World Academy of Science, Engineering and Technology
International Journal of Industrial and Manufacturing Engineering
Vol:11, No:8, 2017

Two Points Crossover Genetic Algorithm for Loop Layout Design Problem

Xu LiYun, Briand Florent, Fan GuoLiang

***Abstract*—**The loop-layout design problem (LLDP) aims at optimizing the sequence of positioning of the machines around the cyclic production line. Traffic congestion is the usual criteria to minimize in this type of problem, i.e. the number of additional cycles spent by each part in the network until the completion of its required routing sequence of machines. This paper aims at applying several improvements mechanisms such as a positioned-based crossover operator for the Genetic Algorithm (GA) called a Two Points Crossover (TPC) and an offspring selection process. The performance of the improved GA is measured using well-known examples from literature and compared to other evolutionary algorithms. Good results show that GA can still be competitive for this type of problem against more recent evolutionary algorithms.

***Keywords*—**Crossover, genetic algorithm, layout design problem, loop-layout, manufacturing optimization.

I. Introduction

FLEXIBLE manufacturing systems (FMSs) are automated production systems designed to produce a variety of parts, where machines and material handling devices are computer controlled. The configuration of the facilities along a production line has a significant influence over the system overall performance. According to Tompkins et al. [1], approximately 8% of the U.S. gross national product is spent every year to build new facilities that need to be planned. Additionally, research by Huang et al. [2] showed that more than 35% of the system efficiency is likely to be lost by applying incorrect layout and location designs. A well-designed line layout is primordial in order to increase output and to decrease the time and manpower required for each task. This can be summed up into the reduction of the material handling cost (MHC). The productivity of the system, as well as the MHC and production times, is greatly influenced by the layout of the machines. Those MHCs can count up to a half of the total operating expenditures in manufacturing and they can be reduced by at least 10 to 30% with an efficient layout [3]. Reducing the MHC means also reduced material movement and throughput times, less product damage, simplified material control and scheduling, and less overall congestion.

The loop layout is a common disposition type of manufacturing system. It comprises a closed ring like network of machines with material handling device such as conveyers, handling robots, path of unidirectional automated guided vehicles (AGV), or overhead monorail systems. Each part has a specified sequence of machines to visit along a unidirectional material handling device in order to complete its processing. Every time a process is finished on a machine, the part is moved to the next machine on the unidirectional material-handling network. If this next workstation is occupied, the part is stored in a buffer waiting until the machine is available. Each station has secondary handling equipment so that part can be brought to and transferred-from the station work ahead to the material handling loop. All the parts enter and exit the system by a unique load/unload station (Fig. 1).

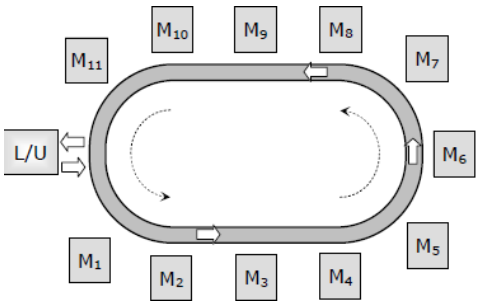

Fig. 1 Unidirectional loop layout

The loop layout shows several advantages over a conventional single line layout: material handling flexibility and agility in accommodating new parts and processes. According to Afentakis [4], this layout has relatively low initial costs since it contains few material handling links to connect all the workstations together. It thus shows a high flexibility degree and can satisfy all the requirements of material handling because all machines are reachable from all other machines. Process changes and new products can easily be integrated when using this type of layout for manufacturing systems. The optimization of the layout is a necessary step for FMSs. It will provide the solid base for an efficient running of the system and drastically reduce the material handling expenses.

The LLDP has been subject to numerous studies during the last decades with all types of methodology and heuristic algorithms. The recent studies have primarily focused on applying new meta-heuristics or extending the problem with multiple objectives. The GA is a widespread and renowned meta-heuristics that already proved its effectiveness for the LLDP. This paper aims at applying recent optimization techniques and newer operators to improve the GA's performance and to compare it with newer algorithms on a common computation data input. The rest of this paper is organized as follows. Section II is a literature review of previous approaches for the LLDP. Section III describes the problem in more details and Section IV introduces the

Xu LiYun, Florent Briand, and Fan GuoLiang are with the School of Mechanical Engineering, Tongji University, 4800 CaoAn Road, ShangHai 201804, China (e-mail: lyxu@tongji.edu.cn, florent.briand56@gmail.com, 2014fanguoliang@tongji.edu.cn).

mathematical model that supports this work. Section V describes the improvement implemented to the algorithm, and Section VI introduces the computational experiments to be discussed in Section VII. Finally, Section VIII summarizes the contribution and introduces some suggestions for future works. First, a review of the previous approaches for the LLDP will be given before introducing the improved algorithm, and finally, we compare its results with the previous studies.

II. Literature Survey

A LLDP deals with the assignment of m_n facilities to m_n candidate locations. Afentakis [4] was the first to address the unidirectional loop layout problem in 1989. He suggested the use of traffic congestion as a measure to evaluate such loop layout. The congestion is defined as the number of times that a part traverses the loop before its processing is completed. There are two kinds of congestion measures commonly used in loop layout design: min_sum and min_max.

- A min_sum problem attempts to minimize the total congestion of all parts;
- A min_max problem attempts to minimize the maximum congestion among a family of parts.

Afentakis proposed a graph-theoretic model for the min_sum loop layout problem. Leung [5] proposed a graph-theoretic model for the min_max loop layout problem. Bozer and Rim [6] used a branch and bound algorithm to solve the LLDP as well as Kouvelis and Kim [7]. As the problem has been proven to be NP-hard [7], the best way to tackle it is through heuristics techniques. Two heuristics called Move and Move/Interchange were presented by Tansel and Bilen [8]. Researchers started using meta-heuristics techniques inspired by nature, since those algorithms are well suited to deal with the combinatorial nature of the LLDP. The min_max congestion measure using a GA inspired by the analogy of population genetics and Darwin's natural selection was proposed by Cheng and Gen [9] and Banerjee and Zhou [10]. Cheng and Gen [11] also used a hybrid GA and neighborhood search. Tian et al. [12] used the simulated annealing algorithm (SAA) which mimics the process of annealing metals. Bennell et al. [13] proposed tabu search (TS) algorithm and an iterated decent randomized insertion algorithm to solve the min_max LLDP. Nearchou [14] approached LLDP with a differential evolution algorithm (DEA), whereas Kumar et al. [15] proposed a particle swarm optimization algorithm (PSO) to solve the same LLDP with min_sum, as well as the artificial immune system (AIS) [16]. Ma et al. [17] proposed dual system method with differential evolution algorithm (DEA) and GA to solve the loop-based station sequencing problem. Zhang et al. [18] applied the GA to solve a concrete case of loop layout in the industry by optimizing a blade workshop, taking into account the material flow between machines. Niroomand and Vizvari [19] suggested a new mathematical model formulated for the closed loop layout with exact distances. Then, they implemented a modified Migrating Birds optimization algorithm to solve the exact distances LLDP [20]. Ramezani et al. [21] presented a robust design for a closed loop supply chain network under an uncertain environment. Saravanan and Kumar [22] also proposed the implementation of the sheep flock heredity algorithm (SFHA) to the LLP as well as Anandaraman [23]. Manita and Korbaa [24] applied the Ant Colony optimization algorithm to the LLP while taking into account proximity constraints and machine dimensions. Hou et al. [25] optimized a manufacturing system composed of multi loop layout around a transfer loop with co-evolutionary methodology.

III. Problem Description

The LLDP solutions can be represented as a permutation of machines (*m1, m2…mn*) assigned to *mn* locations with an additional loading/unloading station at location 0. As previously explained, the optimization of the layout in this study will be centered on the improvement of the traffic congestion as introduced by Afentakis [4] with min-sum and min-max. Both indexes rely on the notion of additional circuits, or reloads, that occur when a part must travel from a machine on a given location to another machine on another location that is upstream of the unidirectional flow of the material handling equipment. The more reloads a given layout (represented as a permutation of machines π) needs, the more congested the traffic of the system will be.

The reloads counting can be done by running a simple test. First, consider the given notations:

- $Pk = \{1, \dots, p\}$: Set of products to be manufactured
- $M = \{1, \dots, m\}$: Set of machines to be arranged around the loop
- $M_k = (\mu_{1k}, \dots, \mu_{n_k k})$: Routing sequence through machines μ for a product k
- n_k : Number of operations required
- π : Ordering of the machines around the loop
- $\lambda_i(\pi)$: Location of machine i in ordering π, with $i \in M$
- $\alpha_{jk}(\pi)$: Reload variable for product k with respect to ordering π when moving from machine μ_{jk} to machine μ_{jk+1}
- $c(\pi)$:Cost of ordering π (min-max and min-sum indexes)

A reload for a given product k occurs if, after completing operation on a machine $\mu_{j,k}$ located on location $\lambda_{\mu_{j,k}}$, it needs to be transferred to another machine $\mu_{j+1,k}$ located on $\lambda_{\mu_{j+1,k}}$ with $\lambda_{\mu_{j,k}} > \lambda_{\mu_{j+1,k}}$.

IV. Mathematical Model

The problem's formulation, for a given layout π:

$$\alpha_{jk}(\pi) = \begin{cases} 1 \ if & \lambda_{\mu_{j,k}}(\pi) > \lambda_{\mu_{j+1,k}}(\pi) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

with

$$\mu_{j,k}(\pi) \neq \mu_{j+1,k}, \forall j \in M_k \quad (2)$$

Equation (2) ensures that the consecutive machines' locations stay different. This test is the base upon which the congestion indexes are then described, i.e. the number of reloads completed by product k with respect to ordering π:

$$c_{min-sum}(\pi) = \sum_{k=1}^{P} \sum_{j=1}^{n_k-1} \alpha_{jk}(\pi) \quad (3)$$

or simplified as:

$$c_{min-sum}(\pi) = \sum_{k=1}^{P} reloads \quad (4)$$

For min-max:

$$c_{min-max}(\pi) = \max\left(\sum_{j=1}^{n_k-1} \alpha_{jk}(\pi)\right) \quad (5)$$

or simplified as:

$$c_{min-max}(\pi) = \max(reloads) \quad (6)$$

V. Design of Improved GA

Developed by Holland [26] in 1975, they have since been applied in a large range of domains such as engineering, social sciences, or physical sciences. It uses the same principle as the biological evolution in nature to improve generation after generation a given sets of solutions. It starts with a population in which each individual, called a chromosome, is a string of numbers corresponding to a layout combination of the facilities. Two chromosomes (called parents) are selected based on their score obtained via the fitness function (the lower the better). The parents mate and generate an offspring through an operation called crossover. Generation after generation, the stronger individuals, i.e. among the best solutions to the problem, are the survivors in a competitive environment. The population thus tends towards an optimal solution for the problem.

A. TPC Crossover Process

Unlike the available genetic operators previously proposed for the LLDP (such as partially mapped crossover, exchange crossover etc.), this study uses a crossover operator so as to search only the feasible space of offspring; thus, we save computational time by avoiding infeasible space. The TPC also considers the relative orders of genes within the two parents and thus creates better offspring able to intersect their parents' characteristics.

The idea of the TPC is to first randomly select two crossover points (gene number 3 and 7 in the following example Fig. 3). All the genes from parent 1 (P1) between those two points (genes 1, 5, 4) are then rearranged considering the order of those same genes in P2, which is 4, 1, 5. Same process for the genes of P2 is rearranged according to P1.

B. Child Competition

The presented algorithm features a selection within the crossover offspring sub-population to select the best children that will be reintroduced within the initial population, replacing the worst parents. A selected child is forbidden to be reintroduced if it already exists in the parents' population. This mechanism ensures genetic variety and prevents premature convergence of the algorithm.

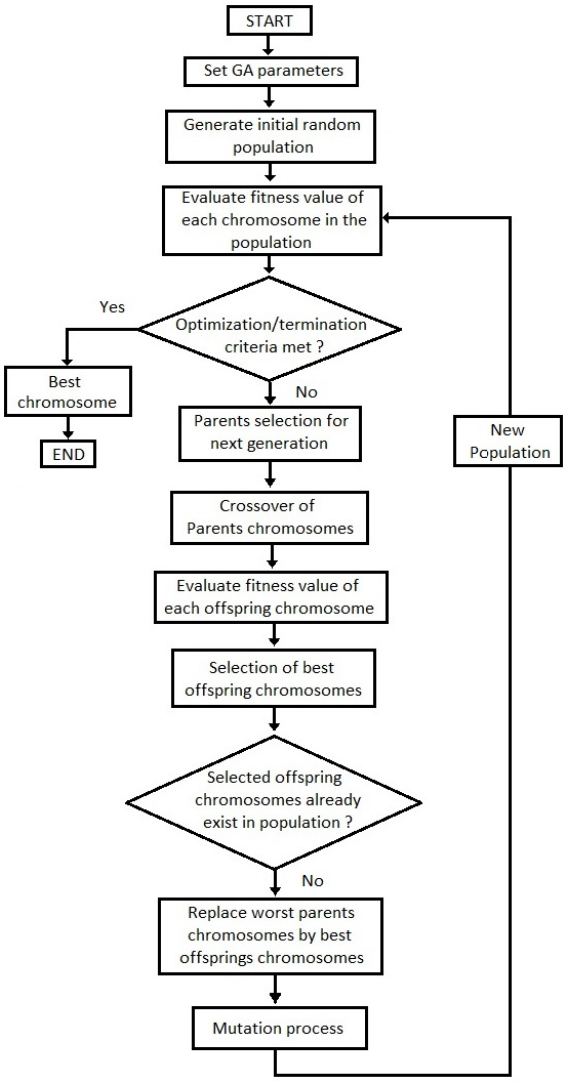

Fig. 2 GA diagram

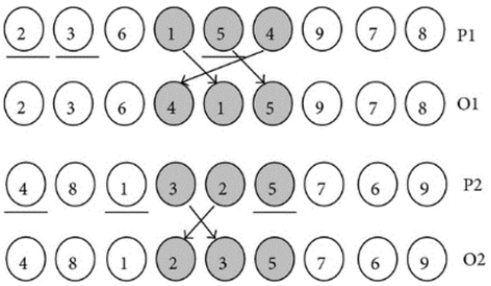

Fig. 3 TPC

C. Mutation Process

A mutation operation is applied after crossover to ensure a greater diversity within the population and thus to consider a greater search space, as well as preventing the population from premature convergence. A chromosome is then randomly selected and two genes are swapped to create a mutant which is re-introduced in the population. The number of mutations occurring in one generation over the total size of the population is the mutation probability. The mutation occurs whether it

affects the quality of the population favorably or unfavorably. Its role is to induce changes and bring in new genetic features that increase the variety of the parents' pool.

The mutation process chosen here was the swap process because it provides a random change in the selected chromosome, where the crossover process uses logic and exchange between chromosomes. This chaotic factor breaks the overall generational continuity of the population and provides a greater genetic diversity.

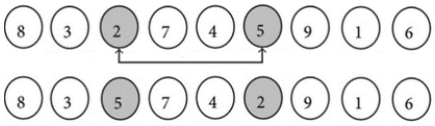

Fig. 4 Swap mutation process

VI. Computational Experiments

This section discusses the performance and computational efficiency of the proposed crossover operator in comparison to other meta-heuristics. Cheng and Gen's GA [9] will be used as comparison against its PMX crossover. The results will be also compared with those of the SAA of Tian et al. [12], the DEA of Nearchou [14], and Kumar et al. studies algorithms (PSO, AIS, SFHA) [15], [16], [22]. All the results for those algorithms are taken from Kumar [22].

The SAA study of Tian et al. [12] contained six different perturbation schemes (PS1–PS6) in order to generate random permutation solutions: random interchange of two adjacent terms (PS1), random interchange of two nonadjacent terms (PS2), single-term insertion (PS3), random movement of a subsequence of terms (PS4), reversion of a randomly selected subsequence of terms (PS5), reversion and/or movement of a randomly selected subsequence of terms (PS6).

The comparison will be run on the examples taken from Nearchou [14] as it is the common examples used in the literature on the LLDP. It contains a wide variety of situations, with randomly generated problems ranging from 10 machines and three parts to 30 machines and 10 parts, described in Table I. Nearchou kept the GA from Cheng and Gen [11] with the original parameters, which are population size = 20, crossover probability Pc = 30% and mutation probability Pm = 30%. To be fair and produce an objective comparison, the different algorithms were left run for a maximum of 20,000 evaluations. An evaluation is defined as a single computation of the objective function of a candidate solution. This study will thus use the same parameters as Cheng and Gen run on the same examples described in Nearchou with the same evaluation limitation.

The Kumar's PSO and SFHA algorithms on the other hand were left running for 40,000 evaluations, so his comparison was obviously not fair compared to the one of Nearchou. A new comparison is then conducted with this new limitation factor for the hereby proposed TPC GA algorithm. The results also feature several measurements such as Solution Effort (SE) describing the efficiency of an algorithm to reach an optimal solution. This factor is calculated as below:

$$SE(\%) = \frac{ne_{opt}}{ne_{total}} \times 100 \quad (7)$$

where ne_{opt} is the number of evaluation performed by the algorithm to achieve its optimal solution, and ne_{total} is the total number of evaluations. A low SE value could imply a fast convergence characterized as premature, whereas a high SE value could suggest that the algorithm has the ability to find even better solutions if running for a greater number of iterations.

The CPU time in seconds until the convergence of the algorithm is also measured. Please note that the Nearchou's results (DEA, GA and SAA) were obtained on a Pentium IV (1.7 GHz) personal computer and programmed in Pascal language. So, the CPU time results are obviously longer than our algorithm run on a Core i5 (2.5 GHz) processor under a Microsoft Windows 10 operating system and programmed in MATLAB. Kumar's results (PSO, AIS and SFHA) are also obtained on a Pentium IV and programmed in Java language.

VII. Results and Discussion

The different computation results are summed up into Table I. They are expressed into percentage of variation from the TPC GA compared to the previous best result from other algorithms on the same problem (a negative percentage means better). The results for the biggest problem (30 machines, 10 parts) are the most meaningful as the small dimension problems have lower values in congestion, which translates into huge variations in the percentages. It can thus be observed a fair improvement of the congestion on the biggest problem, which means the algorithms is worth studying and implementing.

Usually one of the main drawbacks of the classic GA is its fast convergence rate, often premature in regard to the other algorithms rates. It usually prevents the algorithm to find an optimal solution as it is often trapped into a local optimum. But, it also allows it to produce good solutions quite fast compared to the other evolutionary algorithms.

A big difference in SE% can be seen between GA and TPC GA: the original GA of Cheng and Gen tends to converge quite rapidly, while the TPC GA version used hereby has a slower convergence indicator. This is due to the selection process in the offspring population after crossover that refuses any offspring child that already exists in the initial parents' population. This method considerably reduces premature convergence of the algorithm compared to the original version of Cheng and Gen. The genetic diversity is improved throughout the generations and leads to a greater pool of possible solutions. The high SE% observed for the improved GA signifies that the algorithm could be left running even longer to find better solutions. The cross over operator chosen also provides a faster calculation due to its anti-illegal offspring nature that saves the computation time necessary for test and replacement of infeasible children. It is also interesting to note the differences in CPU times measured, representing as previously stated the time for convergence of the algorithms. The large CPU times recorded for the improved algorithm are also a result of this high SE%. But, actually the overall

processing time of the studied GA is lesser than the other algorithms as calculated in Table VI on the most complex problem (30 machines, 10 parts) for meaningful results (deducted for other algorithms from SE% and convergence times). The improved GA is faster than the others algorithms. But, this difference can be attributed to the more powerful CPU used to run this algorithm compared to the previous studies of Nearchou and Kumar.

TABLE I
IMPROVEMENT PERCENTAGE FOR TPC GA

Problem (No. of machines, No. of parts)	No. of computations	Min-Max	Min-Sum
10-3	20,000	0	0
	40,000	0	0
20-5	20,000	+20	-20
	40,000	+6	0
15-9	20,000	0	-14
	40,000	+33	+33
30-10	20,000	-33	-11
	40,000	-2	-12.5

VIII. CONCLUSION

The improved GA presented hereby shows very good results in light of classical examples in literature on the subject. The different mechanisms used for optimization such as selection of offspring after crossover, elitism strategy, anti-illegal offspring crossover procedure and anti-reproduction of already existing chromosomes proved quite efficient and deserves to be implemented on further algorithms. It could also be interesting to apply to the other problems such as the bi-directional loop layout problem or with the other factors such as clearance limitations or multiple load/unload stations, or even shortcuts in the layout. Waiting times or limited capacity buffers could also be taken into account to better reflect the concrete operations in factories.

APPENDIX

This appendix presents the data input for computation (taken from [14]) and the results of calculation for each problem. Bold lines represent the best results.

TABLE II
MACHINE SEQUENCE

Problem No.	No. Of Machines, No. of Parts	Part No.	Required Machine Sequence
1	10, 3	Part 1	2–1–6–5–8–9–3–4
		Part 2	10–8–7–5–9–6–1
		Part 3	9–2–7–4
2	20, 5	Part 1	4–2–3–12–1–9–16–18–5–8–20–15–14–6–11
		Part 2	10–9–1–3–18–17–5–6–2–11–4
		Part 3	17–11–6–8–7–15–16–9–1–20
		Part 4	14–17–11–3–16–5–13–18–20–19–12–10–6–8–15
		Part 5	6–18–8–4–2–7–5–9–14–19–1–20–10–16–11–15–13–12
3	15, 9	Part 1	4–2–5–1–6–8–14–9–11–3–15–12
		Part 2	3–2–15–14–11–1–7–10–4–5–13–6–9
		Part 3	5–6–11–15–2–12–3–4
		Part 4	10–9–4–14–2–3–15–8
		Part 5	11–2–4–14–5–3–15
		Part 6	8–10–12–11–15–13–1–14–4–5–3
		Part 7	5–11–10–3–7–13–8
		Part 8	7–3–2–8–4–10–6–15–13–9–1
		Part 9	11–13–3–1–12–14–4–8–9–2
4	30, 10	Part 1	6–3–4–18–5–1–14–24–26–7–11–30–23–21–13–27–9–16–17–2–25–8–15
		Part 2	17–9–11–8–10–22–24–13–2–29–23–21–25–16–4–20–26–18–15–12–27–6–3–7–28
		Part 3	13–2–6–29–21–3–14–24–12–15–17–8–1–22–28–10–7–30–20–19
		Part 4	7–2–6–11–21–8–16–30–1
		Part 5	3–17–1–2–20–22–8–6–26–19–14–11–15–12–7–16–21–10–28–23–18–4–27–24–25–13–30– 9–5
		Part 6	30–9–2
		Part 7	15–9–30–19–12–3–6–5–8–14–7–28–23–1–29–24–27–2–13–4–26–16–11–10–25–21–22–20–18
		Part 8	7–19–5–4–9–16–3–14–28–13–11–2–21–10–17–22–26–23–29–30
		Part 9	21–4–1–6–11–22
		Part 10	12–6–17–15–13–30–26–18–14–9–7–11–23–2–4–25–24

TABLE III
COMPARISON FOR THE 10-MACHINE, 3-PART PROBLEM

Number of evaluations	Algorithm	Cost min_sum	Cost min_max	SE (%)	CPU time	Congestion for each part	Optimal order of machines
				MIN_SUM approach			
	DEA_1	4	3	1.9	0.2	3–1–0	10–9–2–3–8–6–1–7–5–4
	DEA_2	**3**	**2**	**22.5**	**2.0**	**1–2–0**	**6–5–10–8–9–3–2–7–1–4**
	GA	**3**	**2**	**1.0**	**0.0**	**1–2–0**	**10–5–8–9–3–2–7–4–1–6**
	SA_PS1	5	2	5.2	1.4	2–2–1	6–5–10–8–1–9–7–3–4–2
20,000	SA_PS2	5	2	0.4	0.2	2–2–1	3–2–6–1–7–5–10–8–4–9
	SA_PS3	**3**	**2**	**24.1**	**0.4**	**1–2–0**	**10–8–9–2–7–1–3–6–4–5**
	SA_PS4	**3**	**2**	**9.9**	**1.3**	**2–1–0**	**5–10–8–9–2–3–7–6–4–1**
	SA_PS5	**3**	**2**	**3.7**	**0.8**	**2–1–0**	**5–10–8–9–2–6–1–3–7–4**
	SA_PS6	**3**	**2**	**32.6**	**3.3**	**2–1–0**	**8–9–5–10–1–4–2–6–7–3**
	TPC GA	**3**	**2**	**8.7**	**0.7**	**2-1-0**	**10–5–8–9–3–2–6–7–4–1**
	PSO	**3**	**2**	**18.0**	**–**	**2–1–0**	**5–10–8–9–2–3–7–6–4–1**
40,000	**AIS**	**3**	**2**	**7.2**	**0.0**	**2–1–0**	**10–5–8–9–2–7–3–4–6–1**
	SFHA	**3**	**2**	**6.7**	**0.0**	**1–2–0**	**10–6–5–8–9–2–7–3–4–1**
	TPC GA	**3**	**2**	**77.1**	**7.0**	**1–2–0**	**6–10–5–8–9–3–2–7–4–1**
				MIN_MAX approach			
	DEA_1	6	2	12.8	0.9	2–2–2	2–3–4–6–7–5–10–8–9–1
	DEA_2	4	2	0.1	0.0	2–2–0	9–2–6–3–7–10–5–1–4–8
	GA	6	2	0.4	0.0	2–2–2	2–10–8–3–4–1–9–6–7–5
	SA_PS1	5	2	2.3	0.7	2–2–1	7–5–10–8–9–1–3–4–6–2
	SA_PS2	4	2	0.4	0.5	2–2–0	9–10–6–2–7–5–3–4–1–8
20,000	**SA_PS3**	**3**	**2**	**0.1**	**0.5**	**2–1–0**	**5–10–8–9–2–6–3–7–1–4**
	SA_PS4	6	2	0.1	0.7	2–2–2	10–7–2–5–6–8–9–3–1–4
	SA_PS5	7	3	0.1	0.2	3–3–1	1–5–8–10–9–4–2–7–6–3
	SA_PS6	5	2	0.2	1.3	2–2–1	10–6–2–3–7–1–4–5–8–9
	DEA_2	4	2	0.1	0.0	2–2–0	9–2–6–3–7–10–5–1–4–8
	TPC GA	**3**	**2**	**90.1**	**4.5**	**1–2–0**	**6–5–10–8–9–2–7–3–1–4**
	AIS	**3**	**2**	**8.0**	**0.0**	**1–2–0**	**10–8–9–3–2–7–1–6–5–4**
40,000	**SFHA**	**3**	**2**	**7.8**	**0.0**	**1–2–0**	**5–10–8–9–2–1–3–7–4–6**
	TPC GA	**3**	**2**	**91.5**	**9.5**	**1–2–0**	**6–5–10–8–9–2–3–7–4–1**

TABLE IV
COMPARISON FOR THE 20-MACHINE, 5-PART PROBLEM

Number of evaluations	Algorithm	Cost min_sum	Cost min_max	SE (%)	CPU time	Congestion for each part	Optimal order of machines
					MIN_SUM approach		
	DEA_1	23	7	32.4	2.0	6–3–2–5–7	18–1–2–20–3–14–10–17–5–6–8–7–11–12–15–13–16–4–19–9
	DEA_2	17	6	66.6	3.3	4–2–2–3–6	14–10–18–17–5–6–8–4–2–7–9–1–20–11–3–15–19–13–16–12
	GA	**18**	**5**	**3.4**	**1.1**	**4–2–3–4–5**	**10–9–14–6–19–2–16–12–18–1–17–5–8–11–20–3–7–15–4–13**
	SA_PS1	22	7	53.0	0.6	5–4–2–4–7	12–2–14–6–8–7–1–17–20–10–15–11–3–16–19–18–5–9–13–4
20,000	SA_PS2	22	6	38.8	5.9	6–4–4–4–4	14–13–3–16–17–19–9–1–7–11–5–18–6–8–15–20–4–12–10–2
	SA_PS3	21	5	80.8	6.9	4–4–3–5–5	20–14–2–6–19–17–15–7–10–18–8–12–11–1–9–5–4–3–16–13
	SA_PS4	21	6	27.3	2.3	6–4–3–4–4	10–6–2–17–16–1–18–8–20–11–7–15–5–13–19–12–4–9–3–14
	SA_PS5	22	6	20.1	1.5	4–4–4–6–4	13–4–9–7–14–19–1–2–3–6–12–20–10–16–18–17–11–15–5–8
	SA_PS6	22	5	20.7	3.0	4–4–4–5–5	10–4–16–14–7–2–19–17–5–1–3–6–11–9–18–8–13–12–20–15
	TPC GA	18	6	91.0	6.1	6–3–2–3–4	4–10–2–7–16–17–5–9–1–11–3–13–14–6–18–8–20–15–19–12
	PSO	**16**	**4**	**43.3**	**–**	**4–3–2–3–4**	**10–16–18–5–9–14–6–1–8–4–2–17–20–11–19–7–15–13–3–12**
40,000	**AIS**	**16**	**4**	**55.4**	**2.8**	**4–2–2–4–4**	**10–16–18–17–5–6–9–8–4–1–2–7–11–20–15–14–19–13–3–12**
	SFHA	**16**	**4**	**41.2**	**2.5**	**4–2–2–4–4**	**10–16–18–17–5–6–9–8–1–4–2–7–11–20–15–14–19–13–3–12**
	TPC GA	17	5	62.1	8.9	5–4–2–2–4	5–9–14–6–13–18–8–1–4–2–17–11–20–19–3–7–15–12–10–16
				Number of evaluations			
				MIN_MAX approach			
	DEA_1	28	7	1.3	0.9	6–5–4–7–6	6–12–17–9–20–3–15–14–19–1–16–8–10–11–13–4–2–18–7–5
	DEA_2	23	5	21.4	2.2	5–5–4–4–5	1–4–10–3–2–16–9–17–14–6–18–11–20–5–8–15–19–7–13–12
	GA	24	5	4.1	2.9	5–5–4–5–5	11–9–2–3–18–13–15–20–10–4–17–6–16–14–5–7–8–1–19–12
	SA_PS1	24	5	45.2	2.7	5–5–4–5–5	6–17–13–2–7–18–5–11–20–3–19–12–9–15–10–1–8–16–14–4
	SA_PS2	25	5	62.2	8.0	5–5–5–5–5	3–8–9–12–13–10–14–20–5–15–18–7–6–17–11–16–2–19–1–4
20,000	SA_PS3	24	5	3.8	2.8	5–4–5–5–5	9–3–10–17–14–16–7–5–8–6–19–4–2–13–12–1–11–18–15–20
	SA_PS4	24	5	6.2	1.4	4–5–5–5–5	19–6–13–16–18–4–2–7–5–11–12–10–8–15–1–9–20–3–14–17
	SA_PS5	28	6	42.4	5.7	6–4–6–6–6	3–16–5–9–20–10–15–7–14–1–6–18–8–11–19–4–13–2–17–12
	SA_PS6	28	5	45.9	3.1	4–5–5–5–5	20–15–4–9–16–12–5–11–2–1–13–18–7–17–19–6–14–3–8–10
	TPC GA	**20**	**4**	**39.3**	**3.0**	**4–4–4–4–4**	**14–4–2–10–16–19–17–7–5–1–3–6–13–18–8–9–20–11–12–15**
	AIS	17	4	18.2	1.6	4–4–2–3–4	18–5–9–14–6–1–8–4–2–17–20–11–3–7–15–19–13–12–10–16
40,000	**SFHA**	**16**	**4**	**16.5**	**1.4**	**4–2–2–4–4**	**10–16–18–17–5–9–6–8–4–1–2–7–11–20–15–14–19–13–3–12**
	TPC GA	**16**	**4**	**70.3**	**11.5**	**3–2–4–3–4**	**4–10–9–1–14–6–2–3–16–7–18–17–11–5–8–20–15–13–19–12**

TABLE V
COMPARISON FOR THE 15-MACHINE, 9-PART PROBLEM

Number of evaluations	Algorithm	Cost min_sum	Cost min_max	SE (%)	CPU time	Congestion for each part	Optimal order of machines
						MIN_SUM approach	
20,000	DEA_1	28	4	47.8	12.0	3–4–4–4–2–4–2–3–2	9–4–5–11–13–7–10–3–1–2–12–6–8–15–14
	DEA_2	**24**	**4**	**39.5**	**9.0**	**3–4–3–2–2–4–1–4–1**	**5–7–11–13–10–3–1–6–15–12–8–14–9–2–4**
	GA	**24**	**4**	**31.6**	**3.8**	**2–4–3–2–2–4–1–4–2**	**4–7–5–11–13–10–3–1–6–15–8–14–9–2–12**
	SA_PS1	26	5	77.2	29.9	5–5–2–2–1–3–2–3–3	7–5–6–14–11–3–8–15–13–2–10–1–9–12–4
	SA_PS2	28	5	62.9	16.3	5–4–3–2–2–3–1–3–5	4–12–2–5–3–15–14–6–7–8–11–13–10–9–1
	SA_PS3	26	5	34.7	9.9	5–4–2–2–1–4–1–3–4	10–14–7–3–15–11–1–12–2–13–5–6–9–8–4
	SA_PS4	28	5	29.3	2.7	5–5–3–2–1–3–2–3–4	14–7–5–10–9–12–11–6–3–2–15–4–13–1–8
	SA_PS5	32	6	40.8	6.3	5–3–2–2–3–6–1–5–5	12–10–3–1–15–4–14–5–7–2–13–6–8–9–11
	SA_PS6	26	5	37.7	1.4	4–4–2–2–2–5–1–3–3	7–1–10–8–5–14–11–3–2–6–13–9–4–12–15
	TPC GA	**24**	**4**	**27.4**	**1.2**	**2–4–3–2–3–4–1–4–1**	**5–7–11–13–10–3–1–6–15–8–12–14–9–4–2**
40,000	PSO	24	4	33.3	–	2–4–4–2–2–3–1–4–2	4–5–11–7–10–3–15–13–1–12–6–8–14–9–2
	AIS	**24**	**3**	**31.2**	**4.7**	**3–3–3–3–2–3–1–3–3**	**5–11–7–10–3–2–15–13–1–6–12–8–14–9–4**
	SFHA	**24**	**3**	**31.1**	**4.5**	**3–3–3–3–2–3–1–3–3**	**5–7–11–10–3–2–15–13–1–6–12–8–14–9–4**
	TPC GA	24	4	46.9	4.2	2–4–4–2–2–4–1–4–1	4–5–7–11–10–13–3–1–6–15–12–8–14–9–2
						Number of evaluations	
						MIN_MAX approach	
20,000	DEA_1	33	6	47.7	12.5	4–6–4–4–3–4–2–4–3	9–4–5–11–13–7–10–3–1–2–12–6–8–15–14
	DEA_2	28	4	10.7	14.9	3–4–3–3–3–3–3–3–3	7–11–3–4–2–15–5–13–8–1–10–9–6–12–14
	GA	28	4	13.3	13.3	4–4–3–3–2–4–2–3–3	7–14–4–11–2–10–5–13–12–3–6–8–9–1–15
	SA_PS1	32	4	64.7	17.0	4–3–3–4–3–4–3–4–4	8–7–11–10–3–5–2–15–14–13–1–6–9–12–4
	SA_PS2	31	4	23.7	1.9	4–4–3–4–2–4–3–4–3	4–5–10–13–1–7–8–6–9–11–3–2–12–15–14
	SA_PS3	32	4	20.1	1.5	4–4–4–3–4–4–2–3–4	15–7–5–10–12–14–13–11–9–1–4–3–6–2–8
	SA_PS4	34	4	36.1	2.3	4–4–4–4–3–4–3–4–4	15–13–11–10–4–5–1–7–12–3–6–2–14–8–9
	SA_PS5	30	4	41.8	7.2	4–4–4–3–2–4–2–4–3	14–4–11–5–10–3–2–1–15–13–12–8–9–6–7
	SA_PS6	30	4	45.2	8.3	4–4–4–3–1–3–3–4–4	4–13–1–8–14–7–10–6–9–5–11–12–3–2–15
	TPC GA	**24**	**4**	**70.8**	**3.3**	**2–3–4–3–2–4–1–3–2**	**4–5–7–11–13–10–3–1–6–2–15–8–12–14–9**
40,000	AIS	24	4	28.4	4.4	2–4–3–3–2–3–1–3–3	4–7–5–11–10–3–15–2–13–1–6–8–12–14–9
	SFHA	**24**	**3**	**26.9**	**4.2**	**3–3–3–3–2–3–1–3–3**	**5–11–7–10–3–2–15–13–1–12–6–8–14–9–4**
	TPC GA	24	4	34.4	3.2	3–4–3–2–2–4–1–4–1	7–5–11–10–13–3–1–6–15–8–12–14–9–2–4

TABLE VI
COMPARISON FOR THE 30-MACHINE, 10-PART PROBLEM

Number of evaluations	Algorithm	Cost min_sum	Cost min_max	SE (%)	CPU time (s)	Overall processing time (s)	Congestion for each part	Optimal order of machines
						MIN_SUM approach		
20,000	DEA_1	69	12	40.5	3.3	8.1	10–11–7–4–12 –1–9–8–2–5	12–23–15–3–4–6–17–9–7–13–11–22–5–26–30–16– 25–20–27–18–19–8–21–10–2–1–29–14–24–28
	DEA_2	53	12	74.7	3.6	4.8	5–6–7–2–12 –1–10–7–1–2	12–13–6–29–3–14–27–17–9–7–11–30–23–5–2–21– 4–10–25–1–22–8–20–24–26–19–18–15–28–16
	GA	61	12	8.5	3.0	35.3	8–7–8–3–12 –0–10–6–1–6	7–30–17–19–27–14–28–15–9–23–5–21–8–1–24–16– 12–13–6–11–10–3–2–4–22–20–29–26–25–18
	SA_PS1	68	14	74.2	16.5	22.2	8–9–7–3–11 –1–14–8–1–6	21–9–24–12–19–7–25–3–6–11–28–5–10–4–29–13– 23–17–8–2–18–1–16–27–30–22–15–14–20–26
	SA_PS2	68	11	81.5	11.0	13.5	8–9–8–3–11 –1–11–8–2–7	20–8–23–12–24–9–30–6–29–1–28–5–26–25–19–3– 11–16–4–10–7–14–17–13–18–27–15–21–2–22
	SA_PS3	69	12	61.6	15.2	24.6	10–9–7–4–12 –0–10–7–3–7	4–3–14–27–13–6–28–30–5–10–17–1–22–29–15–8– 25–12–6–9–23–16–11–20–2–21–7–24–18–19
	SA_PS4	68	13	34.9	6.6	18.9	9–7–8–3–13 –1–11–7–2–7	6–13–18–21–3–5–26–28–11–16–29–23–12–8–10–25– 17–4–14–7–22–19–15–27–9–30–1–2–20–24
	SA_PS5	69	14	37.7	1.5	4.0	8–9–8–4–14 –1–10–7–2–6	17–1–28–7–21–15–18–26–29–24–14–9–30–27–20– 19–3–16–22–25–10–11–13–8–5–6–23–2–4–12
	SA_PS6	68	12	35.9	9.3	25.9	8–9–9–3–12 –1–10–8–2–6	26–28–22–25–8–15–4–7–9–2–11–30–13–18–1–6– 27–16–10–12–5–3–17–19–23–29–14–24–21–20
	TPC GA	**53**	**8**	**92.1**	**7.7**	**8.4**	**8–8–6–2–7 –1–8–6–2–5**	**10–3–18–14–20–15–13–12–4–17–27–9–2–6–29–7– 16–11–21–22–25–30–28–24–26–19–5–23–8–1**
40,000	PSO	53	12	63.3	–	–	5–6–7–2–12 –1–10–7–1–2	13–12–6–3–14–29–17–27–9–7–11–30–5–23–2–21– 10–4–1–22–25–20–8–24–26–18–19–28–16–15
	AIS	50	8	27.9	6.4	22.9	7–6–6–2–8 –0–8–7–1–5	21–8–15–12–13–30–3–4–10–17–27–1–14–9–22– 7–2–6–25–20–29–24–26–16–11–19–28–23–18–5

Number of evaluations	Algorithm	Cost min_sum	Cost min_max	SE (%)	CPU time (s)	Overall processing time (s)	Congestion for each part	Optimal order of machines
	SFHA	50	8	26.3	6.0	22.8	7–6–6–2–8 –0–8–7–1–5	21–8–15–12–13–10–3–4–30–17–27–1–14–9–22– 7–2–6–25–20–29–24–26–16–11–19–28–23–18–5
	TPC GA	**49**	**8**	**76.1**	**12.5**	**16.4**	**7–7–7–3–7 –1–8–5–1–3**	**26–15–16–18–12–21–6–3–5–4–10–27–17–14–9–7– 28–13–11–30–23–1–2–20–25–29–22–8–19–24**
MIN_MAX approach								
20,000	DEA_1	79	12	40.5	3.7	9.1	9–10–9–5–12 –1–11–9–3–10	6–16–20–4–28–17–9–23–21–1–8–18–22–24–11– 28–10–30–13–27–25–3–19–29–15–14–2–12–5–7
	DEA_2	68	9	36.2	2.4	6.6	9–9–9–4–9 –1–9–8–3–7	12–28–27–8–11–23–30–3–17–1–18–4–2–21–10– 14–29–7–15–9–6–22–24–20–25–5–13–26–16–19
	GA	74	10	14.6	6.5	44.5	10–10–10–4–10 –1–10–7–2–7	6–9–7–18–10–3–15–17–28–23–19–25–1–13–16– 21– 14–4–29–11–22–30–12–24–2–6–20–27–5–8
	SA_PS1	77	11	14.2	7.8	54.9	11–11–8–5–11 –1–10–10–2–8	3–25–17–1–29–16–14–20–18–13–11–6–21–5–9–27– 22–8–10–2–24–4–19–7–15–30–28–23–12–26
	SA_PS2	76	11	17.0	1.7	10.0	10–8–11–4–11 –1–11–9–2–9	–15–10–21–24–19–12–17–9–4–7–20–5–2–1–25– 29– 26–14–28–16–22–3–27–23–13–18–11–8–30
	SA_PS3	73	11	3.4	3.9	114.7	10–9–10–2–11 –1–11–8–3–8	13–17–8–1–20–18–15–3–30–28–6–21–14–10–12– 4– 22–9–26–29–5–19–2–16–24–11–23–27–25
	SA_PS4	78	11	5.1	6.1	119.6	11–11–9–5–11 –0–11–10–2–8	13–17–8–1–20–18–15–3–30–28–6–21–14–10–12– 4– 22–9–26–29–5–19–2–16–24–11–23–27–25
	SA_PS5	76	11	10.3	3.3	32.0	10–10–9–4–11 –1–11–10–1–9	5–9–20–5–28–11–14–3–29–15–26–10–24–2–25–16– 19–23–13–8–12–30–17–18–21–4–1–27–7–22
	SA_PS6	71	11	7.9	1.0	12.6	10–10–8–4–11 –1–11–7–3–6	–3–25–20–16–23–21–10–6–14–15–22–26–29–9–18– 24–17–13—7–27–11–1–19–30–5–2–12–8–28
	TPC GA	**56**	**8**	**92.1**	**7.4**	**8.0**	**7–8–8–4–7 –1–7–7–1–6**	**20–26–19–23–8–1–18–15–14–12–25–6–9–29–7–16– 5–21–10–11–22–3–30–28–24–13–4–27–17–2**
40,000	AIS	51	8	21.8	5.8	26.6	8–7–6–2–8 –0–8–6–1–5	1–8–15–12–3–13–10–4–30–17–1–22–27–14–9–7–2– 20–28–25–6–26–11–23–29–24–19–18–5
	SFHA	50	8	19.4	5.5	28.4	7–6–6–2–8 –0–8–7–1–5	1–8–15–13–12–30–3–4–10–17–27–1–14–9–22–7–2– 6–25–20–29–24–26–16–11–19–28–23–18–5
	TPC GA	**50**	**7**	**93.5**	**15.5**	**16.6**	**7–7–7–3–7 –0–7–7–1–4**	**6–21–3–13–5–4–11–17–30–20–1–25–22–26–8–18– 19–15–14–24–12–10–9–7–28–23–27–2–16–29**

REFERENCES

[1] Tompkins J. A., White J. A., Bozer Y. A., Tanchoco J. M. A. (2003) Facilities planning, 3rd edn. Wiley, New York.
[2] Huang, C., Wong, C. K., & Tam, C. M. (2010). Optimization of material hoisting operations and storage locations in multi-storey building construction by mixed-integer programming. Automation in Construction, 19(5), 656–663.
[3] Tompkins J. A., White J. A. (1984) Facilities planning. Wiley, New York
[4] Afentakis, P. (1989): A loop layout design problem for flexible manufacturing systems, International Journal of Flexible Manufacturing Systems, 1, 2 175-196.
[5] Leung, J. (1992) A graph-theoretic heuristic for designing loop-layout manufacturing systems. European Journal of Operational Research, 57, 243-252.
[6] Bozer Y. A., Rim S. C. (1989) Exact solution procedures for the circular layout problem. Technical Report 8933. University of Michigan.
[7] Kouvelis P., Kim M. W. (1992). Unidirectional loop network layout problem in automated manufacturing systems. Oper Res 40:533–550.
[8] Tansel B. C., Bilen C. (1998) Move based heuristics for the unidirectional loop network layout problem. Eur J Oper Res 108(1):36–48.
[9] Cheng, R. and Gen, M. Genetic algorithms for designing loop layout manufacturing systems, in Proceedings of the 18th International Conference on Computer and Industrial Engineering, 1995, pp. 187 191.
[10] Banerjee P., Zhou Y. (1995) Facilities layout design optimization with single loop material flow path configuration. Int J Prod Res 33(1):183–203.
[11] Cheng, R., Gen, M. (1998): Loop layout design problem in flexible manufacturing systems using genetic algorithms, Computers and Industrial Engineering, 34, 1 53-61.
[12] Tian, P., Ma, J., Zhang, D.-Mo., 1999. Application of simulated annealing to the combinatorial optimization problem with permutation property: An investigation of generation mechanism. European Journal of Operational Research 118, 81–94.
[13] Bennell J. A., Potts C. N., Whitehead J. D. (2002) Local search algorithms for the min–max loop layout problem. J Oper Res Soc 53:1109–1117.
[14] Nearchou A. C. (2006): Meta-heuristics from nature for the loop layout design problem. Int J Prod Econ 101:312–328.
[15] Kumar R. M. S., Asokan P., Kumanan S. (2008) Design of loop layout in flexible manufacturing system using non-traditional optimization technique. Int J Adv Manuf Technol 38(5–6):594–599.
[16] Kumar R. M. S., Asokan P., Kumanan S. (2009) Artificial immune system based algorithm for the unidirectional loop layout problem in a flexible manufacturing system. Int J Adv Manuf Technol 40(56):553–565.
[17] Ma S., Liu Z. C., Shi Y. J. (2013) A dual system method with differential evolution and genetic algorithm for loop-based station sequencing problem. Inf Technol J 12(4):728–734.
[18] Hu Zhang, Min-min Xia, Li-ling Jiang, Yi Zhang, Tong-tong Lu, (2009) "Research on Applying Unidirectional Loop Layout to Optimize Facility Layout in Workshop Based on Improved Genetic Algorithm".
[19] Niroomand S, Vizvari B (2013) A mixed integer linear programming formulation of closed loop layout with exact distances. J Ind Prod Eng 30(3):190–201.
[20] Niroomand S., Vizvari B. (2016) Modified migrating birds optimization algorithm for closed loop layout with exact distances in flexible manufacturing systems. Expert Systems with Applications, Vol.42 (19) :6586-6597.
[21] Ramezani M., Bashiri M., Moghaddam R. T. (2012) A robust design for a closed loop supply chain network under an uncertain environment. Int J Adv Manuf Technol. 1–19 M.
[22] Saravanan & S. Ganesh Kumar (2014): Design and optimisation of loop layout problems flexible manufacturing system using sheep flock heredity algorithm.
[23] Anandaraman C. (2011) An improved sheep flock heredity algorithm for job shop scheduling and flow shop scheduling problems. Int J Ind Eng Comput 2(4):749–764.
[24] Manita G., Korbaa O., (2013), A Min-Max ANT colony algorithm for machine loop layout problem, 21st Mediterranean Conference on Control & Automation 978-1-4799-0997.
[25] Hou L., Liu Z., Shi Y., Zheng X., (2016) Optimizing Machine Assignment and Loop Layout in Tandem AGV Workshop by Co-Evolutionary Methodology, 20th International Conference on Computer Supported Cooperative Work in Design 978-1-5090-1915.
[26] Holland, J. H., (1975). Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor, MI.