Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2014), *28*, 429–437.
© Cambridge University Press 2014 0890-0604/14 $25.00
doi:10.1017/S0890060413000632

Probability increment based swarm optimization for combinatorial optimization with application to printed circuit board assembly

KEHAN ZENG,[1,2] ZHEN TAN,[1] MINGCHUI DONG,[1] AND PING YANG[3]

[1]Faculty of Science and Technology, University of Macau, Avenida Padre Tomas Pereira, Taipa, Macau Sar, China
[2]Department of Computer Science, Huizhou University, Huizhou, Guangdong, China
[3]School of Mechanical Engineering, Jiangsu University, Zhenjiang, Jiangsu, China

(Received December 29, 2012; Accepted November 7, 2013)

Abstract

A novel swarm intelligence approach for combinatorial optimization is proposed, which we call probability increment based swarm optimization (PIBSO). The population evolution mechanism of PIBSO is depicted. Each state in search space has a probability to be chosen. The rule of increasing the probabilities of states is established. Incremental factor is proposed to update probability of a state, and its value is determined by the fitness of the state. It lets the states with better fitness have higher probabilities. Usual roulette wheel selection is employed to select states. Population evolution is impelled by roulette wheel selection and state probability updating. The most distinctive feature of PIBSO is because roulette wheel selection and probability updating produce a trade-off between global and local search; when PIBSO is applied to solve the printed circuit board assembly optimization problem (PCBAOP), it performs superiorly over existing genetic algorithm and adaptive particle swarm optimization on length of tour and CPU running time, respectively. The reason for having such advantages is analyzed in detail. The success of PCBAOP application verifies the effectiveness and efficiency of PIBSO and shows that it is a good method for combinatorial optimization in engineering.

Keywords: Combinatorial Optimization; Population Evolution; Printed Circuit Board Assembly; State Probability; Swarm Intelligence

1. INTRODUCTION

Intelligent technology has increasingly influenced product design and manufacturing (Jin & Li, 2007; Yang & Zeng, 2009; Shea et al., 2010; Maher & Fisher, 2012). Bionic intelligence, including evolutionary computation, artificial neural networks, swarm intelligence, and so on, is an indispensable part of computational intelligence (Eberhart & Shi, 2007) and designed to mimic one or more aspects of biological intelligence. Swarm intelligence was first introduced by Beni and Wang (1988, 1989) in the context of cellular robotic systems. The extended definition of swarm intelligence includes any attempt to design algorithms or distributed problem-solving devices inspired by the collective behavior of social insect colonies and other animal societies (Bonabeau et al., 1999). With developed theory and its related applications, swarm intelligence has become an important branch of computational intelligence.

Combinatorial optimization problems extensively exist in engineering, such as optimal engineering design (Jin et al., 2005, 2009), best layout of architecture and urban planning (Koenig & Schneider, 2012), optimum synthesis of kinematic chains (Lipson, 2008) and selective disassembly optimization (Wang et al., 2003). Usually, combinatorial optimization problems are formulated as integer programs (Korte & Vygen, 2008).

The printed circuit board assembly optimization problem (PCBAOP) is a typical combinatorial optimization problem in electronic manufacturing industry. Approaches of computational intelligence become increasingly important for solving this problem. Khoo and Ng (1998) proposed a genetic algorithm (GA) for semiautomatic assembly system to assist process engineers to get the near-optimal PCB component placement sequence. Loh et al. (2001) implemented GA on a Quad IIIc insertion machine to optimize the production process. Najera and Brizuela (2005) adapted GA to solve the assembly problem in a pick-and-place machine, and the experimental

Reprint requests to: Kehan Zeng, Room 106, Block 3, University of Macau, Avenida Padre Tomas Pereira, Taipa, Macau Sar, China. E-mail: kehanzeng1980@gmail.com

https://doi.org/10.1017/S0890060413000632 Published online by Cambridge University Press

results showed that GA was better than the typewriter and greedy search mixed algorithm. Chen and Lin (2007) proposed an adaptive particle swarm optimization (APSO) approach to optimize the sequence of component placements on a PCB and the assignment of component types to feeders simultaneously for a pick-and-place machine with multiple heads.

In this paper, a novel swarm intelligence algorithm for combinatorial optimization, named probability increment based swarm optimization (PIBSO), is proposed. In general, for a combinatorial optimization problem, the states in search space are finite. The probabilities of states to be selected are taken into account. The probability increment of a state is proportional to the fitness of the state. Some calculations are defined, and the population evolution mechanism is depicted. In addition, PIBSO is applied to solve PCBAOP. The mathematical model for this problem is formulated, and two types of search states are defined. The numerical results show that PIBSO has much advantage on search capability and search time over GA and APSO.

2. PIBSO

2.1. Initialization

In the initialization of PIBSO, a population of units is generated randomly or under a rule. For a combinatorial optimization problem, the n-dimensional search space is denoted by $\boldsymbol{P}^n$. A state in $\boldsymbol{P}^n$ is denoted by $\boldsymbol{x}_i = [x_{i1}, x_{i2}, \ldots, x_{in}]^{\mathrm{T}} \in \boldsymbol{P}^n$, where i is the sequence number of state. For the kth variable of $\boldsymbol{x}_i$, x_{ik}, the search space is $\boldsymbol{P}_k \subset \boldsymbol{P}^n$. A state in $\boldsymbol{P}_k$ is denoted by s_{km}, and the number of s_{km} is denoted by Num_k. A unit vector, denoted by $\boldsymbol{u}_j = [u_{j1}, u_{j2}, \ldots, u_{jn}]^{\mathrm{T}}$, is a $\boldsymbol{x}_i$, where j is the sequence number of the unit in population. In $\boldsymbol{P}_k$, each s_{km} has a probability to be selected as u_{jk}, and this probability is denoted by p_{km}. Here, p_{km} is initialized in [0, 1] randomly or under a rule and is not necessary to satisfy $\sum_{m=1}^{Num_k} p_{km} = 1$, and $\boldsymbol{u}_j$ is selected from $\boldsymbol{x}_i$. Apparently, the globally optimal $\boldsymbol{u}_j$ is the best combination of s_{km}. The population scale is set to a suited number according to the concrete optimization problem.

2.2. Fitness

The fitness of unit $\boldsymbol{u}_i$, denoted by fit_i, evaluates the suitability of $\boldsymbol{u}_i$ for the objective of optimization, and $\boldsymbol{u}_i$, with higher fitness value approaches closer to the global optimum than others with lower fitness values do. The definition of fitness depends on the concrete optimization problem.

2.3. Unit update

Each fit_i is normalized by

$$norfit_i = \frac{fit_i - fit_{\min}}{fit_{\max} - fit_{\min}}, \tag{1}$$

where $fit_{\max}$ and $fit_{\min}$ are the maximum, and minimum, respectively, among fit_i.

The incremental factor related to $\boldsymbol{u}_i$ is denoted by $w_i \in (0, 1)$ and is calculated by

$$w_i = \frac{dr \times norfit_i}{n}, \tag{2}$$

where dr $\in$(0, 1] is the incremental ratio. Then, for the s_{km} included in $\boldsymbol{u}_i$, that is, u_{jk}, its p_{km} is updated by

$$p_{km}(t+1) = p_{km}(t) \times (1 + w_i), \tag{3}$$

where t denotes the population generation.

If $p_{kq}\ (t+1) > 1$ for s_{kq}, where $1 < q < Num_k$, then $p_{km}\ (t+1)$ $(m \neq \mathrm{q})$ is calculated by

$$p_{km}(t+1) = p_{km}(t) - [p_{kq}(t+1) - 1] \tag{4}$$

and then set as

$$p_{kq}(t+1) = 1. \tag{5}$$

If $p_{kq}\ (t+1) < 0$ for s_{kq}, where $1 < q < Num_k$, then set

$$p_{kq}(t+1) = 0. \tag{6}$$

When $p_{km}(t)$ has been updated to $p_{km}\ (t+1)$, each $u_{ik}\ (t)$ is updated by selecting a s_{km} from $\boldsymbol{P}_k$ to be $u_{ik}\ (t+1)$, based on $p_{km}(t+1)$ and a selection rule defined by users. Usually roulette wheel selection is adopted. Finally, $u_{ik}(t+1)$ forms $u_i(t+1)$.

2.4. Procedure of PIBSO

The procedure of PIBSO is described by the following:

Step 1. Initialize a population of $\boldsymbol{u}_i$ and p_{km} with a suitable scale.

Step 2. Evaluate the fitness fit_i of each $\boldsymbol{u}_i$ and normalize fit_i by Eq. (1).

Step 3. For each $\boldsymbol{u}_i$, find incremental factor w_i by Eq. (2) and update p_{km} of u_{ik} by Eqs. (3), (4), (5), and (6).

Step 4. Update u_{ik} and form new $\boldsymbol{u}_i$ by selecting s_{km} based on p_{km}.

Step 5. The updated $\boldsymbol{u}_i$ forms new generation.

Step 6. Go back to Step 2 until the stop criterion is met.

2.5. Population evolution mechanism of PIBSO

In general, evolution of a $\boldsymbol{u}_i$ means the combination of s_{km} gets better and fit_i increases. Accordingly, each u_{ik} evolves into a better s_{km}. The unit update mechanism indicates that a worse $\boldsymbol{u}_i$ with lower fit_i makes p_{km} of u_{ik} be increased less than a better $\boldsymbol{u}_i$ with higher fit_i does. Thus, a s_{km} selected by more better $\boldsymbol{u}_i$ has a higher probability of being selected. It means that the difference of p_{km} between various s_{km} (which

https://doi.org/10.1017/S0890060413000632 Published online by Cambridge University Press

are selected by bigger amounts of better $\boldsymbol{u}_i$) and s_{km} (which are selected by bigger amounts of worse $\boldsymbol{u}_i$) expands gradually with the increase of generation number. This leads to the situation in which $\boldsymbol{u}_i$ tends to select better s_{km} and approaches the global optimum. Therefore, Steps 3, 4, and 5 of PIBSO push population to reach convergence. In contrast, p_{km} is influenced by all $\boldsymbol{u}_i$, and conversely, p_{km} affects the quality of $\boldsymbol{u}_i$. Therefore, $\boldsymbol{u}_i$ interacts each other. In addition, the incremental ratio dr in Eq. (2) determines the value of incremental factor w_i and controls the incremental velocity of p_{km}.

3. PCBAOP

3.1. Problem statement

The typical sequential pick-and-place machine, shown schematically in Figure 1, consists of a movable placement head, a feeder carrier with slots, feeders, and a board holder. When a PCB has been transferred inside the machine by conveyor, the only movable mechanism, the placement head, moves from the starting point to a feeder to pick up a component. Then, the placement head travels to a prespecified position on the PCB to place the component. Next, the placement head travels to a feeder to pick up another component. This pick-and-place procedure is repeated until the assembly task has been completed. Eventually, when the last component has been placed, the placement head travels back to the starting point. Prior to production, components need to be stored in feeders and feeders need to be loaded on slots. In general, the placement head mounts only one component in a pick-and-place operation and moves directly between a feeder and a placement position. Moreover, a feeder stores only one type of components and a slot loads only one feeder.

During PCB production, the sequential pick-and-place machine completes two types of work: assigning feeders to slots and placing components onto the board in a sequence. To improve efficiency and cut down cost, the traveling time of the placement head, equivalently, the total length of the placement head traveling paths, need be reduced to the minimum. Consequently, optimization of PCB assembly is essentially a combinatorial optimization problem to find the shortest loop of the placement head.

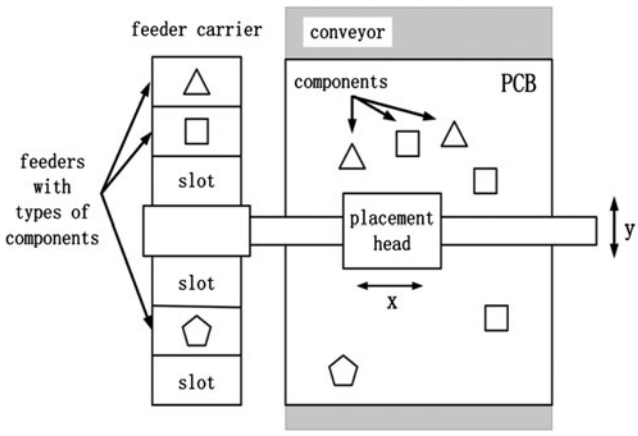

Fig. 1. The typical pick-and-place placement machine.

3.2. Mathematical model

According to Ho and Ji (2007), the integer program model for PCBAOP on the sequential pick-and-place machine can be formulated as minimize

$$z = \sum_{i=0}^{n} \sum_{j=1, j\neq i}^{n} \sum_{t=1}^{\mu} \sum_{l}^{\mu} (d_{il} + d_{lj}) \times x_{ij} \times y_{t_j l} + \sum_{i=1}^{n} d_{i0} \times x_{i0}, \quad (7)$$

subject to

$$\sum_{i=0}^{n} x_{ij} = 1, \quad \forall j = 0, 1, \ldots, n;\ i \neq j \quad (8)$$

$$\sum_{j=0}^{n} x_{ij} = 1, \quad \forall i = 0, 1, \ldots, n;\ i \neq j \quad (9)$$

$$u_i - u_j + n \times x_{ij} \leq n - 1, \quad \forall i, j = 1, 2, \ldots, n;\ i \neq j \quad (10)$$

$$\sum_{t=1}^{\mu} y_{tl} = 1, \quad \forall l = 1, 2, \ldots, \mu \quad (11)$$

$$\sum_{l=1}^{\mu} y_{tl} = 1, \quad \forall t = 1, 2, \ldots, \mu \quad (12)$$

where i, j are the components ($i, j = 0, 1, \ldots, n$); t is the component types ($t = 1, 2, \ldots, \mu$); l are feeders ($l = 1, 2, \ldots, \mu$); d_{0l} is the distance traveled from starting point to feeder l; d_{lj} is the distance traveled from feeder l to position of component j on PCB; d_{il} is the distance traveled from the position of component i to feeder l; d_{i0} is the distance traveled from the position of component i to starting point; u_i is the placement order of component i; $x_{ij} = 1$ if component i is placed immediately prior to component j and 0 otherwise; and $y_{t_j l} = 1$ if component j with component type t is stored in feeder l and 0 otherwise.

3.3. A PCB assembly example

The PCB assembly example PAT is used here to help understand the definitions and PIBSO application. Suppose in PAT, five components of three types will be mounted and four slots are on the feeder carrier. The distance between placement positions and slots, the distance from the starting point to slots, and the distance from the starting point to placement positions are shown in Table 1.

https://doi.org/10.1017/S0890060413000632 Published online by Cambridge University Press

Table 1. *The distances among placement positions, slots, and the starting point in PAT*

	Position 1 (Type 1)	Position 2 (Type 1)	Position 3 (Type 2)	Position 4 (Type 2)	Position 5 (Type 3)	Starting Point
Slot 1	3	7	6	16	12	1
Slot 2	5	4	3	9	7	2
Slot 3	8	3	5	4	2	3
Slot 4	12	5	10	3	6	4
Starting point	4	8	7	17	13	

Note: PAT, a printed circuit board assembly example.

3.4. Definitions

A **tour** of a placement head is a feasible loop of placement head travel, that is, the placement head travels from the starting point, picks and places all components repeatedly, and eventually returns to the starting point. In a tour, a path from a slot to a placement position is called a **placement path**, and a path from a placement position to a slot is called a **pick path**. Apparently, a tour is composed of placement paths, pick paths, and the path from the starting point to a slot as well as the path from a placement position to the starting point.

The **placement distance** is defined as the distance from a slot to a type of components, which is the sum of distances from the slot to placement positions on which components of the type are placed. For example, for PAT, according to Table 1, the placement distance from slot 1 to components of type 1 is the sum of the distance from slot 1 to position 1 and the distance from slot 1 to position 2, equal to 10. The **placement distance matrix (PDM)** is defined as the matrix in which each entry is a placement distance. Moreover, the rows correspond to the slots and the columns correspond to the types of components. As an example, the **PDM** of PAT is depicted as

$$\begin{bmatrix} 10 & 22 & 12 \\ 9 & 12 & 7 \\ 11 & 9 & 2 \\ 17 & 13 & 6 \end{bmatrix}$$

The **pick distance matrix (PiDM)** is defined as the matrix in which the rows and columns correspond to slots and placement positions, respectively, and each entry is a **pick distance**. The starting point corresponds to the last column, with the assumption that the starting point is a placement position when the placement head starts a travel from it to a feeder to pick a component; meanwhile, it corresponds to the last row, with the assumption that it is a slot when the placement head travels from the placement position of the last mounted component back to it. The distance from the starting point to itself is set to positive infinity. As an example, the **PiDM** of PAT is depicted as

$$\begin{bmatrix} 3 & 7 & 6 & 16 & 12 & 1 \\ 5 & 4 & 3 & 9 & 7 & 2 \\ 8 & 3 & 5 & 4 & 2 & 3 \\ 12 & 5 & 10 & 3 & 6 & 4 \\ 4 & 8 & 7 & 17 & 13 & \infty \end{bmatrix}$$

4. APPLY PIBSO TO PCBAOP

4.1. Initialization of probabilities of substates

If the path from the starting point to a slot and the path from a placement position to the starting point are assumed to be pick paths, a tour includes two parts: length of placement paths and length of pick paths. Each entry in **PDM** denotes a substate of placement paths. Each entry in **PiDM** denotes a substate of pick paths. According to Eq. (7), shorter paths are better and should have higher probabilities to be selected. Consequently, the probabilities of two types of substates are initialized as the reciprocals of placement distances and pick distances, respectively. The probability matrix of substates of placement paths is called **feeder assignment probability matrix (FAPM)**. As one example in PAT, it is

$$\begin{bmatrix} 0.100 & 0.045 & 0.083 \\ 0.111 & 0.083 & 0.143 \\ 0.091 & 0.111 & 0.500 \\ 0.059 & 0.077 & 0.167 \end{bmatrix}$$

The probability matrix of substates of pick paths is called **pick probability matrix (PPM)**. As one example in PAT, it is

$$\begin{bmatrix} 0.333 & 0.143 & 0.167 & 0.063 & 0.083 & 1 \\ 0.200 & 0.250 & 0.333 & 0.111 & 0.143 & 0.500 \\ 0.125 & 0.333 & 0.200 & 0.250 & 0.500 & 0.333 \\ 0.083 & 0.200 & 0.100 & 0.333 & 0.167 & 0.250 \\ 0.250 & 0.125 & 0.143 & 0.059 & 0.077 & 0 \end{bmatrix}$$

4.2. Unit generation rules

A unit includes two parts: a feeder assignment subunit (**FASU**) and a pick subunit (**PSU**). The former determines placement paths and the latter determines pick paths. The generation rules of two subunits are described below.

4.2.1. Generating FASU

The **FASU** is a vector, in which indexes denote slots and its elements are feeders (component types). In the procedure of generating a **FASU**, a slot is selected randomly, and then a feeder (component type) is selected for the slot by using rou-

https://doi.org/10.1017/S0890060413000632 Published online by Cambridge University Press

lette wheel selection. This process is repeated until all feeders are loaded. In roulette wheel selection, the roulette wheel consists of probabilities of types of components, that is, the related entries of the row (the row corresponds to the slot) in **FAPM**. When a feeder is selected by a slot, it will not be put in the roulette wheels for selecting feeders for other slots. In **FASU**, for the unused slots, the elements are set to zeros.

As one example in PAT, the process of generating a **FASU** denoted by $fasu_1$ is described as follows. Slot 3 is selected, and the roulette wheel consists of 0.091, 0.111, and 0.500. Thus, the feeder of type 1 is selected. Next, slot 2 is selected, and the roulette wheel consists of 0.083 and 0.142. Thus, the feeder of type 3 is selected. At the end, slot 4 is selected, and the feeder of type 2 is assigned to slot 4. Consequently, $fasu_1$ is worked out as

0 Type3 Type1 Type2

4.2.2. *Generating a PSU associated with a FASU*

When a **FASU** has been generated, an associated **PSU** needs to be generated. The **PSU** is a vector in which the indexes denote placement positions and the last index denotes the starting point, and its elements are the slots selected in the **FASU**. In the procedure of generating a **PSU**, a placement position is selected randomly, and then a slot is selected for the placement position by using roulette wheel selection. This process is repeated until all placement positions are assigned slots. In roulette wheel selection, the roulette wheel consists of probabilities of slots that are the related entries of the column (the column corresponds to the placement position) in **PPM**.

For PAT, as shown in Table 1, position 1 and 2 are of type 1, position 3 and 4 are of type 2, and position 5 is of type 3. Accordingly, for $fasu_1$ in its associated **PSU**, slot 3 and 4 occur twice and slot 2 occurs once. As an example, the process of generating a **PSU** associated with $fasu_1$, denoted by psu_1, is described as follows. Position 4 is selected, and the roulette wheel consists of 0.111, 0.250, 0.333, and 0.059. Thus, slot 3 is selected. Next, position 1 is selected, and the roulette wheel consists of 0.200, 0.125, 0.083, and 0.250. Thus, slot 2 is selected. Next, position 2 is selected, and the roulette wheel consists of 0.250, 0.333, 0.200, and 0.125. Thus, slot 4 is selected. Next, position 3 is selected, and the roulette wheel consists of 0.200, 0.100, and 0.143. Thus, slot 3 is selected. Next, position 5 is selected, and the roulette wheel consists of 0.167 and 0.077. Then, the starting point is selected. At the end, slot 4 is assigned to the starting point. Thus psu_1 is worked out as

slot2 slote4 slot3 slot3 startingpoint slot4

When a **FASU** and an associated **PSU** are generated, a unit is formed. For instance, the placement paths and the pick paths of the above generated unit are shown in Figure 2. For a unit, the length of tour is fixed no matter what the component placement sequence is. For example, as shown in Figure 2, the tour *starting point* → *slot4* → *position4* → *slot3* → *position2* → *slot4* → *position3* → *slot3* → *position1* → *slot2* → *position5* → *starting point*, and the tour *starting point* → *slot4* → *position3* → *slot3* → *position2* → *slot4* → *position4* → *slot3* → *position1* → *slot2* → *position5* → *starting point* have the same length. The reason is that each placement path and pick path need be passed once in a tour.

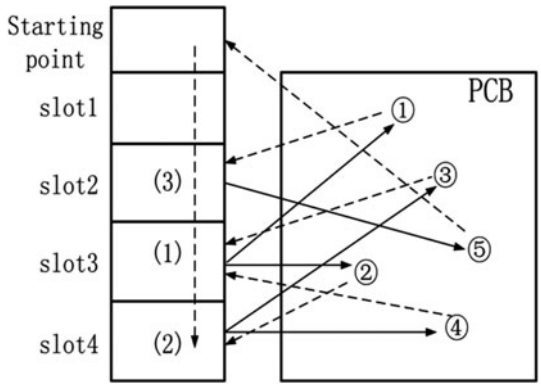

Fig. 2. The placement paths (solid lines) and the pick paths (dash lines) of the unit of $fasu_1$ and psu_1. Here, (1), (2), and (3) denote Type 1, 2, and 3 feeders, respectively, and ①, ②, ③, ④, and ⑤ denote Positions 1, 2, 3, 4, and 5, respectively.

4.3. Population update

The fitness is defined as the reciprocal of the length of a tour, that is, $1/z$ [z from Eq. (7)]. When a slot selects a feeder in **FASU** or a placement position selects a slot in **PSU**, the relevant substate probabilities increase in **FAPM** and **PPM**. The following example shows how **FAPM**, **PPM**, and population update.

For PAT, assume the scale of population is 5; in the first generation, the unit $unit_1$ with length of 67 is

0 ty3 ty1 ty2 (**FASU**)

S2 S4 S3 S3 Stp S4 (**PSU**)

The lengths of other four units, $unit_2$, $unit_3$, $unit_4$ and $unit_5$, are 70, 56, 71, and 80, respectively. For each unit, the length of tour is normalized by Eq. (1), with $fit_{\max} = 80$ and $fit_{\min} = 56$. The results of normalization are

$norfit_1$	$norfit_2$	$norfit_3$	$norfit_4$	$norfit_5$
0.46	0.58	0	0.63	1

Set dr to 1, and then the incremental factors of units are

w_1	w_2	w_3	w_4	w_5
0.09	0.12	0	0.13	0.2

Next, in **FAPM** and **PPM**, the entries relevant to each unit are updated. For $unit_1$, in **FAPM**, entries (2, 3), (3, 1), and (4, 2)

https://doi.org/10.1017/S0890060413000632 Published online by Cambridge University Press

need to multiply $(1 + w_1)$ by Eq. (3), and in **PPM**, entries (1, 2), (2, 4), (3, 3), (4, 3), (5, 5), and (6, 4) need to multiply $(1 + w_1)$ by Eq. (3). The results are

$$\begin{bmatrix} 0.100 & 0.045 & 0.083 \\ 0.111 & 0.083 & \boldsymbol{0.156} \\ \boldsymbol{0.099} & 0.111 & 0.500 \\ 0.059 & \boldsymbol{0.084} & 0.167 \end{bmatrix}$$

and

$$\begin{bmatrix} 0.333 & 0.143 & 0.167 & 0.063 & 0.083 & 1 \\ \boldsymbol{0.218} & 0.250 & 0.333 & 0.111 & 0.143 & 0.500 \\ 0.125 & 0.333 & \boldsymbol{0.218} & \boldsymbol{0.273} & 0.500 & 0.333 \\ 0.083 & \boldsymbol{0.218} & 0.100 & 0.333 & 0.167 & \boldsymbol{0.273} \\ 0.250 & 0.125 & 0.143 & 0.059 & \boldsymbol{0.084} & 0 \end{bmatrix},$$

respectively. In the same way, **FAPM** and **PPM** are modified by $unit_2$, $unit_3$, $unit_4$, and $unit_5$ in sequence. In the next generation, new units are generated by using the updated **FAPM** and **PPM** so that population updates.

4.4. The process of PIBSO for PCBAOP

Set the scale of population to a suitable number. Initialize $i = 0$. Set the threshold thr to a suited value empirically. The process of PIBSO for PCBAOP is described as follows:

Step 1. Let $i = i + 1$, and generate units to form a new generation of population denoted by g_i.

Step 2. Find the average tour length of units, denoted by ave_i. Find the best unit with the shortest tour and denote its length by $best_i$.

Step 3. Find incremental factors of all units and use them to modify **FAPM** and **PPM**.

Step 4. If $i < 0$, go back to Step 1; else go to Step 5.

Step 5. Let $c = i$. Find the average of $|ave_c - ave_{c-1}|$, $|ave_{c-1} - ave_{c-2}|$, . . . , $|ave_{c-18} - ave_{c-19}|$, and denote it by Ata.

Step 6. If $Ata < thr$, then end the process and output the minimum among $best_i$ as the near-optimum result; otherwise, return to Step 1.

The condition of convergence is defined as the average of $|ave_i - ave_{i-1}|$ for the last 20 generationis less than his consideration derives from the following observation. During the

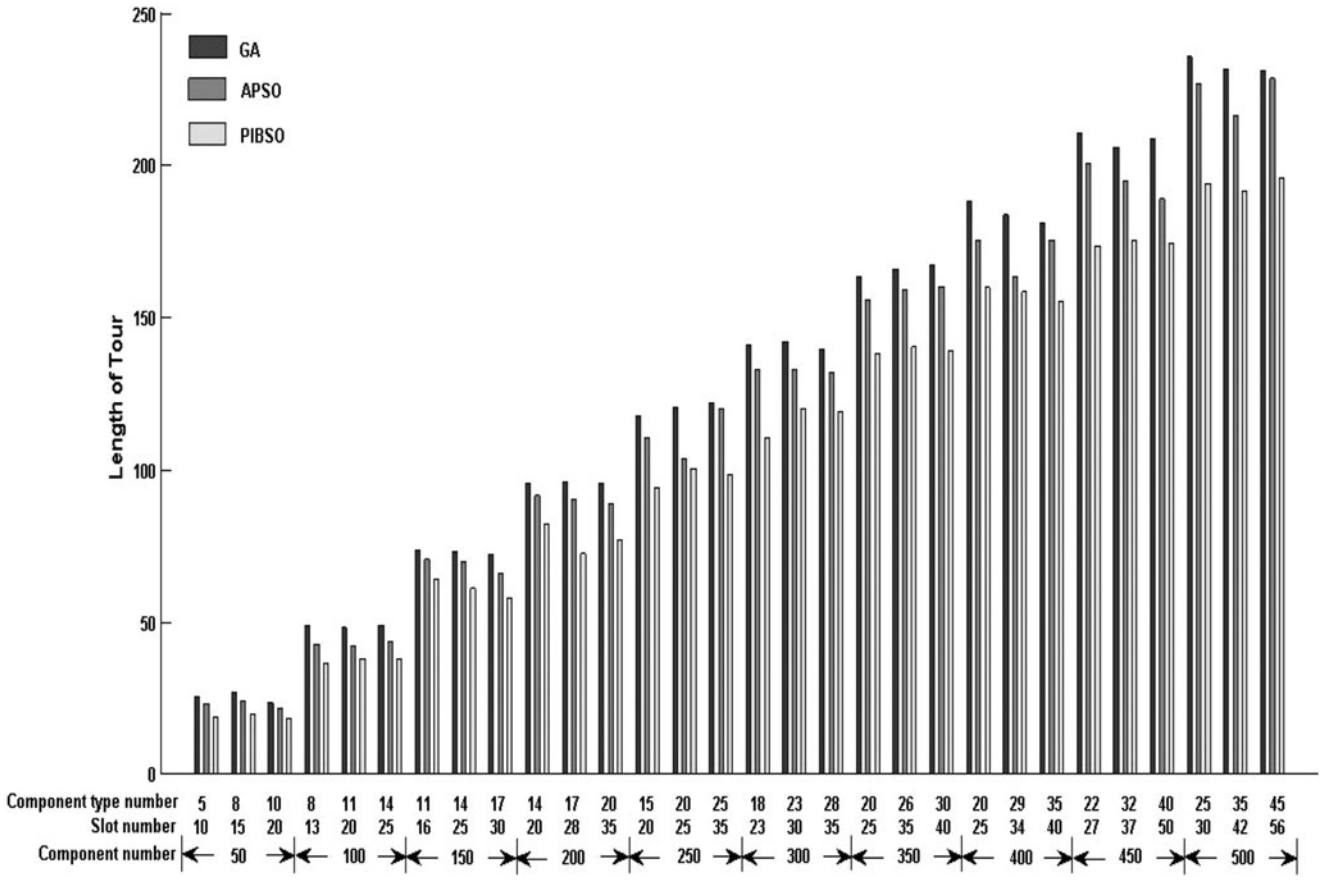

Fig. 3. Comparisons between probability increment based swarm optimization (PIBSO) and two benchmark algorithms on the length of tour.

https://doi.org/10.1017/S0890060413000632 Published online by Cambridge University Press

process of evolution of units, the difference among units tails off and local optima occur frequently. However, it hardly happens that the population falls into a local optimum and cannot jump out in 20 consecutive generations. Thus, if units have little change for 20 consecutive generations, PIBSO reaches a nearly global optimum solution.

5. NUMERICAL EXPERIMENTS AND RESULTS

The effectiveness and efficiency of PIBSO in solving PCBAOP were tested by numerical experiments on a Pentium 1.7-GHz CPU with 1 GB of RAM using MATLAB 7.0. A program generating PCB assembly instances was developed and used to produce **PDM** and **PiDM** of an assembly instance on PCB with the size of 300×500 mm^2. In order to inspect performance of PIBSO comprehensively, PCB assembly instances with component number of 50, 100, 150, 200, 250, 300, 350, 400, 450, and 500 are generated, respectively. For a certain component number, three types of assembly instances with different component type numbers and slot numbers are generated. For example, when component number is set to 50, three pairs of component type number and slot number, that is, (5, 10), (8, 15), and (10, 20), are used to produce assembly instances of three cases.

PIBSO is compared with two benchmark algorithms: GA (Najera & Brizuela, 2005) and APSO (Chen & Lin, 2007). The parameters of PIBSO are set as population scale is 40 ($dr = 1.0$, $thr = 0.01$). A PCB instance is generated by the aforementioned program first. Then three algorithms are run 30 times on it. The average length of tour and CPU running time are recorded and graphically shown in Figure 3 and Figure 4, respectively.

It is easy to observe from Figure 3 that for each assembly instance, no matter what settings of component type number, slot number, and component number are, PIBSO can find a shorter tour than GA and APSO do. Moreover, tours found by GA are always the longest. The reason could be analyzed as follows. Crossover and mutation operators of GA are wholly stochastic, while selection strategy leads GA to search global optimal solution through iterations. Instead, APSO memorizes the current and historic optima so that it finds better solution than GA. The critical weakness of GA and APSO is easy to fall into local optimum. However, the mechanism of PIBSO avoids local optimum by roulette wheel selection and probability updating. Roulette wheel selection increases the randomness of a search, while a probability updating operator makes better states more likely to be chosen. The exploitation and exploration in search process are balanced, and thus

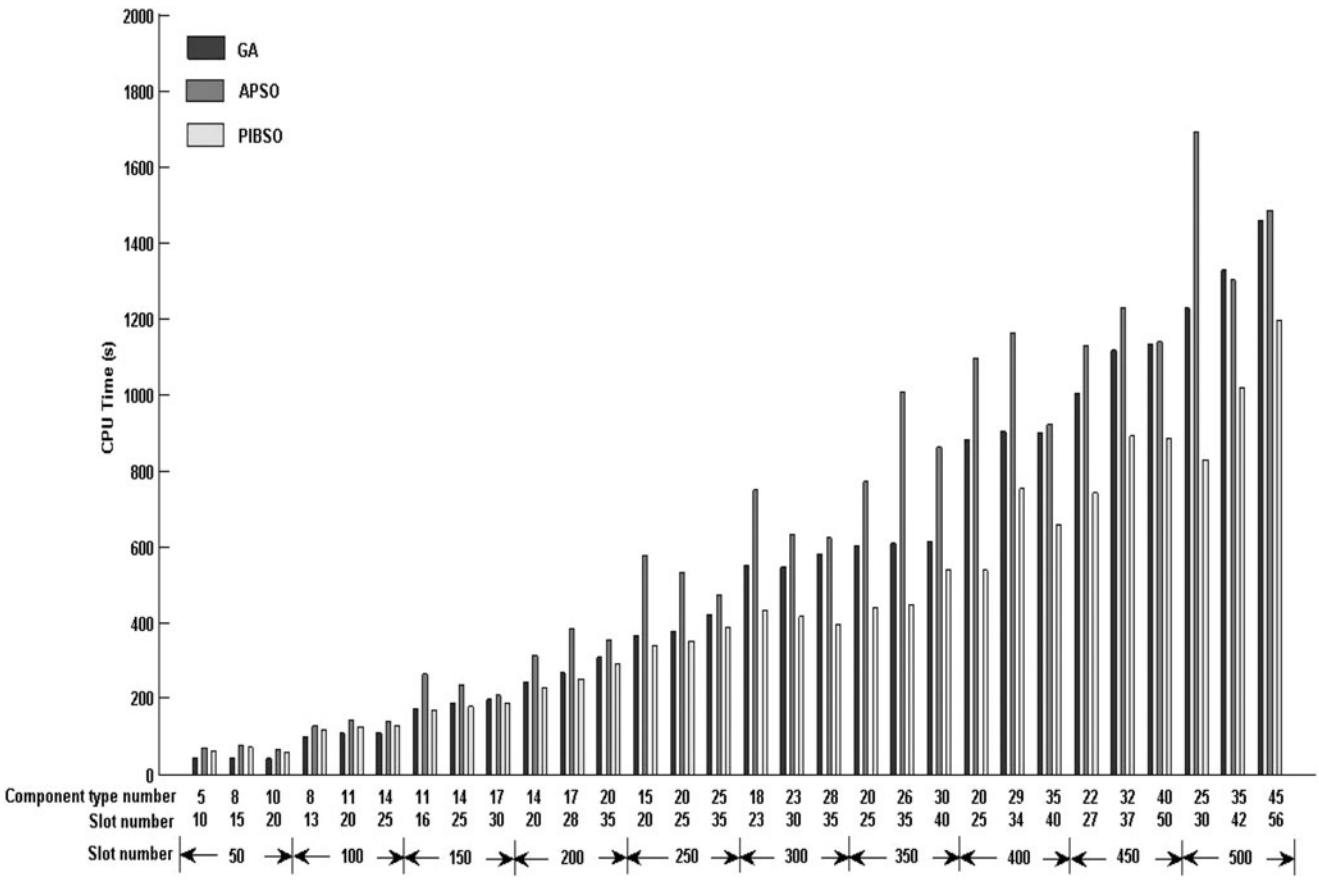

Fig. 4. Comparisons between probability increment based swarm optimization (PIBSO) and two benchmark algorithms on CPU running time.

https://doi.org/10.1017/S0890060413000632 Published online by Cambridge University Press

PIBSO has superior search performance and gets a shorter tour.

As shown in Figure 4, when component number is less than 100, GA is faster than APSO and PIBSO; and PIBSO has shorter search time than GA and APSO with growth of component number. As mentioned before, GA has powerful stochastic search operators, crossover and mutation; thus, when the search space is not very large for a small-scale PCBAOP with component number less than 100, GA can find optimum quickly. However, when component number increases, search space grows tremendously and GA cannot find global optimum in a short time. Conversely, roulette wheel selection and probability updating of PIBSO produce a trade-off between global and local searches, which makes PIBSO more stable and free from influence of optimization scale growth. Thus, as component number increases, PIBSO costs less time than GA to get near-global optima. In addition, APSO spends longer time than GA, as shown in Figure 4. Its reason is that when search space grows, the velocity calculation in APSO becomes more complicated.

6. CONCLUSIONS

In this paper, a novel swarm intelligence approach for combinatorial optimization, named PIBSO, is proposed. It has a complete mechanism of population evolution. Each state of search space has a probability to be selected, and the probability is updated by incremental factor and related operators. Incremental factor is determined by fitness of the state and relevant operators. PIBSO searches the optimal state based on the probabilities of states, and usually roulette wheel selection is used.

When PIBSO is applied to PCBAOP, **PDM** and **PiDM** are defined. Accordingly, **FAPM** and **PPM** are defined as well. PIBSO finds the near-optimal solution by updating the two probability matrices and roulette wheel selection. Experimental results indicate that PIBSO not only gets shorter tour but also costs less CPU running time than GA and APSO. The superior performance of PIBSO attributes to the population evolution mechanism in which roulette wheel selection and probability updating make a trade-off between global and local search.

ACKNOWLEDGMENTS

This work was partially supported by the Research Committee of the University of Macau under Grant MYRG184 (Y1-L3)-FST11-DMC, the Science and Technology Development Fund of Macau SAR under Grant 018/2009/A1, the National Natural Science Foundation of China under Grant 61076098, and the Natural Science Foundation of Huizhou University under Grant 2012YB16.

REFERENCES

Beni, G., & Wang, J. (1988). The concept of cellular robotic system. *Proc. IEEE Int. Symp. Intelligent Control*, Los Altimos, CA.

Beni, G., & Wang, J. (1989). Swarm intelligence in cellular robotic systems. *Proc. NATO Advance Workshop on Robots and Biological Systems*, Tuscany, Italy.

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. New York: Oxford University Press.

Chen, Y., & Lin, C. (2007). A particle swarm optimization approach to optimize component placement in printed circuit board assembly. *International Journal of Advanced Manufacturing Technology 35(5–6)*, 909–925.

Eberhart, R.C., & Shi, Y. (2007). *Computational Intelligence: Concepts to Implementations*. Burlington, VT: Morgan Kaufmann.

Ho, W., & Ji, P. (2007). *Optimal Production Planning for PCB Assembly*. London: Springer–Verlag.

Jin, Y., & Li, W. (2007). Design concept generation: a hierarchical coevolutionary approach. *Journal of Mechanical Design 129*, 1012–1022.

Jin, Y., Li, W., & Lu, S.C.Y. (2005). A hierarchical co-evolutionary approach to conceptual design. *CIRP Annals Manufacturing Technology 54(1)*, 155–158.

Jin, Y., Zouein, G.E., & Lu, S.C.Y. (2009). A synthetic DNA based approach to design of adaptive systems. *CIRP Annals—Manufacturing Technology 54(1)*, 153–156.

Khoo, L.P., & Ng, T.K. (1998). A genetic algorithm-based planning system for PCB component placement. *International Journal of Production Economics 54(3)*, 321–332.

Koenig, R., & Schneider, S. (2012). Hierarchical structuring of layout problems in an interactive evolutionary layout system. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 26(2)*, 129–142.

Korte, B., & Vygen, J. (2008). *Combinatorial Optimization: Theory and Algorithms*. Berlin: Springer.

Lipson, H. (2008). Evolutionary synthesis of kinematic mechanisms. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 22*, 129–142.

Loh, T.S., Bukkapatnam, S.T.S., Medeiros, D., & Kwonb, H. (2001). A genetic algorithm for sequential part assignment for PCB assembly. *Computer & Industrial Engineering 40(4)*, 293–307.

Maher, M.L., & Fisher, D.H. (2012). Using AI to evaluate creative designs. *Proc. Int. Conf. Creative Design, ICDC'2012*. Glasgow: Design Society.

Najera, A.G., & Brizuela, C.A. (2005). PCB assembly: an efficient genetic algorithm for slot assignment and component pick and place sequence problems. *Proc. IEEE Congr. Evolutionary Computation CEC'05*. Edinburgh.

Shea, K., Ertelt, C., Gmeiner, T., & Ameri, F. (2010). Design-to-fabrication automation for the cognitive machine shop. *Advanced Engineering Informatics 24(3)*, 251–268.

Wang, J.F., Liu, J.H., Li, S.Q., & Zhong, Y.F. (2003). Intelligent selective disassembly using the ant colony algorithm. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 17*, 325–333.

Yang, P., & Zeng, K. (2009). A high-performance approach on mechanism isomorphism identification based on an adaptive hybrid genetic algorithm for digital intelligent manufacturing. *Engineering With Computers 25(4)*, 397–403.

Kehan Zeng is currently a PhD student in the Department of Electrical and Computer Engineering, University of Macau, and a part-time lecturer in the Department of Computer Science, Huizhou University. He received a BS in computer science and applications from the National University of Defense Technology of China in 2002, and a ME in electronic and mechanical engineering from Jiangsu University in 2009. His research interests include wavelets, graph algorithms, computational intelligence, optimization, machine learning, and applications in biomedical engineering.

Zhen Tan is a first-year master's student in the Department of Electrical and Computer Engineering at the University of Macau. He received a BS in electronic science from Huizhou University in 2012. His research interests include optimization, R&D of circuits, and systems in biomedical engineering.

https://doi.org/10.1017/S0890060413000632 Published online by Cambridge University Press

Mingchui Dong is a Professor and PhD supervisor with the Department of Electrical and Computer Engineering at the University of Macau and the Automation Department at Tsinghua University. He received an MS degree in electronic engineering at Tsinghua University. Prof. Dong's main research interests are artificial intelligence and its application in biomedical engineering, CIMS, fault diagnosis, machine translation, and so forth.

Ping Yang is a Professor in the Laboratory of Advanced Design, Manufacturing & Reliability for MEMS/NEMS/ODES and the Laboratory of Materials & Micro-Structural Integrity, School of Mechanical Engineering, Jiangsu University. He received a PhD in mechanical engineering from Huazhong University of Science and Technology in 2001. Dr. Yang's research interests include the theoretical aspect and computer-aided design of mechanical systems for design and control.

https://doi.org/10.1017/S0890060413000632 Published online by Cambridge University Press