A Monte Carlo Hyper-Heuristic To Optimise Component Placement Sequencing For Multi Head Placement Machine

Masri Ayob And Graham Kendall
ASAP Group, CSiT, University of Nottingham, Nottingham NG8 1BB, UK
mxa|gxk@cs.nott.ac.uk

Abstract: In this paper we introduce a Monte Carlo based hyper-heuristic. The Monte Carlo hyper-heuristic manages a set of low level heuristics (in this case just simple 2-opt swaps but they could be any other heuristics). Each of the low level heuristics is responsible for creating a unique neighbour that may be impossible to create by the other low level heuristics. On each iteration, the Monte Carlo hyper heuristic randomly calls a low level heuristic. The new solution returned by the low level heuristic will be accepted based on the Monte Carlo acceptance criteria. The Monte Carlo acceptance criteria always accept an improved solution. Worse solutions will be accepted with a certain probability, which decreases with worse solutions, in order to escape local minima. We develop three hyper-heuristics based on a Monte Carlo method, these being Linear Monte Carlo Exponential Monte Carlo and Exponential Monte Carlo with counter. We also investigate four other hyper-heuristics to examine their performance and for comparative purposes. To demonstrate our approach we employ these hyper-heuristics to optimise component placement sequencing in order to improve the efficiency of the multi head placement machine. Experimental results show that the Exponential Monte Carlo hyper-heuristic is superior to the other hyper-heuristics and is superior to a choice function hyper-heuristic reported in earlier work.

Keywords: Intelligent Search, Heuristic, Printed Circuit Board Assembly, Optimisation.

1. Introduction

SMD (surface mount device) placement machines are used to assemble components onto a PCB (printed circuit board). Optimising the pickup and placement sequence is a significant factor in determining the efficiency of an SMD placement machine. In this work, we study a component placement sequencing problem of a multi head SMD placement machine. The machine has a fixed feeder carrier, a fixed PCB table and a positioning arm head that is equipped with a number of nozzles that is used to grasp the components. The feeder carrier consists of several feeder slots where the components are located. The PCB table holds the PCB in a locked position during a pick-place operation. The head and arm (or sometimes called robot arm) is movable in the X-Y direction simultaneously. The tour of the robot arm begins by picking up a number of components from the feeder. Then, it travels in the X and Y direction simultaneously and positions itself at the point where the component will be mounted. Then the robot arm moves down (Z-direction) and mounts the component on the board before returning to its original position and repeating these steps for the next locations on the board that have to be mounted on the same tour. After completing a tour, the robot arm returns to the feeder location to begin another tour. Figure 1 is an example of multi head placement machine.

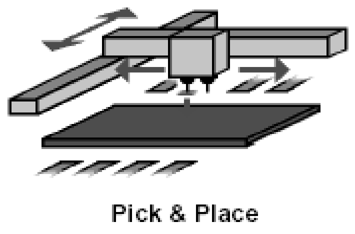

Figure 1: An Example Of Pick Place Multi Head Placement Machine (Taken From [1])

Various methods have been applied to improve the efficiency of SMD placement machines such as [2,3,4], but none have used a hyper-heuristic approach. Hence, we are going to introduce a new hyper-heuristic approach based on a Monte Carlo method [5], to optimise the component placement sequencing in order to improve the efficiency of the multi head SMD placement machine.

The motivation for investigating the hyper-heuristic approach instead of the other meta-heuristics is that the hyper-heuristic framework provides a way to combine many heuristics in solving a problem. This feature is useful for solving the component placement sequencing problem of a multi head SMD placement machine since this problem requires a number of heuristics. Most previous works on hyper-heuristics (such as [6,7,8,9,10,11]) focus on sequencing the calls of the low level heuristics (LLHs). They report successful results. However, in this work we investigate on improving the acceptance criteria. Of course, the sequence of LLH is important but in our problem domain we are dealing with the LLHs that randomly create the neighbour solutions. Since the neighbour's solution is randomly generated by the LLHs, we cannot measure the performance of each LLH based on the historical performance as the quality of the obtained solution does not represent the efficiency of the LLH. This statement is supported by the experimental results from this work where the Choice-Function described in [7] does not perform well in all test problems.

There are many determining factors in minimising the assembly cycle time of multi head placement machine such as optimisation of the pickup sequence, the placement sequence, nozzle assignment, sub-tour grouping and sequencing the sub-tour. However, optimising one factor may increase the cost of another factor(s). The complexity of this problem causes a difficulty in devising a good strategy to minimise the assembly cycle time. The advantage of using the hyper-heuristic, compared to the other meta-heuristics, is the ability to combine a number of heuristics. By applying a hyper-heuristic approach, we do not have to concern ourselves with the trade-off between the optimisation of the important factors as this will be catered for within the hyper-heuristic.

2. The Scheduling Model

The quality of a schedule can be evaluated by a placement time function, known as the assembly cycle time, for a pick and place multi head placement. We model a pick place multi head placement machine that has a single head equipped with *G* number of nozzles, fixed PCB table and fixed feeder carrier (same as in [12,13]). This placement machine was categorised in [14] as a multi head placement machine that has an arm and head which can move in the X-Y axis simultaneously but the feeder carrier and PCB table are fixed. The following notations are used to describe the scheduling model:

CT	: the assembly cycle time to assemble all components;
N	: the number of PCB points on the PCB;
Q	: the total number of available PCB points to be scheduled, where $Q \leq N$.
K	: the number of component types (each feeder slot holds multiple copies of one component type);
G	: the number of nozzles per head;
B	: the total number of sub tours;
M	: the total number of feeder slots where $K \leq M$;
$c(j,h)_{x,y}$	: the X,Y coordinate on the PCB which will have a component placed there in the hth placement sequence of the jth sub tour;
V	: the robot speed (average);
λ	: the time for picking up a component;
θ	: the time for placing a component;
r	: the rth slot number where $r \in \{0,1,2,\ldots,(M-1)\}$;
i	: the ith component type where $i \in \{1,2,\ldots,K\}$;
k	: the kth pickup sequence in a sub tour where $k \in \{1,2,\ldots,G\}$;
h	: the hth placement sequence in a sub tour where $h \in \{1,2,\ldots,G\}$;
j	: the jth sub tour number where $j \in \{1,2,\ldots,B\}$;
$I(j,k)$	: the time taken for the robot arm to travel from feeder carrier to PCB point and place a component in the kth placement sequence of the jth sub tour;
$P(j,k)$	: the time taken for the robot arm to travel from PCB point to feeder carrier and pick a component in the kth pickup sequence of the jth sub tour;
$\Phi_i(j,k)$	: the nozzle used to pick ith component in the kth pickup sequence of the jth sub tour;
$\Omega_i(j,h)$	: the nozzle used to place ithcomponent in the hth placement sequence of the jth sub tour;
$S(r)$	: the rth slot distance referring to the origin of feeder slot, s(0) where s(0)=0 and $r \geq K$;
$R(j,k)$	: the slot distance for the kth pickup sequence of the jth sub tour;
$d(j,h)$	: the $\max\{\lvert d(j,h)_x\rvert, \lvert d(j,h)_y\rvert\}$ where *x* and *y* is the X,Y robot traveling distance to place a component in the hth placement sequence of the jth sub tour, where the distance is measured as a Chebychev distance (dictated by the maximum of X or Y traveling distance as the robot arm moves concurrently in X-Y axis);
$m(j,k)$	: the $\max\{\lvert m(j,k)_x\rvert, \lvert m(j,k)_y\rvert\}$ where *x* and *y* is the X,Y robot traveling distance to pick a component in the kth pickup sequence of the jth sub tour, where the distance is measured as a Chebychev distance ;
F	: the gap between each two adjacent feeder slots;
L	: the gap between each two successive (adjacent) nozzles.

The objective function is to minimise the assembly cycle time, CT, by minimising the traveling distance of the robot to perform pick and place operation:

$$\text{Minimise } CT = \sum_{j=1}^{B}\left[\sum_{k=1}^{G} P(j,k) + \sum_{k=1}^{G} I(j,k)\right] \quad (1)$$

s.t.

$$\sum_{i=1}^{K} g_{ir} \leq 1, \forall r; \quad \text{Each feeder slot may hold one component type only.} \quad (2)$$

$$\sum_{r=0}^{M-1} g_{ir} = 1, \forall i; \quad \text{Assuming no feeder duplication.} \quad (3)$$

$$g_{ir} = \begin{cases} 1: & \text{if component type i is assigned to feeder slot r,} \\ 0: & \text{otherwise} \end{cases} \quad (4)$$

$$\Phi_i(j,k) \in \{0,1,2,..(G-1)\};\ \varphi(j,k)\neq\varphi(j,l) \text{ if } k\neq l;\quad j=1,2,..,B;\ k=1,2,..,G; \tag{5}$$

$$\Omega_i(j,h) \in \{0,1,2,..(G-1)\};\ \omega(j,h)\neq\omega(j,l) \text{ if } h\neq l;\quad j=1,2,..,B;\ h=1,2,..,G; \tag{6}$$

$$\Phi_i(j,k)= \Omega_i(j,h) \tag{7}$$

$$z(j,k) = \begin{cases} 1 : \text{if there is a nozzle assigned to pick or place a component in the } k^{th} \text{ sequence of } j^{th} \text{ sub tour.} \\ 0 : \text{otherwise} \end{cases} \tag{8}$$

$$\sum_{k=1}^{G} z(j,k) \leq G; \tag{9}$$

$$b_r(j,k)= \begin{cases} 1: \text{ if there is a component to be picked up from feeder slot } r \text{ in the } k^{th} \text{ pickup sequence of the } j^{th} \text{ sub tour,} \\ 0: \text{ otherwise} \end{cases} \tag{10}$$

$$\sum_{r=0}^{M-1} b_r(j,k) \leq 1, \forall j, \forall k; \quad \text{Only one component can be picked up in each pickup sequence.} \tag{11}$$

where:

$$B = \begin{cases} Q/G & \text{if } Q \text{ MOD } G=0; \\ 1 + Q/G & \text{if } Q \text{ MOD } G\neq 0; \end{cases} \tag{12}$$

$$P(j,k) = (\frac{m(j,k)}{V} + \lambda) * z(j,k); \tag{13}$$

$$I(j,k) = (\frac{d(j,k)}{V} + \theta) * z(j,k); \tag{14}$$

$$m(j,k)_x = \begin{cases} R(j,k) - \varphi(j,k)*L & \text{if } j=k=1; \\ [R(j,k)-\varphi(j,k)*L] - [c(j-1,G)_x-\omega(j-1,G)*L] & \text{if } k=1, j>1; \\ [R(j,k)-R(j,k-1)] - [(\varphi(j,k)-\varphi(j,k-1))*L] & \text{if } k>1; \end{cases} \tag{15}$$

$$d(j,h)_x = \begin{cases} [c(j,h)_x-R(j,G)] - [(\omega(j,h)-\varphi(j,G))*L] & \text{if } h=1; \\ [c(j,h)_x-c(j,h-1)_x] - [(\omega(j,h)-\omega(j,h-1))*L] & \text{if } h>1; \end{cases} \tag{16}$$

$$m(j,k)_y = \begin{cases} 0 - [c(j-1,G)] & \text{if } k=1, j>1; \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

$$d(j,h)_y = \begin{cases} c(j,h)_y & \text{if } h=1; \\ [c(j,h)_y - c(j,h-1)_y] & \text{if } h>1; \end{cases} \tag{18}$$

$$S(r) = r * F \tag{19}$$

$$R(j,k) = \sum_{r=0}^{M-1} \left[S(r) * b_r(j,k) \right] \tag{20}$$

The assembly cycle time, CT is a function of robot traveling distance divided by the robot speed, plus a components' pickups (λ) and a placements'(θ) time. In our model, all elements are fixed (that are the PCB table, feeder carrier and nozzles) except the arm and head that moveable in X-Y axis concurrently. Since the robot (i.e. the arm and head of SMD placement machine) can move simultaneously in the X-Y axis, the robot traveling distance dictates by the maximum of X or Y traveling distance (i.e. a Chebychev distance). A complete tour of a robot consists of *B* sub tours and each sub tour has at most G pairs of pick and place points (equation 1).

3. Monte Carlo Algorithm

Let us define a solution space *S*, an objective function *f* and a neighbourhood structure *n*. A basic Monte Carlo (MC) method for minimisation problem can be expressed by the following algorithm [5]:

```
Step 1:  (Initialisation)
    (A) Choose a starting solution $S_0 \in S$;
    (B) Record the best obtained solution, $S_{best} = S_0$
        and $f(S_{best}) = f(S_0)$;
Step 2:  (Choice and termination)
    (A) Randomly choose $S_c \in n(S_0)$;
    (B) Compute $\delta = f(S_c) - f(S_0)$;
    (C) If $\delta \leq 0$ then accept $S_c$ (and proceed to Step 3);
    (D) Else: Accept $S_c$ with a probability that
        decreases with increases in $\delta$. If $S_c$ is rejected
        and stopping condition=false, then return to
        Step2(A);
    (E) Terminate by a stopping condition.
Step 3:  (Update)
    Re-set $S_0 = S_c$, and if $f(S_c)<f(S_{best})$, perform
    Step1(B). Return to Step2 if stopping
    condition=false.
```

Figure 2: A Basic Monte Carlo (MC) Algorithm[5]

In this work, we develop three types of acceptance probability (referring to Step2(D) Figure 2):

i. Linear Monte Carlo (LMC). The probability is computed by (*M-δ*) where *M* is a constant valued between 0 and 100. Based on our preliminary test, the LMC works well with M=5 (for our test data, different values of M may be required for different problem instances). The test shows that the LMC is parameter sensitive. The LMC will perform almost similar to a steepest descent approach with a small value of M since the probability of accepting worse solution is too small. On contrary, larger M will more likely to accept worse solution and cannot converge. The new solution, S_c is accepted if a generated random number is less than (*M-δ*).
ii. Exponential Monte Carlo (EMC). The probability is computed by $e^{-\delta}$ where $\delta=f(S_c)-f(S_0)$. The probability of accepting a worse solution decreases as the δ increases. The new solution, S_c is accepted if a generated random number is less than $e^{-\delta}$.
iii. Exponential Monte Carlo with counter (EMCQ). The probability is computed by $e^{-\theta/\tau}$ where $\theta=\delta*t$

and $\tau=\rho(Q)$. t is a computation time (in our case we use minutes as a unit time). θ and τ are defined such that we ensure that the probability of accepting a worse solution decreases as the time increases and δ increases. The factor of time is included in this formulation as an intensification factor. At the beginning of the search, the moderately worse solution is more likely to be accepted but as the time increases the worse solution is unlikely to be accepted. However, the probability of accepting a worse solution increases as the counter of consecutive none improvement iterations, Q increases. This is a diversification factor. $\rho(Q)$ is a function to intelligently control the Q. In this work we use $\tau=v*Q$ where $0\leq v\leq 1$, in order to limit the acceptance probability. However, our preliminary experiment on parameter sensitivity of v shows that the EMCQ algorithm with $v=1$ performs the best. Therefore, we set $\tau=Q$. The new solution, S_c is accepted if a generated random number is less than $e^{-\theta/\tau}$.

The formulation of the acceptance probability of EMC and EMCQ is quite similar to the acceptance criteria of a simulated annealing approach. The difference, for the EMC and EMCQ is that we do not have a cooling schedule. EMCQ will exponentially increase the acceptance probability as we have been unable to find a better solution for a long time (i.e. too long being trapped in local optima). However, the EMCQ will exponentially reduce the acceptance probability as the searching time increases (similar to a simulated annealing approach). As the EMC and EMCQ do not have parameters which have to be tuned (all the parameters are automatically controlled based on the solution quality (with exploring time and the duration of being trapped in local optima in the case of EMCQ)), these methods are simple and robust heuristic technique.

The EMC and EMCQ algorithm work as follows. First, an initial solution is chosen. Then, for each iteration, a neighbour of the current solution is generated. The 'qualities' of the two solutions are compared. A decision is made whether the new solution should be accepted. An improved solution is always accepted. However, in order to escape from the local optima we accept a worse solution with a probability that depends on δ (and the duration we have been trapped in the local optima in the case of EMCQ). A worse solution is more likely to be accepted if the δ is small (and we cannot find a better solution for a long time for the case of EMCQ). The idea of EMCQ is to ensure that we only accept a moderately worse solution after most of the neighbours of the current solution have been explored and none of them is better than the old solution. Table 1. shows the difference among the three proposed approaches.

Method	**Acceptance Probability**
LMC	$x<(M-\delta)$
EMC	$e^{-\delta}$
EMCQ	$e^{-\theta/\tau}$ where $\theta=\delta*t$ and $\tau=Q$

Table 1: A Comparison Of Acceptance Probability For LMC, EMC And EMCQ.

4. Hyper-heuristic

Many heuristics are problem-dependent [15]. However, there are some heuristics that are not problem-specific such as hyper-heuristic [6,7,8] approaches that aims to be a general-purpose heuristic that can handle a wide range of problems. Hyper-heuristics are (meta-)heuristics that can operate on (meta-)heuristics [8].

The motivation for investigating hyper-heuristic approaches is that the hyper-heuristic framework provides a way of combining many heuristics in solving a problem. The hyper-heuristic framework manages a set of low level heuristics which operates at a higher level of abstraction without having access to the problem domain-knowledge [8]. Generally, the low level heuristics (LLH) are a simple local search, k-opt operator or other heuristics that are problem-dependent. In this work we develop six 2-opt operators as LLHs. Each LLH is responsible for creating a unique neighbour that may be impossible to reach by other LLHs. In fact, each LLH plays a unique role in minimising the cycle time. For example, one aims to optimise the pickup sequence, whilst others aim to optimise the placement sequence, the pickup nozzle assignment, the sub-tour's grouping etc. Whatever factor is being optimised the overall aim is to minimise CT. There is no good strategy for selecting which LLH to apply at a given time. However, continuously applying single LLH will quickly lead to a local optima. Previous work has attempted to combine simple heuristics (in a hyper-heuristic framework) using a Choice-Function [6] and Hyper-genetic algorithm (Hyper-GA) [9]. The Choice-Function hyper-heuristic [6] adaptively ranks the LLHs based upon the historical performance of individual LLHs in order to suggest the next LLH to apply. The Hyper-GA hyper-heuristic [9] evolves the sequence of calling the LLHs by representing a gene in a chromosome with a LLH. The chromosome is evaluated by the quality of the solution obtained when applying the LLHs in the sequence denoted by the chromosome. Other works in the hyper-heuristic area can be found in [10,11]. A general framework of hyper-heuristic is shown in Figure 3.

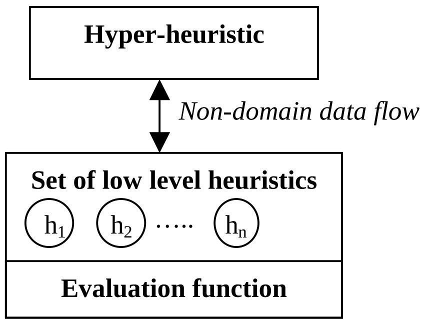

Figure 3: Hyper-heuristic Framework

The communication between the hyper-heuristic and the LLHs uses a standard interface. Only non-domain specific data such as the solution's quality and the computation time is allowed to cross the barrier between the hyper-heuristic and the LLHs. The hyper-heuristic only knows that it has a certain number of heuristics on which to operate and whether the objective function is being minimised or maximised. The general structure of the hyper-heuristic algorithm is shown in Figure 4.

```
Step 1: (Initialisation)
    (A) Choose a starting solution $S_0 \in S$;
    (B) Define H as a set of LLH;
    (C) Record the best obtained solution, $S_{best} = S_0$ and $F(S_{best}) = F(S_0)$;
Step 2: (Choice and termination)
    (A) Choose an $H_c \in H$;
    (B) Apply $H_c$ to produce $S_c \in n(S_0)$;
    (C) Compute $\delta = f(S_c) - f(S_0)$;
    (D) If the acceptance criteria is true, then accept $S_c$ (and proceed to Step 3);
    (E) If $S_c$ is rejected and stopping condition=false, then return to Step2(A);
    (F) Terminate by a stopping condition.
Step 3: (Update)
    Re-set $S_0 = S_c$, and if $f(S_c)<f(S_{best})$, perform Step1(C). Return to Step2 if stopping condition=false.
```

Figure 4: A General Structure Of A Hyper-Heuristic Algorithm

In this work, we investigate seven hyper-heuristic approaches with different acceptance criteria (see 2(D), Figure 4):

1) AM (All Move): Randomly select LLH and accept any solution returned by the LLH.
2) OI (Only Improving): Randomly select LLH and only accept an improved solution returned by the LLH.
3) OICF (Only Improving Choice Function): Select LLH based on historical performance [7] and only accept an improved solution returned by the LLH.

   $F(N_k)=max\{\alpha*f_1(N_k)+ \beta*f_2(N_j,N_k)+ \sigma*f_3(N_k)\}$ *Where*

   $F(N_k)$ is a Choice Function of the kth LLH that has the largest $F(N_k.)$. $f_1(N_k)$ is the cumulative performance rate of heuristic N_k, $f_2(N_j,N_k)$ is the cumulative performance rate of consecutive pairs of heuristics (heuristic N_j followed by N_k) and $f3(N_k)$ is the CPU time which has elapsed since heuristic N_k was last called. Details of the algorithm can be found in [6,7]. If the time taken by each LLH to make a swap is too short (approximate to zero millisecond), then we set the duration as 1.
4) AMCF (All Move Choice Function): Same as OICF but in this case we accept all solution returned by the LLH.
5) LMC (Linear Monte Carlo): Randomly select LLH and accepts S_c returned by the LLH based on the accepting criteria in Table 2.
6) EMC (Exponential Monte Carlo): Randomly select LLH and accepts S_c returned by the LLH based on the accepting criteria in Table 2.
7) EMCQ (Exponential Monte Carlo with Counter): Randomly select LLH and accepts S_c returned by the LLH based on the accepting criteria in Table 2.

The LMC, EMC and EMCQ algorithms have been discussed in Section 3. Table 2 shows the difference among these hyper-heuristics.

Hyper-heuristic	**Accepting Criteria**
AM	Accept all moves.
OI	Accept S_c if $\delta\leq 0$, otherwise reject S_c.
OICF	Accept S_c if $\delta\leq 0$, otherwise reject S_c.
AMCF	Accept all moves
LMC	Accept S_c if $\delta\leq 0$, otherwise accept S_c if $x<(M-\delta)$ where $\delta=f(S_c)-f(S_0)$.
EMC	Accept S_c if $\delta\leq 0$, otherwise accept S_c if $x<e^{-\delta}$ where $\delta=f(S_c)-f(S_0)$.
EMCQ	Accept S_c if $\delta\leq 0$, otherwise accept S_c if $x<e^{-\theta/\tau}$ where $\theta=(\delta*t)$ and $\tau=Q$.

Note: x is a generated random number (value between 0 to 100 for LMC or 0 to 1 for EMC and EMCQ) .

Table 2: A List Of Hyper-Heuristics With Their Acceptance Criteria.

5. Low Level Heuristic

The low level heuristics are implemented based on the problem domain. A set of *simple* LLHs provides more flexibility for the hyper-heuristic. A set of *complex* LLH, such as steepest descent that finds the best neighbour is computationally expensive and indirectly influences the hyper-heuristic to behave as a steepest descent method (for example). This results in simple hyper-heuristics (such as AM) and more complex hyper-heuristics (such as EMC and EMCQ) producing similar results when using a *complex* set of LLH. This is due to the fact that the *complex* set of LLH is able to find good solutions without having to be guided by a hyper-heuristic. Of course, it takes a lot longer to implement *complex* LLH when the problem domain changes. Therefore, if we are dealing with a set of *complex* LLHs, it may be worth applying a simple hyper-heuristic such as an AM hyper-heuristic. However, in this work we prefer to use a set of *simple* LLHs with an intelligent hyper-heuristic as this allows us to solve a wider range of problems.

Our LLH is a set of 2-opt operations. There are many factors involved in determining the efficiency of pick-place operations of multi head placement machine such as the grouping of PCB points (also referred to as placement points) to a sub tour, nozzle assignment, pickup and placement sequencing etc. As the robot arm is equipped with a number of nozzles, the problem is to determine the sets of PCB points that will be visited by the robot arm (i.e. to place a component) in a same route (or a sub tour). A sub tour consists of a set of pickup and placement point that will be visited by robot arm in a tour. For example if the robot arm is equipped with 8 nozzles, we may have 8 pickup points and 8 placement points in a sub tour. As the sub tour of the robot arm begins by picking up a number of component (from the feeders) sequentially, then travels in X-Y direction concurrently for placing components onto the PCB, these incur a number of scheduling problems. These are:

i. Assigning the pickup's nozzle. The robot arm has a number of nozzles. In this work we assume all components are the same size (we ignore the nozzle size selection). The issue is to determine which nozzle should be used to pickup a component such that we minimise the robot arm traveling distance. We must ensure that the PCB points will receive the correct component type. Therefore, if nozzle *A* picks up component type X and nozzle B picks up component type Y, the nozzle A must place component X at a placement point which is expecting a component of type X in the sub tour (similarly with nozzle B). However, if both nozzle A and B pick up a component type X then the sub tour scheduling is easier as either nozzle A or B can be used to place X at a relevant point on the PCB. As the nozzles are located at a fixed position at the end of heads, the cost of picking up the next component is dependent on the nozzle used, the current location of the head and the current nozzle used to pick up the current component.
ii. Sequencing the components pickups. The problem is to determine the sequence of picking up components in a sub tour to optimise the pickups.
iii. Sequencing the placement operation. The problem is to determine the sequence of placing components in a sub tour to optimise the placements.
iv. Assigning the placement's nozzle. Again, the issue is to optimise the placement and we must ensure that the PCB points will receive the correct component type.
v. Assigning PCB points to a sub tour.
vi. Sequencing the sub tours. The aim is to optimise the sequence of sub tours in order to minimise the CT.

The aim of optimising the pickups and the placements is to minimise the CT. However, there is a trade-off between optimising the pickups and optimising the placements. Applying the hyper-heuristic over of a set of LLH simplifies the problem such that we do not have to compromise between optimisation of picking and placing.

On the contrary, in solving this problem, a typical meta-heuristic approach has to intelligently deal with the trade-off between optimising the picking and placement. A meta-heuristic approach is also forced with a decision as to which sequence of factors need to be minimised and how far each factor should be minimised.

However, the problem can be simplified by applying a hyper-heuristic over a set of LLHs. In this work, we develop six *simple* LLHs:

H1: Swap the pickup sequence in a sub tour. Two randomly selected points from a sub tour, say i^{th} and j^{th} pickup sequence in k^{th} sub tour are swapped. The nozzle assignments remain the same.

H2: Swap the placement sequence in a sub tour. Two points are randomly selected from a sub tour, say i^{th} and j^{th} placement sequence in k^{th} sub tour are swapped. The nozzle assignments remain the same. The pickup sequence and the placement sequence in a sub tour are independent operations.

H3: Swap the pickup nozzles in a sub tour. The nozzle's used in the two points are randomly selected from a sub tour, say i^{th} and j^{th} pickup sequence in k^{th} sub tour are swapped. If the swapping operation involves two different component types, then we modify the appropriate placement nozzle in the sub tour such that the PCB points will receive the correct component type.

H4: Swap the placement nozzles in a sub tour. The nozzle's used for two points are randomly selected from a sub tour, say i^{th} and j^{th} placement sequence in k^{th} sub tour are swapped. If the swapping operation involves two different component types, then we modify the appropriate pickup nozzle in the sub tour such that the component will be picked with the correct nozzle.

H5: Swap the PCB points among the sub tours. Two points are randomly selected from different sub tours, say i^{th} and j^{th} placement sequence in k^{th} and l^{th} sub tour, respectively, and are swapped.

If the swapping operation involves two different component types, then we modify the appropriate pickup component in the appropriate sub tour such that the pickup components are valid in both sub tours.

H6: Swap the sub tour sequence order. Two sub tours are randomly selected, say i^{th} and j^{th} and are swapped.

6. Experiments And Results

An initial solution is generated using either a randomised or an ordered constructive heuristic that we proposed in [12]. As in [12], we assume that the gap between the feeder carrier and the PCB is 10 unit length, the nozzle's gap is equal to the size of feeder slot (chosen as 4 unit length) and all allocated components are the same size (that is we ignore the nozzle size selection problem). We assume that all components use the same nozzle type and the speed of robot arm is constant for all component types. We further assume that the placement machine can only pickup one component at a time but the number of components that can be pickup in a sub tour is dependent on the number of nozzles per head (in this test the number of nozzle, G=8). In our formulation, we consider that the robot makes a positive traveling distance when it moves in increasing X or Y direction and a negative traveling distance otherwise. Since we model the same placement machine as in [12], we also apply the same experimental parameters.

To simulate the pick-place operation of the placement machine, we set the speed of the robot arm, V=10 unit distance/unit time, the pickup and placement time, $\lambda = \theta = 0.5$ unit time. For the purpose of generating the random placement points, we will set the length and the width of the PCB, such that the random PCB points fall within the limits. In the experiment we use two data sets (data set N80K20_A and N240K40_F). Data set N80K20_A has 80 PCB points (N) consisting of 20 component types (K) with board width, BW=200 and board length, BL=600. Data set N240K40_F has N=240, K=40, BW=600 and BL=1800. These data sets are randomly generated using our random PCB generator software called PCBgen. PCBgen allows the user to set the required N, K, BW and BL. The data sets and PCBgen software are available at http://www.cs.nott.ac.uk/~mxa/.

We ran the experiments using an Intel® Pentium®4 PC with a 1.5GHz processor and 256 MB RAM. In this work we set *M=5 (for LMC), α=1.0, β=0.01 and σ=0.5 (for OICF and AMCF).* The parameter values are chosen based on the best result obtained from our preliminary test on parameter sensitivity. The parameter sensitivity's test shown that the LMC, OICF and AMCF performance are very sensitive to their parameters and the best value for the parameters is subject to the problem size. Table 3 shows the experimental results of the average of ten runs on data set N80K20 and N240K40 with each run being given one hour of computation time as a termination criteria. However, any other termination criteria also applicable. For example, if we apply these methods for an adaptive scheduling that we proposed in [12] we can continually search for an improved schedule until there are no more PCB's to be assembled. The initial CT for data set N80K20_A and N240K40_F is 1061.04 and 8385.98 unit time, respectively. The figures in Table 3 show the average of the best obtained solution's qualities with the iteration number (when the best solution is found) and the computation time. The percentage of the CT's improvement (Δ) is computed as:

$$\Delta = (\text{Initial CT-best CT})*100/\text{Initial CT}$$

The results in Table 3 show that the OI hyper-heuristic, that only accepts an improved solution (i.e. typical random descent method), rapidly converges to a local optima. For example, in data set N80K20_A, the OI hyper-heuristic gets trapped in local optima in 2.58 minutes and cannot find a better solution even after one hour. However, for the larger size data set, N240K40_F, the OI hyper-heuristic can still improve the solution (not yet in local optima). In all tests, the OICF [7] hyper-heuristic does not perform well, being even worse than OI hyper-heuristic for data set N80K20_A. This indicates that the historical performance and the time taken by the LLH to produce a neighbour solution are not applicable in this case study. This is because the performance of the LLH is unpredictable since it generates a random neighbour every time it is called and historical performance counts for nothing. In fact, the LMC, OI, EMC and EMCQ hyper-heuristics that randomly call the LLH are superior to OICF and AMCF, with the EMCQ hyper-heuristics being superior to the other hyper-heuristics (for data set N80K20_A). This indicates that the formulation of an exponential acceptance criteria is effective in searching for better solutions. The exponential formulation produces a lower probability of acceptance for higher search times and worse evaluation, such that it is adequate to guide the direction of the hyper-heuristic. Injecting a counter of consecutive unimproved solutions into the EMCQ formulation give a significant impact in guiding the search direction and it also provides a way to diversify the search when we trapped in local optima.

	Data Set N80K20_A				Data Set N240K40_F			
	I	CT	T	Δ(%)	I	CT	T	Δ(%)
AM	717566.6	720.15	31.43	32.13	163458	7602.68	25.76	9.34
LMC	1239791.2	266.8	50.2	74.85	488035.7	2350.204	58.38	71.97
EMC	605639.6	281.84	26.96	73.44	360348.5	2350.62	55.84	71.97
EMCQ	1090483.9	248.8	45.8	76.55	388236	2370.3	59.22	71.73
OICF	243268.0	342.98	10.20	67.68	175331.3	2968.45	27.00	64.60
AMCF	290887.5	804.3	18.2	24.20	33307.6	8116.27	5.31	3.22
OI	1310363.2	288.98	54.39	72.76	378998	2771.10	57.72	66.96

Note: I =No of iteration; CT=Assembly cycle time(unit time);
Δ=CT's Improvement; T=computation time(minutes).

Table 3: An Average Result Of Ten Runs On Each Data Set (Test Duration: 1 Hour)

This work is actually a continuation of our previous work in [12,13] where we employ a real-time or adaptive schedule to continually search for an improved solution while the placement machine is picking up and placing components onto the PCB. These approaches require a fast searching technique that is capable of finding good quality solutions in short timescales. Thus, we examine the performance of the hyper-heuristics after five minutes to identify a good hyper-heuristic technique that is able to operate over a shorter timescale. These results are shown in Table 4 which demonstrate that the LMC, EMC and EMCQ hyper-heuristics also perform well in short timescales. These results demonstrate that the EMCQ is a good and fast hyper-heuristic approach as well as operating well over longer time periods (Table 3). Moreover the EMCQ is a parameter free heuristic. The exact behavior of the hyper-heuristics can be observed in Figure 5.

	Data Set N80K20_A				Data Set N240K40_F			
	I	CT	T	Δ(%)	I	CT	T	Δ(%)
AM	70943.0	750.83	5.0	29.24	31625	8873.95	5	-5.82
LMC	123531.4	317.2	5.0	70.10	41556.5	3169.99	5	62.20
EMC	109080.1	332.54	5.0	68.66	32214.8	3291.70	5	60.75
EMCQ	97842.4	307.2	5.0	71.05	32381.1	3197.7	5	61.87
OICF	118716.6	347.67	5.0	67.23	32385.8	4279.74	5	48.97
AMCF	131991.2	862.5	9.6	18.71	3490.6	8269.00	5	1.39
OI	120173.8	322.63	5.0	69.59	32660	3346.14	5	60.10

Table 4: An Average Result Of Ten Runs On Each Data Set (Test Duration: 5 Minutes)

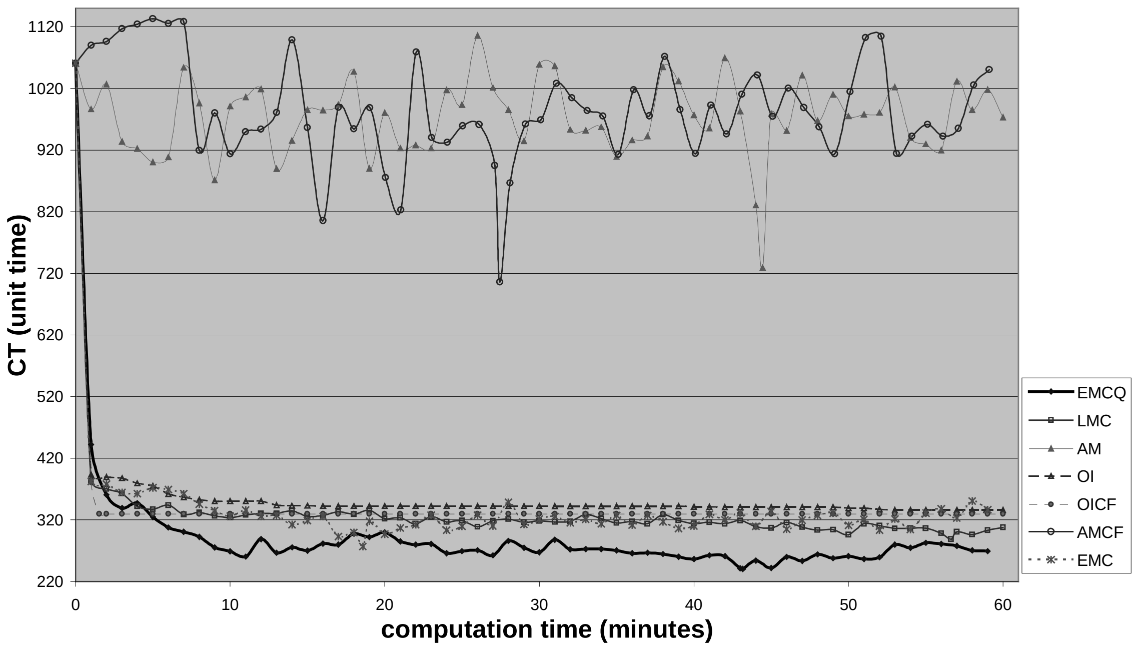

Figure 5. A comparison of hyper-heuristics behaviour

Figure 5 show that the AMCF is the worst among the other hyper-heuristics. This indicates that the heuristic's selection based on Choice Function criteria does not perform well in this case study. The OI and OICF hyper-heuristics get trapped in local optima whilst the LMC, EMC and EMCQ continually find improved solutions. As the performance of the OI and OICF hyper-heuristics are almost the same in this case study, we can conclude that the heuristic's selection in OICF is arbitrary.

7. Conclusion

We have demonstrated the ability of hyper-heuristics approaches in solving the component placement sequencing problem of multi head SMD placement machine. We examined seven hyper-heuristic approaches, these being AM, OI, OICF, AMCF, LMC, EMC and EMCQ. The OICF and AMCF are the hyper-heuristics that choose the next LLH to be applied based on previous performance, whilst the other hyper-heuristics randomly select the next LLH to be called. Since all the LLHs produce a random neighbour solution each time they are called, their performance is unpredictable. Thus, hyper-heuristics based on previous LLH's performance (OICF and AMCF) are unable to perform well. On the other hand, the other hyper-heuristic demonstrated a good performance (except AM hyper-heuristic), especially the LMC and EMCQ hyper-heuristics. For example, for data set N80K20_A, the EMCQ minimised the CT by 76.55% respectively (with respect to the initial solution). Generally, the LMC and EMCQ shows almost equal performance in this case study, but the EMCQ has a better formulation which include the intensification (time, t) and diversification (counter for consecutive unimproved, Q) factors. Moreover, the EMCQ is a parameter free heuristic whereas LMC is sensitive to the parameter, M. Therefore, the EMCQ is a good, fast, robust and parameter free heuristic. In future, we plan to explore these hyper-heuristics methods with steepest descent, random descent and other intelligent low level heuristics. We also believe that these methods may also work in other problem domains which have similar characteristics.

Acknowledgements

This work has been supported by the Public Services Department of Malaysia (JPA) and the National University of Malaysia (UKM).

References

[1] Bentzen, B. SMD placement. The SMT in FOCUS, Nov. 28, 2000, url:http://www.smtinfocus.com/PDF/SMD_placement.pdf (Sept. 25, 2002)

[2] Kazaz, B. and Altınkemer, K., Optimization of multi-feeder (depot) printed circuit board manufacturing with error guarantees. European Journal of Operational Research, 150(2), pp.370-394, 2003.

[3] Broad, K., Mason, A., Rönnqvist, M. and Frater, M. Optimal robotic component placement. J. Operational Research Society, 47, pp.1343—1354, 1996.

[4] Khoo, L.P. and Ng, T.K. A genetic algorithm-based planning system for PCB component placement. J. Production Economics, 54, pp.321-332, 1998.

[5] Glover, F. and Laguna, M., Tabu search, ch. 3. In: Reeves, C. R.(ed) Modern heuristic techniques for combinatorial problems, McGraw-Hill, pp.70-150, 1995.

[6] Cowling, P., Kendall, G. and Soubeiga, E., Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. Proc. of 2nd European Workshop on Evolutionary Computation in Combinatorial Optimisation, EvoCOP2002, Springer LNCS, pp.1-10, 2002.

[7] Cowling, P., Kendall, G. and Soubeiga, E., A hyperheuristic approach to scheduling a sales summit, In Burke, E. and Erben, W., editors, Selected Papers Of The Third International Conference On The Practice And Theory Of Automated Timetabling, PATAT'2000, Springer Lecture Notes in Computer Science, pp.176-190, 2001.

[8] Burke E., Hart E., Kendall G., Newall J., Ross P. and Schulenburg S. Hyper-Heuristics: An emerging direction in modern search technology, ch. 16. In: Glover, F. and Kochenberger, G. (ed) Handbook of Meta-Heuristics. Kluwer, pp.457-474, 2003.

[9] Cowling, P., Kendall, G. and Han, L., an investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. Proc. of Congress on Evolutionary Computation, CEC2002, Hawaii, May 12-17, pp.1185-1190, 2002.

[10] Ross, P., Schulenburg, S., Marín-Blázquez, J.G. and Hart, E., Hyper-heuristics: Learning to combine simple heuristics in bin-packing problem. Proc. of the Genetic and Evolutionary Computation Conference, GECCO 2002, Morgan Kauffmann, pp.942-948, 2002.

[11] Schulenburg, S, Ross, P., Marín-Blázquez, J.G. and Hart, E., A hyper-heuristic approach to single and multiple step environments in bin-packing problems. Proc. of the Fifth International Workshop on Learning Classifier Systems 2002, IWLCS-02.

[12] Ayob M. and Kendall G. An investigation of an adaptive scheduling for multi headed placement machines. Proc. of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications, MISTA 2003, Nottingham, UK, pp.363-380, 13-16 Aug 2003.

[13] Ayob, M. and Kendall, G., Real-time scheduling for multi headed placement machine. Proc. of the 5th IEEE International Symposium on Assembly and Task Planning, ISATP'03, Besançom, France, pp.128-133, 9-11 July 2003.

[14] Ayob, M., Cowling, P. and Kendall, G., Optimisation for surface mount placement machines, Proc. of the IEEE ICIT'02, Bangkok, pp.498-503, 11-14 Dec. 2002.

[15] Reeves, C.R. and Beasley, J.E., Introduction, ch. 1. In: Reeves, C. R.(ed) Modern heuristic techniques for combinatorial problems, McGraw-Hill, pp.1-19, 1995.