/10.1080/00207543.2011.648276