Available online at www.sciencedirect.com

ScienceDirect

Procedia Manufacturing 17 (2018) 944–951

Procedia MANUFACTURING

www.elsevier.com/locate/procedia

28th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2018), June 11-14, 2018, Columbus, OH, USA

Comparison of Four Population-Based Meta-Heuristic Algorithms on Pick-and-Place Optimization

Tian He[a], Haifeng Wang[a], Sang Won Yoon[a,*]

[a]*Department of Systems Science and Industrial Engineering, Binghamton University, State University of New York, Binghamton, NY, 13902, USA*

Abstract

This paper applies four population-based classical meta-heuristic algorithms to solve a pick-and-place optimization problem for a surface mounter in a PCB assembly environment. A mathematical model of this optimization problem is formulated as an integrated problem of the capacitated vehicle routing problem and the quadratic assignment problem, which are well-known NP-hard problems. A brief description of each method is presented and special operators for the integer encoded solutions are developed. Ten real-world PCB samples are tested and optimized using all the four algorithms. The experiment results show that the genetic algorithm has better performance than the others in terms of solution quality, especially the deviation of results from multiple trials, and computation time.

Keywords: Genetic algorithm, Particle swarm, Ant colony, Shuffled frog leaping, Optimization

1. Introduction

Meta-heuristic algorithms are problem-independent algorithmic frameworks that provide a set of guidelines or strategies to search near-optimal solutions. Most meta-heuristic algorithms are inspired by the processes of nature biological evolution and/or the social behavior of species. To mimic the learning, adaptation, and evolution behavior of species, various meta-heuristic algorithms have been developed. Some of these algorithms focus on modifying and improving a single candidate solution, such as simulated annealing, tabu search, and iterated local search. While others are population-based, i.e., multiple candidate solutions are maintained and improved during the search. In this research, four population-based meta-heuristic algorithms, genetic algorithm (GA), particle swarm optimization (PSO), ant colony optimization (ACO), and shuffled frog leaping algorithm (SFLA), are applied to a printed circuit board (PCB) placement problem and their performances are compared and analyzed.

* Corresponding author. Tel.: +1-607-777-5935 ; fax: +1-607-777-4094.
E-mail address: yoons@binghamton.edu

10.1016/j.promfg.2018.10.112

The PCB placement procedure, where hundreds or thousands of electronic components are placed on a PCB by surface mounters, is a critical part of PCB assembly. A surface mounter, which is also known as a surface mount technology placement machine, has three elementary parts: a feeder bank, a PCB placement table, and a moving system that can pick up components from the feeder bank and place them onto the PCB. To optimize this pick-and-place process, at least two decisions must be made: 1) how to arrange component feeders in the feeder bank, and 2) how to decide placement sequences of components. In this research, a gantry-type surface mounter is used to place components, which has a movable gantry, an unmovable feeder bank, and a stationary PCB table during placement. Multiple identical PCBs are placed by this surface mounter one by one. The placement process of a PCB can be described as follows. When a PCB is delivered by a conveyer into the machine, the gantry moves from its last placement position of the previous PCB to the feeder bank, and starts the placement process of the PCB. First, the gantry head, which have multiple nozzles, will rotate one step at a time to pick up components. Then, the gantry head needs to move to a camera, whose position is fixed, to scan and check quality. Finally, the gantry head leaves the camera and places the components onto the PCB. In the next cycle, the gantry head will return to the feeder bank and repeats the pick-and-place process until all the required components are placed on the PCB. It is assumed that the pickup sequences of components in a cycle are the same as their placement sequences. Because the camera scanning process is fast, the scanning time is ignored.

The objective of this research is to minimize the total time of placing one PCB, which is called cycle time. To simplify the model as a generalized problem, it is assumed that the gantry moving speed is a constant, and the rotation speed of the gantry head from one component to the next is always faster than the gantry's moving speed. Therefore, the feeder arrangement and the component pick-and-place sequence decisions are converted to a quadratic assignment problem (QAP) and a capacitated vehicle routing problem (CVRP), respectively, both of which are well-known NP-hard problems. A mathematical model of this integrated problem is developed, which can be solved by constraint programming using CPLEX. However, the computational time of exact algorithms on this pick-and-place optimization problem is extremely long when hundreds of components are required to be placed. Therefore, four population-based meta-heuristic algorithms are developed. Their performances are compared and analyzed in the numerical experiments.

The remaining part of this paper is organized as follows. Section 2 reviews the applications of GA, PSO, ACO and SFLA to the pick-and-place optimization problems and to other related integer programs. Section 3 models the pick-and-place problem mathematically. Section 4 explains the four meta-heuristic algorithms in details. Section 5 summarizes the comparison of algorithm performances. Finally, conclusions are provided in Section 6.

2. Literature review

Table 1. Number of research related to the four algorithms from 2008 to 2018

Algorithms	Year of first work	Total number	PNP in PCB placement	CVRP	QAP
GA	1970s	20,800	2	38	20
PSO	1995 [3]	16,700	0	11	11
ACO	1992 [2]	5,940	0	10	10
SFLA	2003 [5]	544	1	0	0

Population-based meta-heuristic algorithms are a class of approaches that search near-optimal solutions by maintaining a set of candidate solutions and using population characteristics to guide the search iteratively. This class of algorithms includes (but is not restricted to) GA, PSO, ACO and SFLA. Among these four algorithms, GA [7, 6] and SFLA [8] have been applied in the pick-and-place (PNP) optimization of PCB placement processes and have shown their computational efficiency in searching high-quality solutions. To enhance the performance of algorithms, some commonly used methods are hybridizing the original search strategies with other methods to improve population quality, for example, clustering [7] and greedy algorithms [6], or introducing some new operators to increase search efficiency [8]. PSO and ACO are selected in this research because they have been commonly used in the CVRP and QAP optimizations. Table 1 shows a literature search result by using Google Scholar to find the articles with titles that have the algorithms' names and/or the problems' key words since 2008. This result indicates that GA and PSO are the most widely used methods among these four algorithms, and there is limited research in applying PSO and ACO in the pick-and-place optimization problem.

3. Mathematical model of the pick-and-place problem

In this section, a mathematical model is formulated for the pick-and-place optimization problem. All the notations used in the model are explained in Table 2. In the model, the cardinality of a set I is represented as $|I|$.

Table 2. List of notations used in this research

Categories	Notations	Explanations
Subscripts	i	Placement location on a PCB
	j	Component type
	k	Feeder slot
Parameters	D_i^{CL}	Distances from the camera to each placement location i
	D_{k_1,k_2}^{FF}	Distances between two feeder slots k_1 and k_2
	D_k^{FC}	Distances from each feeder slot k to the camera
	D_{i_1,i_2}^{LL}	Distances between two placement locations on a PCB
	D_i^{LF}	Distances from each placement location i to the center of the feeder bank
	H	Total number of nozzles on the gantry head
	N	Total number of cycles to place all components on a PCB
Sets	I	Set of all placement locations on a PCB
	I_j	Set of all placement locations that require component type j, $I_j \subseteq I$
	J	Set of component types that are required to be placed
	K	Set of all feeder slots in the feeder bank
Decision variable	u_i	Placement sequence of location i in a cycle
	v_{jk}	1-if component type j is loaded into feeder slot k, 0-otherwise
	w_i	1-if location i is the first to be placed in a cycle, 0-otherwise
	x_{i_1,i_2}	1-if location i_2 is placed directly after location i_1 in a cycle, 0-otherwise
	y_{i_1,i_2}	1-if locations i_1 and i_2 are placed in the same cycle, 0-otherwise
	z_i	1-if location i is the last to be placed in a cycle, 0-otherwise

$$\min Distance = \sum_{i\in I} D_i^{LF} z_i + \sum_{i\in I} D_i^{CL} w_i + \sum_{i_1\in I}\sum_{i_2\in I, i_2\neq i_1} D_{i_1,i_2}^{LL} x_{i_1,i_2} + \sum_{k_1\in K}\sum_{k_2\in K} D_{k_1,k_2}^{FF}\left(\sum_{j_1\in J}\sum_{j_2\in J}\left(\sum_{i_1\in I_{j_1}}\sum_{i_2\in I_{j_2}, i_2\neq i_1} x_{i_1,i_2}\right) v_{j_1,k_1} v_{j_2,k_2}\right) + \sum_{j\in J}\sum_{k\in K}\left(D_j^{FC} v_{jk}\sum_{i\in I_j} z_i\right) \quad (1)$$

Subject to

$$1 + \sum_{i_2\in I, i_2\neq i_1} y_{i_1,i_2} \leq H, \forall i_1 \in I \quad (2)$$

$$y_{i_1,i_2} + y_{i_2,i_3} - y_{i_1,i_3} \leq 1, \forall i_1 \neq i_2 \neq i_3, i_1, i_2, i_3 \in I \quad (3)$$

$$x_{i_1,i_2} - y_{i_1,i_2} \leq 0, \forall i_1 \neq i_2, i_1, i_2 \in I \quad (4)$$

$$w_{i_2} + \sum_{i_1\in I, i_1\neq i_2} x_{i_1,i_2} = 1, \forall i_2 \in I \quad (5)$$

$$z_{i_1} + \sum_{i_2\in I, i_2\neq i_1} x_{i_1,i_2} = 1, \forall i_1 \in I \quad (6)$$

$$x_{i_1,i_2} + x_{i_2,i_1} \leq 1, \forall i_1 < i_2, i_1, i_2 \in I \quad (7)$$

$$u_{i_1} - u_{i_2} + |I| \times x_{i_1,i_2} \leq |I| - 1, \forall i_1 \neq i_2, i_1, i_2 \in I \quad (8)$$

$$\sum_{j\in J} v_{jk} \leq 1, \forall k \in K \quad (9)$$

$$\sum_{k\in K} v_{jk} = 1, \forall j \in J \tag{10}$$

Because the pick-and-place problem integrates CVRP and QAP, the mathematical model in this section clearly shows this integration feature: the first three parts in Equation (1) plus Equations (2-8) can be treated as a CVRP model, and the remaining parts of this mathematical model is a QAP model. The CVRP part and the QAP part are connected in the objective function. The objective function values, *Distance*, are not known until the CVRP and QAP parts are both solved. The PCB placement cycle time should be calculated by

$$CycleTime = \frac{Distance}{GantrySpeed} + NozzlePickTime + NozzlePlaceTime. \tag{11}$$

Because the gantry's moving speed is assumed to be a constant, and the vertical nozzle pick-and-place time is also a constant given a specific value of $|I|$, minimizing the cycle time is equivalent to minimizing the gantry's total moving distance.

The objective function in Equation (1) consists of five parts. The first part sums all the distance from the last placement locations in each cycle to the feeder bank. The second part sums all the distance from the camera to the first placement locations in each cycle. The third part represents the total distance between any two consecutive placements. The fourth part counts all the distances when the gantry moves over the feeder bank. Note that the pickup sequences are the same as the placement sequences. The last part is a summation of all the moving distances from the last pickup locations of each cycle on the feeder bank to the camera. Equation (2) is a gantry head capacity constraint. Each cycle cannot place more than H components. Equation (3) defines that if i_1 and i_2, i_2 and i_3 are placed in the same cycle, then i_1 and i_3 must be placed in the same cycle. Equation (4) tells that if i_2 is placed right after i_1 in a cycle, then they must be placed in the same cycle. Equation (5) depicts the first placement location of a cycle as a location that has no preceding placements in the same cycle. Similarly, Equation (6) depicts the last placement location of a cycle as a location that has no succeeding placements in the same cycle. Equation (7) limits that each location is visited only once. Equation (8) is a subtour elimination constraint. Equation (9) means that each feeder can load at most one type of component. Equation (10) forces each component feeder to use only one feeder slot.

Although this mathematical model has a non-linear objective function, it can be solved by constraint programming in CPLEX, which is a popular optimization software package. Although a global optimum can be reached by using the branch-and-bound method in CPLEX, the computational burden is extremely heavy. For example, in a pick-and-place problem of 12 placement locations, 3 component types, 3 nozzles and 8 feeder slots, the total number of constraints and decision variables are 1,697 and 324, respectively. To prove the optimality of solutions, around 10 million branches should be explored, which results in a computational time of more than 91 hours. In real-world practice, the total number of components to be placed on a PCB by one surface mounter is typically from 100 to 500. Because the problem complexity increases exponentially with the increase of problem size, it is unrealistic to find the global optimum within reasonable time. Therefore, meta-heuristic algorithms are applied to find near-optimal solutions.

4. Algorithms explanation

In this section, four population-based meta-heuristic algorithms are developed to solve the pick-and-place optimization problem. For the implementation pseudocodes of these algorithms, please refer to [4].

4.1. Encoding

Both the feeder arrangements and pick-and-place sequences are encoded by integers in the algorithms. Each candidate solution has two parts to represent the pick-and-place sequences and the feeder arrangement separately, as shown in Table 3. The pick-and-place sequences part has $N \times H$ capacities in total, where N and H represent the number of cycles and the number of nozzles, respectively. Note that the total number of placements $|I|$ may be less than $N \times H$, because the total number of placements is not always an integer multiplier of the total number of nozzles. In this case, there should be $N \times H - |I|$ times that a nozzle is skipped during the placement process. In the encoding of pick-and-place sequences, "-1" is used to indicated the skipped nozzle, while the other cells have integer numbers to represent placement location IDs. Each location ID must appear once and only once in a solution. In the feeder arrangement part, a feeder slot can have no component feeders, which is denoted by “-1”, or have a feeder of component type $j \in J$. Each component type must appear once and only once.

Table 3. Candidate solution format in the four algorithms

	Pick-and-place sequences					Feeder arrangement			
Sequence indexes	1	2	...	$N \times H$	Slot indexes	1	2	...	$\|K\|$
Values	-1 or $i \in I$	-1 or $i \in I$	...	-1 or $i \in I$	Values	-1 or $j \in J$	-1 or $j \in J$	...	-1 or $j \in J$

4.2. Operators for integer solutions

Some meta-heuristic methods, such as PSO and SFLA, were originally developed on a continuous definition domain. In the classical algorithms, addition and subtraction operators may be used to search for new solutions. Because the solutions of the pick-and-place optimization problems are encoded as a permutation of all placement locations and feeders, the traditional addition and subtraction operators may yield infeasible solutions. Therefore, the operators defined in [1] are applied to generate new feasible solutions.

In this research, an operator *step* is defined, which is applied to a solution during one time move to get another solution. For a permutation of integers, solution P, a *step* with length L is defined by

$$step = ((a_1, b_1), ..., (a_r, b_r), ..., (a_L, b_L)), a_r \in P, b_r \in P, r = 1, ..., L, \tag{12}$$

which represents a series of exchanges: “exchange nodes a_1 and b_1, then exchange a_2 and b_2, ... , and at last exchange nodes a_L and b_L.” Two parentheses are used in Equation (12) because *step* is a list of pairs.

The addition operator $\oplus$ represents a move from the current position to a new position in the search space. The new position $P' = P \oplus step$ is found by applying the series of exchanges to P. In the example $P = (5, 4, 3, 2, 1)$, $step = ((1, 2), (2, 3))$, the first exchange is to swap 1 and $2 \in P$ and the second exchange is to swap 2 and $3 \in P$. So the final result of $P \oplus step$ is $(5, 4, 2, 1, 3)$, with an intermediate result of $(5, 4, 3, 1, 2)$. Besides the addition operator between a position and a step, it is necessary to define another addition operator between two steps, which is denoted by $\uplus$. Let $step1$ and $step2$ be two steps, then $step1 \uplus step2$ is defined as concatenating nodes of these two steps.

The subtraction of two solutions, $P_2 \ominus P_1$, is defined as the move *step* found by a given algorithm, so that $P_1 \oplus step = P_2$. For example, $(5, 4, 2, 1, 3) \ominus (5, 4, 3, 2, 1) = ((2, 3), (1, 3))$. Finally, a multiplication operator $\otimes$ is defined for a coefficient times a step. Let c be a real number between 0 and 2. If $c = 0$, then $c \otimes step = \emptyset$. If $0 < c \leq 1$, *step* is truncated from length L to length $\lceil cL \rceil$ by randomly removing elements. If $1 < c \leq 2$, $c \otimes step$ is defined as $step \uplus ((c - 1) \otimes step)$.

4.3. Explanation of four meta-heuristic algorithms

GAs are inspired by the process of natural selection. The crossover operator is applied to produce offspring by exchanging information between parent chromosomes. Let $Parent1$ and $Parent2$ be the chromosomes of two selected parents, $r \sim U(0, 1)$ be a random number following a uniform distribution between 0 and 1. The crossover process is defined in Equation (13). On the other hand, the mutation operator is needed to introduce new genes to the evolutionary process. In this pick-and-place optimization problem, a mutation is performed by applying a randomly generated *step* to a chromosome. Four parameters affect the performance of GAs: population size, number of generations, crossover rate, and mutation rate.

$$\begin{aligned} Offspring1 &= Parent1 \oplus r \otimes (Parent2 \ominus Parent1) \\ Offspring2 &= Parent2 \oplus r \otimes (Parent1 \ominus Parent2) \end{aligned} \tag{13}$$

PSO is inspired by the social behavior of a flock of migrating birds to an unknown destination. In PSO, the birds (called particles) in the population (called swarm) evolve their social behaviors by learning from their own best-known position in the search space as well as the entire swarm's best-known position. The concept of velocity in the PSO is analogous to the variable *step*. In tth iteration, the new position of a particle is updated by

$$\text{new position } P_t = \text{current position } P_{t-1} \oplus \text{new velocity } V_t. \tag{14}$$

Let P^p and P^g be the best-known positions of the particle and the entire swarm, respectively. Then, the new velocity of the particle is given by

$$V_t = [\omega \otimes V_{t-1}] \uplus [r \times c_1 \otimes (P^p \ominus P_{t-1})] \uplus [r \times c_2 \otimes (P^g \ominus P_{t-1})], \tag{15}$$

where ω is an inertia weight employed to control the impact of the previous history of velocities on the new velocity, and c_1 and c_2 are learning factors ($c_1 = c_2 = 2$ in this research). The second and third parts in Equation (15) represent private thinking and social collaboration of particles, respectively. In this PSO algorithm, the main parameters are: population size, number of iterations, and ω. In the experiments of this research, the controllable inertia weight factor ω was defined as a variable that decreases linearly with the increase of generations. At the nth generation, the value of ω was given by

$$\omega = \omega_N + (\omega_0 - \omega_N) \times \tfrac{N-n}{N-1}, \tag{16}$$

where N represents the maximum number of generations, such that the value of ω is in the range ω_N to ω_0.

The ACO is inspired by the behavior of ants seeking the shortest route between their colony and a source of food. This search is done using pheromone trails, which ants deposit whenever they travel, as a form of indirect communication [4]. In the ACO algorithm, each candidate solution represents an ant. The process starts by generating M random ants. The pheromone concentration associated with each path (i, j) at iteration t is changed as follows.

$$\tau_{ij,t} = (1-\rho)\tau_{ij,t-1} + \Delta\tau_{ij}, \tag{17}$$

where ρ is the pheromone evaporation rate and $\Delta\tau(i, j)$ is defined by

$$\Delta\tau_{ij} = \sum_{m=1}^{M} \begin{cases} Q/Distance_m & \text{if path } (i, j) \text{ is chosen by ant } m \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

where Q is a constant and $Distance_m$ is the value of the objective function calculated for ant m.

Once the pheromone is updated after an iteration, the next iteration starts by changing the ants' routes. The probability of choosing path (i, j) for an ant at iteration t is as follows.

$$p_{ij,t} = \frac{(\tau_{ij,t})^\alpha (\eta_{ij})^\beta}{\sum_{j' \in allowed} (\tau_{ij',t})^\alpha (\eta_{ij'})^\beta} \tag{19}$$

where η_{ij} is a heuristic factor to indicate the desirability of path (i, j), which is typically $1/d_{ij}$, and d_{ij} is the distance of path (i, j). α and β control the relative importance of pheromone concentration versus the heuristic factor. The main parameters used in ASO include population size M, number of iterations, exponents α and β, pheromone evaporation rate ρ, and pheromone reward factor Q.

In SFLA, the population consists of F frogs that is divided into m subsets, which are referred to as memeplexes. Each memeplex consists of n frogs satisfying $F = mn$. After the initial population of F frogs is generated randomly, the frogs are sorted in a descending order according to their fitness, i.e., rank 1 frog is the global best, which is denoted as P^g. Next, the entire population is partitioned into m memeplexes following the rule that ith memeplex consists of all the frogs ranked $(j-1)m+i, \forall j = 1, ..., n$. Within each memeplex, q frogs are selected to construct a submemeplex based on a probability distribution respecting their ranks of fitness j' in the memeplex, as shown in Equation (20).

$$p_{j'} = \frac{2(n+1-j')}{n(n+1)}, j' = 1, ..., n \tag{20}$$

In a submemeplex, the frogs with the best and the worst fitness are identified as P^b and P^w, respectively. Then, a movement is applied to create new frogs, as shown in Equation (21). If P_{t1} has higher fitness than P^w_{t-1}, replace P^w_{t-1} by P_{t1}. Otherwise, check if P_{t2} has higher fitness than P^w_{t-1}. If the fitness of P_{t2} is higher than P^w_{t-1}, replace P^w_{t-1} by

P_{t2}. Otherwise, a random solution is generated to replace the worst frog P^w_{t-1}. After the replacement, the frogs in the submemeplex are put back into the memeplex and one iteration of local exploration is completed. From the same memeplex, another submemeplex can be constructed and repeats the evolutionary process. Thus, the total number of submemeplexes constructed from one memeplex represents the number of iterations for local exploration, s. After all memeplexes have finished local exploration, all the F frogs are mixed and reordered in a descending order of their fitness. Then, the frogs are partitioned into m memeplexes again. In other words, before one shuffling iteration, the local exploration, which is the operation of updating the local worst frogs in every submemeplex, is performed $m \cdot s$ times. Accordingly, the main parameters of SFLA are: population size F, number of memeplex m, number of frogs in a submemeplex q, number of local iterations s, and number of shuffling iterations g.

$$\begin{aligned}\text{new frog } P_{t1} &= P^w_{t-1} \oplus r \otimes (P^b_{t-1} \ominus P^w_{t-1}),\\ \text{new frog } P_{t2} &= P^w_{t-1} \oplus r \otimes (P^g_{t-1} \ominus P^w_{t-1}).\end{aligned} \tag{21}$$

5. Experiment Results

All the four meta-heuristic algorithms explained above have been coded using MATLAB R2013a on a computer with 3.4GHz CPU and 16G RAM. The performances of the four algorithms are compared on 10 PCB samples from industry. All the PCBs are placed by a surface mounter with 16 nozzles and 30 feeder slots. The population size of all the four algorithms is set as 120 and the termination criteria is defined as meeting either of the following two conditions: 1) algorithm running time reaches 10 minutes; 2) the number of iterations reaches 2,000. Because the algorithms use random numbers, each algorithm is executed 20 times on one PCB sample.

Table 4. Parameter settings of algorithms.

	GA		PSO		ACO		SFLA
Crossover rate	***0.85***, 0.90, 0.95	ω_0	***1.2***, 1.6, 2.0	ρ	0.1, ***0.4***	m	12, 15, ***20***
Mutation rate	0.05, 0.10, ***0.15***	ω_N	0.2, ***0.4***, 0.6	Q	2, ***10***	q	***2***, 4, 6
				α	***0.5***, 1.0	s	5, ***10***
				β	***2.0***, 2.5		

The performance of meta-heuristic algorithms largely depends on the parameter settings. Therefore, design of experiments (DOE) techniques were used to select the most appropriate values for the algorithm parameters. The initial settings of the algorithm parameters were established referring to the values previously reported in the literature [7, 4]. Then, the parameter values were changed and full factorial experiments were conducted. Two PCB samples, one requiring 116 components and the other 424 components, were tested in the experiments. Under a combination of factor levels, an algorithm was run 20 replications on each PCB sample. In total, the results of 2,080 runs were monitored in terms of solution quality. All the levels of factors tested in the experiments are shown in Table 4. The set of parameters that yielded the best average result was selected as the final parameter setting for an algorithm when testing all the ten samples. The selected parameter values are indicated in italic and bold in Table 4.

Table 5. Results summary of four algorithms from 20 trials on each PCB sample (meter)

	GA			PSO			ACO			SFLA		
Sample	Avg.	Std.	Min, Max	Avg.	Std.	Min, Max	Avg.	Std.	Min, Max	Avg.	Std.	Min, Max
1	11.22	0.17	10.96, 11.53	12.60	0.22	11.95, 12.92	12.53	0.14	12.22, 12.79	12.97	0.24	12.66, 13.59
2	18.52	0.30	18.05, 19.28	20.77	0.24	20.34, 21.07	19.21	0.23	18.81, 19.56	21.21	0.42	20.47, 22.01
3	28.42	0.25	27.98, 28.95	32.33	0.58	31.46, 33.42	28.83	0.25	28.23, 29.27	32.90	0.50	31.68, 33.89
4	24.57	0.27	24.14, 25.03	27.59	0.46	26.89, 28.30	22.78	0.30	22.12, 23.26	27.63	0.41	26.98, 28.32
5	24.55	0.32	23.92, 25.02	27.72	0.37	27.18, 28.36	25.60	0.67	24.38, 26.70	28.04	0.43	27.38, 28.97
6	27.19	0.35	26.65, 27.95	31.82	0.44	31.09, 32.76	25.65	0.24	25.24, 25.99	32.45	0.54	31.41, 33.52
7	28.54	0.21	28.13, 28.95	27.15	0.21	26.69, 27.57	27.47	0.50	26.54, 28.37	25.58	0.22	25.10, 25.90
8	40.32	0.31	39.56, 40.90	46.53	0.79	44.67, 47.69	40.90	0.70	39.54, 42.17	46.72	0.55	45.49, 47.50
9	36.96	0.30	36.38, 37.46	43.48	0.60	42.48, 44.47	39.81	1.13	37.39, 42.07	43.60	0.73	42.16, 44.84
10	37.97	0.31	37.26, 38.52	44.16	0.48	43.19, 45.13	35.56	0.80	33.45, 37.23	44.25	0.61	42.73, 45.17
Mean	27.83	0.28	27.30, 28.36	31.41	0.44	30.59, 32.17	27.83	0.50	26.79, 28.74	31.53	0.47	30.61, 32.37

The results found by the four meta-heuristic algorithms on 10 PCB samples are summarized in Table 5. The average results obtained in all trials by each algorithm were compared, as well as the best and worst results and standard deviations. It is noted that, on average, GA and ACO yielded solutions with the same quality, which were better than the other two algorithms. ACO yields the minimal cycle time from the 20 trials, and GA had a relatively small standard deviation among the four algorithms.

The average computational times for each trial and for one population-level evolutionary iteration are shown in Table 6. The average processing time of one iteration is used to measure the speed of each algorithm because many instances stop after running 10 minutes because of the stopping criterion. According to the computational time summary in Table 6, GA is the fastest of all the four algorithms. Also, as the number of placement locations I increased, the processing time of one iteration increased linearly on all algorithms (p-value < 0.0001).

Table 6. Computational time summary (second)

	Specifications		Avg. CPU time per trial				Avg. CPU time per iteration			
Sample	$\|I\|$	$\|J\|$	GA	PSO	ACO	SFLA	GA	PSO	ACO	SFLA
1	116	22	166	227	600	494	0.08	0.11	4.78	0.25
2	180	20	327	401	600	600	0.16	0.20	10.78	0.31
3	244	23	436	532	600	600	0.22	0.27	19.37	0.41
4	264	12	422	539	600	600	0.21	0.27	23.32	0.43
5	268	17	410	545	600	600	0.20	0.27	25.06	0.45
6	308	20	470	600	600	600	0.23	0.32	33.05	0.54
7	320	18	490	600	600	600	0.24	0.33	35.85	0.50
8	388	27	581	600	600	600	0.30	0.42	49.72	0.68
9	424	21	593	600	600	600	0.30	0.48	56.29	0.73
10	424	22	600	600	600	600	0.33	0.46	56.34	0.73
	Mean		450	525	600	590	0.23	0.31	31.46	0.50

6. Conclusion

A pick-and-place optimization problem of a surface mounter in PCB assembly environment is investigated in this research. A mathematical model is formulated as an integrated problem of the capacitated vehicle routing problem and the quadratic assignment problem. Four population-based meta-heuristic algorithms, GA, PSO, ACO and SFLA, are applied to solve this pick-and-place optimization problem. A brief description of each method is presented. Ten different PCB samples are tested and optimized using all the four algorithms. Finally, the GA and ACO approach are found to have better performance than the other two algorithms in terms of solution quality. In addition, GA produces results with a relatively low deviation by each run, and it has fast searching speed. To apply PSO and SFLA to solve this pick-and-place problem, algorithm structures should be well designed to improve their performance.

References

[1] Clerc, M., 2004. Discrete Particle Swarm Optimization, illustrated by the Traveling Salesman Problem. Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 219–239.

[2] Dorigo, M., 1992. Optimization, learning and natural algorithms. PhD Thesis, Politecnico di Milano .

[3] Eberhart, R., Kennedy, J., 1995. A new optimizer using particle swarm theory, in: Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on, IEEE. pp. 39–43.

[4] Elbeltagi, E., Hegazy, T., Grierson, D., 2005. Comparison among five evolutionary-based optimization algorithms. Advanced engineering informatics 19, 43–53.

[5] Eusuff, M.M., Lansey, K.E., 2003. Optimization of water distribution network design using the shuffled frog leaping algorithm. Journal of Water Resources planning and management 129, 210–225.

[6] Ho, W., Ji, P., Dey, P.K., 2008. Optimization of pcb component placements for the collect-and-place machines. The International Journal of Advanced Manufacturing Technology 37, 828–836.

[7] Kulak, O., Yilmaz, I.O., Günther, H.O., 2007. Pcb assembly scheduling for collect-and-place machines using genetic algorithms. International Journal of Production Research 45, 3949–3969.

[8] Zhu, G.Y., Zhang, W.B., 2014. An improved shuffled frog-leaping algorithm to optimize component pick-and-place sequencing optimization problem. Expert Systems with Applications 41, 6818–6829.