ROBOTIC ASSEMBLY SEQUENCE PLANNING AND OPTIMIZATION BY CUCKOO SEARCH ALGORITHM

Atul Mishra1, Sankha Deb2

FMS and Computer Integrated Manufacturing Laboratory, Mechanical Engineering Department,
Indian Institute of Technology Kharagpur, Kharagpur-721302, India.
[1] atulmishra_chdi@yahoo.in
[2] sankha.deb@mech.iitkgp.ernet.in

Abstract
The assembly sequence optimization problem is challenging as number of sequences rises staggeringly with part count making it combinatorially explosive, tedious and time consuming to solve. A Computer-Aided Process Planning (CAPP) system for robotic assembly by a discrete cuckoo search algorithm has been proposed by us to generate feasible and optimal sequences considering minimization of orientation changes, tool/gripper changes, assembly stability and base component location. A case study and comparison results with an established soft computing approach are presented to show its effectiveness. It is further integrated with an upstream assembly product database to extract information on assembly directions, contact details, assembly precedence constraints and tool/grippers. Further the assembly sequence output from CAPP system is integrated with a robot task planner developed using Knowledge Based System capable of automatically generating executable robot level program for performing the assembly.

Keywords: assembly sequence optimization, robot task planning, cuckoo search algorithm, knowledge based system.

1 INTRODUCTION AND LITERATURE REVIEW

Most mechanical products are composed of multiple components necessitating assembly operations to put them together. In recent years, industries are witnessing a boom in application of industrial robots to increase the flexibility and efficiency of assembly operations. There are, however, several critical issues and challenges in robotic assembly process planning. The order or the sequence in which the components are put together affects the time necessary for performing the assembly operations and hence the manufacturing cost of the product. Manually determining the optimal assembly sequence can be a tedious task, which becomes even more challenging as number of components in the assembly increases and consequently the number of feasible assembly sequences possible also increases, which makes it a difficult combinatorial optimization problem to solve. To overcome the above problems, researchers have been attempting to develop methodologies for automating the generation of optimal assembly sequence. Off late, one such methodology that is fast emerging as a promising optimization technique is soft-computing based evolutionary optimization. Various algorithms that have been explored for assembly sequence optimization include Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Ant colony Optimization (ACO), Memetic Algorithm (MA), Imperialist Competitive Algorithm (ICA), Harmony Search, etc.

Hong and Cho [1] developed a robotic assembly sequence optimization approach using Hopfield neural network and expert system to infer assembly constraints from product liaison data. Bonneville et al. [2] proposed a GA approach to generate valid and good assembly plans but its performance was slow, and did not guarantee optimum plans. Mishra and Deb [3] developed a knowledge based system for selection of assembly tools, and a GA based approach to determine optimal assembly sequences considering minimum tool changes and assembly direction changes. Wang et al. [4] proposed an ACO approach for assembly sequence optimization based on least reorientations. Mishra and Deb [5] proposed a GA based parameter meta-optimization of ACO algorithm for solving assembly sequence optimization. Cao and Xiao [6] developed Immune Optimization Algorithm (IOA) based on bionic principles of Artificial Immune Systems (AIS). Assembly sequences were evaluated based on number of components, assembly directions, tools changes and base component location. Chang et al. [7] developed an AIS approach for assembly sequence planning using connector concepts. Wang and Liu [8] developed a PSO approach where assembly cost was subjected to geometrical constraints, local assembly precedence, number of the unstable parts, assembly direction and tool changes, connector changes. Lv and Lu [9] developed a discrete PSO considering tool and orientation changes, and operation type changes and interference times. Gao et al. [10] presented an MA approach, with a chromosome representing an sequence consisting of genes containing part number and direction. They considered times of the direction changes. Zhou et al. [11] combined Bacterial Chemotaxis with GA where assembly sequences were coded as chromosomes and a gene treated as bacterium. Fitness function contained length of longest sub-sequence, orientation and tool changes. Li et al. [12] presented an Improved Harmony Search (IHS) algorithm. An initial harmony memory (HM) established using the opposition based learning (OBL) strategy and a local search strategy were proposed. Changes of orientation and tool and stability criteria were considered in fitness function.

Next to implement the sequence for robotic assembly, the sequence must be converted into a task level plan and a robot level program, which is also a tedious and time consuming task to perform manually. Researchers have been attempting to develop methodologies to automate this task. Rabemanantsoa and Pierre [13] focused on the feasibility of coupling Artificial Intelligence (AI) with robotics for intelligent knowledge based Computer Integrated Manufacturing (CIM). A knowledge based system was used

to generate assembly sequence in tree form from CAD design data and forward chaining inference for generating task level commands. Mosemann and Wahl [14] used AND/OR graphs for sequence representation and by analyzing algorithmically hyperarcs of graphs, complex sequences of assembly operations were decomposed into skill primitives. Unified modeling language (UML) was used to model robot tasks and skill primitives. Thomas and Wahl [15] developed a complete process chain for assembly starting from specification of assembly tasks based on CAD models via assembly sequence planning through task planning and finally task execution. They used algorithmic and graph theoretic approaches. Neto and Mendes [16] proposed an intuitive offline robot programming method in which extraction of motion data from CAD, mapping data from CAD model to real environment and process of automatic robot program generation were done. Stenmark and Malec [17] developed a knowledge integration framework (KIF) (containing robotics ontologies, data repository and services provided for stored knowledge), a robot programming system and a robot task execution system. The KIF provides a planning service transforming an assembly graph to an operation sequence by natural language programming. The natural language service outputs a preliminary form of program statements containing actions and arguments. Actions are mapped to robot program templates describing actions of picking, placing, moving, locating and the arguments mapped to physical objects in the workspace.

From the above literature review, it is evident that soft computing based algorithms hold lot of promise in automating the assembly sequence planning and optimization and should be explored further. To implement the optimal assembly sequence on a robot, it should be integrated with a robot task planner for automatically generating an executable robot program offline. With the above in mind, in this paper a CAPP system has been proposed for robotic assembly using a discrete cuckoo search algorithm to generate feasible and optimal sequences. Furthermore, a robot task level planner has been proposed and implemented by a knowledge-based expert system, CLIPS for offline generation of the robot level program in a commercial robot programming language, INFORM. Finally its implementation has been shown on a commercial offline robot programming and simulation environment, MotoSim EG to demonstrate the working of the proposed system. The performance of the system has been illustrated with the help of a fourteen component assembly.

PlungerPumpAssembly | DirectionMatrix | PrecedenceMatrix | StabilityMatrix

ID	PartNo	PartName	PartCount	BaseComponen	ComponentType	ToolNumber	ToolType
1	1	Base	1	Yes	Functional	1	reconfigurable_gripper
2	2	Screw 1	1	No	Fastener	2	reconfigurable_gripper
3	3	Screw 2	1	No	Fastener	2	reconfigurable_gripper
4	4	Screw 3	1	No	Fastener	2	reconfigurable_gripper
5	5	Washer	1	No	Fastener	3	reconfigurable_gripper
6	6	Bearing 1	1	No	Functional	3	reconfigurable_gripper
7	7	Cam rod	1	No	Functional	4	v-grooved_gripper
8	8	Bearing 2	1	No	Functional	5	reconfigurable_gripper
9	9	Plate	1	No	Functional	5	reconfigurable_gripper
10	10	Bolt set 1	4	No	Fastener	6	tool_mounted_gripper
11	11	Cylinder	1	No	Functional	5	reconfigurable_gripper
12	12	Spring	1	No	Functional	5	reconfigurable_gripper
13	13	Head	1	No	Functional	5	reconfigurable_gripper
14	14	Bolt set 2	3	No	Fastener	6	tool_mounted_gripper

Fig 1: An extract of Assembly Product Database

2 PROPOSED METHODOLOGY FOR ASSEMBLY SEQUENCE PLANNING

This section discusses the format for description of assembly product information, our proposed approach for assembly sequence optimization, automatic offline generation of the robot level program and its implementation on an offline robot programming and simulation environment.

2.1 Assembly Product Database

The Microsoft Access Database Management software has been used to store all relevant information about the assembly product (such as component identifiers, their types i.e. functional part or fastener, part count, assembly direction, precedence and stability matrices, etc.) as well as the database of manufacturing resources (like tools, grippers, etc.). A screenshot of the Assembly Product Database is shown in the figure 1.

2.2 Generation of Optimal Assembly Sequence by Cuckoo Search (CS) algorithm

Cuckoo search (CS) is an optimization algorithm first proposed by Yang and Deb [18] for solving continuous optimization problems. It was inspired by the obligate brood parasitism of some cuckoo species by laying their eggs in the nests of other host bird species. If a host bird discovers this, it either throws away these alien eggs or simply abandons its nest and builds a new nest. Yang and Deb proposed the following three idealized rules: (i) each cuckoo lays one egg at a time, and dumps it in a randomly chosen nest; (ii) the best nests with high quality eggs (solutions) will carry over to next generations; (iii) the number of available host nests is fixed, and a host can discover an alien egg with a probability $p_a \in [0, 1]$. In this case, the host bird can either throw away the egg or abandon the nest to build a new one. For simplicity, this last assumption can be approximated by a fraction p_a of the n nests being replaced by new nests. For a maximization problem, the quality or fitness of a solution can simply be proportional to the objective function. Based on these three rules, the basic steps of the Cuckoo Search (CS) can be summarized as in Figure 2. When generating new solutions $x_i^{(t+1)}$ for cuckoo *i*, a Levy flight is performed as follows.

$$x_i^{t+1} = x_i^t + \alpha \oplus \text{Levy}(\lambda) \quad (1)$$

where $\alpha > 0$ is the step size which should be related to the scales of the problem of interest. In most cases, $\alpha = O(1)$.

Objective function f(x), x = $(x_1, ..., x_d)$T ;
Initial a population of n host nests x_i (i = 1, 2, ..., n);
while (t < MaxGeneration) or (stop criterion);
    Get a cuckoo (say i) randomly by Levy flights;
    Evaluate its quality/fitness F_i;
    Choose a nest among n (say j) randomly;
    if ($F_i > F_j$),
        Replace j by the new solution;
    end
    Abandon a fraction (p_a) of worse nests
    [and build new ones at new locations via Levy flights];
    Keep the best solutions (or nests with quality solutions);
    Rank the solutions and find the current best;
end while
Post process results and visualization;

Figure 2: Cuckoo Search Algorithm.

The product $\oplus$ means entry wise multiplications. Levy flights essentially provide a random walk while their random steps are drawn from a Levy distribution for large steps given below which has an infinite variance with an infinite mean.

$$\text{Levy} \sim \text{u} = t^{-\lambda}, (1 < \lambda \leq 3) \quad (2)$$

Here the consecutive jumps/steps of a cuckoo essentially form a random walk process which obeys a power law step length distribution with a heavy tail.

Proposed discrete CS algorithm for assembly sequence optimization

The main challenge in applying CS algorithm for assembly sequence optimization is continuous nature of the basic CS of Yang and Deb [18]. The search space of basic CS is a real space domain, whereas in assembly sequence optimization, the search space is discrete. Therefore, some modifications have been proposed by us in the CS algorithm, as given below.

Proposed approach for Population Generation and Evaluation

For the given problem of assembly sequence optimization, a random population of individuals (i.e. cuckoo eggs representing assembly sequences) is generated which can be feasible or infeasible. This population is evaluated using a fitness function (*FF*) given in equation 3, consisting of number of tool changes (n_t) direction changes (n_d), base component location in assembly sequence (*B*), and stability index (*SI*). Different parts need to be assembled from different directions. Moreover, the robot manipulator needs to use different assembly tools/grippers for grasping and manipulating the parts depending on their type and geometry. Therefore changes in assembly directions and tools are inevitable. Hence, the criteria chosen by us for assembly sequence optimisation include minimization of both direction changes and tool/gripper changes so as to minimize the assembly time and cost. Stability of the parts in subassembly/assembly is also very important as instability will cause the parts to separate from each other. Therefore, maximization of the stability of the parts has chosen as another criterion for optimization. Further, in an assembly sequence, the base part is the part with which assembly starts. Hence in the assembly sequence, we have ensured that the first component in the assembly sequence is the base component. To ensure feasibility of final individuals, a feasibility criterion is incorporated that is measured in terms of number of feasibility violations (*FV*) due to components' locations in the sequence.

$$FF = w_1 \frac{(n-1-n_d)}{(n-1)} + w_2 \frac{(n-1-n_t)}{(n-1)} + w_3 * B + w_4 * \frac{SI}{(2n-2)} + \frac{1}{\left(\frac{FV}{n}+1\right)} \quad (3)$$

where *n* represents number of assembly components and w_1, w_2, w_3 and w_4 are weight coefficients for direction changes, tool changes, base component location and stability index respectively with their corresponding values as 0.25, 0.25, 0.1 and 0.4.

Proposed implementation of Levy flight

An individual (i.e. cuckoo egg) other than the best egg in the population is randomly selected. This egg (i.e. assembly sequence) is modified using Levy flights. The modification is performed on all the elements of the egg (an element corresponding to an assembly component) using equation 1. It may result in some elements repeated or missing. Since in a feasible assembly sequence none of the components should be repeated or absent, the modified egg is repaired following the repair strategy illustrated in figure 3. It assists to feasibilize the sequence and move it to feasible search space. Its fitness is compared with a randomly picked egg, which it replaces, if found to be better.

Proposed strategy for new cuckoo egg generation

The population is sorted based on fitness, and a fraction of worst eggs is thrown out. To replace them, random eggs are generated following a strategy where the next candidate element is selected based on a probability set by user. While building the sequence, a random number is generated and if it is less than the set probability, the component having least number of precedences is selected and added as next candidate. Otherwise, a component is selected randomly from the list of components absent from the sequence built so far. After adding a component, it is removed from to be assembled components list.

Sequence	Remark
[1 4 3 3 4 4]	Note that the length of assembly sequence is 6. Components [2, 5, 6] are missing and components [3, 4] are repeated. Arrange the missing component list in ascending order of no. of precedences to be complied with. Say, after rearrangement the list becomes [2, 6, 5].
[***1*** 4 3 3 4 4]	Now, start with the first component, i.e. component '1' which is not repeated so move to next component.
[1 4 3 3 4 4]	Next component is component '4', which is repeated thrice. Check if the number of precedences to be complied by component '4' is less than the number of precedences to be complied by component '2' (first component from the list of missing elements). If yes, then replace component 2 with component '4'. And remove component '2' from missing list. Replace the second component '4' with component '2' since it has lesser no. of precedences. If no, then move to the next component.
[1 ***2*** 3 3 4 4]	
[1 2 ***3*** 3 4 4]	Go on for next component, i.e. '3', in the same way, until the missing list is empty.

Figure 3: Repair Strategy.

2.3 Robot task level planning for implementation of generated assembly sequence

To implement the generated sequence for robotic assembly, it must be converted into a task level plan and executable robot level program. A task level planning strategy for robotic assembly has been developed by a knowledge-based system implemented in Expert System shell, CLIPS [19]. It takes as input an optimal assembly sequence generated by the sequence planner described in the previous section and the assembly product data and manufacturing resources data from the MS Access database. It generates as output the executable robot level program in commercial robot programming language, INFORM. Figure 4 shows the architecture of the developed knowledge based system.

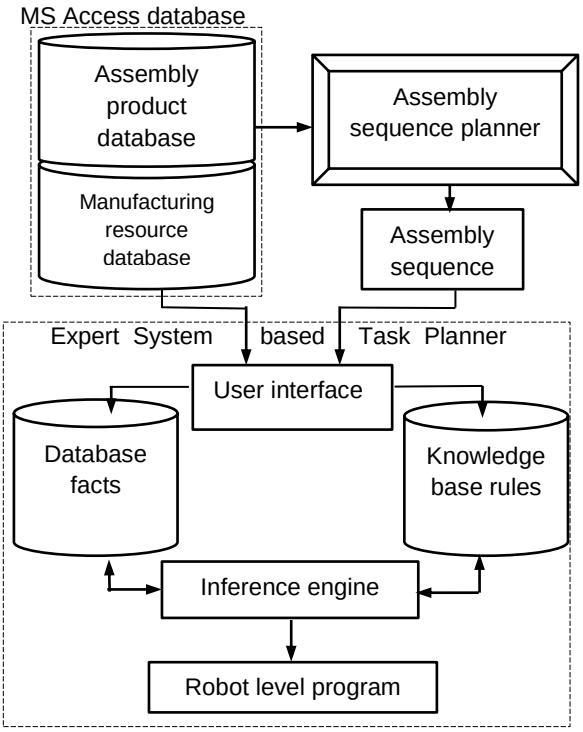

Figure 4: Architecture of the knowledge based system for task level planning.

The validation of the robot program has been carried out in offline robot programming environment and simulation tool, MotoSim EG [20]. Using the 3D solid modeling capability of MotoSim, a virtual model of the robotic assembly workcell is created comprising of the MOTOMAN industrial robot, an assembly fixture, various components to be assembled and their storage racks. Figure 5 gives a screenshot of the MotoSim software showing the layout of the robotic workcell. The task planner is used for automatic offline generation of the robot level program in INFORM, which is loaded onto the MotoSim environment for its execution. Through robot simulation experiments, the cycle time analysis and collision avoidance have been also performed.

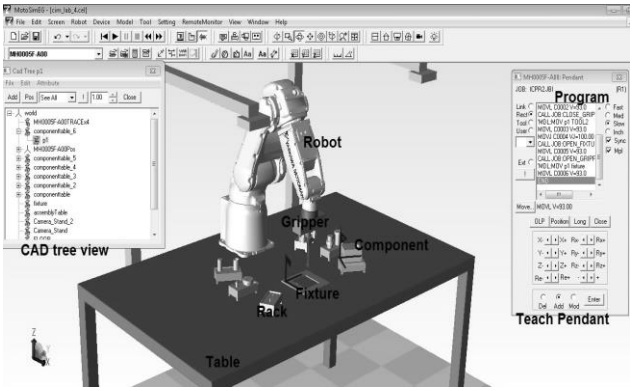

Figure 5: Screenshot of MotoSim EG software showing layout of the robotic workcell.

3 RESULTS AND DISCUSSIONS

To demonstrate the working of the proposed system, a fourteen component product assembly shown in figure 6 has been used. The assembly product data as well as manufacturing resources data are stored in MS Access database. Figures 7 (a) and (b) show an extract of the precedence matrix and stability matrix of the above assembly respectively. The sections below present the results obtained by the developed assembly sequence planner and the robot task level planner.

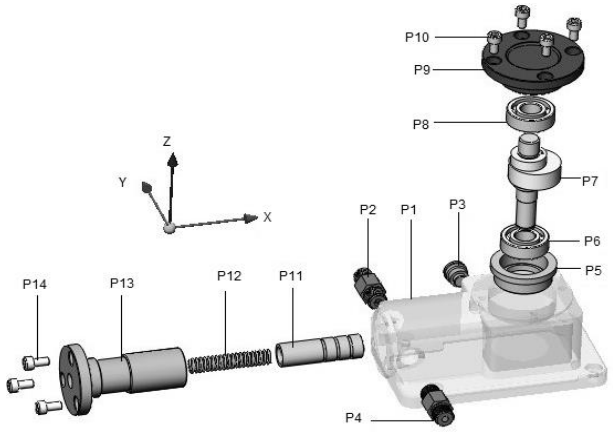

Figure 6: Fourteen component assembly [21].

```
PM = [0 0 0 0 0 0 0 0 0 0 0 0 0 0;
      1 0 0 0 0 0 0 0 0 0 0 0 0 0;
      1 0 0 0 0 0 0 0 0 0 0 0 0 0;
      1 0 0 0 0 0 0 0 0 0 0 0 0 0;
      1 0 0 0 0 0 0 0 0 0 0 0 0 0;
      1 0 0 0 1 0 0 0 0 0 0 0 0 0;
      1 0 0 0 1 1 0 0 0 0 0 0 0 0;
      1 0 0 0 1 1 1 0 0 0 0 0 0 0;
      1 0 0 0 1 1 1 1 0 0 0 0 0 0;
      1 0 0 0 0 0 0 0 0 0 0 0 0 0;
      1 0 0 0 0 0 0 0 0 1 0 0 0 0;
      1 0 0 0 0 0 0 0 0 1 1 0 0 0;
      1 0 0 0 0 0 0 0 0 1 1 1 0 0;
      1 0 0 0 0 0 0 0 0 1 1 1 1 0];

SM = [0 2 2 2 0 0 0 0 0 0 0 0 0 0;
      1 0 0 0 0 0 0 0 0 0 0 0 0 0;
      0 0 0 0 0 0 0 0 0 0 0 0 0 0;
      1 0 0 0 0 0 0 0 0 0 0 0 0 0;
      0 0 0 0 0 1 0 0 0 0 0 0 0 0;
      0 0 0 0 0 0 0 0 0 0 0 0 0 0;
      0 0 0 0 0 0 0 1 0 0 0 0 0 0;
      0 0 0 0 0 0 0 0 1 0 0 0 0 0;
      0 0 0 0 0 0 0 0 0 2 0 0 0 0;
      0 0 0 0 0 0 0 0 0 0 0 0 0 0;
      1 2 0 0 0 0 0 0 0 0 0 0 0 0;
      0 0 0 0 0 0 0 0 0 0 0 0 0 0;
      0 0 0 0 0 0 0 0 0 0 0 0 0 2;
      0 0 0 0 0 0 0 0 0 0 0 0 0 0];
```

Figure 7: (a) Precedence matrix (PM) and (b) Stability matrix (SM) for the assembly shown in Figure 6.

3.1 Results of proposed CS algorithm for assembly sequence optimization

The main parameters affecting the performance of CS algorithm are population size, fraction of worst nests (p_a), probability used in generating new eggs and number of iterations. To study their effects on performance of the CS algorithm, a sensitivity analysis is carried out by running CS algorithm 10 times independently and the results are given in Table 1. The optimum set of parameters found to be able to provide the optimal/near optimal assembly sequence in reasonable time is population size of 10, fraction of worst eggs that are thrown out 10%, probability used in generating new eggs to replace those that are thrown out 90%.

After the CS algorithm is run using the above set of parameters, the optimal assembly sequence obtained is [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14] with fitness value of 1.5269, requiring 7 tool changes, 4 reorientation changes, stability index of 9, having the base component at the first location of the assembly sequence, and most importantly the sequence was found to have no feasibility (precedence) violations. In order to evaluate the effectiveness of the proposed CS algorithm, it is compared with Improved Harmony Search (IHS) algorithm [12]. The best sequence generated by the IHS algorithm is found to be [1, 4, 2, 3, 5,

6, 11, 12, 13, 14, 7, 8, 9, 10] with fitness value of 1.5077, requiring 7 tool changes, 5 reorientation changes, stability index of 9, having base component at the first location of the assembly sequence, no feasibility violations but is clearly sub-optimal when compared to the sequence generated by CS algorithm.

Figure 8 shows comparison between the convergence graphs of the CS and IHS algorithms from which it can be concluded that the proposed CS algorithm clearly outperforms IHS in terms of convergence speed as it is able to reach the global best fitness value in lesser number of iterations. The CS algorithm reached the global best solution in 3421 iterations, while the IHS algorithm took 4754 iterations to reach to its best solution which is found to be sub-optimal compared to that given by CS. Table 2 gives a summary of the results of comparison between the CS and the IHS algorithms. It is found that in 9 times out of 10 runs, the CS algorithm could find the optimal solution of 1.5269, which gives a 90 % probability of finding the best solution or assembly sequence. In contrast, the probability of finding the best solution by IHS is equal to zero (as it could, at best, generate a suboptimal assembly sequence with fitness value of only 1.5077).

Table 1: Effects of the parameters on the CS Algorithm

Population size	Average of the best fitness	Mean average fitness
5	1.5019	1.4770
10	1.5257	1.5013
15	1.5211	1.4732
20	1.5227	1.4643
25	1.5219	1.4330

fraction of worst eggs	Average of the best fitness	Mean average fitness
0.1	1.5257	1.5013
0.2	1.5177	1.4483
0.3	1.5127	1.3936
0.4	1.5065	1.3448
0.5	1.5046	1.2891

Probability used in generating new eggs	Average of the best fitness	Mean average fitness
0.5	1.5014	1.4647
0.6	1.5173	1.4963
0.7	1.5219	1.4773
0.8	1.5227	1.5002
0.9	1.5257	1.5013

Table 2: Comparison between the results of the IHS and CS Algorithms

	IHS	CS
Best fitness value	1.5077	1.5269
Mean of the average fitness values for 10 runs	1.4792	1.5013
Mean of the best fitness values for 10 runs	1.4792	1.5250
Number of runs out of 10 independent runs in which convergence to global best fitness was achieved	0	9
% of obtaining the optimal assembly sequence	0	90

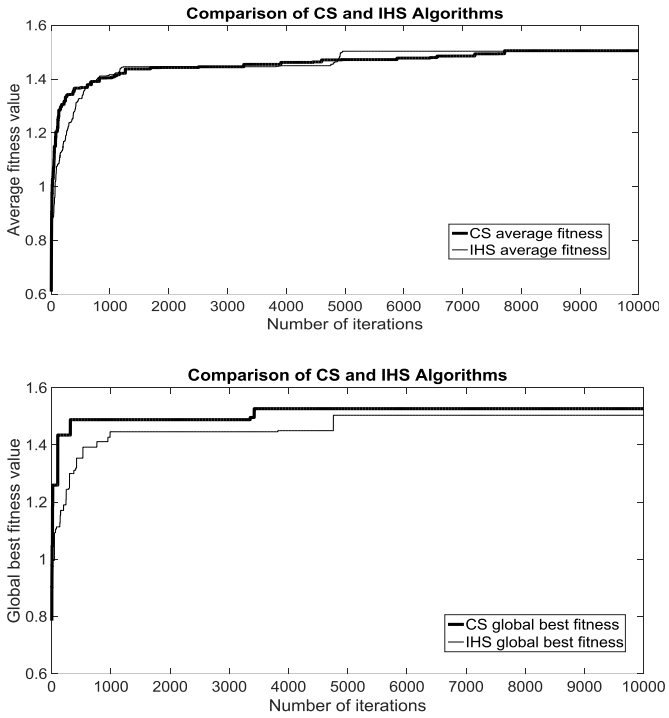

Figure 8: Comparison between the convergence graphs of the proposed CS Algorithm and IHS Algorithm.

3.2 Results of robotic assembly task level planning

The executable robot program for assembling the fourteen component product that is generated by the task planner is loaded onto the MotoSim environment for execution and simulation experiments are conducted. Figure 9 shows an extract of the robot program in INFORM to direct the robot to retrieve the base component from its storage rack and place it onto the assembly fixture.

```
2  //initialize model of component 1
3  CALL JOB:INIT
4  //Move Robot's TCP to a safe intermediate point
5  MOVJ C0000 VJ=50
6  //Move Robot's TCP above the component 1
7  MOVJ C0001 VJ=50
8  //Open the robot gripper by function 'OPEN_GRIPPER'
9  CALL JOB:OPEN_GRIPPER
10 //Move TCP linearly towards component 1
11 MOVL C0002 V=93
12 //Close robot gripper by function 'CLOSE_GRIPPER'
13 CALL JOB:CLOSE_GRIPPER
14 //Move Robot's TCP above the component 1
15 MOVL C0003 V=93
16 //Move Robot's TCP above the fixture
17 MOVJ C0004 VJ=50
18 //Open assembly fixture
19 CALL JOB:OPEN_FIXTURE
20 MOVL C0005 V=93
21 //Close assembly fixture
22 CALL JOB:CLOSE_FIXTURE
23 CALL JOB:OPEN_GRIPPER
24 //Move Robot's TCP above the fixture
25 MOVL C0006 V=93
```

Figure 9: Extract of executable robot level program in INFORM generated by the task planner.

In figure 9, MOVJ and MOVL are the commands for movements of the robot using joint interpolation and linear interpolation respectively; C0001, C0002, etc. are the position variables for storing the TCP (tool center point)

coordinates of the robot along its trajectory; V and VJ are the TCP speed and Joint speed respectively; OPEN_GRIPPER and CLOSE_GRIPPER are the functions for executing the opening and closing of the pneumatic gripper respectively. The cycle time for completing the assembly estimated from the simulation is 102.07s. The above cycle time analysis is useful for optimal layout of the robotic workcell.

4 CONCLUSIONS

Since assembly accounts for a significant proportion of total manufacturing cost, proper process planning and optimization are important. An important decision in process planning is to determine the feasible and optimal sequence. It is challenging because there are various constraints driving the choice of a feasible sequence and moreover with increase in part count, the number of sequences rises staggeringly. This makes the assembly sequence optimization problem combinatorially explosive, arduous and hence time consuming to solve manually. Moreover, to implement the sequence for robotic assembly, the sequence must be converted into a task level plan and the robot level program which is also a tedious task when performed manually. To automate these, a CAPP system has been proposed in this paper for assembly using a discrete cuckoo search algorithm with capability to automatically generate feasible and optimal assembly sequences, giving due considerations to optimization criteria like minimization of orientations and tool/robot gripper changes, assembly stability criteria and location of base component. To implement the optimal assembly sequence for robotic assembly, the assembly sequence output from the sequence planner has been integrated with a robot task planner developed using a Knowledge Based System that is capable of automatically generating an executable robot level program for performing the assembly. The entire CAPP system has been integrated with an upstream assembly product database and manufacturing resources database in MS Access. It is anticipated that by using the proposed integrated assembly sequence planner and task level planner, the assembly process planning can be accomplished automatically by investing very limited amount of time, making it an attractive and cost effective solution for industry. A future research direction could be integration of the proposed assembly sequence planning and task planning methodologies with a CAD based feature extraction module to minimize the user intervention and further reduce the time for generation of the assembly process plans.

REFERENCES

[1] Hong D. S., Cho H. S., 1993, Optimization of robotic assembly sequences using neural network, IEEE/RSJ Int Conf on Intelligent Robots and Systems, Japan, 26-30, 232-239.

[2] Bonneville F., Perrard C., Henrioud J. M., 1995, A genetic algorithm to generate and evaluate assembly plans, INRIA/IEEE Symposium on Emerging Technologies and Factory Automation, France, 231-239.

[3] Mishra A., Deb S., 2016, An Intelligent Methodology for Assembly Tools Selection and Assembly Sequence Optimization. In: Mandal D.K., Syan C.S. (eds) CAD/CAM, Robotics and Factories of the Future. Lecture Notes in Mechanical Engg. Springer, 323-333.

[4] Wang J. F., Liu J. H., Zhong Y. F., 2005, A novel ant colony algorithm for assembly sequence planning, Int J Adv Manuf Technol, 2005, 25, 1137–1143.

[5] Mishra A., Deb S., 2016, A GA based parameter meta-optimization of ACO algorithm for solving assembly sequence optimization, 46th International Conferences on Computers and Industrial Engineering (CIE 2016), Tianjin (China).

[6] Cao P. B., Xiao R. B., 2007, Assembly planning using a novel immune approach, Int J Adv Manuf Technol, 31(7), 770–782.

[7] Chang C. C., Tseng H. E., Meng L.P., 2009, Artificial immune systems for assembly sequence planning exploration, Engineering Applications of Artificial Intelligence, 22, 1218–1232.

[8] Wang Y., Liu J.H., 2010, Chaotic particle swarm optimization for assembly sequence planning. Robot Computer Integrated Manufacturing, 26(2), 212–222.

[9] Lv H.G., Lu C., 2010, An assembly sequence planning approach with a discrete particle swarm optimization algorithm, Int J Adv Manuf Technol, 50(5–8), 761–770.

[10] Gao L., Qian W. R., Li X. Y., Wang. J. F., 2010, Application of memetic algorithm in assembly sequence planning, Int J Adv Manuf Technol, 2010, 49(9–12), 1175–1184.

[11] Zhou W., Zheng J. R., Yan J. J., Wang J. F., 2011, A novel hybrid algorithm for assembly sequence planning combining bacterial chemotaxis with genetic algorithm, Int J Adv Manuf Technol, 52(5–8), 715–724.

[12] Li X., Qin K., Zeng B., Gao L., Su J., 2016, Assembly sequence planning based on an improved harmony search algorithm, Int J Adv Manuf Technol, 84, 2367-2380.

[13] Rabemanantsoa M., Pierre S., 1993, A knowledge-based system for robot assembly planner, Proceedings of Canadian Conference on Electrical and Computer Engineering, Vancouver BC, 2, 829-832.

[14] Mosemann H., Wahl F. M., 2001, Automatic Decomposition of Planned Assembly Sequences Into Skill Primitives, IEEE Transactions on Robotics and Automation, 17(5), 709-718.

[15] Thomas U., Wahl F. M., 2010, Assembly Planning and Task Planning -Two Prerequisites for Automated Robot Programming, Robotic Systems for Handling and Assembly, STAR 67, 333-354.

[16] Neto P., Mendes N., 2013, Direct off-line robot programming via a common CAD package, Robotics and Autonomous Systems, 61(8), 896–910.

[17] Stenmark M., Malec J., 2015, Knowledge-based instruction of manipulation tasks for industrial robotics, 2015, Robotics and Computer-Integrated Manufacturing, 33, 56-67.

[18] Yang X. S., Deb S., 2010, Engineering Optimization by Cuckoo Search, Int. J. Mathematical Modelling and Numerical Optimization, 1, 4, 330–343.

[19] CLIPS User's Guide, 2007, Available: http://clipsrules.sourceforge.net/documentation/v630/ug.pdf [Accessed: 28- Apr- 2017]

[20] MotoSim EG Ver4.01 Operational Manual for Windows, 2015, Yaskawa Electric Corporation, Japan.

[21] Guo J., Sun Z., Tang H., Yin L., Zhang Z., 2015, Improved Cat Swarm Optimization Algorithm for Assembly Sequence Planning, The Open Automation and Control Systems, 7, 792-799.