Journal of Advanced Manufacturing Systems
2017 (In Press)
© World Scientific Publishing Company

A METHODOLOGY FOR ASSEMBLY SEQUENCE OPTIMIZATION BY HYBRID CUCKOO-SEARCH GENETIC ALGORITHM

G. V. S. K. KARTHIK and SANKHA DEB

FMS and Computer Integrated Manufacturing Laboratory,
Department of Mechanical Engineering,
Indian Institute of Technology Kharagpur,
Kharagpur 721302, India.
sankha.deb@mech.iitkgp.ernet.in

In this paper, we have proposed and implemented a methodology for assembly sequence optimization by using a nature-inspired meta-heuristic algorithm, known as hybrid cuckoo-search genetic algorithm (CSGA) algorithm. The cost criteria for optimization in the present formulation take into consideration the total assembly time and the number of reorientations during the assembly process. To demonstrate the application of the CSGA algorithm, an example assembly containing 19 parts has been presented and the results have been compared with those of another meta-heuristic algorithm, Genetic Algorithm (GA). From the results, it has been observed that for the given problem, the CSGA algorithm not only produces optimal assembly sequences with costs comparable to that of GA, but the convergence of CSGA algorithm has been found to be faster than the GA algorithm.

Keywords: assembly sequence optimization; Computer-Aided Process Planning; Hybrid cuckoo-search genetic algorithm.

1. Introduction

Assembly is one of the most important processes during the manufacturing of a product because it is during the assembly phase that the final product is obtained from the individual components. More than 40% of the manufacturing cost is estimated to be spent on assembly, and assembly accounts for almost 20 – 70% of all the manufacturing work. The assembly process planning involves detailing the plan for the assembly of the product according to the design specifications. Assembly process planning can be divided into two tasks namely, assembly sequence planning and assembly task planning. An assembly sequence plan (ASP) is the sequence of assembly operations of individual components in a particular order. Determining the optimal assembly sequence for a given assembly is considered as a combinatorially exponential problem. With the increasing number of parts within the assembly, the number of assembly sequences grows exponentially, so the optimal assembly sequence determination by searching the entire solution space can be very tedious. The different optimization criteria of an ASP include, for example, degree of difficulty of assembly operation, subassembly stability, time necessary to accomplish assembly tasks, costs of tooling and hardware required, costs of fixturing, change in the direction of the assembly etc. There are two main approaches to the generation of assembly plans: manual planning and computer aided process planning. Manual planning is based on the process planner's expertise and information on the connection between pairs of parts and the feasibility of the assembly process. This method can be tedious for assemblies consisting of large number of components because of its complexity as well as considerably large amount of time needed. Computer aided process planning methods are mainly based on graph search techniques to search for an optimal path in the graph representation of assembly operations, cut-set algorithms where the assembly is decomposed into all possible subassemblies, and evaluating optimal assembly sequence from the information on the individual subassemblies etc. Computer aided process planning techniques can be divided into techniques that involve traditional search algorithms like graph search algorithms, and AI and soft computing techniques like Knowledge based system, Fuzzy reasoning based approaches, population search based approaches like genetic algorithms (GA), memetic algorithm, particle swarm optimization (PSO), ant colony optimization (ACO), Cuckoo Search (CS) etc. Traditional approaches tend to search for the entire solution space and obtain the optimal sequence. They are time consuming (exponential in nature) as assembly sequence planning problem is an NP hard problem. Soft computing techniques have been proved to be effective for solving NP hard problems.

The research work reported in this paper aims at obtaining the optimal assembly sequence of a given assembly using hybrid cuckoo-search genetic algorithm and also using genetic algorithm, both being nature inspired meta-heuristics (soft computing techniques), and then comparing the results obtained by both techniques.

2. Review of previous research

Since the early 1980s, numerous research works have been reported on the assembly sequence generation and planning. Bourjault [1] presented the liaisons diagram (the 'graphe de liaisons fonctionelles'), which is a graph devised to represent an assembly. Based on this diagram, a list of ''Yes–No'' questions are generated, and by answering these questions about the feasibility of assembly of parts, the assembly sequence is determined. But for an assembly of n parts, his method unfortunately requires answering a minimum of 2n2 "yes or no" questions and hence is very time consuming. De Fazio and Whitney [2] were able to improve upon this work and were able to reduce the number of questions to 2n by altering the question's form. Their questions are not ''yes–no'', and require geometric reasoning and anticipation by the user. While this improvement made Bourjault's work more applicable, it is still unreasonable to have to ask the user so many questions for components with large number of parts. Heemskerk [3] et al. proposed an approach for assembly sequence planning, using heuristics in several stages. In the first stage parts are grouped together to reduce the combinatorial complexity. In later stages, iterative search techniques were used. Homem and Sanderson [4] proposed an AND/OR graph representation for all the possible configurations of the assembly and generated the assembly sequences for a product using a disassembly or decomposition method based on the assumption that the disassembly sequence is the reverse of a feasible assembly sequence. Gottipolu and Ghosh [5] proposed a novel approach which involved extracting geometric and mobility constraints extracted directly from the CAD model of the assembly and translating them into relevant matrices, and an algorithmic procedure is used to generate all feasible assembly sequences from these two matrices. Alfadhlani et al [6] presented a methodology for automatic retrieval of geometrical data of components directly from the CAD system and then using an algorithm for automatic detection of the collision-free path of an assembly based on mating type, the normal vector direction, and the surface shape of contacted component. Ben Arieh and Kramer [7] presented a methodology and algorithms to generate consistently all feasible assembly sequences considering the various combinations of subassembly operations. The algorithms are implemented using a LISP program. Also, Ben-Arieh [8] used a fuzzy set based method to evaluate the degree of difficulty of each assembly operation and then selected a "best" sequence of assembly operations. Dong et al. [9] proposed a knowledge-based approach to the assembly sequence planning problem. They proposed a CSBAT (connection-semantics-based assembly tree) hierarchy which provides an appropriate way to consider both geometric information and non-geometric knowledge. They proposed the structure of the KBASP (knowledge-based assembly sequence planning) and they proposed different ways to construct plans for a CSBAT: by retrieving the typical base, by retrieving the standard base, and by geometric reasoning. Various soft computing based approaches have also been developed by researchers for assembly sequence planning and optimization. One of the earliest attempts to solve the combinatorial explosion problem of Assembly Sequence Planning (ASP) using GA was by Bonnevile et al [10]. They used GA on the possible assembly sequences given by an experienced assembly expert as the starting point of the evolution computing, and encoded every possible assembly plan as an assembly tree in a chromosome. Chen [11] also adopted a genetic ASP method which uses five genetic operators to generate offspring and introduces a double level genetic structure to adjust the control parameters of computing dynamically. Guan et al. [12] proposed a gene-group-based evolution approach to obtain sequences for the whole assembly process plan rather than just assembly sequence plan. They have designed a compound chromosome encode which has all the information regarding a particular operation, to represent abundant assembly process information, and several specific genetic operators are used to generate offspring individuals. Wang et al. [13] studied the application of ACO algorithm proposed by Dorigo et al [14] to the ASP problem. They considered the assembly by disassembly technique. They used the concept of interference matrix (named as disassembly matrix) to generate dynamically and implicitly the graph connecting various disassembly operations which are taken as nodes and edges between them to represent the feasibility of successive operations. Gao et al. [15] used the memetic algorithm to determine the Assembly sequences. The memetic algorithm (MA) combines the parallel global search nature of evolutionary algorithms with local search to improve individual solutions. However, the fitness function used for individual solution depends not only on the assembly costs but also on the feasibility of the sequence provided so the algorithm may not necessarily provide feasible solutions. Li et al. [16] attempted to combine the genetic algorithm and tabu search techniques and applied it to the ASP problem. Choi et al. [17] considered application of genetic algorithm for multi-criteria assembly sequence planning. Total assembly time and number of reorientations are taken as the optimization criteria. The operators used ensure the feasibility of the produced sequences. Sung et al. [18] introduced an evolutionary method for solving travelling salesman problem with precedence constraints. The various parameters within the algorithm are adapted (changed) with the number of iterations based on the performance over the iterations. Yun et al. [19] proposed a GA approach with topological sort-based representation procedure for solving various types of PCSP. Yang et al. [20] proposed cuckoo search algorithm, a nature inspired meta-heuristic which is inspired from the obligate brood parasitic behaviour of certain cuckoo species. The proposed algorithm is largely applicable to continuous optimization problems. Lim et al. [21] proposed a hybrid cuckoo-search genetic algorithm approach for optimization of hole-making operations, which is a combinatorial optimization problem. Nilakantan et al. [22] proposed a hybrid cuckoo search and particle swarm optimization (CS-PSO) approach for robotic assembly line balancing, which is a well-known NP-hard problem with the objective of minimizing the cycle time.

In the current paper, we propose to solve the problem of assembly sequence planning and optimization using a nature inspired soft computing based meta-heuristic algorithm known as hybrid cuckoo-search genetic algorithm (CSGA) and show the results of comparison with the genetic algorithm.

3. Proposed Methodology for the assembly sequence optimization

3.1 *Optimization Criteria*

The optimization criteria used in the present paper are minimization of total assembly time and number of reorientations. A reorientation is a change in the direction of assembly or insertion. Oftentimes, while performing the assembly of the final product, we cannot assemble the present component in the same direction as that of the previously assembled component. This requires rotating the subassembly by some angle about the axis of insertion or about an axis perpendicular to the insertion axis. This operation is called reorientation. A combined fitness function [17] to minimize the above two criteria is used. A fitness function is an empirical function which provides a score for each assembly sequence (solutions to the problem) that we generate using the algorithm. In the present scenario, the lower the score for the assembly sequence provided by the fitness function, the better is the assembly sequence. The total assembly time is quantified as follows. The setup time is given by:

$$T_{Setup}(i) = p_{i0} + \sum_{j=1}^{n} p_{ij} x_{ij} \qquad (1)$$

where

n	number of components
i	component to be assembled
p_{i0}	setup time for product i being the first component in the assembly
p_{ij}	contribution to the setup time due to the presence of part j when entering part i

$$x_{ij} = \begin{cases} 1, \mathit{if\ component\ j\ has\ already\ assembled}, \mathit{for}\ i = 1, \dots, n \\ 0, \mathit{otherwise} \end{cases} \qquad (2)$$

The total assembly time is the summation of setup time and actual assembly time

$$T_{Assembly} = \sum_{i=1}^{n} \left(T_{Setup}(i) + A_i\right) \qquad (3)$$

where A_i is the assembly time for component i.

The number of reorientations is calculated as follows:

$$R_i = \begin{cases} 1, if\ reorientation\ is\ needed \\ when\ entering\ component\ i\ for\ i = 1, \dots, n \\ 0, otherwise \end{cases}$$

(4)

The total number of reorientations is:

$$R = \sum_{i=1}^{n} R(i) \qquad (5)$$

The combined objective function is given by:

$$Z = w_1 * T_{Assembly} + w_2 * R \qquad (6)$$

Here, Z represents the combined fitness function considering both the criteria, i.e., amount of assembly time and number of reorientations, and w_i represents the weight or priority that we give to individual criteria. More the weight given to a particular criterion, more is the likelihood that the final solution will satisfy that criterion better.

Z	combined fitness function
w_i	weight of the individual function

Our objective is to minimize the above function.

3.2 *Solution Representation*

In this paper, the solution is represented by the assembly sequence, i.e., the sequence of parts represents the assembly sequence. For example if the number of components is 6, numbered 1 to 6, then 2 – 1 – 3 – 4 – 6 – 5 represents a possible solution. The feasible solutions must satisfy the precedence constraints for each part. The precedence constraints are represented by a matrix PM, where

$$PM\ (i, j) = \begin{cases} 1, if\ assembly\ of\ component\ i \\ requires\ j\ to\ be\ present\ in\ the\ assembly \\ 0, otherwise \end{cases}$$

(7)

We define for each component i, i = 1,…,n

$$P(i) = \sum_{j=1}^{n} P(i,j) * y(i,j) \tag{8}$$

where

$$y(i,j) = \begin{cases} 1, if\ component\ j\ is\ assembled\ before\ i \\ 0, otherwise \end{cases} \tag{9}$$

$$P = \sum_{i=1}^{n} P(i) \tag{10}$$

If for a particular solution or assembly sequence, P is zero, then that solution is feasible, else infeasible. Here non-zero value of P for a solution indicates that one of the components that requires the presence of a later assembled component, is assembled early. This is infeasible.

To estimate the total assembly time, the contribution of a component to the assembly time of a given component is given by the Setup matrix (SM). SM(i, j) gives the contribution of component j to the assembling time of component i. For example if components 2, 5, 7 are already assembled and component 1 is being assembled, then the setup time for component 1 is:

ST(1) = SM(1, 2) + SM(1, 5) + SM(1, 7) (11)

where ST(i) is the setup time required for component ‘i’

Reorientation of the subassembly may be required based on the component being assembled because the component may have geometric constraints with the components already assembled. This information can be obtained from the reorientation matrix (RM), where the row number indicates the component being assembled and the elements in the row, the components already assembled in that order. If for the components being assembled, the components already assembled and their order matches exactly with any one of the elements present in that row, then there will be a reorientation for that particular component [16].

Consider the reorientation matrix provided in table 3. Let the current subassembly consists of components 0, 3, and 4 and the subassembly is formed by components assembled in that order. If the current component being assembled is 2, then as the assembly order 0 – 3 – 4 is present in the reorientation matrix for component 2, there will be a reorientation in this case.

3.3 *Genetic Algorithm (GA)*

The algorithm used searches only feasible solution space to find the optimal or near-optimal solution. The various operators used are inspired from [18]. The functionality and the operators used in the algorithm are given below:

3.3.1 *Generation of initial population*

The initial population can be generated based on topological sort [19]. The procedure used to generate one individual of the population is given below:

1. Create an empty list which denotes the chromosome to be built.

2. While there are still nodes left:

3. Get the set of all nodes which have no precedence constraints.

4. Randomly select one of the nodes from the above set and append the node to the chromosome list.

5. Remove the node and its precedence dependencies from the precedence graph or matrix.

The initial population can be generated by using the above procedure the required number of times.

3.3.2 *Crossover*

The crossover operator requires two chromosomes and produces a single new chromosome. The offspring chromosome is created by incremental inclusion of selectable nodes [19]. The procedure to crossover two chromosomes is given below. Here L is the length of each chromosome i.e., the number of components in the assembly:

1. Create a graph G = (N, A) of TSPPC. Set l = 1

2. Create a selectable node set E from G.

3. Select two lth genes from both parent chromosomes. We have four possible cases as follows.

Case 1. Two selected genes are different and found in E. Select one arbitrarily.

Case 2. Two selected genes are same and found in E. Then select that one.

Case 3. Only one gene is selected and found in E. Then select that one.

Case 4. No selected genes are found in E. Then select a gene from E arbitrarily.

4. Delete the selected node in ‘3’ with the corresponding arcs.

5. If $l = L$, terminate, else goto ‘2’.

3.3.3 *Mutation*

Mutation is performed on a chromosome to find a better solution or to jump from local minima. In a given chromosome, two positions are chosen randomly and all the genes present in between these two selected positions (including the genes at these positions) are sorted by topology. This procedure ensures the feasibility criteria [18].

The procedure for the algorithm is given below:

```
1. Initialize the various parameters like population size (n), the number of times a
crossover is performed in an iteration (n_c), the initial population, crossover probability
(p_c), mutation probability (p_m), maximum number of iterations (iter_max), and set
present iteration count (iter_count) to 1

2. While iter_count <= iter_max:

3. Create a new child set

4.  Loop for n_c times:

5.  Generate a random number, r. The random number will be between 0 and 1.

6. If r < p_c:

7. Get two chromosomes parent1 and parent2 from the population randomly, with
probability of selecting a chromosome inversely proportional to its fitness function

8. Perform crossover to get a new child chromosome child1

9. Generate a random number, r. The random number will be between 0 and 1.

10. If r < p_m:

11. Perform mutation on child1

12. Add child1 to the child list
```

13. Choose the n best individuals from population and the child list and assign the population to this new set of n individuals

14. Identify the current best solution and if the current best solution is better than the overall best solution, replace the overall best solution by the current best solution

15. Increase iter_count by 1

3.3.3 *Proposed Hybrid Cuckoo Search Genetic Algorithm*

Cuckoo search (CS) is a nature-inspired meta-heuristic proposed by Yang and Deb (2009) [19], which was inspired by the obligate brood parasitism of some cuckoo species, which lay their eggs in the nests of other host birds. Because the use of the operators in the original CS may lead to infeasible sequences, the CS operators are hybridized with that of the GA operators mentioned above to maintain and produce feasible solutions. For the initial population, crossover and mutation, the same operators as mentioned in the GA are used. The overall procedure for the CSGA algorithm inspired from [21] is given below:

1. Initialize the number of nests (n), fraction of nests to be rejected (p_a), random solution in each nest, total number of iterations (iter_max), and present iteration count (iter_count) to 1.

2. While iter_count <= iter_max:

3. Select two random solutions and perform crossover to get a new solution

4. Get a random nest from all the nests.

5. If the fitness of the new solution is better than fitness of the solution in the selected nest:

6. Replace the solution in the nest with the new solution.

7. Select a random nest and perform mutation on the solution in the nest.

8. Nests are sorted by their fitness and the n*p_a number of low fitness solutions in the nests are replaced with new solutions.

9. Identify the current best solution and if the current best solution is better than the overall best solution, replace the overall best solution by the current best solution.

10. Increase iter_count by 1.

4. Illustrative example: Results and Discussions

A product with 19 components taken from literature [17] is considered. Table 1 shows the Precedence Matrix (PM) for the product, Table 2 gives the Setup Matrix (SM) and Table 3 gives the Reorientation Matrix (RM). 1000 iterations have been performed on both the algorithms with an initial population of 15. The numbering of the components starts from zero, i.e., if there are 5 components in a product, then the components are numbers as 0, 1, 2, 3, 4. Both the algorithms are run 100 times for getting the average behaviour during a typical run. The algorithms are implemented in the python programming language and executed on a core 2 duo 2.2Ghz processor with 3GB RAM on Linux Mint 13 operating system.

4.1. *Results of GA*

The number of times crossover is performed in every iteration is taken equal to the population count, the crossover probability is taken as 0.8 and mutation probability as 0.2. The GA convergence plots for average fitness (i.e. mean of the fitness values across the entire population) and the best fitness are shown in Fig. 1. (In this case, the plot for average fitness is found to closely follow the plot for best fitness, that is why the two plots appear to be overlapping in the figure).
The best sequence obtained with GA and its fitness are 1, 0, 3, 8, 2, 11, 12, 15, 4, 14, 17, 10, 5, 6, 7, 9, 13, 16, 18 and 528.3 respectively.

4.2. *Results of CSGA*

The fraction of nests to be replaced during each iteration (p_a) is taken to be 0.25. The CSGA convergence plots for average fitness and best fitness are shown in Fig. 2.
The best sequence obtained with CSGA and its fitness are 1, 0, 3, 8, 2, 11, 12, 15, 4, 14, 17, 10, 5, 6, 7, 9, 13, 16, 18 and 528.3 respectively, which is the same as that obtained by GA.
The average time taken for the execution of both the algorithms is shown in Table 4. Clearly the time taken for convergence by GA is far more than that of CSGA. In this paper, we have implemented a methodology for assembly sequence optimization by using CSGA and compared the results with the GA.

5. Conclusions

The research work reported in this paper aims at obtaining the optimal assembly sequence of a given assembly using hybrid cuckoo-search genetic algorithm and also using genetic algorithm, and then comparing the results obtained by both the techniques. The optimization criteria considered are the total assembly time and the number of reorientations. Both CSGA and the GA are population based techniques but the operators used in these algorithms ensure the feasibility of the produced sequences. From the results, it has been observed that for the given problem, the CSGA algorithm not only produces optimal assembly sequences with costs comparable to that of GA, but the convergence of CSGA algorithm has been found to be faster than the GA algorithm. The scope of future work includes the use of other criteria such as subassembly stability, fixture changes etc. to determine the optimality of assembly process plans.

References

1. A. Bourjault, Contribution à une approche méthodologique de l'assemblage automatisé: élaboration automatique des séquences opératoires, *PhD Thesis*, Université de Franche-Comté (France 1984).
2. T.L. De Fazio and D.E. Whitney, Simplified generation of all mechanical assembly sequences, *IEEE Trans Robot Automat.* 3(6) (1987) 640–58.
3. C.J.M. Heemskerk, The Use of Heuristics in Assembly Sequence Planning, *Annals of the CIRP*. 38(1) (1989).
4. L.S. Homen de Mello and A.C. Sanderson, AND/OR graph representation of assembly plans, *IEEE Transactions on Robotics and Automation*. 6(2) (1990) 188-199.
5. G.B. Reddy and K. Ghosh, A simplified and efficient representation for evaluation and selection of assembly sequences, *Computers in Industry*. 50 (2003) 251–264.
6. Alfadhlani, T. M. A. Ari Samadhi and Anas Maruf. Automatic Collision Detection for Assembly Sequence Planning Using a Three-Dimensional Solid Model, Journal of Advanced Manufacturing Systems, 10(2) (2011) 277-291.
7. D. Ben-Arieh and B. Kramer, Computer-aided process planning for assembly: generation of assembly operations sequence, *International Journal of Production Research*. 32(3) (1994) 643-656.
8. D. Ben-Arieh, A methodology for analysis of operation's difficulty, *International Journal of Production Research*. 32(8) (1994) 1879-1895.
9. T. Dong., R. Tong., L. Zhang, and J. Dong, A knowledge-based approach to assembly sequence planning, *Int. J. Adv. Manuf. Technol*. 32 (2006) 1232-1244.
10. F. Bonneville, C. Perrard and J.M. Henrioud, A genetic algorithm to generate and evaluate assembly plans, *Proceedings of IEEE Symposium on Emerging Technologies and Factory Automation* (Paris 1995).
11. S.F. Chen, Assembly planning – a genetic approach, in *ASME Proceedings of Design Engineering Technical Conferences* (Atlanta, GA 1998).
12. Q. Guan, J.H. Liu, and Y.F. Zhong, A concurrent hierarchical evolution approach to assembly process planning, *Int. J. Prod. Res.* 40 (2002) 3357-3374.
13. J.L. Wang, J.H. Liu, and Y.F. Zhong, A novel ant colony algorithm for assembly sequence planning, *Int. J. Adv. Manuf. Technol*. 25 (2005) 1137–1143.
14. M. Dorigo, and G.D. Caro, Ant Colony Optimization: A New Meta-Heuristic, *Proceedings of the 1999 Congress on Evolutionary Computation*. 2 (1999) 1470-1477.
15. L. Gao, W.Qian , X. Li and J. Wang, Application of memetic algorithm in assembly sequence planning, *Int. J. Adv. Manuf. Technol*. 49 (2010) 1175–1184.
16. J.R. Li, L.P. Khoo and S.R. Tor, A Tabu-enhanced genetic algorithm approach for assembly process planning, *Journal of Intelligent Manufacturing*. 14 (2003) 197-208.
17. Y.K. Choi, , M.L. Dong and Y.B.Cho, An approach to multi-criteria assembly sequence planning using genetic algorithms, *Int. J. Adv. Manuf. Technol*. 42 (2008) 180-188.
18. J. Sung and B. Jeong, An adaptive evolutionary algorithm for travelling salesman problem with precedence constraints, *The Scientific World Journal*. Article ID 313767 (2014).
19. Y. Yun, and C. Moon, Genetic algorithm approach for precedence-constrained sequencing problems, *Journal of Intelligent Manufacturing*. 22(3) (2011) 379-388.
20. X.S. Yang and S. Deb, Cuckoo search via levy flights. IEEE World congress on Nature & biologically inspired computing, NaBIC 2009. (2009) 210-214.

21. W.C.E. Lim, G. Kanagaraj and S.G. Ponnambalam, A hybrid cuckoo-search genetic algorithm for hole-making sequence optimization,. *J. Intell. Manuf.* (2014) doi: 10.1007/s 10845-014-0873-z.
22. J. Mukund Nilakantan, S. Ponnambalam, N. Jawahar and G. Kanagaraj, Bio-inspired search algorithms to solve robotic assembly line balancing problems. Neural Computing & Applications, 26(6), (2015) 1379-1393. doi:10.1007/s00521-014-1811-x

Table 1 Precedence Matrix

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
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	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	1	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
8	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	1	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	1	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	1	0	1	0	1	1	1	1	1	0	0	0	1	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	1	0	1	0	1	1	1	1	1	0	0	0	1	0	0	1	0	0

Table 2 Setup Matrix

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	10	1	2	3	4	5	6	7	8	9	3.2	4.3	7	6.1	1.2	3.4	0	0	7.4
1	1.5	10	2	2	2	2	2	2	2	2	0	3.1	6	4.3	2.7	4.8	0	3	0.5
2	1	2.3	10	0	4	5	0	4	2.3	4.3	9.8	2.4	5	1.2	3.4	4.5	5.6	3.4	3.1
3	0	2	3.4	10	4.5	0	4	0	8	0	3.4	5.6	5	0	0	3.4	0	0	9.8
4	1.2	1	2	3	10	7.9	8.9	0	1.2	2	2.3	0	3	0	3.6	0	2.8	9.8	0
5	9.8	4.5	0	1.2	3.6	10	3.4	4	0	2.3	4.6	5.6	0	4	3	2	0	0.4	3.2
6	0.5	1.4	2.3	0.5	1.9	1	10	13.4	1.2	4	2.3	0	3	5.7	8.3	2	0.1	0	0.5
7	0	0	0	0	0	1.8	9.8	10	2.3	3	8.9	2.3	0	0	2.3	0.5	9.8	0	2.3
8	1	3	4.5	2.3	4.6	9.8	7.5	6.8	10	6	2.3	3.4	5	12.3	3.4	5.61	1	0	0
9	2.3	4.5	2.3	0	2.3	0	2.1	0	4.5	10	1.1	2.3	2	0	0	2.1	1.2	5.4	9.2
10	1	1	2	3	4	5	6	7	8	9	10	4.5	3	6.1	1.2	3.4	0.3	0	1.3
11	1.5	0	2	2	2	2	2	1	2	2	11.2	10	6	4.3	2.7	4.8	0	3	0.5
12	1	2.3	0	0	4	5	0	4	2.3	4.3	9.8	2.4	10	1.2	2.4	4.5	1.6	2.4	3.1

13	0	2	3.4	0	4.5	0	4	0	8	0	3.4	5.6	5	10	2.1	1.4	1	0	2.8
14	1.2	1	2	3	0	7.9	8.9	0	1.2	2	1.3	4	3	1.4	10	1.3	9.8	9.8	2
15	9.8	4.5	0	1.2	3.6	0	3.4	4	0	2.3	4.6	3.6	0	4	3	10	1.5	0	3.2
16	1	3	4	5	0	5	4	3.4	1.2	4	1.3	0	2	3.7	4.3	2.3	10	3.8	10
17	0.6	0.5	3.4	1.2	3	2	9.8	2	2.3	3	5.9	2.3	0	1.0	2.3	0.5	9.8	10	2.3
18	1	3	4.5	2.3	4.6	9.8	7.5	6.8	0	6	3.3	3	2	3.3	4.4	2.6	0.3	2.5	10

Table 3 Reorientation Matrix

Part	Set of Reorientation
0	{3, 8} {8, 10} {3, 5, 10} {10, 6} {6, 8} {5, 4} {12, 11, 17}
2	{4, 1} {0, 3, 4} {15, 17} {1, 0, 5} {1, 0, 3, 4} {12, 8, 3} {8, 5, 1} {5, 1, 9} {17, 16}
6	{4, 11, 5} {11, 5, 0} {8, 5, 15} {18, 1, 3, 8, 7} {11, 5} {6, 4} {8, 4} {9, 7, 5, 3}
7	{5, 6, 15} {8, 5, 4, 6} {11, 6} {14, 1} {15, 14, 4} {8, 4, 0}
9	{15, 7} {5, 6, 7} {14, 0} {6, 2} {8, 3} {18, 16, 2}
10	{3, 8, 15} {11, 12} {11, 4, 8} {12, 5, 15} {12, 11} {15, 17, 2}
15	{5, 6} {11, 5} {10, 8, 5} {4, 2, 17, 10, 12} {12, 11, 3, 18, 17, 5, 4, 2, 7}
16	{13, 14} {4, 1} {9, 13, 14} {12, 11, 8} {14, 12, 4} {18, 6}

Table 4 Average Execution times of GA and CSGA

	GA	CSGA
Example (19 parts, 1000 iterations)	14.2s	6.7s

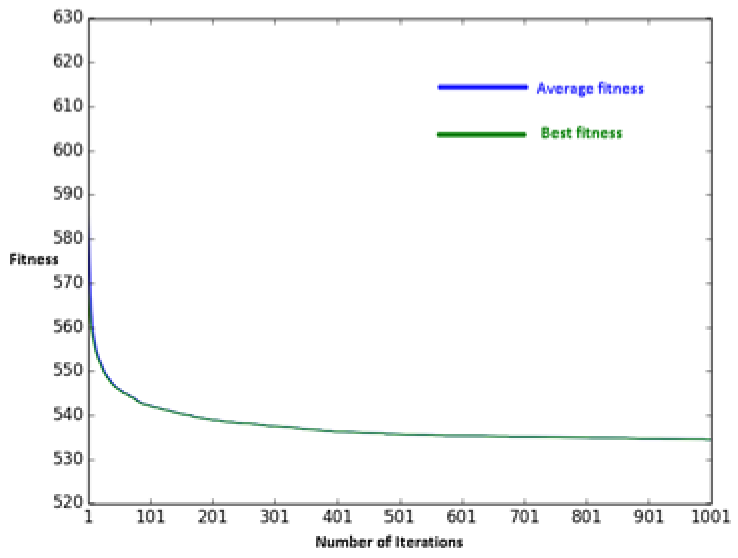

Fig. 2 Results of GA

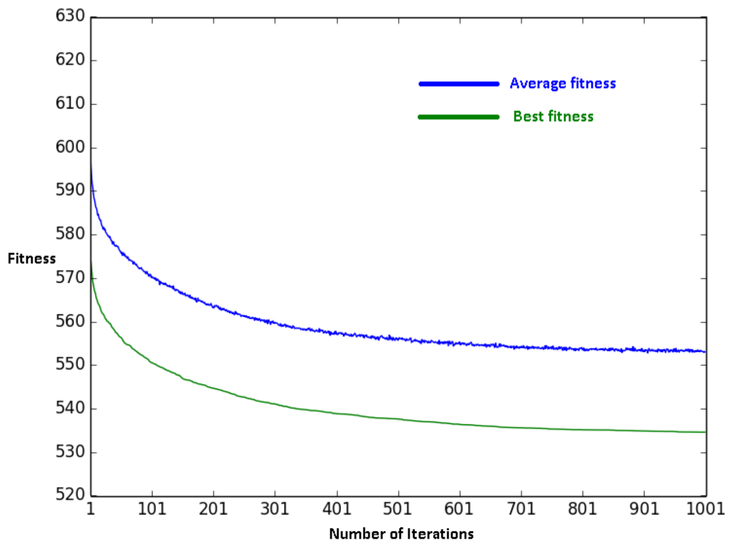

Fig. 3 Results of CSGA