A Local Search Algorithm for Large-Scale MCM Substrate Testing

Keisuke MURAKAMI

Department of Industrial and Systems Engineering, Aoyama Gakuin University
5–10–1 Fuchinobe, Chuo-ku, Sagamihara, Kanagawa 252–5258, Japan
E-mail: murakami@ise.aoyama.ac.jp

Abstract
A multi-chip module substrate is designed for packing two or more semiconductor chips. On a substrate, there may exist two types of faults; one in a wiring and the other at an intersection of two wires, called via, and the faults must be detected. The testing is performed by a pair of probes, which touch the two ends of an inter-chip wiring. The completion time of the whole testing is the time when it has been confirmed that no fault exists on the substrate. A two-probe routing problem is considered to design efficient routes of the two probes for minimizing the completion time of the whole testing. In this paper, the two-probe routing problem is formulated as a constrained shortest path problem, and a heuristic algorithm by local search techniques is proposed. Computational experiments demonstrate that the proposed heuristic algorithm outperforms a known two-phase heuristic.

Key words: MCM Substrates Testing, Two-Probe Routing Problem, Local Search Algorithm

1. Introduction

Multi-chip module (MCM) substrates are designed for packing two or more semiconductor chips. Many different chip dies are connected through a wired interconnect on these substrates. The end of a wired interconnect is called a pin, the intersection of two wires is called a via, a part of an interconnecting wire between a via and a pin (or another via) is called a wire segment and a set of pins, vias and wire segments that are to be electrically connected is called a net (Fig.1). MCM technology provides dense assembly capability and increased chip packaging at relatively low cost. Because of the high-density assembly, MCM has some advantages in terms of power consumption, package volume, and so on. However, high-density assembly causes faults on MCM substrates. These faults are classified as open faults, short faults, near-open faults, and near-short faults. An open fault is an electrical disconnection between two points (pins) that are to be connected. A short fault is an electrical connection between two different nets that are not to be connected. A near-open fault is nearly a wire break, but not an electrical disconnection; hence, this fault causes a high-resistance connection. A near-short fault is an insufficient spacing between nets (pins); thus, this fault may result in a short fault. In order to detect these faults, we measure the resistance or capacitance in a wire, and the measured value shows which faults, if any, are present in the wire.

For dense assemblies such as MCMs, the checking of these faults is too time-consuming. An efficient method for checking faults therefore needs to be established. Electron-beam testing and probe testing have been proposed for fault detection. Electron-beam testing injects a charge into the nets and then scans the nets. The advantage of this method is that it is adaptable to many substrate layouts without shifting the hardware. In addition, electron-beam testing avoids a large number of mechanical contacts between the testing equipment and the substrate. This method may therefore be used on sensitive pins and fragile substrates. However, the vacuum-based electron-beam system has the drawbacks of long testing times...
In the probe testing, two probes simultaneously touch two pins, and the open faults of wires between the two pins are checked. The open faults also exist at the vias, which are cracked. All these cracked-vias might not be detected even if all faults of wires are checked. For example in Fig. 1, if via $r_1$ is cracked, we can detect the fault of the via while the faults of wires are checked. This is because when three wire segments which are adjacent to via $r_1$ are checked, the fault of via $r_1$ is certainly detected. Thus, we need not check the fault of via $r_1$. For the same reason, we also need not check the faults of vias $r_3$ and $r_4$. On the other hand, if via $r_2$ is cracked such as Fig. 1, we might not detect the fault while the faults of wires are checked, because, if we check the four wire segments which are adjacent to via $r_2$ by touching simultaneously $p_1$ and $p_3$, and $p_4$ and $p_5$, this fault of via cannot be detected. Therefore, if a via is adjacent to four wire segments, we need an extra check in order to detect the fault of the via; this makes it more difficult to find an efficient routes of two probes. Note that it is quite unlikely that there are the vias that are adjacent to more than four wire segments.

1.1. Related works and our contributions

Several previous studies have proposed probe-testing methods. Yao et al. proposed a method using a two-phase approach with two probes. The first phase produces a set of pairs of pins (for short, pair set) and the pair set checks all the faults. In the second phase, the routes of the two probes are designed. The second phase corresponds to solving the traveling salesman problem (TSP). Yao et al. extended their method from two probes to $k$ probes. Kahng et al. proposed a method for producing minimum pair sets (the pair set size is minimum) in the first phase of the two-phase method. Also, Chou et al. proposed an approach that iterates the two-phase method and gradually improves the solution. However, we think that two-phase methods are inadequate because it is too difficult to produce appropriate pair sets in the first phase. When we produce an inappropriate pair set in the first phase, the final solution is bad, even if we obtain an optimal route for the probes in the second phase.

We therefore propose a new approach. We model a probe testing problem as a constrained shortest path problem (CSP). The CSP is $NP$-hard as it reduces to a TSP when two probes need to touch “all” pairs of pins in order to check the faults, where each pair is regarded as a node of TSP (the details will be given later). Then, we use a local search technique for
solving large-scale problem. We expect that our local search outperforms the existing two-phase method even for large-scale substrates.

In this paper, the number of probes is two and the substrate testing problem is named the “two-probe routing problem”. The objective of the two-probe routing problem is to minimize the completion time for testing. The main contribution of this paper is that we propose a novel approach (model and heuristic algorithm) to large-scale problem. This paper is organized as follows. Section 2 and 3 describe the two-probe routing problem and model the two-probe routing problem as a CSP. In Section 4, we propose a heuristic algorithm based on local search. In Section 5, we compare the results obtained using our CSP approach with the existing two-phase method. Section 6 concludes this paper.

2. Problem description

We describe the two-probe routing problem as follows:

1) The testing detects the open faults of wires and (cracked-)vias.
2) An MCM substrate is aggregation of nets. Then, a net is a tree and an MCM substrate is a forest.
3) The features of two probes are same exactly, so two probes are not distinguished.
4) In a net, two probes simultaneously touch two pins, respectively and then the wire segment(s) between the two pins is(are) checked. Let pair(a, b) denote a pair of pins a and b, touched simultaneously by two probes. Let wire(a, b) be the wire between pair(a, b); wire(a, b) implies the path between pin a and b on the tree (net).
5) The type of cracked-via fault we must check is only one, such as Fig.2 (b), because the others, such as Fig.2 (c), can be detected while faults of wires are checked.
6) In order to check cracked-via faults at via r, we define four sets of pins as P1, P2, P3 and P4 such as Fig. 2 (a), where each of P1, P2, P3 and P4 represents a set of pins connecting with each side of a via. For example, at via r2 in Fig.1, P1 = {p1, p2}, P2 = {p3}, P3 = {p5, p6, p7} and P4 = {p4}. In addition, we assume that P1 faces P3 and P2 faces P4; this implies the wire(a ∈ P1, c ∈ P3) and wire(b ∈ P2, d ∈ P4) intersect at the via. Then, at least one of the following conditions must be satisfied in order to check the fault of a via.
6)-a) When some pairs of pins are touched, the via must be crossed by the wires between at least three of the pairs. For example, in the left side of Fig.3, when pair(a ∈ P1, b ∈ P2), pair(a ∈ P1, d ∈ P2) and pair(c ∈ P3, d ∈ P4) are given, the via is crossed by the wires between these three pairs. Note that even if two pairs are given from same two sets such as pair(a ∈ P1, b ∈ P2) and pair(a′ ∈ P1, b′ ∈ P2), we consider that via r is crossed by the wire between only one pair, not two pairs.
6)-b) When some pairs of pins are touched, the wires between the two of the pairs must intersect at the via. For example, in the right side of Fig.3, when pair(a ∈ P1, c ∈ P3) and pair(b ∈ P2, d ∈ P4) are given, the wire(c, d) and wire(b, d) intersect at the via.
7) Two probes must check all wire segments and all vias that are adjacent to four wire segments on an MCM substrate.
8) Probes are allowed to travel from a net to another one before the testing of a net is completed.
9) Testing start and termination points (pins) of each probe are fixed and the pair of start and termination pins is same.

10) The objective is to minimize the completion time of testing and we assume that a probe’s travel time is equal to the Euclidean distance.

![Fig. 3 Detection of fault of via](image)

3. Model

In this section, we model the two-probe routing problem as a CSP. We generate “all pairs of pins” in each net. We consider each pairs of pins to be a (pseudo) node and let V be the set of the nodes (pairs of pins). Let E be a directed edges set and we define \( E = \{(i, j) \in V \times V \} \).

The CSP network is represented by a directed graph \( G = (V, E) \). Let \( S \) be the set of wire segments and \( R \) be the set of vias. When a node (pair of pins) is visited, the faults of some wire segments and vias are checked. The objective of the CSP is to minimize the completion time for the testing and there are constraints to ensure that the faults of all wire segments (\( \forall s \in S \)) and all vias (\( \forall r \in R \)) are checked.

We define the travel times between two (pseudo) nodes; that is the travel times of two probes between two pairs of pins. Two probes simultaneously touch two pins, respectively. If one probe arrives at a pin before the other probe arrives at the other pin, one probe must wait till the other probe arrives at the other pin. Hence, the travel times are computed as follows.

Let \( p^1_i \) and \( p^2_i \) be the two pins of pair \( i \) and let \( t(k, h) \) be the travel time from pin \( k \) to pin \( h \). There are only two possible paths for the two probes (Fig. 4 (a)). Path 1 is that one probe travels from \( p^1_i \) to \( p^1_j \), and the other probe travels from \( p^2_i \) to \( p^2_j \). Path 2 is that one probe travels from \( p^2_i \) to \( p^1_j \), and the other probe travels from \( p^1_i \) to \( p^2_j \). If Path 1 is used, the travel time between pair \( i \) and \( j \) is expressed by \( \max\{t(p^1_i, p^1_j), t(p^2_i, p^2_j)\} \). If Path 2 is used, the travel time is expressed by \( \max\{t(p^2_i, p^1_j), t(p^1_i, p^2_j)\} \).

Since the two probes are not distinguished, the travel time between \( i \) and \( j \) is expressed by

\[
c_{i,j} = \min\{\max\{t(p^1_i, p^1_j), t(p^2_i, p^2_j)\}, \max\{t(p^2_i, p^1_j), t(p^1_i, p^2_j)\}\} \tag{1}
\]

Equation (1) implies that the travel time \( c_{i,j} \) is the smaller of the travel times for the two paths. Then, each travel time between two pairs of pins is regarded as a cost between the two (pseudo) nodes (Fig. 4 (b)). Hence, the two probes can be transformed into a (pseudo) probe; this makes the two-probe routing problem much easier to solve, because we have only to find the path of a (pseudo) probe instead of two probes. This is the main advantage of modeling as the CSP. Note that all nodes might need not to be visited in the CSP; though all pins must be touched in the original two-probe routing problem. The feasible path implies a path from the start to termination points and satisfies all constraints (checks the faults of all wire segments and all vias).

4. Algorithm

We propose a local search algorithm for the CSP and the outline is described in subsection 4.1. We also present the method for improving a feasible solution in subsection 4.2. Furthermore, we discuss the size of the neighborhood in our local search algorithm in subsection 4.3.
4.1. Local Search Algorithm

For the CSP, we propose a solution method based on local search. Our solution method first generates an initial feasible path by using the existing method (Fig.5 (a)). The feasible path can check the faults of all wire segments and all vias. Next, we repeatedly ruin and recreate a part of the feasible path in order to improve the cost. In the ruin phase, we choose a part of the feasible path and remove it. Then, a tentative path is generated when one end of the part of the feasible path is directly connected to the other (Fig.5 (b)). The tentative path is not feasible, because it cannot check the faults of all wire segments and all vias. In recreate phase, we insert nodes into the tentative path one by one until the tentative path develops into a feasible one (Fig.5 (c)).

If the faults of all wire segments are completely checked, the fault of each via can be checked by at most one node. This is because the nodes checking the faults of all wire segments contribute to the check for each via at least twice. For example in the left side of Fig.3, \( \text{pair}(a \in P_1, b \in P_2) \) and \( \text{pair}(c \in P_3, d \in P_4) \) are touched when the faults of the wire segments are checked, we have only to touch one pair \( \text{pair}(a \in P_1, d \in P_4) \) or \( \text{pair}(b \in P_2, d \in P_4) \) to check the faults of the via. This makes it easy to determine the nodes which are inserted into the tentative path, because we need not confirm the complicated conditions (i.e. 6)-a), 6)-b) in Section 2). Therefore, we first complete the check for the faults of the wire segments, and then we check the faults of the vias.

Let \( S' \subseteq S \) be a set of wire segments which are not checked by the tentative path, and let \( S'_i \subseteq S' \) be the set of wire segments which are checked by node \( i \). Let \( R' \subseteq R \) be a set of vias which are not checked by the tentative path, and let \( R'_i \subseteq R' \) be the set of vias for which node \( i \) contributes to the check (i.e. node \( i \) implies \( \text{pair}(a, b) \), \( \text{pair}(a, d) \) or \( \text{pair}(c, d) \) in 6)-a), or \( \text{pair}(a, c) \) or \( \text{pair}(b, d) \) in 6)-b) of Section 2). We first enumerate the nodes which can check wire segments \( s \in S' \) and/or contribute to the check for vias \( r \in R' \), and let \( V' \) denote the set of the nodes; we call \( V' \) a set of candidate nodes for inserting into the tentative path. Let \( V' \) be the set of nodes on the tentative path. We assume that each node \( i \) in \( V' \) is inserted and we calculate an increment cost by the following equation:

\[
d_i = \min_{j \in V'} c_{ij} + c_{i,k} - c_{j,k} \quad \forall i \in V'
\]

where node \( i \) is inserted between node \( j \) and \( k \); \( j \) is the previous node of \( k \) on the tentative path. When we consider the check for the faults of wire segments, we calculate the evaluation value of node \( i \) by the following equation:

\[
e_i = \frac{|S'_i|}{d_i} \quad \forall i \in V'
\]

We actually insert a node with largest evaluation value into the tentative path. With respect to the faults of vias, we similarly determine the node which is inserted into the tentative path by using the following equation:

\[
e'_i = \frac{|R'_i|}{d_i} \quad \forall i \in V'
\]
Equations (3) and (4) are similar to an evaluation value of a greedy algorithm for the set-covering problem. The greedy algorithm for the set-covering problem (SCP)\(^{12}\) first has been proposed by Chvatal. We briefly explain this algorithm below. Let \( x_i \) be 0-1 decision variable. Let \( D_i \) be the covered set if \( x_i \) is one and let \( cost_i \) denote the cost associated with variable \( x_i \). We find the variable with largest evaluation value; that is \( x_i' = \arg \max_{x_i} |D_i|/cost_i \), and then the variable \( x_i' \) is set to one. This is repeated until the covering constraints are satisfied. This heuristic is likely to find a good feasible solution quickly. Furthermore, for unweighted set-covering problem (i.e. for all \( x_i \), \( cost_i = 1 \)), Slavik showed that the approximation ratio of the greedy algorithm is \( \ln n - \ln \ln n + \Theta(1) \), where \( n \) is the number of elements which must be covered\(^{13}\). Feige proved that the unweighted set-covering problem cannot be approximated within a ratio of \( (1 - \epsilon) \ln n \) for any \( \epsilon \) if \( NP \) does not have \( n^{O(\log \log n)} \) times deterministic algorithms\(^{14}\). Raz and Safra proved that unless \( P = NP \) for a certain constant \( c \), the unweighted set-covering problem cannot be approximated within a ratio of \( c \log n^{1/5} \). These results show that the greedy algorithm is an asymptotically best possible approximation algorithm for the (weighted) set-covering problem\(^{16}\).

For the weighted \( k \)-set covering problem, the greedy algorithm proposed by Chvatal is an \( H_k \)-approximation algorithm, where \( H_k = 1 + 1/2 + 1/3 + \cdots + 1/k \). Hallodrossn proposed an approximation algorithm within a ratio of \( H_k = 1/3 \)\(^{17}\). Duh and Furur further presented an \( (H_k - 1/2) \)-approximation algorithm\(^{18}\). For the weighted \( k \)-set covering problem, Fujito and Okamura proposed \( (H_k - 1/12) \)-approximation algorithm where weights are 1 or 2. Hassin and Levin presented \( (H_k - (k - 1)/8k^3) \)-approximation algorithm for general weighted \( k \)-set covering problem\(^{16}\).

We repeatedly ruin and recreate a feasible path until a termination condition is satisfied. The local search algorithm is described as follows. Let \( \sigma^* \) denote a current feasible path and \( \sigma' \) be a tentative path:

**LOCAL SEARCH (LS)**

Step 0 Find an initial feasible path, which is set to the current feasible path \( \sigma^* \).

Step 1 Choose a part of \( \sigma^* \) and remove it from \( \sigma^* \) temporarily and generate a tentative path \( \sigma' \).

Step 2 Find \( V^* \) and \( V' \), and obtain \( S'_i \) and \( R'_i \) for all \( i \).

Step 3 Compute \( d_i = \min_{j \in V'} |c_{ji} + c_{i,k} - c_{j,k}| \) and \( e_i = |S'_i|/d_i \) for all \( i \in V' \).

Step 4 Obtain \( i^* = \arg \max_{i \in V'} e_i \).

Step 5 Insert node \( i^* \) into the tentative path \( \sigma' \), then update \( V^* \) and \( V' \), and update \( S'_i \) and \( R'_i \) for all \( i \).

Step 6 If \( \sigma' \) checks the faults of all wire segments, go Step 7; otherwise, return to Step 3.

Step 7 Compute \( d_i = \min_{j \in V'} |c_{ji} + c_{i,k} - c_{j,k}| \) and \( e'_i = |R'_i|/d_i \) for all \( i \in V' \).

Step 8 Obtain \( i^* = \arg \max_{i \in V'} e'_i \).

Step 9 Insert node \( i^* \) into the tentative path \( \sigma' \). If \( \sigma' \) develops into a feasible path, let \( \tilde{\sigma} \) be the feasible path and remove redundant nodes from \( \tilde{\sigma} \), then go to Step 10; otherwise, update \( V^* \), \( V' \) and \( R'_i \) for all \( i \in V' \), then return to Step 7.

Step 10 If the cost of \( \tilde{\sigma} \) is smaller than that of \( \sigma^* \), set \( \sigma^* \) to \( \tilde{\sigma} \).
Step 11. If the termination condition is satisfied, exit; otherwise, return to Step 1.

Step 0 finds an initial feasible path by using existing two-phase method. In Step 1, a tentative path is generated, and then we find each set in Step 2. In Steps 3-9, we determine a node and insert the node into the tentative path; this is iterated until the tentative path develops into a feasible path. In Step 9, we also remove redundant nodes; the detail is explained in the subsection 4.2. Step 10 updates the current feasible path. Finally, the LS outputs a feasible path.

4.2. Removal of redundant nodes

In Step 9 of the LS, we remove the redundant nodes from the feasible path \( \hat{\sigma} \), because several redundant nodes might be included in \( \hat{\sigma} \). We also propose the method for the removal of redundant nodes; this method removes redundant nodes one by one. Let \( \hat{V} \) denote the set of nodes on \( \hat{\sigma} \). We enumerate critical nodes from \( \hat{V} \). If the faults of all wire segments and vias are not checked by a node \( i \in \hat{V} \), the node \( i \) is a critical node. Then, let \( \text{Cr}(\hat{\sigma}) \subseteq \hat{V} \) denote the critical node set in feasible path \( \hat{\sigma} \) and let \( M_{\text{Cr}(\hat{\sigma})} \subseteq S' \cup R' \) be the set of wire segments and vias checked by \( \text{Cr}(\hat{\sigma}) \). Also let \( \bar{\text{Cr}}(\hat{\sigma}) \) denote the complement of \( \text{Cr}(\hat{\sigma}) \). The critical nodes \( \text{Cr}(\hat{\sigma}) \) cannot be removed because if one of the critical nodes is removed, all \( m \in S' \cup R' \) are not checked. So, we select some removal nodes from \( \bar{\text{Cr}}(\hat{\sigma}) \). Note that we can preliminarily remove the node checking only subset of \( M_{\text{Cr}(\hat{\sigma})} \), from \( \bar{\text{Cr}}(\hat{\sigma}) \), if such nodes exist, because the node is determinately redundant. Since it might take a long time to select the optimal removal nodes, we use the following method to find removal nodes quickly.

We compute saving costs of each node in \( \text{Cr}(\hat{\sigma}) \) in order to decide which node to remove. The saving cost of a node implies decrement when that node is removed. The saving cost of node \( i \in \text{Cr}(\hat{\sigma}) \) is computed by

\[
f_i = c_{k,i} + c_{i,h} - c_{k,h} \quad \forall i \in \text{Cr}(\hat{\sigma})
\]

where \( k \) and \( h \) represent the previous and following nodes of node \( i \) on the feasible path \( \hat{\sigma} \). The node that has the largest saving cost is then removed. After we remove a redundant node, we connect the previous node to the following of the removed node. This is how the feasible path \( \hat{\sigma} \) is updated. Then, \( \text{Cr}(\hat{\sigma}) \) and \( \bar{\text{Cr}}(\hat{\sigma}) \) are also updated. This is iterated until \( \text{Cr}(\hat{\sigma}) \) is empty. The method for the removal of redundant nodes is described as follows.

**REMOVAL OF REDUNDANT NODES (RRN)**

Step 0. Initialize \( \text{Cr}(\hat{\sigma}) \) and \( \bar{\text{Cr}}(\hat{\sigma}) \).
Step 1. Compute \( f_i = c_{k,i} + c_{i,h} - c_{k,h} \) for all \( i \in \bar{\text{Cr}}(\hat{\sigma}) \), where \( k \) and \( h \) are the previous and following nodes, respectively, of node \( i \).
Step 2. Set node \( j' \leftarrow \arg \max \limits_{j \in \text{Cr}(\hat{\sigma})} f_j' \).
Step 3. Remove node \( j' \) from \( \hat{\sigma} \).
Step 4. Update \( \sigma, \text{Cr}(\hat{\sigma}) \) and \( \bar{\text{Cr}}(\hat{\sigma}) \).
Step 5. If \( \text{Cr}(\hat{\sigma}) = \emptyset \), exit; otherwise, return to Step 1.

In Step 1, we compute the saving costs. In Step 2 and 3, we find the node that has the largest saving cost and that node is removed. Step 4 updates the feasible path \( \hat{\sigma} \), the set of critical nodes \( \text{Cr}(\hat{\sigma}) \) and that complement \( \bar{\text{Cr}}(\hat{\sigma}) \). Finally, the RRN outputs the feasible path \( \hat{\sigma} \) where no redundant nodes are included.

4.3. How to choose a part of a current path

In this subsection, we discuss how to choose a part(s) of a current path in Step 1 of the LS. We call the part of a current path “sub-path”. The neighborhood within our LS depends on the sub-path(s). Let \( z = (z_1, z_2, ..., z_E) \) denote 0-1 decision variables associated with edges, where if edge \( l \in E \) is included in the current feasible path, \( z_l = 1 \) otherwise, \( z_l = 0 \). We
define the neighborhood within our LS as the solution set obtained by flipping \( q \) variables in \( z \). Then, the size of the neighborhood is \( \binom{|E|}{2q} \times 2^q \), where \( \binom{|E|}{2q} \) implies the combinatorial number when we choose \( q \) variables in \( z \), and \( 2^q \) represents the number of solutions using \( q \) variables. Therefore, the size of the neighborhood is influenced by \( q \). Furthermore, the number of variable \( q \) depends on the size of the candidate nodes \( V' \), and \( |V'| \) is almost proportional to the size of sub-path(s). So, it is important for our LS to choose appropriate sub-path(s). Also, we should consider the costs of edges between the candidate nodes, where smaller costs are preferred because they are likely to improve the current path effectively. With consideration for these, we choose sub-paths which consist of the nodes (pairs of pins) belonging to “a net”.

In general, when \( |V'| \) is larger, the computation time of the LS is also larger. When we choose sub-paths included in a net, the size of \( V' \) is smaller. For example, in Fig. 6, the nodes (pairs of pins) are generated as follows:

\[
\begin{align*}
v_1 &= \text{pair}(p_1, p_2), \quad v_2 = \text{pair}(p_1, p_3), \quad v_3 = \text{pair}(p_4, p_3), \quad v_4 = \text{pair}(p_5, p_3), \\
v_5 &= \text{pair}(p_5, p_8), \quad v_6 = \text{pair}(p_5, p_6), \quad v_7 = \text{pair}(p_5, p_7), \quad v_8 = \text{pair}(p_5, p_4), \\
\end{align*}
\]

where a current feasible path is expressed by \( v_1 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow v_7 \rightarrow v_8 \). If we choose a single sub-path \( v_3 \rightarrow v_4 \); both nodes belong to “a” net, the set of candidate nodes \( V' \) is given by \( \{v_1, v_2, v_4, v_5, v_6, v_7, v_8\} \). On the other hand, when we choose a sub-path \( v_4 \rightarrow v_5 \); these nodes belong to two “different” nets, the set of candidate nodes \( V' \) is represented by \( \{v_1, v_2, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}\} \). That is, when two nodes of the sub-path belong to “a” net, \( |V'| = 6 \), and when two nodes of the sub-path belong to “different” net, \( |V'| = 10 \). Hence, we basically choose a net and then obtain a sub-path(s). However, if \( |V'| \) is too small, we choose more (two or three) nets because the current path might not be improved. (In our computational experiments, if \( |V'| \) is less than one hundred, we choose one more net located near the net which has already been chosen.)

![Fig. 6 Two nets](image)

Also, the current feasible path is improved more effectively, because the costs between the nodes belonging to a net are likely to be smaller. In addition, our method might choose multiple sub-paths where the nodes are close to each other, such as Fig. 7 (a). If a single sub-path is always chosen, the nodes which are far away from each other on the current feasible path cannot be interchanged, even though the nodes are close to each other in the space defined by equation (1). On the other hand, our method can interchange such nodes. For example, in Fig. 7, though node \( v_i \) is close to node \( v_6 \) and \( v_5 \) in the space defined by equation (1), node \( v_i \) is far away from \( v_5 \) and \( v_6 \) on the path. If we choose a single sub-path, node \( v_i \) is not likely to be inserted between \( v_5 \) and \( v_6 \). However, if we choose multiple sub-paths, node \( v_i \) might be inserted between \( v_5 \) and \( v_6 \) (Fig. 7 (b)). We expect that our choice method of sub-paths is appropriate for LS.

5. Computational experiments

We compare our LS with the two-phase method (TPM). In the first phase of the TPM, we enumerate some sets of pairs of pins (nodes) which can check the faults of all wire segments and all vias in each net. Then, a set of pairs of pins is selected in each net. In the second phase, we solve a TSP, where the nodes are equivalent to the sets of pairs of pins (nodes) selected...
in the first phase, and the cost is computed using equation (1). The TSP is then solved using the nearest neighbor and 2-opt heuristics\(^{19}\). Finally, we remove the redundant pairs of pins (nodes) from the solution by using the RRN, which is presented in subsection 4.2. The TPM is iterated until a termination condition is satisfied; we call this method “Iterated TPM”. We find the best in the solutions obtained by using Iterated TPM. Also, a solution to the TPM (not Iterated TPM) is used as an initial solution of our LS.

The LS and TPM are implemented in C++ using STL vector. All tests were performed on a 2.5 GHz Core i5 with a Windows operating system using 2 GB of RAM memory. Note that none of the implementations used multi-cores.

We prepare two kinds of nets with 4 pins, which are not topological equivalence. We name the nets “net4.4” and “net4.5”, respectively (Fig. 8). The latter number represents the number of wire segments. Note that even if multiple net4.4 (net4.5) are used on an instance, their configuration (location of pins) are not same in spite of topological equivalence. In a similar way, we prepare “net6.7”, “net6.8”, “net8.10”, “net8.11”, “net10.13” and “net10.14”.

We generate seven instances by combination of net4.4, ..., net10.14 at random; they are named “ins100”, “ins400”, “ins900”, “ins1600”, “ins2500”, “ins3600” and “ins4900”, respectively. The latter numbers 100, 400, ..., 4900 represent the number of nets in each instance. The data of each instance is shown in Table 1, where “#pin” represents the number of pins in the instance and the data under #pin expresses the ratio of the net in the instance.

<table>
<thead>
<tr>
<th>ins</th>
<th>100</th>
<th>400</th>
<th>900</th>
<th>1600</th>
<th>2500</th>
<th>3600</th>
<th>4900</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pin</td>
<td>682</td>
<td>2,792</td>
<td>6,276</td>
<td>11,182</td>
<td>17,424</td>
<td>25,104</td>
<td>34,216</td>
</tr>
<tr>
<td>4.4</td>
<td>17%</td>
<td>14%</td>
<td>14%</td>
<td>13%</td>
<td>13%</td>
<td>13%</td>
<td>12%</td>
</tr>
<tr>
<td>4.5</td>
<td>16%</td>
<td>15%</td>
<td>13%</td>
<td>13%</td>
<td>13%</td>
<td>13%</td>
<td>13%</td>
</tr>
<tr>
<td>6.7</td>
<td>10%</td>
<td>11%</td>
<td>12%</td>
<td>13%</td>
<td>13%</td>
<td>13%</td>
<td>13%</td>
</tr>
<tr>
<td>6.8</td>
<td>10%</td>
<td>10%</td>
<td>12%</td>
<td>13%</td>
<td>13%</td>
<td>13%</td>
<td>13%</td>
</tr>
<tr>
<td>8.10</td>
<td>11%</td>
<td>12%</td>
<td>11%</td>
<td>11%</td>
<td>12%</td>
<td>12%</td>
<td>12%</td>
</tr>
<tr>
<td>8.11</td>
<td>9%</td>
<td>11%</td>
<td>12%</td>
<td>12%</td>
<td>12%</td>
<td>13%</td>
<td>13%</td>
</tr>
<tr>
<td>10.13</td>
<td>18%</td>
<td>14%</td>
<td>13%</td>
<td>13%</td>
<td>13%</td>
<td>12%</td>
<td>13%</td>
</tr>
<tr>
<td>10.14</td>
<td>9%</td>
<td>14%</td>
<td>13%</td>
<td>13%</td>
<td>13%</td>
<td>12%</td>
<td>12%</td>
</tr>
</tbody>
</table>

The LS is terminated if the solution is not improved #pin times in a row. The TPM is iterated until the computation time reaches that of the LS; that is, the computation time of the Iterated TPM equals to that of the LS. We show an initial solution obtained by performing the TPM once, and the solution obtained from the LS and the Iterated TPM. The (initial) solutions are improved using the LS, and then when termination condition is satisfied, the solutions are finally found. The “cost” and “time” represent the objective function and the computation...
time (seconds). Note that the computation times of the LS and the Iterated TPM are the same, as described above.

We also show the rate of improvement of the LS from the initial solution; the "rate of improvement" is computed as follows:

\[
\text{rate of improvement} = \frac{(\text{Initial solution cost}) - (\text{LS cost})}{(\text{LS cost})} \times 100\% \tag{6}
\]

where Initial solution cost represents the cost of initial solution and LS cost implies the cost obtained from the LS. The rate of improvement means the effect of the LS, that is, when the rate of improvement is large, our LS performs well.

In a similar way, we show cost gap between the LS and the Iterated TPM; the “cost gap” is computed as follows:

\[
\text{cost gap} = \frac{(\text{Iterated TPM cost}) - (\text{LS cost})}{(\text{LS cost})} \times 100\% \tag{7}
\]

where Iterated TPM cost represents the cost obtained from the Iterated TPM. When the cost gap is larger, our LS is superior to the Iterated TPM.

<table>
<thead>
<tr>
<th>ins</th>
<th>Initial solution</th>
<th>LS</th>
<th>Iterated TPM</th>
<th>rate of improvement</th>
<th>cost gap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cost</td>
<td>time</td>
<td>cost</td>
<td>cost</td>
<td>time</td>
</tr>
<tr>
<td>100</td>
<td>2,323</td>
<td>1.45</td>
<td>1,836</td>
<td>2,251</td>
<td>2.32</td>
</tr>
<tr>
<td>400</td>
<td>9,631</td>
<td>2.34</td>
<td>7,610</td>
<td>9,301</td>
<td>20.01</td>
</tr>
<tr>
<td>900</td>
<td>21,526</td>
<td>4.66</td>
<td>16,876</td>
<td>20,976</td>
<td>125.46</td>
</tr>
<tr>
<td>1600</td>
<td>38,242</td>
<td>9.89</td>
<td>30,068</td>
<td>37,513</td>
<td>125.46</td>
</tr>
<tr>
<td>2500</td>
<td>59,778</td>
<td>20.40</td>
<td>46,496</td>
<td>58,477</td>
<td>1,104.13</td>
</tr>
<tr>
<td>3600</td>
<td>85,688</td>
<td>41.93</td>
<td>67,695</td>
<td>84,475</td>
<td>1,512.34</td>
</tr>
<tr>
<td>4900</td>
<td>116,536</td>
<td>85.30</td>
<td>91,506</td>
<td>114,962</td>
<td>5,284.25</td>
</tr>
</tbody>
</table>

In Table 2, the costs obtained using our LS are much smaller than those of initial solutions and those obtained using the Iterated TPM. The rate of improvements of all instances are more than 26%; even for larger-scale instances the rate of improvement are not worsening. The cost gaps of all instances are more than 22%; they are likely to be larger when the scales of instances are larger. When the scales of instances are larger, the computation times also are larger; this is because the sizes of sub-path are almost same in any instance and more sub-paths are generated in larger-scale instances. The computation times are practical even for large-scale instances. These results show that our LS performs well and it considerably outperforms the Iterated TPM, and our LS is useful even for large-scale instances with respect to cost and computation time.

6. Conclusion

We modeled an MCM substrate testing problem as a CSP. We proposed a solution method for the CSP, based on local search. In this local search, we applied the Chvatal’s greedy heuristic algorithm for SCP. Then, the local search was repeated until the termination condition was satisfied. Computational experiments showed that our local search outperformed the existing two-phase method in all instances (ins100,...,ins4900). Even for large-scale instances, our local search performed well within practical time. Therefore, our solution method is useful for large-scale MCM substrate testing problem.

References


