Applied Soft Computing 49 (2016) 570–589

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Nature inspired algorithms to optimize robot workcell layouts

Zhen Yang Lim [a], Ponnambalam S.G. [a,*], Kazuhiro Izui [b]

[a] Advanced Engineering Platform and School of Engineering, Monash University Malaysia, 47500 Bandar Sunway, Malaysia
[b] Department of Mechanical Engineering and Science, Kyoto University, Yoshida-Honmachi Sakyo-Ku, Kyoto 606-8501, Japan

ARTICLE INFO

Article history:
Received 28 September 2015
Received in revised form 4 August 2016
Accepted 27 August 2016
Available online 31 August 2016

Keywords:
Robot workcell layout
Sequence-pair representation
B*-Tree representation
Multiobjective optimization
Nature-inspired algorithms

ABSTRACT

Multi-objective layout optimization methods for the conceptual design of robot cellular manufacturing systems are proposed in this paper. Robot cellular manufacturing systems utilize one or more flexible robots which can carry out a large number of operations, and can conduct flexible assemble processes. The layout design stage of such manufacturing systems is especially important since fundamental performances of the manufacturing system under consideration are determined at this stage. Layout area, operation time and manipulability of robot are the three important criteria when it comes to designing manufacturing system. The use of nature inspired algorithms are not extensively explored to optimize robot workcell layouts. The contribution in this paper is the use of five nature-inspired algorithms, viz. genetic algorithm (GA), differential evolution (DE), artificial bee colony (ABC), charge search system (CSS) and particle swarm optimization (PSO) algorithms and to optimize the three design criteria simultaneously. Non-dominated sorting genetic algorithm-II is used to handle multiple objectives and to obtain pareto solutions for the problems considered. The performance of sequence pair and B*-Tree layout representation schemes are also evaluated. It is found that sequence pair scheme performs better than B*-Tree representation and it is used in the algorithms. Numerical examples are provided to illustrate the effectiveness and usefulness of the proposed methods. It is observed that PSO performs better over the other algorithms in terms of solution quality.

1. Introduction

A typical automated manufacturing usually use one or more robots to perform different operation and manufacturing tasks. The focus of this paper is to propose efficient nature inspired algorithms to optimize certain criteria for Robotic workcell layouts. Cellular manufacturing assumed in this paper performs only assembly tasks. Cell concept is one of the applications from group technology where similar parts are grouped together. Robotic workcells offer advantages and potential to reduce cycle time and provides a more flexible production at a lower cost [1]. In robotic cellular workcells, human workers are replaced by robot in order to increase the efficiency [1]. The main advantage of this is that it is more flexible and efficient. The goal of these systems are to increase the productivity and production cycle time. Therefore, the design stage of a workcell is very important. Similar to other manufacturing system, layout design stage is the fundamental requirement when it comes to determining the performance of a manufacturing system. Since the path taken by the robot arm to complete a task will affect the cycle time of the overall system the initial layout planning is therefore crucial [2].

The layout design of a manufacturing system will have a huge impact on the manufacturing performance [2]. This is because most of the manufacturing performance parameters are highly depending on the layout design. A good layout design will reduce the operation time and increase productivity. Many layout design problem have already been proposed by other researches [3–5]. In order to achieve an optimum layout design, optimization techniques are required to assist in the design process. The main criterion of a layout design is to minimize the layout area. General layout criteria for manufacturing system may include the adjacency between components (work cells) to reduce material traveling distance.

The organization of the paper is as follows. Section 2 briefly reviews the nature inspired algorithms adopted in this paper to solve robotic workcell layout problems. Section 3 details the problem environment and the assumptions used. Section 4 explains the implementation details of these algorithms and details of parameters fine-tuned. Section 5 discusses on the evaluation of two

* Corresponding author.
E-mail addresses: zhenyang_28@hotmail.com (Z.Y. Lim), sgponnambalam@monash.edu, sgponnambalam@gmail.com (P. S.G.), izui@me.kyoto-u.ac.jp (K. Izui).

http://dx.doi.org/10.1016/j.asoc.2016.08.048
1568-4946/

layout representation schemes and the convergence details of these schemes. Section 6 presents the results and discussion. The conclusions are presented in Section 7.

2. Literature survey

The literature on the workcell layout problems, layout representation schemes and brief review on nature inspired algorithms used in this paper are presented in this section. The consolidation of the literature review is presented in Table 1.

2.1. Robot workcell layout problems

A good placement and design of a manufacturing system will often affect the overall efficiency of the manufacturing system. Therefore many researchers addressed the layout problem in the past. Layout problem does not only applied to manufacturing system as it is also a popular research problem for printed circuit board (PCB) design. A simple layout problem for PCB design is presented in [6]. Simple layout representation scheme is applied in PCB design as well. The overall idea of designing the layout is similar in both cases. However, the main focus of this paper is based on manufacturing system layout.

One of the earliest papers on facility layout problem is proposed by Koopmans & Beckmann [7]. They emphasized the importance of minimizing the layout area in order to lower the cost and distance for transporting material in a facility. Similar facility layout problem can be found in [8,9]. Drira et al. [8] considered static and dynamic layout problems. In a static layout, the layout arrangement is constant once the design is done. However, in dynamic layout problem, there will be a series of layouts with different arrangement according to the types of tasks.

Lueth [10] proposed an approach to plan robot workcells in the Cartesian configuration space, where the problem is formulated as the quadratic set covering problem. Barral et al. [11] adopted the placement heuristic from [12] and implemented simulated annealing as the optimization algorithm. Drezner & Nof [13] studied the problem of planning a robot assembly station in which component parts are picked from bins and assembled. However, they did not optimize every component in the layout.

Cheng [14] conducted robotic workcell simulation via Deneb's IGRIP robotic simulation technology. He showed that developing faithful models for a robotic workcell simulation study is a complex process that requires the multifaceted knowledge in diverse disciplines.

Yap et al. [15] proposed a virtual reality based support system for layout planning in robotic workcells. They used virtual reality (VR) technology to improve human-robot interface, whereby complicated commands or programming knowledge is not required. They proposed a solution, known as VR-based Programming of a Robotic Work Cell (VR-Rocell), consists of two sub-programmes, which are VR-Robotic Work Cell Layout (VR-RoWL) and VR-based Robot Teaching System (VR-RoT). VR-RoWL is developed to assign the layout design for an industrial robotic work cell, whereby VR-RoT is developed to overcome safety issues and lack of trained personnel in robot programming.

Pai et al. [16] presented an augmented reality–based robotic work cell consisting of a virtual robot arm, conveyor belt, pallet and computer numerical control machine that simulates an actual manufacturing plant environment.

Zhang et al. [17] proposed a three step method and a system for automatic optimization of the placement of a plurality of workstations in a robot workcell in order to improve robot performance and increase productivity of a robotic work cell. In the first step, each robot task is handled separately. A preferred region that can ensure best robot performance in terms of kinematics and kinetics is employed to determine the best position of each individual robot task. In the second step, all robot tasks are put into consideration together to seek a best order of the workstations to be placed according to the robot operational sequence. In the third step, an optimization method that can ensure a better and better solution, such as but not limited to Simulated Annealing (SA), is used to finally fine-tune the placement of all workstations. The optimization in the third step is done with regard to robot performance, such as cycle time, stress, or energy. In the last two steps, all robot tasks are treated equally and handled simultaneously for a global optimization.

Tao et al. [18] presented a layout optimization approach based on differential evolution (DE) to solve facility layout problem. They developed a digital simulation platform for the automated layout optimization to realize three-dimensional visualization demonstration of the optimal layout solution.

Jian & Ai-Ping [19] addressed the machine layout problem in a flexible manufacturing cell served by a robot and presented a GA to minimize robot travel distance which in turn minimize the cycle time.

Barral et al. [11] adopted the placement heuristics of Tay & Ngoi [12] as the underlying placement methodology within a broader implementation of Simulated Annealing algorithm (SAA). Instead of picking out only one option for the placement of the next station in the layout, different options are explored and the final option picked would be based on SA parameters.

Islier [20] proposed the utilization of GA as a means to optimize multi-criteria FLP specifically for an office area with multiple departments. He studied a layout for the placement of different departments on a single rectangular floor space which would facilitate the best possible flow of parts between one department and another in terms of cost, distance between these departments and the actual physical load of the flow.

Sim et al. [21] present a GA approach to optimizing robot workcell layout by using the distance covered by the robot arm as a means of gauging the degree of optimization. The approach is constructive: the different stations within the workcell are placed one by one in the development of the layout. The placement method adopted is based on the spiral placement method first proposed by Islier [20].

2.2. Layout representation schemes

The main challenge in designing a layout is to avoid overlapping of departments. Many methods have been proposed in order to overcome the overlapping problem. For example, in [22,23], multi-step optimization techniques are suggested. In these methods, the initial layout is required to undergo a few steps of optimization. By adding in some constraints in each step, the overlapping can be reduced after each step and finally a non-overlapping layout can be obtained. The major flaw of this method is that the computation time is long as many steps and constraints checking are involved. There are different layout representation schemes available in the literature, which are useful to avoid overlapping.

Tay & Ngoi [12] described a heuristic algorithm that uses a spatial representation technique to solve a robot workcell layout problem.

Tree structure representation scheme such as B*trees and O-tree are among some of the popular schemes. The effectiveness and flexibility of B*trees is presented in [24]. The use of O-tree representation scheme is given in [25]. By comparing these two methods, B*tree is said to be performing better than O-tree as O-tree is less flexible and harder for implementation [24].

Another layout representation method that is also being widely used is the sequence pair representation scheme that is proposed by

Table 1
Literature on robot workcell layout problems.

Algorithms used	Objective function	Layout representation method	Problem type	Reference
Simulated Annealing, Genetic Algorithm	Review paper	Min-cut placement	VLSI Cell placement	[6]
Assignment algorithm	Minimize cost of inter-plant transportation	n/a	Plant allocation to locations	[7]
Exact and approximation algorithms	Review paper	Discrete and continuous layout schemes	A tree representation of layout problems is presented	[8]
Quadratic programming, exhaustive slicing procedure	Slicing optimization, area utilization	n/a	VLSI palcement	[9]
Optimal and heuristic algorithms	Minimize cost of the layout	Cartesian coordinate space	Robot workcell	[10]
Simulated annealing	Minimize the cycle time	constructive approach	Assembly workcell layout	[11]
Heuristic algorithm	Minimization of total path of travel of a robot arm	Spatial representation	Robot workcell layout	[12]
Heuristic algorithms	Minimize expected times and maximal time	n/a	Robotic assembly layout	[13]
Simulation using IGRIP simulator	n/a	n/a	Robotic workcell layout	[14]
Virtual reality approach	Improve human-robot interface, reduce robot downtime	n/a	Robotic workcell layout	[15]
Augmented reality-based programming and simulation approach	Collision free operation	n/a	Robotic workcell layout	[16]
Genetic algorithm, simulated annealing	Robot performance: cycle time, stress and energy	n/a	Robotic workcell layout	[17]
Differential Evolution algorithm	Minimize total material handling cost and maximize area utilization rate	Three dimensional geometric characteristics, pose and position of facilities are used.	Facility layout problem	[18]
Genetic algorithm	Minimize cycle time, Optimizing the locations and orientations of machines	n/a	Machine layout problem in a flexible manufacturing cell served by a robot	[19]
Genetic algorithm	Minimizing the transportation load, maximizing the compactness of the departmental areas and minimizing the difference between requested and available areas.	Different placing procedures	Facility layout problem	[20]
Genetic algorithm	Minimization of cycle time	Constructive approach	Robot workcell layout	[21]
Genetic algorithm	Minimization of the material handling and penalty costs	n/a	Machine cell formation	[28]
Genetic algorithm	Minimize the cost of shifting departments and material flow	permutation of N department numbers	Dynamic facility layout problem	[29]

Murata et al. [26]. The concept of sequence pair is to represent the modules on an oblique grid and construct some constraints graphs based on the grid produced. Based on the constraints graphs, the coordinates of each component in the layout can be found.

2.3. Nature inspired algorithms

Many researchers use nature inspired algorithms to handle complex optimization problems. In addition to the algorithms discussed in this paper, there are many algorithms being proposed by researchers. For example, Caraveo et al. [27] proposed a new bio-inspired optimization algorithm based on the self-defence mechanism of plants. A survey on nature-inspired optimization algorithms with fuzzy logic for dynamic parameter adaptation could be found in [28]. Valdez [29] described the optimization of a set of mathematical functions using bio-inspired algorithms.

The nature inspired algorithms dealt in this paper are briefly reviewed in this section.

2.3.1. Genetic algorithm

The layout problem does not stop after selecting the layout representation scheme; optimization is needed to find the best layout solution. There are many popular Evolutionary Algorithms (EAs) available. One of the commonly used EAs by many researches is Genetic Algorithm (GA). Many optimization problems have been solved using GA and one of the classical examples is travelling salesman problem [30]. Similar concept of using GA can also be applied to layout optimization. Rajagopalan & Fonseca [31] proposed a GA problem to optimize a cell design which aims to minimize the material handling cost. Pourvaziri and Naderi [32] proposed a hybrid multi-population genetic algorithm to solve the dynamic facility layout problem.

Solving optimization problem based on one objective is often seen as insufficient and ineffective. Therefore, now-a-days, researchers are moving from single objective to multi-objective optimization. A multi-objective optimization method based on GA is developed by Deb et al. [33] and is named as Non-dominated Sorting Genetic Algorithm (NSGA). However, even though NSGA is an

effective algorithm it is heavily criticized due to the fact that it lack of elitism and is complex to compute [34]. Therefore, an improved version of NSGA, called NSGA-II is proposed by Deb et al. [33]. The new NSGA-II is a better version of NSGA as it includes elitism, the complexity is reduced as well and it does not requires the need for specifying the sharing parameter [34]. NSGA-II quickly became a popular choice when it comes to optimizing multi-objective problems. Many researches implemented it to optimize manufacturing facility layout problem [35,36]. NSGA-II is used in the algorithms proposed in this paper to handle multiple objectives.

2.3.2. Differential evolution algorithm

Differential Evolution (DE) is proposed by Storn & Price [37]. DE is popular due to its simplicity, effectiveness and robustness when it comes to solving optimization problems [38]. DE consists of simulation annealing mechanism and the main concept of differential evolution is the added difference between the individuals [39]. DE also includes mutation and crossover process and the parameters involved often changes depending on the optimization problems. Therefore the choices of parameters are important in DE as it will affect the overall performance and convergence of the solution [40].

Similar to GA there are also multi-objective optimization algorithms created based on DE. Multi-objective Differential Evolution (MODE) is introduced by Xue et al. [41]. Solving multi-objective problem is always harder than single objective problem due to a more complicated selection process. Thus, in MODE they first need to identify the Pareto optimum solution. It can be done by ranking the individual according to the objective functions and this process is similar to the method used in NSGA-II.

Another multi-objective DE approach is proposed by Robič & Filipič [42]. They introduced Differential Evolution for Multi-Objective (DEMO) and are able to achieve the goal of producing a good convergence to Pareto front and a uniform spread of individuals along the Pareto. They also tested the proposed method with three different variants but the results are not significant as all variants produced similar results.

2.3.3. Artificial colony algorithm

Artificial Bee Colony (ABC) is a relatively new algorithm inspired by the intelligent behaviour of honey bees and is originally used to solve numerical problems. Since ABC is a new algorithm, it has not been applied to any layout optimization for manufacturing system. The performance and efficiency of ABC is compared with others algorithms such as GA, DE and PSO [43,44]. From [43], the results show that ABC algorithm is able to solve multi-dimensional and multi variable problems effectively. Rubio-Largo et al. [45] proposed hybrid Multiobjective artificial bee colony algorithm for a multiple sequence alignment problem.

An enhanced version of ABC algorithms is proposed by Szeto et al. [46] to solve a capacitated vehicle routing problem. They concluded that the enhanced version of ABC is performing better than the original ABC.

2.3.4. Charged system search algorithm

Charged System Search (CSS) algorithm, developed by Kaveh & Talatahari [47] is one of the latest population based metaheuristic optimization techniques inspired by the governing laws of electrostatics in physics and the governing laws of motion from the Newtonian mechanics. Kaveh & Talatahari [47] used CSS algorithm for design of skeletal structures. Özyön et al. [48] applied CSS algorithm to solve a multi-objective environmental economic power dispatch problems.

2.3.5. Particle swarm optimization algorithm

Reeves[49] one of the early pioneers, as part of his work at Lucas film, implemented a particle system that used many individuals working together to form what appeared to be a fuzzy object (such as a cloud or an explosion). A particle system stochastically generates a series of moving points, these typically being initialised to predefined locations. Each particle is assigned an initial velocity vector. Reynolds [50] used Reeves' particle system as the basis for his higher order (in terms of the objects being modelled) flocking algorithm. He took the particle movement and added orientation and inter-object communication. Kennedy & Eberhart [51] extended the model by Reynolds [50] to reflect social behaviours. Kennedy & Eberhart [52] developed the first discrete version of PSO for binary problems. They solved the problem of moving the particle through the problem space by changing the velocity in each vector to the probability of each bit being in one state or the other.

PSO offers rapid optimization of complex multidimensional search spaces, and is a popular contemporary algorithm for a wide range of search and optimization problems [53,54]. Banks et al. [53,54] reviewed the theoretical development of PSO and its applications extensively. Chen et al. [55] proposed a multi-objective endocrine particle swarm optimization algorithm to solve multi-objective optimization problems.

Based on the literature survey, it is found that mostly GA and SAA in addition to few heuristic algorithms are proposed to solve the robotic workcell layout problem. The criteria to optimize the layouts are mostly focusing on minimization of assembly cycle time and the end effector travel time. So far, the power of DE, ABC, CSS and PSO algorithms are not explored. Izui et al. [31] used a Multi-objective GA to optimize the layout area, processing time and manipulability concurrently and shown the performance of the proposed GA with a sample problem.

The use of nature inspired algorithms are not extensively explored to optimize robot workcell layouts. The contribution in this paper is the use of five nature-inspired algorithms, viz. genetic algorithm (GA), differential evolution (DE), artificial bee colony (ABC), charge search system (CSS) and particle swarm optimization (PSO) algorithms and to optimize the three design criteria simultaneously. The performance of different layout representation schemes are also evaluated. There are no benchmark problems available in the literature for robot workcell layouts. Seven problems are generated and used to illustrate the effectiveness and usefulness of the proposed methods.

3. Multi-objective robot workcell layout optimization problem

The problem considered in this paper is a robotic workcell that perform assembly operations. There are four main components in this workcell. The components are assembly robot, assembly table, part boxes and dummy blocks. The assembly robot used to illustrate the problem is Mitsubishi RV-6SQ [35]. The details of robot dimensions are shown in Fig. 1. Assembly table is the location where the assembly robot performs assembly operations. The use of dummy blocks is to represent empty spaces between the part boxes and assembly robot. The dummy blocks are useful to provide spacing between robot and other components [35].

Ten problems are generated randomly to test the performance of the algorithms. Each problem will have one assembly robot and one assembly table. However, each problem will have different number of part boxes and dummy blocks. The dimension of the dummy blocks varies for each problem. The total number of operations required to complete the assembly operation also varies for each problem. The test problems used and the setup specification details are provided in Appendix.

Three objective functions are taken into consideration. They are: minimization of layout area; minimization of operation time and; maximization of manipulability index. All seven problems are

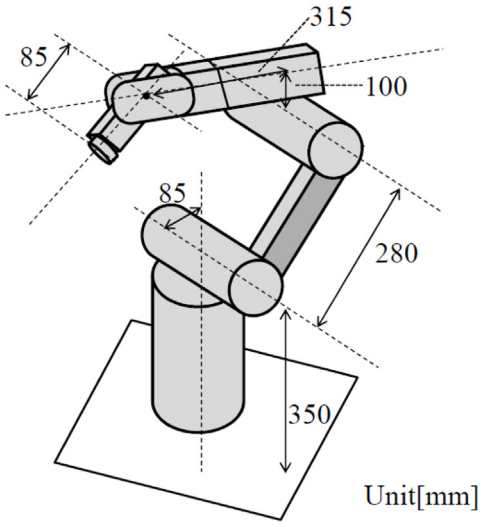


Fig. 1. Dimemsions of Mitsubishi RV-6SQ.

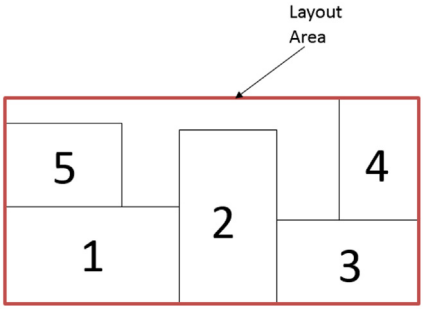


Fig. 2. Layout area representation.

optimized using five different algorithms (including GA) with the integration of NSGA-II to handle multiple objectives. The objective functions used are explained in Section 3.1, the fitness value formulation is explained in Section 3.2 and the details of layout representation schemes tested are explained in Section 3.3.

3.1. Objective functions

This section describe the three objective functions namely layout area, operation time and manipulability index.

3.1.1. Objective 1: layout area

The layout area is the area of the smallest rectangle which can contain all allocated components as shown in Fig. 2.

3.1.2. Objective 2: operation time

The second objective for layout design is operation time. After forming the layout plan, it is now possible to compute the operation time. However, it is not possible to calculate it precisely due to the fact that detailed motion trajectory is normally not given during layout design stage. Therefore, it is only possible to evaluate the robot motion time based on the rotational angle, θ of each robot arm. The joint angle can found by using inverse kinematics since the final position is known. The final position is assumed to be the midpoint of each module.

By using the maximum angular velocity, ω of the robot arm at each particular joint, the motion time of the k^{th} joint performing the i^{th} operation is calculated as shown in Eq. (1).

$$T_k^i = \frac{|\bar{\theta}_k^i|}{\omega_k} \tag{1}$$

Where $\bar{\theta}_k^i$ is the angle through which the k^{th} joint moves during the i^{th} motion.

Another assumption is that all joint can simultaneously have their angle changed during each operation and thus the motion time for each operation is the maximum joint motion time. The equation for motion time is given in Eq. (2).

$$T_i = \max_k \left(\frac{|\theta_k^i|}{\omega_{\boldsymbol{k}}} \right) \tag{2}$$

Therefore, the total operation time for one assembly cycle T can be calculated by adding the motion time for each operation in an assembly task as shown in Eq. (3). N represents the number of operation in each assembly task.

$$T = \sum_{i=1}^{N} T_i \tag{3}$$

3.1.3. Objective 3: manipulability

For a robot to perform skilled assembly task, it has to be able to avoid singularity. Singularity is defined as the configuration where the value of joint position causes the manipulator loses degree of freedom. Singularity can be calculated by finding the determinant of Jacobian matrix. If the determinant is zero then the robot is in singularity.

Let W represent the manipulability of a non-redundant robot. The manipulability for each i^{th} operation is as given in Eq. (4).

$$W_i = |\det\left(J\left(\theta\right)\right)| \tag{4}$$

In Eq. (4) above, when W is close to zero there is a high chances that the robot will encounter singularity point. Therefore, the value of W should be as large as possible. The total manipulability for one assembly task can be calculated by the weighted summation of manipulability value (W) at the various task positions which is given in Eq. (5).

$$\boldsymbol{W} = \sum_{\boldsymbol{i}=1}^{\boldsymbol{N}} \boldsymbol{W_i} \tag{5}$$

N represents the number of operation in one assembly cycle and i represent a weightage coefficient. In this paper, i is assumed as 1.

3.2. Combined fitness value

Fitness value is summation of all objective values. Eqs. (6)–(8) are used to calculate the fitness value.

$$Fitness_{area} = \frac{1}{(1 + area)} \tag{6}$$

$$Fitness_{operation\,time} = \frac{1}{10 * (1 + operation\,time)} \tag{7}$$

$$Fitness_{manipulability} = \frac{1}{(1 - manipulability)} \tag{8}$$

$$Fitnessvalue = Fitness_{area} + Fitness_{operationtime} + Fitness_{manipulability} \tag{9}$$

Eq. (9) is used to find the combined fitness values, which are presented in Tables 5–7 for comparative evaluation of the algorithms.

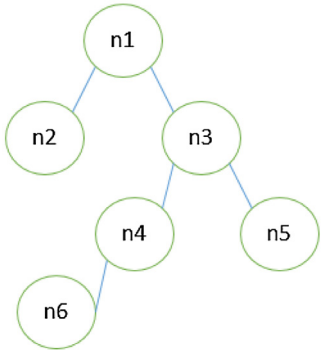


Fig. 3. B*tree diagram.

3.3. Layout representation scheme

3.3.1. Scheme 1: B*tree layout representation

The binary tree in B* Tree algorithm proposed by Chang et al. [24] is generated randomly. Fig. 3 shows an example of B*tree constructed using six nodes. The layout placement position of all modules can be obtained starting from the root. The root of the B*tree correspond to the module on the bottom left corner with a coordinate of (0, 0) in the layout. While the other modules are placed based on the following rules.

The rules for forming the layout are as below:

3.3.1.1. Rule 1. If node n_j is the left child of node n_i, then module j will be placed on the right hand side and adjacent to module i. Thus the x-coordinate of module j can be computed using Eq. (10).

$$x_j = x_i + w_i \tag{10}$$

where x is x-coordinate and w is width of the module.

3.3.1.2. Rule 2

If node n_j is the right child of node n_i, then module j will be placed above module i. Thus the x-coordinate of module j can be computed using Eq. (11).

$$x_j = x_i \tag{11}$$

3.3.1.3. Rule 3

For computing y-coordinates, first find the permutation P that determines the vertical position of the module when two modules have proper overlap in their x-coordinate projections.

For each node n_i, let $\pi(i)$ be the set of module n_j with its order lower than n_i in permutation P and interval $(x_j, x_j + w_j)$ overlaps interval $(x_i, x_i + w_i)$ by a non-zero length.

If $\pi(i)$ is non-empty then the y-coordinates of node n_i can be calculated as follow:

$y_i = \max(x_j + h_j), j \in \pi(i)$ otherwise $y_i = 0$

where, h is the height of the module.

3.3.2. Scheme 2: sequence pair layout representation

Murata et al. [26] introduced the sequence pair representation scheme. Consider an example where there are six modules or stations in a manufacturing system and let the sequence pairs P1 = [431625] and P2 = [635412]. The oblique grid of the sequence pair can be formed as shown in Fig. 4. After forming the grid, the intersection point between the same block numbers is considered as nodes. For example, the dimensions for the six blocks are shown in Table 2.

From the oblique grid, a horizontal and vertical constraints graph can be constructed as shown in Figs. 5 and 6.

Table 2
Dimensions of blocks.

Block No.	Width	Height
1	4	6
2	3	7
3	3	3
4	2	3
5	4	3
6	6	4

Table 3
Pareto Solutions generated using sequence pair.

Pareto Solutions	1	2	3	4	5	6	7	8	9
Operation time (s)	3.07	***1.74***	2.13	2.47	1.78	2.08	2.53	2.85	1.86
Manipulability	***0.88***	0.78	0.84	0.85	0.78	0.82	0.86	0.86	0.79
Area	*0.82*	0.84	0.86	0.76	0.84	0.71	0.85	0.81	0.80

Bold values indicate the best function values obtained.

Table 4
Pareto Solutions generated using B*tree.

Pareto Solution	1	2	3	4	5	6	7
Operation time (s)	3.57	***2.22***	2.51	2.44	2.93	3.25	3.19
Manipulability	***0.75***	0.62	0.73	0.66	0.75	0.77	0.76
Area	*0.85*	0.70	0.90	0.76	0.64	0.86	0.60

Bold values indicate the best function values obtained.

Table 5
Comparison 1–Area vs Operation Time.

	GA	DE	MABC	PSO	CSS
Problem 1	1.0734	1.4329	1.3938	***1.4628***	1.4266
Problem 2	0.7928	0.8395	0.9126	***0.9337***	0.8021
Problem 3	0.5831	0.6783	0.6454	***0.7048***	0.6472
Problem 4	0.7712	0.7915	0.7188	***0.8193***	0.7346
Problem 5	0.5219	0.5359	0.4840	***0.5435***	0.4909
Problem 6	0.5084	0.5222	0.5167	***0.5250***	0.4982
Problem 7	0.4587	0.4618	0.4642	***0.4695***	0.4618
Problem 8	0.4199	0.4442	0.4434	***0.4490***	0.3971
Problem 9	0.4485	0.5470	0.5394	***0.5761***	0.4689
Problem 10	0.4919	0.6668	0.6187	***0.6936***	0.6373

Bold values indicate the best function values obtained.

Table 6
Comparison 2–Area vs Manipulability.

	GA	DE	MABC	PSO	CSS
Problem 1	0.6307	0.7217	0.7148	***0.7429***	0.5735
Problem 2	0.7059	0.7211	0.5896	***0.7321***	0.6680
Problem 3	0.7120	0.7062	0.6314	***0.7694***	0.6745
Problem 4	0.9355	0.8697	0.9721	***1.0588***	0.8540
Problem 5	0.7754	0.8167	0.7722	***0.8710***	0.7436
Problem 6	0.9552	0.9461	0.9494	***0.9667***	0.8254
Problem 7	0.9776	0.9679	1.0031	***1.0042***	0.9173
Problem 8	0.6719	0.7175	0.7344	***0.7408***	0.6862
Problem 9	0.8465	0.9914	0.9607	***1.0665***	0.8932
Problem 10	0.8631	1.1325	1.1466	***1.1710***	0.9753

Bold values indicate the best function values obtained.

Table 7
Comparison 3–Manipulability vs Operation Time.

	GA	DE	MABC	PSO	CSS
Problem 1	1.2581	1.3010	1.2711	***1.3099***	1.2621
Problem 2	0.9469	0.9577	0.9535	***0.9719***	0.9003
Problem 3	0.8794	0.8795	0.8828	***0.8938***	0.8663
Problem 4	1.2764	1.2750	1.2751	***1.3655***	1.2362
Problem 5	0.9393	0.9494	0.9163	***0.9509***	0.9006
Problem 6	0.9630	0.9722	0.9607	***0.9793***	0.9592
Problem 7	0.9835	0.9887	1.0036	***1.0383***	0.9154
Problem 8	0.9977	1.0458	1.0986	***1.1368***	1.0086
Problem 9	1.0010	1.1243	1.1274	***1.1493***	1.0709
Problem 10	1.1336	1.1918	1.1755	***1.2201***	1.1378

Bold values indicate the best function values obtained.

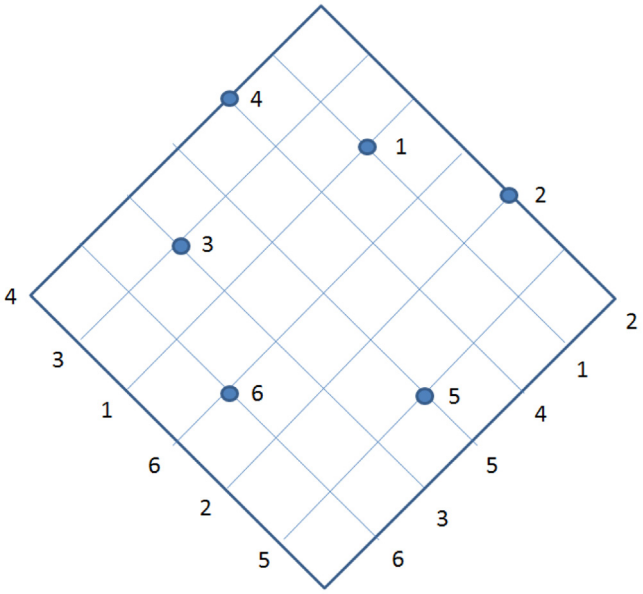


Fig. 4. Oblique grid for sequence pair.

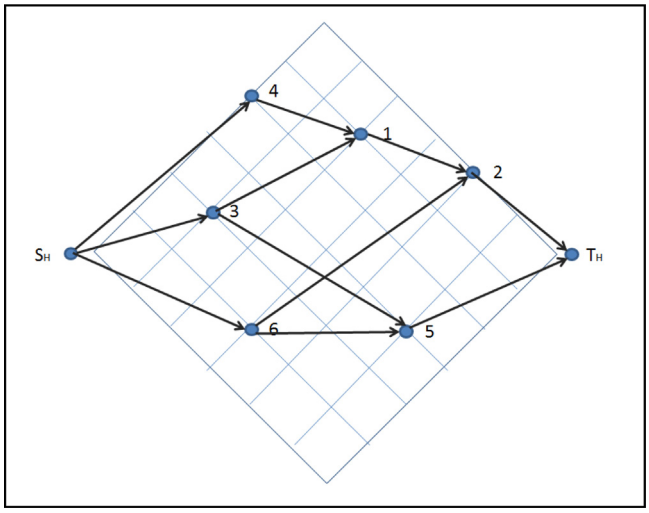


Fig. 5. Horizontal constraint graph.

The construction of these two graphs is based on simple rules as stated below:

IfP1(< ..bi..bj.. >)andP2(< ..bi..bj.. >)thenbiistotheleftbj

IfP1(< ..bj..bi.. >)andP2(< ..bi..bj.. >)thenbiisbelowbj

3.3.3. Longest path algorithms

Longest path algorithm [56] is used to determine the x and y coordinates of each block. A weighted value is assigned to each node. For source node, it will have a weight zero. While the other nodes will have a weighted value based on their width (for horizontal constraint graph) or height (for vertical constraint graph).

The coordinates of each block are the coordinates of the lower left corner the block. For example, from Fig. 5, the x-coordinate of block 4, 3 and 6 are zero since the source node weight is zero. Next the x-coordinate for block 1 is based on the longest distance between the paths of block 3 and block 4 and the x-coordinate for block 5 is based on the longest distance between the paths of block 3 and block 6. The steps are repeated in a similar fashion until all x-coordinates of the blocks are found.

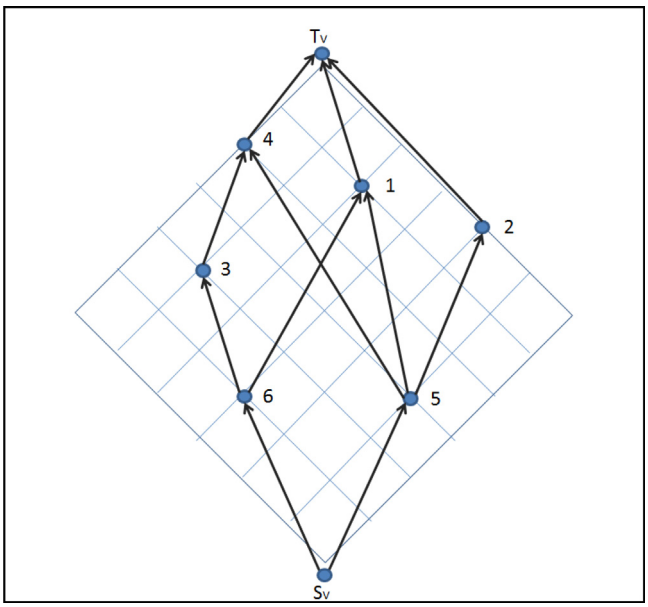


Fig. 6. Vertical constraint graph.

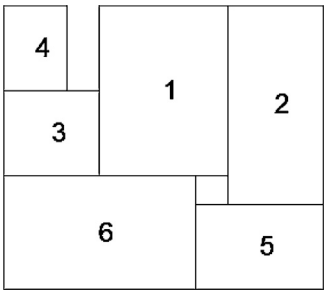


Fig. 7. Final layout.

For y-coordinates, the method is similar but now it is based on the vertical constraint graph and instead of checking the longest path based on the length of the width it is now based on the height. The final layout for the example is shown in Fig. 7.

Finally from the layout planning, the layout area can be computed using Eq. (12).

$$\text{Layout area, } a = \text{Longest path from} S_H \text{ to } T_H \times \text{Longest path from } S_V \text{ to } T_V \quad (12)$$

4. Natural inspired algorithms to optimize robot workcell layouts

Algorithm 1a. Genetic Algorithm (GA) for B*tree.

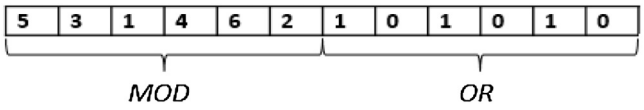


Fig. 8. MOD and OR vector representation for B*tree layout.

*GA pseudocode for B*Tree representation:*

Step 1: Initialize population randomly
Step 2: Check feasibility of each individual
WHILE** number of generations < MAX_GENERATION **DO
Step 3: Evaluate the population /equation (9) to get combined fitness/
Step 4: Perform Binary Tournament Selection and check feasibility
Step 5: Perform First order crossover and check feasibility
Step 6: Perform Inversion Mutation and check feasibility
END

GA is simple algorithm that starts off by evaluating the initial population and selects the evaluated population for crossover and mutation. The operation of crossover and mutation are explained further below. If the termination condition is not met, the algorithm continues its evaluation and the whole process is repeated until the condition is met thus producing the final population.

In B*tree scheme, the vector is represented by the modules and orientation. An example is shown in Fig. 8.

4.1. Crossover

The crossover operator implemented here is known as the first basic variant of order crossover (O_{X1}). In order to perform this crossover, two parents are first selected. Each parent is represented as in the vector form as above. Each parent will then breakdown into two smaller vector (MOD vector and OR vector). MOD vector and OR vector will undergo the crossover procedure before combining again to form the offspring vector.

For example, consider two MOD vector in parents M1 and M2. Let M1 = (2–6,1) and M2 = (6,2,3,5,1,4). First select two cut points as shown below (cut point are marked by "|")

$$M1 = (23|45|61) \text{and} M2 = (62|35|14)$$

The modules between the cut points are copied directly into the offspring vector.

$$\text{Off\_M1} = (--|45|--) \text{and} \text{Off\_M2} = (--|35|--)$$

Then, starting from the second cut point of one parent, the modules from the other parent are copied in the same order, excluding those modules which already exist.

The sequence of modules starting from the second cut point in the second parent is 1 4 6 2 3 5.

From the sequence, by removing 4 and 5 which already exist in the first offspring, the sequence is now left with 1 6 2 3.

Now placing the sequence back into Off_M1 at the second cut point, we have

Off_M1 = (2 3 4 5 1 6) and similarly for Off_M2, we have Off_M2 = (2 4 3 5 6 1).

The similar process is repeated for OR vectors to get Off_OR1 and Off_OR2. Finally both Off_M1 and Off_OR1 vector will combine again to form the offspring vector.

4.2. Mutation

Inversion mutation (IM) operator is used in the paper. The process is done by selecting a sub-sequence and inverting the modules inside the sub-sequence.

For example, the offspring vector is (5 3 1 4 6 2 1 0 1 0 1 0). Once again the vector is separated into two parts namely MOD and OR.

Thus MOD = (5 3 1 4 6 2) and OR = (1 0 1 0 1 0). Then two cut points are selected. The cut point is represented as "|".

$$\text{MOD} = (5|314|62) \text{and} \text{OR} = (1|01|010)$$

By inverting the modules inside the two cut point, we get MOD = (5 4 1 3 6 2) and the same process is applied for OR to get OR = (1 1 0 0 1 0). The final vector is form by combining MOD and OR again. Therefore the final vector will be (5 4 1 3 6 2 1 1 0 0 1 0).

4.3. B*tree representation scheme

In order to implement B*tree layout representation scheme, any two vectors are introduced. They are predecessor vector and position vector. Predecessor vector will record the parent node of each elements contain in the module vector. Position vector will record whether the node is on the left (represented by 0) or right (represented by 1).

For example, the modules vector given is as shown in Fig. 9. The first element is always taken as the root node which in this case is 5. Thus in the predecessor vector the first element will be 0 and for position vector it will be represented as 2.

Moving on to the second step, it is always assumed that the root node will have two child nodes. Thus, the root node is inserted, which is 5 in the second and third element of predecessor vector.

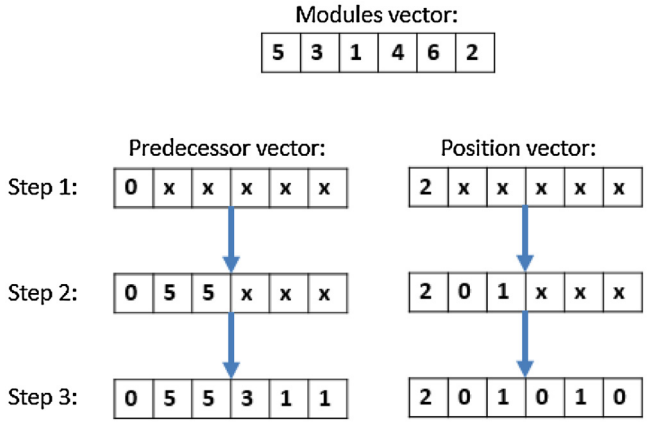


Fig. 9. B*tree representaion.

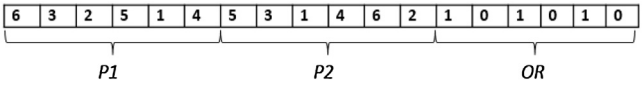


Fig. 10. Chromosome representation in GA.

Since one of the nodes must be left and the other must be right, in position vector, 0 (left) and 1 (right) is randomly inserted in the second and third element. For this example, 0 is inserted in the second element and 1 is in the third element. This means that module 3 will be the left child node of 5 and module 1 will be the right child node for 5.

In the third step, first need to determine how many child nodes module 3 and module 1 should have. The number of child node can be range from 0 to 2. For this example, module 3 will have one child node while module 1 will have two child nodes. Therefore module 4 will be child node of module 3 while module 6 and 2 will be the child node for module 1. Once again the position of module 4 is randomly generated and for this example it is 0 which is the left child node of module 3. Similarly, module 6 is the right child node and module 2 is the left child node of module 1.

After finding the predecessor and position vector, following the B*tree layout representation scheme rules as explained earlier in Section 3.3, the layout design can be obtained.

Algorithm 1b. Genetic Algorithm (GA) for Sequence Pair

The procedure flow of GA for sequence pair is similar to the one used for B*tree. The only differences are the crossover and mutation operators. The changes in these operators are to suit to the chromosomes (vectors) generated using sequence pair representation. The crossover and mutation used for sequence pair are explained in the following sections. The pseudocode for GA with sequence pair representation is given below.

GA pseudocode for sequence pair representation:

Step 1: Initialize population randomly
Step 2: Check feasibility of each individual
WHILE** number of generations < MAX_GENERATION **DO
Step 3: Evaluate the population /equation (9) to get combined fitness/
Step 4: Perform Binary Tournament Selection and check feasibility
Step 5: Perform PPEX crossover and check feasibility
Step 6: Perform swapping Mutation and check feasibility
END

4.4. Crossover

The crossover operator used is PPEX crossover which is adopted from [57]. PPEX selects modules within a pre-defined window in an oblique grid (same as the one use in sequence pair) and swap the modules with another one. A random value is selected to define the size of the window. In GA, individuals are represented as chromosome. Fig. 10 shows an example chromosome. The vector is formed by combining the individual vector of *P1, P2* and *OR*. *P1* indicates the first pair and *P2* is the second pair. For orientation vector (*OR*), 0 is used to indicate the original dimension while 1 means swapping the original dimension (width to length and length to width).

PPEX offspring creation steps (illustration shown in Fig. 11):

1 Select two individuals as parent 1 and parent 2.
2 Form an oblique grid using *P1* and *P2* for each parent.
3 Generate a window that can fit into the oblique grid for each parent. The size of window is generated randomly.
4 Let the M_c be the set of modules inside the window.
5 Each module of M_c is exchanged according to the sequence of its partner parent and is copied to the child.
6 The other modules outside the window are copied directly to the child.

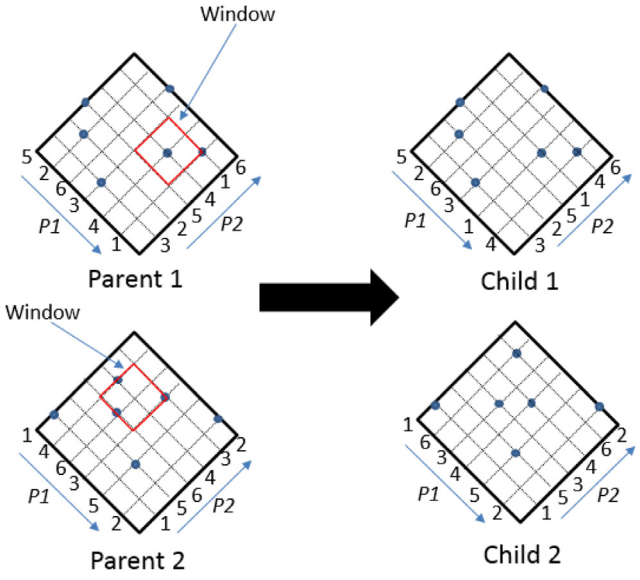


Fig. 11. PPEX Crossover.

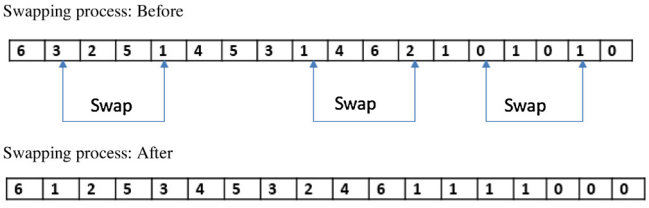


Fig. 12. Swapping Mutation process.

4.5. Mutation

For mutation process, randomly select and exchange the position of two elements for P1, P2 and OR. Fig. 12 shows the vectors before and after mutation.

4.6. Parameters fine tuning

The parameters fine-tuned for GA are crossover rate and mutation rate. In order to select the best parameter value, a fine tuning process is conducted. The crossover rates tested are 0.1, 0.3, 0.5, 0.7 and 0.9. The Pareto front plot for each crossover rate is as shown in Fig. 13. The effect of crossover rate could be observed in Fig. 13. Based on Fig. 13, crossover rate with a value of 0.9 produces the best solution. The pareto front is plotted considering the operation time and manipulability.

Similary the mutation rate is also tested using 0.1, 0.01, 0.001, and 0.0001. Once again, the Pareto front for each mutation rate is plotted. Fig. 14 shows that the pareto front with mutation rate of 0.001 could obtain the best solution.

Algorithm 2. Differential Evolution (DE)

There are many variants of DE reported in the literature [58]. The variant used in this paper is DE/rand-to-best/1. The DE algorithm used in this paper is adopted from [59]. An example vector for DE is shown in Fig. 15.

4.7. Mutation

The variant used in this paper is DE/rand to best/1. Eq. (13) is used to do mutation for this variant.

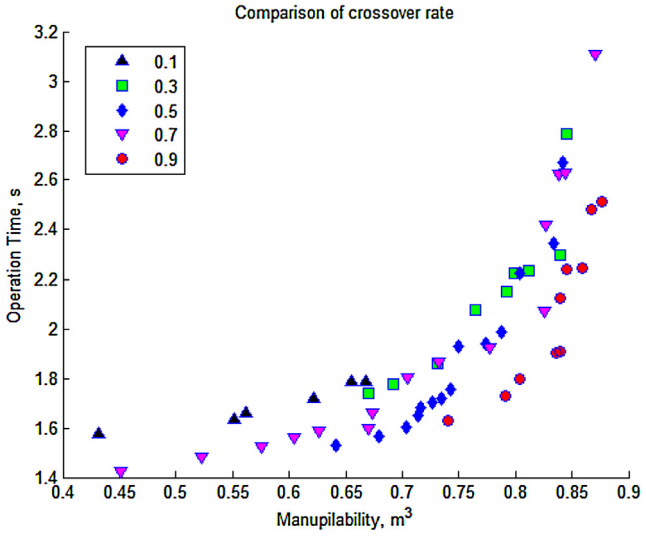


Fig. 13. Tuning of Crossover rate.

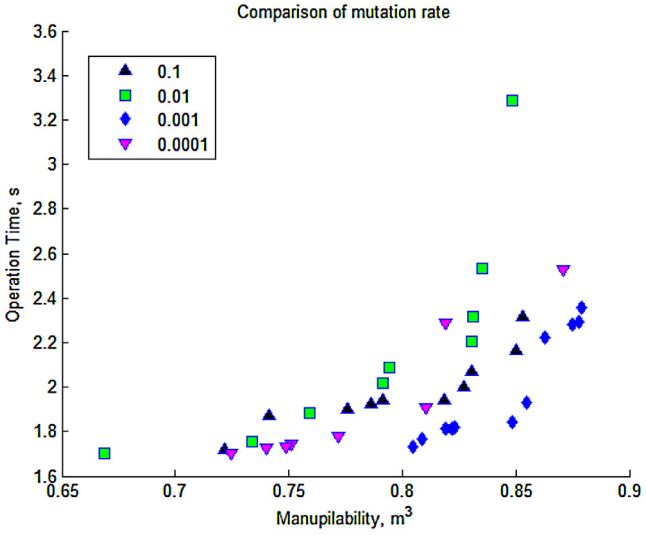


Fig. 14. Tuning of Mutation rate.

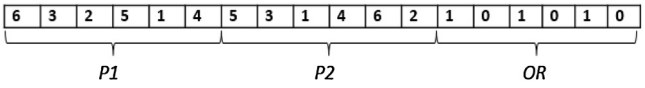


Fig. 15. Vector representation in DE using sequence-pair.

Mutation for DE/rand to best/1:

$$V_i = x_{r1} + F_1\,(x_{r2} - x_{r3}) + F_1\,(x_{best} - x_{r1}) \quad (13)$$

Where, V_i = Donor Vector

x_{r1}, x_{r2}, x_{r3} = Random Target Vectors

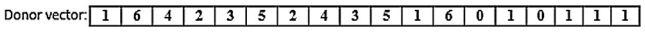


Fig. 17. Donor vector generated.

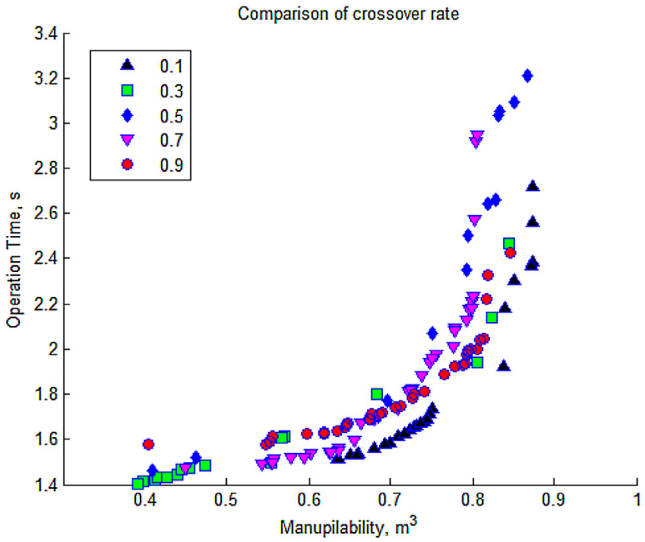


Fig. 18. selection of best crossover rate.

x_{best} = Target Vector with the best fitness.

x_{best} is selected based on the fitness value calculated using Eq. (9).

F_1 = Mutation factor

DE mutation: An illustration:

Let F_1 = 0.5 and the vectors used for illustration are shown in Fig. 16.

Repair algorithm is used to make sure the vector elements are as required.

The steps of repair algorithm are as follow:

1) All the negative elements are change to positive.
2) All decimal elements are rounded.
3) Repeating elements are randomly replaced by another missing element.
4) All elements that are greater than the number of modules will be replaced by another missing element.

By using Eq. (13) and applying the repair algorithm, the final donor vector generated is as shown in Fig. 17.

4.8. Crossover

A mixture of elements from target vector and donor vector will form a trial vector, y. If a randomly generated number, R is less than or equal to crossover rate (C_r), then select the element from donor vector. If R is more than the crossover rate (C_r), then select the element from donor vector. Fig. 18 illustrate the explanation

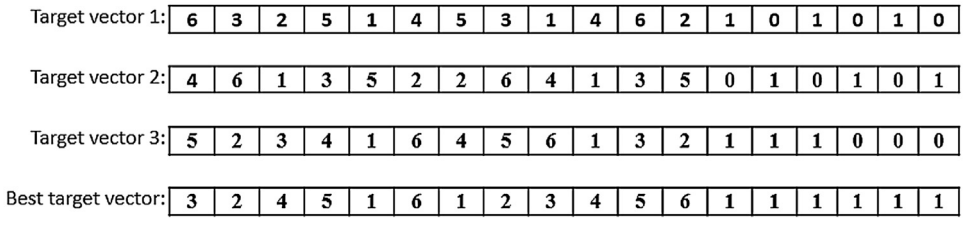


Fig. 16. Target vectors and best vector selected for mutation.

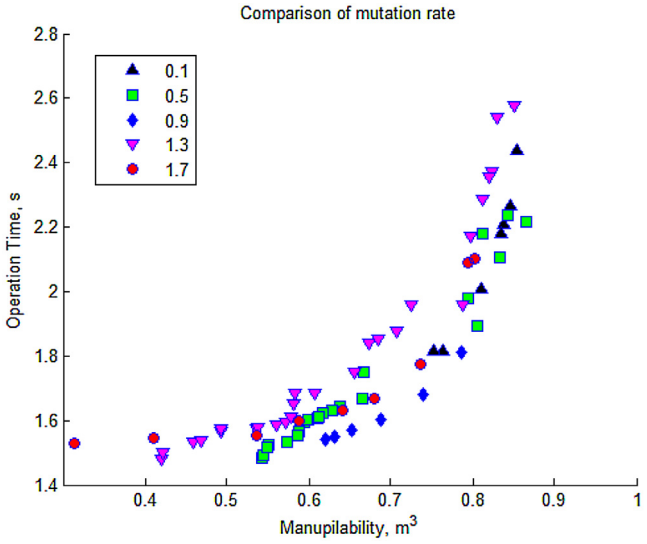


Fig. 19. Selection of best mutation factor.

above. Let target vector elements are represented in white colour and donor vector elements are in grey colour. By using Eq. (14), the trial vectors are generated.

$$y_{i,k}^{j} = \begin{cases} \hat{x}_{i,k}^{j} \text{if} R^{j} \leq C_R \\ x_{i,k}^{j} \text{if} R^{j} > C_R \end{cases} \quad (14)$$

where,

$y_{i,k}^{j}$ *is the trial vector element*,

$\hat{x}_{i,k}^{j}$ *is the donor vector element*,

$x_{i,k}^{j}$ *is the target vector element*, and

C_R *is the crossover rate* $\in [0, 1]$

4.9. Selection

The selection is done by comparing the fitness values of each target vectors $y_{i,G}$) and trial vectors ($x_{i,G}$). If the trial vector has a better fitness then target vector then it will be selected into the final population otherwise target vector will be selected, which is explained in Eq. (15).

$$x_{i,G+1} = \begin{cases} y_{i,G} iff \left(y_{i,G}\right) > f\left(x_{i,G}\right) \\ x_{i,G} \text{otherwise} \end{cases} \quad (15)$$

4.10. Parameters fine tuning

The crossover rate is tested for 0.1, 0.3, 0.5, 0.7 and 0.9. The Pareto front plot obtained is provided in Fig. 18. Based on the plot, the best crossover rate selected is 0.1.

The mutation factor is also tested for 0.1, 0.5, 0.9, 1.3 and 1.7. The Pareto front plot for these values is shown in Fig. 19 and based on this the best mutation factor is set to 0.9.

Algorithm 3. Modified Artificial Bee Colony (MABC)

Artificial Bee Colony (ABC) algorithm uses three different types of bees which are divided into employed, onlooker and scout bees. Employed bees will first exploit new food sources and gathering

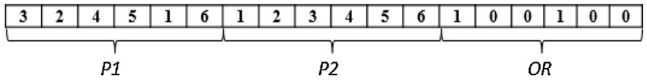


Fig. 20. Food source representation in MABC.

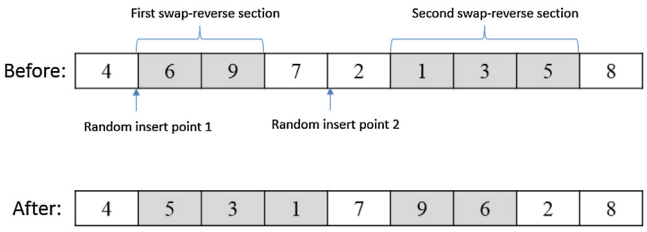


Fig. 21. Swap-reverse combination operator.

information about the food sources. It will then pass on the information gathered to onlooker bees. Onlooker bees will then explore the food sources further. Scout bees come into play when there are abandoned food sources. Scout bees are required to look for new food sources to replace the abandoned food sources. A neighbourhood operator is used to obtain a new solution from the current solution. Szeto et al. [46] proposed modified version of ABC with a set of neighbourhood operators to create new food sources. The steps of the modified ABC proposed by Szeto et al. [46] is adopted in this paper. The implementation details are presented in this section.

4.11. Initialization of population

The food sources are generated randomly and the length of the food source is determined by the number of modules. The food source is made out of three different vectors (*P1*, *P2* and *OR*). An example of the food source generated is shown in Fig. 20.

4.12. Neighbourhood operator

Szeto et al. [46] introduced few neighbourhood operators and compared their performances. The best operator recommended by Szeto et al. [46] is implemented in this paper, which is called swap-reverse combination operator.

4.13. Swap-reverse process

Firstly, two swap section and two insert points are selected. The order of elements in each swap section is reversed and it is then inserted on the random insert point. Fig. 21 shows the vector before and after the swap reverse combination operator.

4.14. Evaluation of food sources

For each food sources, we applied the neighbourhood operator as explained above. Then we compare the fitness value (using Eq. (9) of each existing food sources and the new food sources (after applying neighbourhood operator). The food sources with the higher fitness values are selected into the next phase (Onlooker bee phase).

4.15. Onlooker bee phase

At each iteration, the onlooker bees select a random number of food sources. The selected food sources will then undergo the neighbourhood operator process. Once again, the newly produced food sources are compared with existing food sources and if the fitness value of the new food source is higher than the old food

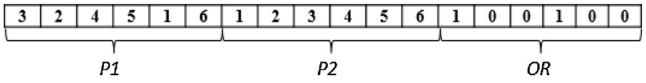


Fig. 22. Vector representation in CSS.

source fitness value, the new food source is replaced by the old food source. if not the old food source remains.

4.16. Scout bee phase

If the fitness value of a food source did not improve after a predetermined number of trials, the food source is assumed as an abandoned food source. Then a new food source which is generated randomly will replace it.

Algorithm 4. Charged System Search (CSS)

Charged System Search [47,60] is relatively a new optimization algorithm based on some principle from physics and mechanics. The principals involved are Coulomb law and Newton law. Readers are advised to refer [60] for the pseudo-code of CSS algorithm, which is adopted in this paper. The implementation details of CSS is discussed in this section. The structure of the vectors in the population is the same as used in the other algorithms as shown in Fig. 22.

4.17. Initialization stage

CSS start off with initialization stage. For each charged particle, it will have a magnitude of charge (q_i). The formula for q_i is given below.

$$q_i = \frac{fit(i) - fitworst}{fitbest - fitworst}, i - 1, 2, 3, \ldots..N \tag{16}$$

where *fitbest* and *fitworst* are the so far best and the worst fitness of all particles; *fit(i)* represents the objective function value or the fitness of the agent *i*; and *N* is the total number of CPs.

The separation distance r_{ij} between two charged particles is defined as follows:

$$r_{ij} = \frac{||X_i - X_j||}{||(X_i + X_j)/2 - X_{best}|| + \varepsilon} \tag{17}$$

where,

Xi and ***Xj*** are the positions of the *i* th and *j* th CPs, ***Xbest*** is the position of the best current CP, and ε is

a small positive number to avoid singularities.

4.18. Search stage

In order to determine the best charged particle, attractive forces are required. A charged particle can attract another if its electric charged amount (fitness) is better than the other. A simple probability function is used to determine whether a charged particle can be attracted to another as shown in Eq. (18).

$$p_{ij} = \begin{cases} 1 \dfrac{fit(i) - fitbest}{fit(j) - fit(i)} > rand \vee fit(j) > fit(i) \\ 0 else \end{cases} \tag{18}$$

The value of the resultant electrical force acting on a charged particle is determined using Eq. (19).

$$F_j = q_j . \sum_{i,i \neq j} \left(\frac{q_i}{a^3} r_{ij} . i_1 + \frac{q_i}{r_{ij}^2} . i_2 \right) . p_{ij} . \left(X_i - X_j \right),$$

$$\begin{cases} j = 1, 2, \ldots, n \\ i_1 = 1, i_2 = 0 \Leftrightarrow r_{ij} < a \\ i_1 = 0, i_2 = 1 \Leftrightarrow r_{ij} \geq a \end{cases} \tag{19}$$

Where, F_j is the resultant force acting on the j^{th} charged particle and *a* is set to 1.

After finding the resultant force, the new position and velocity of each charged particle can be determined by using Eqs. (20) and (21).

$$X_{j,new} = 0.5 rand_{j1} . \left(1 + iter/iter_{max}\right) . F_j + 0.5 rand_{j2} . \left(1 + iter/iter_{max}\right) . V_{j,old} + X_{j,old} \tag{20}$$

$$V_{j,new} = X_{j,new} - X_{j,old} \tag{21}$$

where *iter* is the current generation number and $iter_{max}$ is the maximum number of generation.

Algorithm 5. Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a population based meta-heuristic search technique in which a swarm consist of a set of particles [51,52]. Readers are advised to refer [51,52] for the structure of PSO algorithm. The implementation details are presented in this section.

Each particle holds a position and a velocity. Each particle will adjust its position based on the velocity and move towards the global optimum.

4.19. Velocity generation

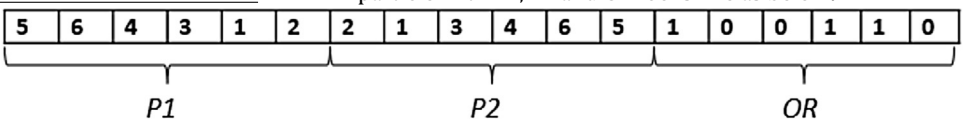
Initially, the length of velocity for each particle, ||V|| is generated randomly between 0 to *n (the total length of individual vector)*. List of transpositions are generated based on the length of velocity. For example, if ||V|| = 2, the transpositions can be V_1 = ((1, 3), (2, 4)). Similarly for the remaining (n-1) particles, velocity lengths and the corresponding lists are generated randomly.

4.20. Position update

Particle *i* at time step *t* is represented by P_i^t, and P_i^{t+1} is computed using equation (22).

$$P_i^{t+1} = P_i^t + V_i^{t+1} \tag{22}$$

A particle with P1, P2 and OR looks like as below.



4.21. Velocity update

Each final particle vector adjusts its position toward the global optimum according to Eq. (23).

$$V_i^{t+1} = V_i^t + C_1 U_1 \left({}^e P_i^t - P_i^t \right) + C_2 U_2 \left(G - P_i^t \right) \tag{23}$$

Where, $U_1 and U_2$ are random number in the range of [0,1]; $C_1 and C_2$ are weights of present fitness; ${}^e P_i^t$ is the particle best and G is global best particle.

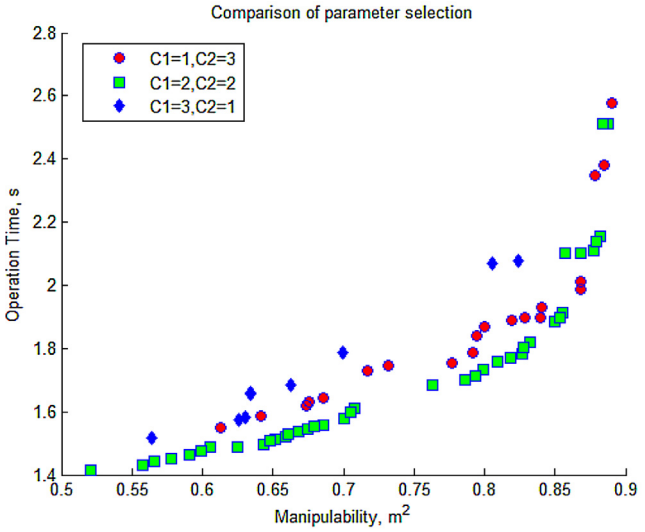


Fig. 23. Performance of C1 and C2 combinations in PSO.

4.22. Parameter fine tuning

The parameters used in PSO are C1 and C2. C1 controls the importance of personal best particle and C2 controls the importance of neighbourhood best. Based on Beielstein et al. [61], the summation of C1 and C2 should be less than or equal to 4. Therefore, in the tuning process, C1 and C2 are set in three different combinations. The combinations are (C1 = 1, C2 = 3), (C1 = 2, C2 = 2) and (C1 = 3, C2 = 1).The plot of Pareto front for each combination of C1 and C2 are shown in Fig. 23. The combination of (C1 = 2, C2 = 2) clearly performs better than the other combinations.

Algorithm 6. Non-dominated Sorting Genetic Algorithm II (NSGA-II)

NSGA-II [34] is used to handle multiple objectives in all the algorithms. The first step is to initialize an initial population of size N. An offspring population of size N is then created using a simple mutation method called inversion mutation. The inversion mutation operator randomly selects two positions on each individual and inverts the substring between these two positions. The two populations are merged and the size of the merged population is now 2N.

Second step is to apply Non-dominated sorting method to divide the population into different rank or front. The ranks assigned are based on the objective functions. Then crowding distance is used to get a new population of size N. Then the algorithms (GA/DE/MABC/CSS/PSO) are applied on the population generated.

5. Comparison of layout representation scheme and justification of number of generations

The performance of B*tree and Sequence pair schemes are evaluated. The analysis to find the optimal number of generations is also conducted.

5.1. Comparison of B*tree and sequence pair

For layout scheme comparison, the algorithm used is GA with NSGA-II. The parameters used are: population size is 50, the number of generations is 100, the crossover rate is 0.8 and mutation rate is *(1/num)/2*, where *num* represents the total number of stations. Simulation experiments are conducted using the test problems given in Appendix. The three objective function values are computed for the two layout representation schemes. The results obtained for the non-dominated (pareto) solutions generated and are presented in Tables 3 and 4.

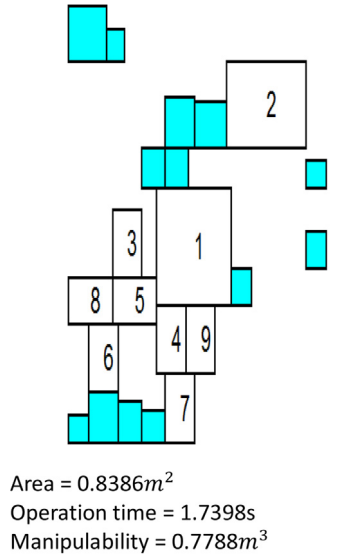


Fig. 24. Sequence pair layout (Problem 4).

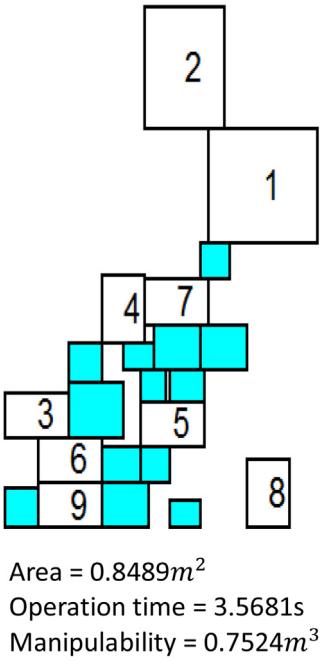


Fig. 25. B*tree layout (Problem 4).

Sequence pair provides the lowest operation time of 1.74 s while the lowest operation time from B*tree is 2.22 s. Sequence pair also produce the highest manipulability of 0.88, while B*tree only managed to produce 0.78. It is found that sequence pair is performing better than B*tree. Another advantages of using sequence pair is that its computation time is almost ten times faster than the computation time of B*tree. The average computation time for sequence pair is 16 s while B*tree took around 120 s for the sample problem (problem 4) tested.

The layout generated by the two schemes are shown in Figs. 24 and 25. All the three objective function values are better for the layout generated using sequence pair, as shown in Fig. 24. It is observed that sequence pair performs better than B*tree. Thus sequence pair is used to generate layouts in this paper.

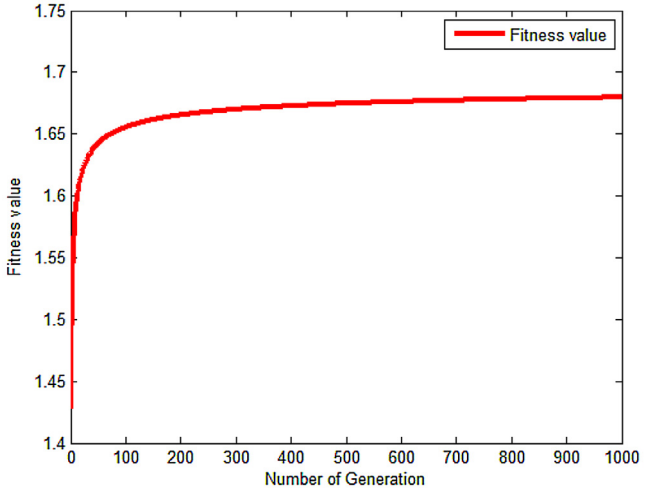


Fig. 26. convergence graph for all algorithms.

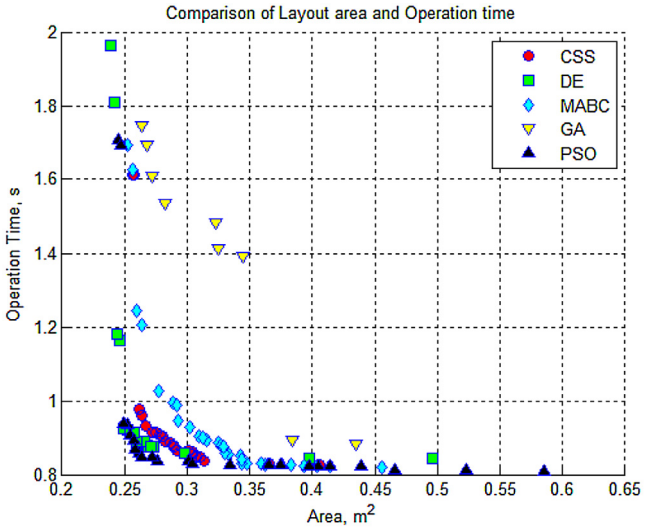


Fig. 27. Comparison of layout area and operation time for all algorithms.

5.2. Justification of generation number

The number of generations is initially set to 1000. The convergence graph of fitness value over number of generations for each individual algorithm are plotted. The convergence graph for all the algorithms is almost the same. Fig. 26 shows the convergence graph obtained for each algorithm. Eq. (9) is used to calculate the combined fitness value to plot the graph. It is observed that for all the algorithms the convergence starts to stabilize after 500 generations. Therefore, the number of generations selected for all the algorithms is 800.

6. Results and discussion

In order to evaluate the performance of the algorithm, ten problems are randomly generated and the results of these problems are presented in Tables 5–7. All values recorded in Tables 5–7 are calculated based on Eq. (24), where f (Eq. (9)) represent the fitness value for each non-dominated solution and n represent the total number of non-dominated solutions in the Pareto front.

$$Averagefitnessvalue = \frac{\sum (f_1 \ldots f_n)}{n} \tag{24}$$

For illustrative purpose, pareto fronts generated by the algoroithms are presented in Figs. 27–29 for test problem 4.

The results presented in Tables 5–7 clearly shows the better performance of PSO over other algorithms considered in this paper. To further justify the results, the Pareto front (for probeolm 4) shown in Figs. 27–29 indicate that the better performance of PSO.

It is also found that the performance of PSO is quite consistent throughout the simulation. DE is performing very close to PSO. Both PSO and DE Pareto fronts are very close to each other as shown in Fig. 27. It is also observed that the overall computation time required for PSO is also faster than DE. GA was not performing well in most cases which could be seen in Fig. 27. The Pareto front of GA is quite far away from PSO and the fitness values are the lowest in most cases.

6.1. Performance of PSO algorithm and the effect of objective functions for two objective case

Since PSO is performing better over other algorithms, sample layouts are generated using PSO for the test problem 4 considering two objective combinations; layout area with operation time and layout area with manipulability. The pareto front sloutions for

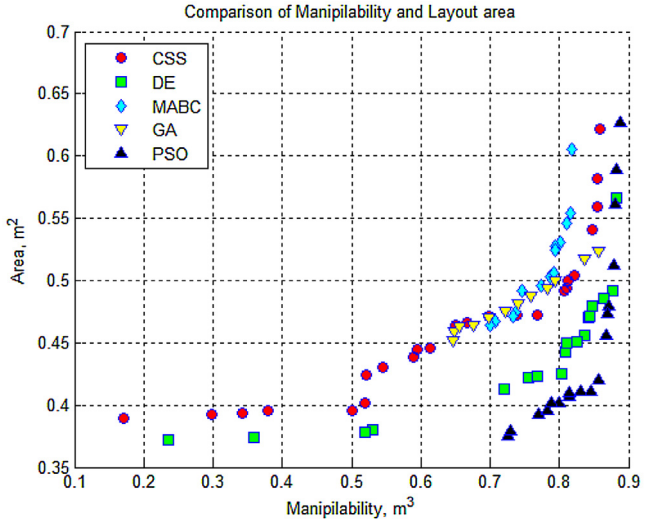


Fig. 28. Comparison of layout area and manipulability for all algorithms.

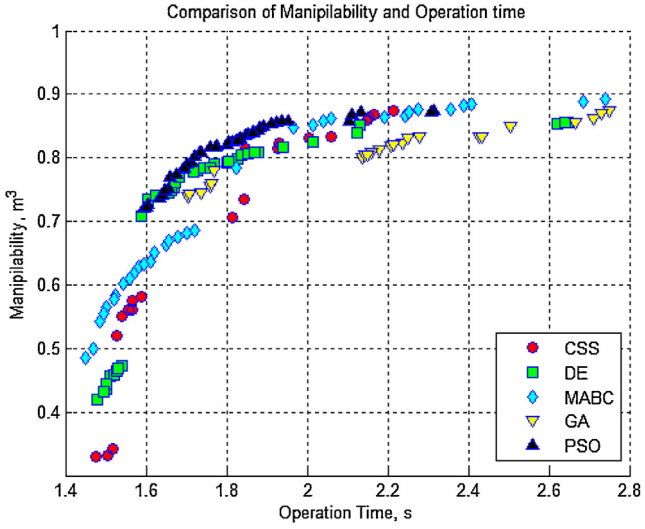


Fig. 29. Comparison of manipulability and operation time for all algorithms.

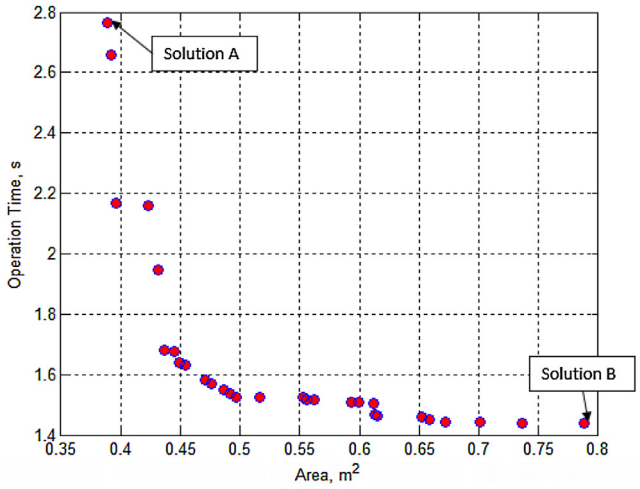


Fig. 30. Pareto front slutions for layout area and operation time combination.

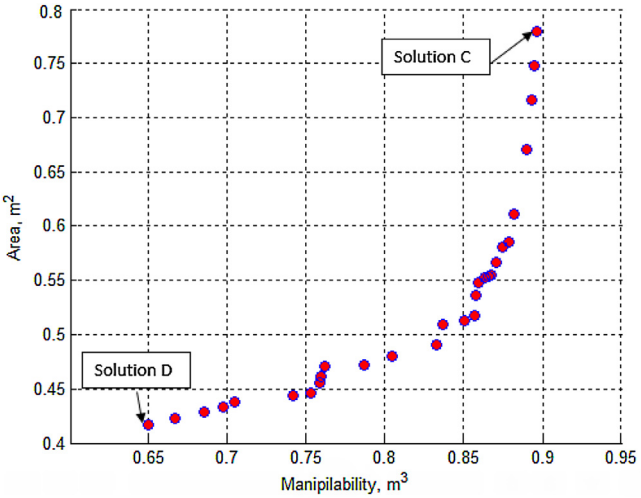


Fig. 31. Pareto front slutions for layout area and manipulability combination.

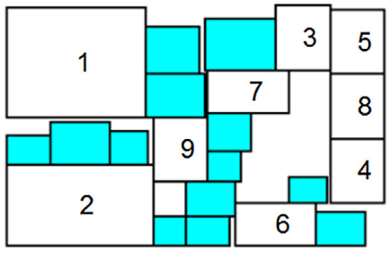


Fig. 32. Layout of solution A (Layout area = 0.3892m^2, Operation time = 2.7670s).

these combinations are presented in Figs. 30 and 31. Two extreme solutions are selected for both cases. Solution A has the smallest layout area while solution B has the shortest operation time. Similarly solution C has the largest manipulability and solution D has the smallest layout area. These four solutions clearly shows that there are trade-off relationship among the three objectives, such as, when the layout area is smallest the results obtained for operation time and manipulability are poor.

Figs. 32–35 shows the layouts for the four (A–D) solutions. The trade-off relationship could be further justified based on the layouts presented in these figures. As shown in Figs. 32 and 35, both solution A and D have the smallest layout area but since all the components are closely pack to each other, the robot arm requires more time and extra movements during assembly operations to avoid collisions thus increasing the operation time and decreasing manipulability in the process.

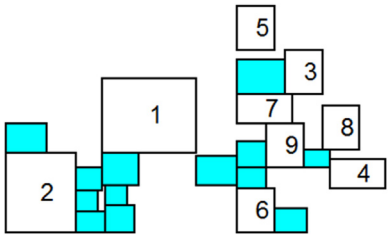


Fig. 33. Layout of solution B (Layout area = 0.7877m^2, Operation time = 1.44s).

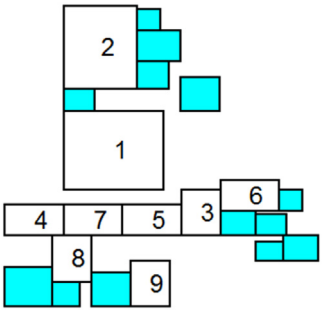


Fig. 34. Layout of solution C (Layout area = 0.78m^2, Manipulability = 0.8967s).

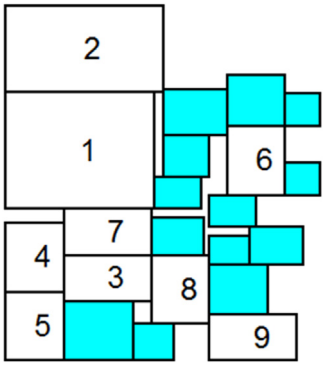


Fig. 35. Layout of solution D (Layout area = 0.4174m^2, Manipulability = 0.6492s).

The workcell layout for solution B and C layout are shown in Figs. 33 and 34 respectively. Layouts for solution B and C in figures have a similar design where most of the components are spread out and the surrounding of the robot station (Box 1) has more empty space. This is completely different from the case of solution A and D where there are limited free space around the robot station. Therefore, this effectively reduced the operation time needed and increased the manipulability as the robot arm can move freely as it has a lesser tendency for collision to occurs. However, this comes at the cost of having a larger layout area.

6.2. Performance of PSO algorithm and the effect of objective functions for three objective case

Concurrent optimization of all the three objectives is also considered in this paper. PSO algorithm and the test problem 4 is used to explain the performance. Fig. 36 shows the non-dominated solutions for three objective optimization. Fig. 37 shows the extracted plane view of comparison between manipulability and operation time from Fig. 37.

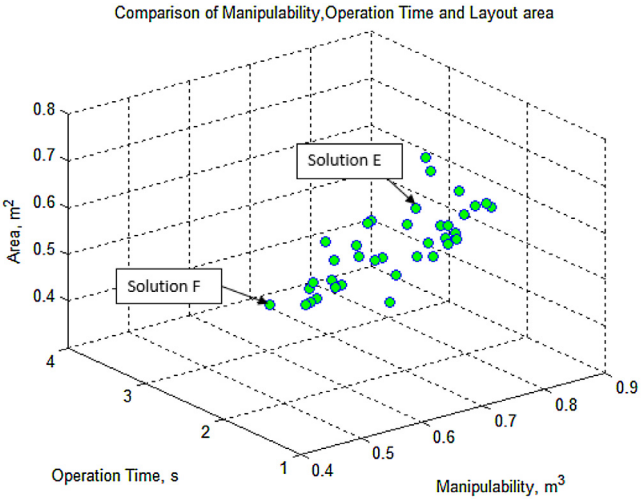


Fig. 36. Non-dominated solutions for three objective comparisons.

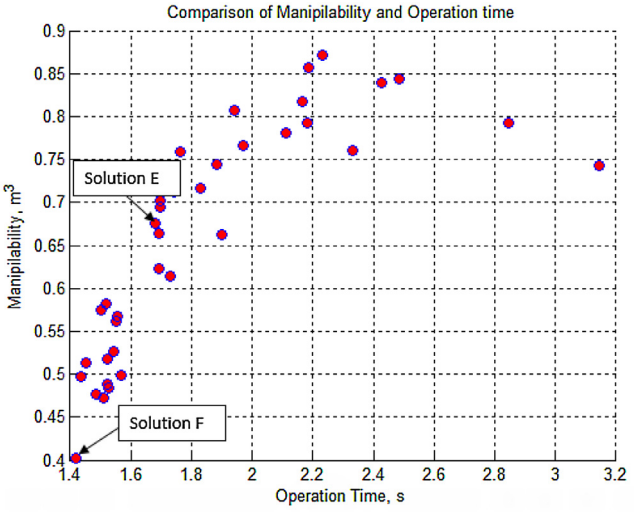


Fig. 37. Non-dominated solutions mapped to the manipulability-operation time plane.

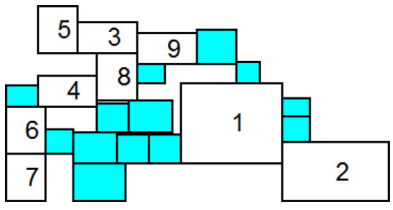


Fig. 38. Layout of solution E (Layout area = 0.5983m^2, Operation time = 1.6783s, Manipulability = 0.6753m^3).

Solution F has the best value in terms of operation time but worst in terms of manipulability index. Layout for solution F is shown in Fig. 39, it can be seen that the components (Box 4–7) are located next to each other while components (box 8, 9 & 3) are group together. These arrangements causes difficulty for the robot arm during assembly operations as it requires more radical movements to move from one component to the other. Therefore, this results in poor manipulability index.

Due to the trade-off relationship as explained above, there will not be a perfect layout design solution. Thus, in order to obtain good manipulability and operation time, layout area will need to be compromised. Fig. 38 shows the layout design for solution E, which is

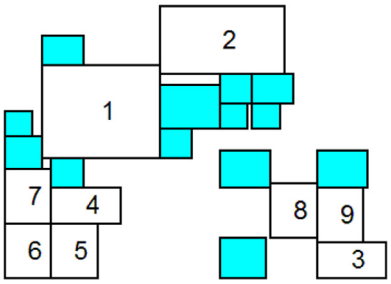


Fig. 39. Layout of solution F (Layout area = 0.4023m^2, Operation time = 1.418s, Manipulability = 0.5887m^3).

Table 8
Modules specifications.

Station ID	Size (height, width) mm	No. of operations
1 (Robot)	255 2 55	–
2 (Assembly Table)	270 188	6
3 (Part Box 1)	150 100	1
4 (Part Box 2)	150 100	2
5 (Part Box 3)	150 100	2
6 (Part Box 4)	150 100	1

consider as the best workcell layout solution as it has a reasonable performance values for every criterion and shows a good compromise among the three objectives. As shown in Fig. 38, there are empty spaces around the robot station (Box 1) and the components (Box 3–9) are arranged nearby each other. The free space provides the robot arm with more manipulability while the compactness of the components arrangement allows it to maintain a reasonable layout area and operation time.

7. Conclusion

Genetic algorithm, differential evolution, particle swarm optimization, artificial bee colony and charged system search algorithms are proposed to solve robot workcell layout problems. The performance of these five algorithms are evaluated using ten problems generated randomly. Three objective functions are considered in this paper, viz, Layout area, operation time and manipulability of robot. The implementation details of all the algorithms are presented. Two layout representation schemes, B*tree and sequence pair are evaluated and found that sequence pair representation scheme performs better. The parameters of the five nature inspired algorithms considered in this paper are fine-tuned to get better results. The assembly robot used to illustrate the problem is Mitsubishi RV-6SQ. Two objective case and three case of the problems are analysed separately and the results are presented. NSGA-II is used to handle multiple objectives and to optimize them simultaneously. It is found that PSO performs better over other algorithms. Best layouts for the two-objective three-objective cases are generated using PSO. The best layout design was found by optimizing three objectives simultaneously as there is trade-off among the objective functions.

Future research work could expand the cell size with more number of machines and more number of robots. Another research direction could be to develop layout representation schemes, which are efficient and fast to generate layouts.

Appendix.

Problem 1 setup details

See Tables 8 and 9

Table 9
Dummy blocks specifications.

Dummy ID	Size (height, width) mm
7	70 80
8	100 110
9	80 90
10	90 100
11	50 60
12	90 100

Table 10
Modules specifications.

Station ID	Size (height, width) mm	No. of operations
1 (Robot)	255 2 55	–
2 (Assembly Table)	270 188	10
3 (Part Box 1)	150 100	3
4 (Part Box 2)	150 100	2
5 (Part Box 3)	150 100	2
6 (Part Box 4)	150 100	2
7 (Part Box 5)	150 100	1

Table 11
Dummy blocks specifications.

Dummy ID	Size (height, width) mm
8	110 120
9	60 70
10	50 60
11	100 110
12	90 100
13	80 90
14	110 120
15	70 80

Table 12
Modules specifications.

Station ID	Size (height, width) mm	No. of operations
1 (Robot)	255 2 55	–
2 (Assembly Table)	270 188	15
3 (Part Box 1)	150 100	2
4 (Part Box 2)	150 100	2
5 (Part Box 3)	150 100	3
6 (Part Box 4)	150 100	1
7 (Part Box 5)	150 100	3
8 (Part Box 6)	150 100	2

Table 13
Dummy blocks specifications.

Dummy ID	Size (height, width) mm
9	80 90
10	60 70
11	90 100
12	110 120
13	90 100
14	80 90
15	110 120
16	70 80
17	100 110
18	70 80

Problem 2 setup details

See Tables 10 and 11

Problem 3 setup details

See Tables 12 and 13

Table 14
Modules specifications.

Station ID	Size (height, width) mm	No. of operations
1 (Robot)	255,255	–
2 (Assembly Table)	270,188	10
3 (Part Box 1)	150,100	2
4 (Part Box 2)	150,100	2
5 (Part Box 3)	150,100	2
6 (Part Box 4)	150,100	1
7 (Part Box 5)	150,100	1
8 (Part Box 6)	150,100	1
9 (Part Box 7)	150,100	1

Table 15
Dummy blocks specifications.

Dummy ID	Size (height, width) mm
10	60,70
11	70,80
12	80,90
13	100,110
14	120,130
15	60,70
16	70,80
17	80,90
18	100,110
19	60,70
20	70,80
21	80,90
22	100,110

Table 16
Modules specifications.

Station ID	Size (height, width) mm	No. of operations
1 (Robot)	255 2 55	–
2 (Assembly Table)	270 188	17
3 (Part Box 1)	150 100	2
4 (Part Box 2)	150 100	1
5 (Part Box 3)	150 100	2
6 (Part Box 4)	150 100	1
7 (Part Box 5)	150 100	3
8 (Part Box 6)	150 100	3
9 (Part Box 7)	150 100	3
10 (Part Box 8)	150 100	2

Table 17
Dummy blocks specifications.

Dummy ID	Size (height, width) mm
11	80 90
12	60 70
13	100 110
14	110 120
15	50 60
16	80 90
17	120 130
18	110 120
19	50 60
20	80 90
21	110 120
22	100 110
23	60 70
24	70 80

Problem 4 setup details

See Tables 14 and 15

Problem 5 setup details

See Tables 16 and 17

Table 18
Modules specifications.

Station ID	Size (height, width) mm	No. of operations
1 (Robot)	255 2 55	–
2 (Assembly Table)	270 188	18
3 (Part Box 1)	150 100	3
4 (Part Box 2)	150 100	3
5 (Part Box 3)	150 100	1
6 (Part Box 4)	150 100	3
7 (Part Box 5)	150 100	3
8 (Part Box 6)	150 100	1
9 (Part Box 7)	150 100	1
10 (Part Box 8)	150 100	1
11 (Part Box 9)	150 100	2

Table 19
Dummy blocks specifications.

Dummy ID	Size (height, width) mm
12	80 90
13	90 100
14	50 60
15	50 60
16	120 130
17	70 80
18	60 70
19	70 80
20	90 100
21	110 120
22	70 80
23	80 90
24	100 110
25	70 80

Table 20
Modules specifications.

Station ID	Size (height, width) mm	No. of operations
1 (Robot)	255 2 55	–
2 (Assembly Table)	270 188	19
3 (Part Box 1)	150 100	1
4 (Part Box 2)	150 100	1
5 (Part Box 3)	150 100	1
6 (Part Box 4)	150 100	3
7 (Part Box 5)	150 100	2
8 (Part Box 6)	150 100	3
9 (Part Box 7)	150 100	1
10 (Part Box 8)	150 100	2
11 (Part Box 9)	150 100	3
12 (Part Box 1 0)	150 100	2

Problem 6 setup details

See Tables 18 and 19

Problem 7 setup details

See Tables 20 and 21

Problem 8 setup details

See Tables 22 and 23

Problem 9 setup details

See Tables 24 and 25

Problem 10 setup details

See Tables 26 and 27

Table 21
Dummy blocks specifications.

Dummy ID	Size (height, width) mm
13	110 120
14	90 100
15	120 130
16	120 130
17	80 90
18	60 70
19	60 70
20	70 80
21	100 110
22	90 100
23	60 70
24	80 90
25	110 120
26	50 60

Table 22
Modules specifications.

Station ID	Size (height, width) mm	No. of operations
1 (Robot)	255 2 55	–
2 (Assembly Table)	270 188	20
3 (Part Box 1)	150 100	3
4 (Part Box 2)	150 100	2
5 (Part Box 3)	150 100	2
6 (Part Box 4)	150 100	1
7 (Part Box 5)	150 100	2
8 (Part Box 6)	150 100	1
9 (Part Box 7)	150 100	2
10 (Part Box 8)	150 100	1
11 (Part Box 9)	150 100	2
12 (Part Box 1 0)	150 100	1
13 (Part Box 1 1)	150 100	3

Table 23
Dummy blocks specifications.

Dummy ID	Size (height, width) mm
14	120 130
15	70 80
16	100 110
17	90 100
18	90 100
19	80 90
20	60 70
21	100 110
22	110 120
23	80 90
24	60 70
25	70 80
26	110 120
27	120 130

Table 24
Modules specifications.

Station ID	Size (height, width) mm	No. of operations
1 (Robot)	255 2 55	–
2 (Assembly Table)	270 188	21
3 (Part Box 1)	150 100	1
4 (Part Box 2)	150 100	2
5 (Part Box 3)	150 100	2
6 (Part Box 4)	150 100	1
7 (Part Box 5)	150 100	2
8 (Part Box 6)	150 100	2
9 (Part Box 7)	150 100	3
10 (Part Box 8)	150 100	2
11 (Part Box 9)	150 100	1
12 (Part Box 1 0)	150 100	2
13 (Part Box 1 1)	150 100	1
14 (Part Box 1 2)	150 100	2

Table 25
Dummy blocks specifications.

Dummy ID	Size (height, width) mm
15	50 60
16	70 80
17	90 100
18	80 90
19	60 70
20	110 120
21	100 110
22	50 60
23	120 130
24	80 90
25	110 120
26	100 110
27	70 80
28	70 80

Table 26
Modules specifications.

Station ID	Size (height, width) mm	No. of operations
1 (Robot)	255 2 55	–
2 (Assembly Table)	270 188	22
3 (Part Box 1)	150 100	2
4 (Part Box 2)	150 100	2
5 (Part Box 3)	150 100	1
6 (Part Box 4)	150 100	2
7 (Part Box 5)	150 100	1
8 (Part Box 6)	150 100	2
9 (Part Box 7)	150 100	1
10 (Part Box 8)	150 100	2
11 (Part Box 9)	150 100	2
12 (Part Box 1 0)	150 100	2
13 (Part Box 1 1)	150 100	1
14 (Part Box 1 2)	150 100	1
15 (Part Box 1 3)	150 100	3

Table 27
Dummy blocks specifications.

Dummy ID	Size (height, width) mm
16	60 70
17	120 130
18	110 120
19	70 80
20	100 110
21	50 60
22	50 60
23	120 130
24	80 90
25	70 80
26	60 70
27	110 120
28	80 90
29	100 110

References

[1] C. Renzi, F. Leali, M. Cavazzuti, A. Andrisano, A review on artificial intelligence applications to the optimal design of dedicated and reconfigurable manufacturing systems, Int. J. Adv. Manuf. Technol. 72 (2014) 403–418.
[2] N. Ngampak, B. Phruksaphanrat, Cellular manufacturing layout design and selection: a case study of electronic manufacturing service plant, Proceedings of International MultiConference of Engineers and Computer Scientists (2011) 16–18.
[3] A. Agarwal, Partitioning bottleneck work center for cellular manufacturing: an integrated performance and cost model, Int. J. Prod. Econ. 111 (2008) 635–647.
[4] S. Angra, R. Sehgal, Z. Samsudeen Noori, Cellular manufacturing—a time-based analysis to the layout problem, Int. J. Prod. Econ. 112 (2008) 427–438.
[5] B. Javadi, F. Jolai, J. Slomp, M. Rabbani, R. Tavakkoli-Moghaddam, An integrated approach for the cell formation and layout design in cellular manufacturing systems, Int. J. Prod. Res. 51 (2013) 6017–6044.
[6] K. Shahookar, P. Mazumder, VLSI cell placement techniques, ACM Comput. Surv. (CSUR) 23 (1991) 143–220.
[7] T.C. Koopmans, M. Beckmann, Assignment problems and the location of economic activities, Econom.: J. Econom. Soc. (1957) 53–76.
[8] A. Drira, H. Pierreval, S. Hajri-Gabouj, Facility layout problems: a survey, Ann. Rev. Control 31 (2007) 255–267.
[9] J.M. Kleinhans, G. Sigl, F.M. Johannes, K.J. Antreich, GORDIAN: VLSI placement by quadratic programming and slicing optimization, Comput.-Aided Des. Integr. Circuits Syst. 10 (1991) 356–365, IEEE Transactions on.
[10] T.C. Lueth, Automated planning of robot workcell layouts, Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference On, IEEE (1992) 1103–1108.
[11] D. Barral, J.-P. Perrin, E. Dombre, A. Liegeois, Simulated annealing combined with a constructive algorithm for optimising assembly workcell layout, Int. J. Adv. Manuf. Technol. 17 (2001) 593–602.
[12] M.M. Tay, B. Ngoi, Optimising robot workcell layout, Int. J. Adv. Manuf. Technol. 12 (1996) 377–385.
[13] Z. Drezner, S.Y. Nof, On optimizing bin picking and insertion plans for assembly robots, IIE Trans. 16 (1984) 262–270.
[14] F.S. Cheng, Design and optimization of cellular manufacturing systems: a methodology for developing robotic workcell simulation models, Proceedings of the 32nd Conference on Winter Simulation, Society for Computer Simulation International (2000) 1265–1271.
[15] H.J. Yap, Z. Taha, S.Z.M. Dawal, S.-W. Chang, Virtual reality based support system for layout planning and programming of an industrial robotic work cell, PLoS One 9 (2014) e109692.
[16] Y.S. Pai, H.J. Yap, R. Singh, Augmented reality-based programming, planning and simulation of a robotic work cell, Proc. Inst. Mech. Eng. Part B: J. Eng. Manuf. 229 (2015) 1029–1045.
[17] E.-D. Zhang, L.-L. Qi, S. Murphy, Method and system for optimizing the layout of a robot work cell, in, Google Patents, 2013.
[18] J. Tao, P. Wang, H. Qiao, Z. Tang, Facility layouts based on differential evolution algorithm, Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference On, IEEE (2013) 1778–1783.
[19] Z. Jian, L. Ai-Ping, Genetic algorithm for robot workcell layout problem, Software Engineering, 2009. WCSE'09. WRI World Congress On, IEEE (2009) 460–464.
[20] A. Islier, A genetic algorithm approach for multiple criteria facility layout design, Int. J. Prod. Res. 36 (1998) 1549–1569.
[21] S.-K. Sim, M.-L. Tay, A. Khairyanto, Optimisation of a robotic workcell layout using genetic algorithms, ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers (2005) 921–930.
[22] A.E. Dunlop, B.W. Kernighan, A procedure for placement of standard cell VLSI circuits, IEEE Trans. Comput.-Aided Des. 4 (1985) 92–98.
[23] M. Gen, L. Lin, H. Zhang, Evolutionary techniques for optimization problems in integrated manufacturing system: state-of-the-art-survey, Comput. Ind. Eng. 56 (2009) 779–808.
[24] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, S.-W. Wu, B*-Trees a new representation for non-slicing floorplans, Proceedings of the 37th Annual Design Automation Conference, ACM (2000) 458–463.
[25] P.-N. Guo, C.-K. Cheng, T. Yoshimura, An O-tree representation of non-slicing floorplan and its applications, Proceedings of the 36th Annual ACM/IEEE Design Automation Conference, ACM (1999) 268–273.
[26] H. Murata, K. Fujiyoshi, S. Nakatake, Y. Kajitani, VLSI module placement based on rectangle-packing by the sequence-pair, Comput.-Aided Des. Integr. Circuits Syst. 15 (1996) 1518–1524, IEEE Transactions on.
[27] C. Caraveo, F. Valdez, O. Castillo, Bio-inspired optimization algorithm based on the self-defense mechanism in plants, in: Advances in Artificial Intelligence and Soft Computing, Springer, 2015, pp. 227–237.
[28] F. Valdez, P. Melin, O. Castillo, A survey on nature-inspired optimization algorithms with fuzzy logic for dynamic parameter adaptation, Expert Syst. Appl. 41 (2014) 6459–6466.
[29] F. Valdez, Bio-inspired optimization methods, in: Springer Handbook of Computational Intelligence, Springer, 2015, pp. 1533–1538.
[30] V. Dwivedi, T. Chauhan, S. Saxena, P. Agrawal, Travelling salesman problem using genetic algorithm, IJCA Proceedings on Development of Reliable Information Systems, Techniques and Related Issues (DRISTI 2012) (2012) 25.
[31] R. Rajagopalan, D.J. Fonseca, A genetic algorithm approach for machine cell formation, J. Adv. Manuf. Syst. 5 (2006) 27–44.
[32] H. Pourvaziri, B. Naderi, A hybrid multi-population genetic algorithm for the dynamic facility layout problem, Appl. Soft Comput. 24 (2014) 457–469.
[33] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, Evol. Comput. 6 (2002) 182–197, IEEE Transactions on.
[34] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II, Lect. Notes Comput. Sci. 1917 (2000) 849–858.
[35] K. Izui, Y. Murakumo, I. Suemitsu, S. Nishiwaki, A. Noda, T. Nagatani, Multiobjective layout optimization of robotic cellular manufacturing systems, Comput. Ind. Eng. 64 (2013) 537–544.
[36] B.H. Tang, Z.X. Zhang, Multi-objective optimization of manufacturing workshop layout based on improved genetic algorithm, in: Advanced Materials Research, Trans Tech Publ, 2014, pp. 2664–2668.
[37] R. Storn, K. Price, Differential Evolution-a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces, ICSI, Berkeley, 1995.
[38] D. Zaharie, Influence of crossover on the behavior of differential evolution algorithms, Appl. Soft Comput. 9 (2009) 1126–1138.

[39] C.-W. Chien, Z.-R. Hsu, W.-P. Lee, Improving the performance of differential evolution algorithm with modified mutation factor, International Conference on Machine Learning and Computing IPCSIT (2011).
[40] R. Mallipeddi, P.N. Suganthan, Differential evolution algorithm with ensemble of parameters and mutation and crossover strategies, in: Swarm, Evolutionary, and Memetic Computing, Springer, 2010, pp. 71–78.
[41] F. Xue, A.C. Sanderson, R.J. Graves, Multi-objective differential evolution and its application to enterprise planning, Robotics and Automation, 2003 Proceedings. ICRA'03. IEEE International Conference On, IEEE (2003) 3535–3541.
[42] T. Robič, B. Filipič, DEMO: differential evolution for multiobjective optimization, in: Evolutionary Multi-Criterion Optimization, Springer, 2005, pp. 520–533.
[43] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, J. Global Optim. 39 (2007) 459–471.
[44] D. Karaboga, B. Basturk, On the performance of artificial bee colony (ABC) algorithm, Appl. Soft Comput. 8 (2008) 687–697.
[45] Á. Rubio-Largo, M.Á. Vega-Rodríguez, D.L. González-Álvarez, Hybrid multiobjective artificial bee colony for multiple sequence alignment, Appl. Soft Comput. 41 (2016) 157–168.
[46] W. Szeto, Y. Wu, S.C. Ho, An artificial bee colony algorithm for the capacitated vehicle routing problem, Eur. J. Oper. Res. 215 (2011) 126–135.
[47] A. Kaveh, S. Talatahari, Optimal design of skeletal structures via the charged system search algorithm, Struct. Multidiscip. Optim. 41 (2010) 893–911.
[48] S. Özyön, H. Temurtaş, B. Durmuş, G. Kuvat, Charged system search algorithm for emission constrained economic power dispatch problem, Energy 46 (2012) 420–430.
[49] W.T. Reeves, Particle systems—a technique for modeling a class of fuzzy objects, ACM Trans. Graphics (TOG) 2 (1983) 91–108.
[50] C.W. Reynolds, herds and schools: a distributed behavioral model, in: ACM Siggraph Computer Graphics, ACM, 1987, pp. 25–34.
[51] J. Kennedy, R. Eberhart, Particle swarm optimization, Neural Networks 1995 Proceedings., IEEE International Conference on vol. 1944 (1995) 1942–1948.
[52] J. Kennedy, R.C. Eberhart, A discrete binary version of the particle swarm algorithm, Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference On, IEEE (1997) 4104–4108.
[53] A. Banks, J. Vincent, C. Anyakoha, A review of particle swarm optimization. Part I: background and development, Nat. Comput. 6 (2007) 467–484.
[54] A. Banks, J. Vincent, C. Anyakoha, A review of particle swarm optimization. Part II: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications, Nat. Comput. 7 (2008) 109–124.
[55] D.-b. Chen, F. Zou, J.-t. Wang, A multi-objective endocrine PSO algorithm and application, Appl. Soft Comput. 11 (2011) 4508–4520.
[56] E.L. Lawler, Combinatorial optimization: networks and matroids, Courier Corp. (1976).
[57] S. Nakaya, T. Koide, S.i. Wakabayashi, An adaptive genetic algorithm for VLSI floorplanning based on sequence-pair, Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium On, IEEE (2000) 65–68.
[58] A.K. Qin, P.N. Suganthan, Self-adaptive differential evolution algorithm for numerical optimization, Evolutionary Computation, 2005. The 2005 IEEE Congress On, IEEE (2005) 1785–1791.
[59] M. Ali, M. Pant, A. Abraham, Simplex differential evolution, Acta Polytech. Hungarica 6 (2009) 95–115.
[60] A. Kaveh, S. Talatahari, A novel heuristic optimization method: charged system search, Acta Mech. 213 (2010) 267–289.
[61] T. Beielstein, K.E. Parsopoulos, M.N. Vrahatis, Tuning PSO Parameters Through Sensitivity Analysis, Universität Dortmund, 2002.