Hybrid Floorplanning Based on Partial Clustering and Module Restructuring

Takayuki Yamanouchi, Kazuo Tamakashi, and Takashi Kambe
Precision Technology Development Center, Sharp Corporation

Abstract

In this paper, we propose a hybrid floorplanning methodology. Two hierarchical strategies for avoiding local optima during iterative improvement are proposed: (1) Partial Clustering, and (2) Module Restructuring. These strategies work for localizing nets connecting small modules in small regions, and conceal such small modules and their nets during the iterative improvement phase. This method is successful in reducing both area and wire length in addition to reducing the computational time required for optimization. Although our method only searches slicing floorplans, the results are superior to the results obtained even with non-slicing floorplans.

We applied our method to the largest MCNC floorplan benchmark example, ami49, and industrial data. For the ami49 benchmark, we obtained results superior to any published results for both estimated area and routing results.

1 Introduction

Recently, floorplanning has become more complex. This is due to the recent use of hardware description languages and synthesis tools which increase the number and variation in size of modules, and nets.

Many approaches to floorplanning have been proposed, and they can generally be classified into two categories, constructive and iterative improvement methods.

Constructive methods divide the problem into two steps. In the first step, they determine the relative placement among modules subject to minimizing wire length. Partitioning based methods determine relative placement by top-down partitioning [1,2,3,4]. Graph based methods are another approach for relative placement [3,5,6]. In the second step, module sizing is applied to minimize the area [2,3,4,7,8]. These methods are successful in handling a large number of modules, but the relative placement in the first step restricts optimization of floorplan.

Iterative improvement methods optimize wire length and area simultaneously. Wong et al. [9] proposed a normalized Polish expression representation for slicing structures [10]. Iterative improvement methods, such as simulated annealing or genetic algorithms, are formulated using this representation [9,11]. Murata et al. [12] proposed the Sequence-Pair representation which has higher flexibility than the slicing structure. Their approach has been shown to be successful in area optimization for large problems. However, because iterative improvement methods search large solution spaces, they can be very time-consuming. Moreover, we have found that the solution potentially falls into local optimum. We will discuss this problem in detail later.

Onodera et al. [13] proposed a branch-and-bound placement method. Theoretically, this can obtain an optimal solution. However, for practical problems, hierarchical partitioning must be applied before the branch-and-bound placement because the applicable number of modules is limited to six. Consequently, this method still depends on the initial partition.

In this paper, we propose a hybrid floorplanning approach based on hierarchical strategies. These strategies work for localizing nets in small regions, and also work for avoiding undesirable solutions for iterative improvement. Experimental results demonstrate that our method improves the quality of resulting floorplans substantially.

The paper is organized as follows. Section 2 gives an overview of the method. Sections 3, 4 and 5 then describe our partial clustering, cluster placement, and module placement methods. Section 6 presents some experimental results. We conclude in Section 7.

2 Overview of the Method

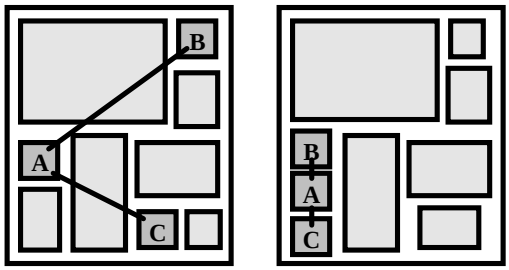
2.1 Motivation

For an iterative improvement method, a cost function must be defined for evaluating the current floorplan. The cost function depends on both area and total wire length in order to optimize both of them. However, we found a serious problem in which the solution potentially falls into a local optimum even with hill-climbing methods, such as simulated annealing. Fig. 1 illustrates this problem in the

ICCAD '96
1063-6757/96 $5.00 © 1996 IEEE

case of simulated annealing. Suppose that module **A** is tightly connected with **B** and **C**. Let us consider that the annealing temperature goes down and the area has been optimized. Once the current floorplan becomes Fig. 1 (a), it is necessary to increase the area temporarily for the later improvements if we expect the reduction of the wire length. However, at low temperatures, the increase of area cost is too high to accept, and thus the solution falls into a local optimum while there exist better solutions with the same area but shorter wire length (Fig. 1(b)).

The same kind of problem arises from the difference in module size, which is one of the distinctive characteristics of large floorplans. At high temperatures, placement improvements for small modules can be canceled later because of the wasted area caused by large modules. On the other hand, at low temperatures, it is necessary to move large modules temporarily for optimizing small module placement, but this improvement is difficult to accept.



(a) (b)

Fig. 1 Example of a local optimum solution

2.2 Basic Concepts

Based on the above discussions, it can be seen that iterative improvement will produce a much better solution if we prevent it from falling into undesirable solutions. By putting tightly connected small modules into a limited region, we can prevent them from being spread too widely, and the wires connecting them will be accordingly shortened (Fig. 2). Furthermore, differences in module size will be reduced. This operation will effectively avoid undesirable solutions for iterative improvement, and also reduce computational time.

For this purpose, we introduce two hierarchical strategies. One is *partial clustering*, and the other is *module restructuring*.

Partial clustering is applied to only small modules, and it takes difference in module size into account. The number of modules in a cluster is restricted.

Module restructuring consists of two operations, *merge* and *split*. The *merge* operation combines two modules. The *split* is the inverse operation which splits a merged module in two. We can apply *module restructuring* to modules used in cell based layout styles. During iterative improvement, modules are restructured for optimization.

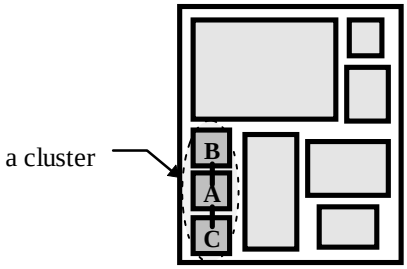


Fig. 2 Example of partial clustering

2.3 Outline of Our Method

Our floorplanning method consists of the following three stages.

1 Partial clustering
2 Cluster placement
3 Module placement

In stage 1, tightly connected small modules are clustered. In stage 2, we enumerate possible shapes for each cluster, then determine the placement of clusters. This variation in cluster shape increases the flexibility of cluster placement. During cluster placement, we use module restructuring operations. After cluster placement, we determine placement of all modules at stage 3 in which we focus on wire length.

The floorplan model in our method is a *slicing structure* [10]. Wong et al. proposed a *normalized Polish expression* to represent such structures [9]. In this representation, operands correspond to modules, and operators (+ or *) correspond to (horizontal or vertical) rectangular bisections of the region (Fig. 3). We can represent a cluster or merged module by changing a part of an expression.

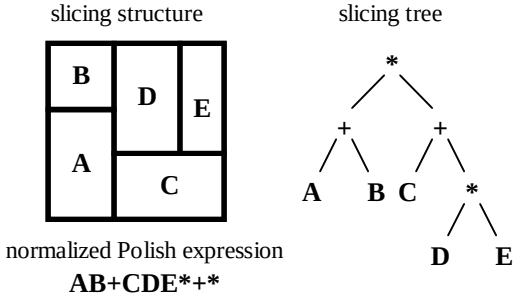


Fig. 3 Slicing structure and normalized Polish expression

3 Partial Clustering

Conventional methods of clustering focus mainly on the number of connections, either among clusters or within a cluster. It is difficult to optimize area in a cluster because of the differences in module size (Fig. 4). To solve this problem, we introduce a new connectivity that takes module sizes into account.

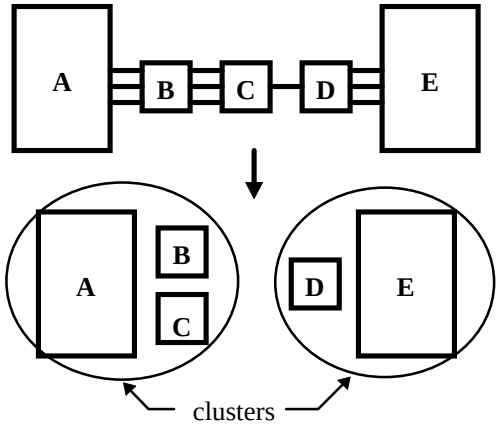


Fig. 4 Problem of the conventional clustering

3.1 Selection of Clustering Candidates

Our clustering strategy is to create a cluster of small modules whose sizes are smaller than a given threshold. The larger modules are handled individually, because they will widely influence the whole placement region.

3.2 Connectivity

Connectivity C_{ij} between two modules M_i and M_j is defined in eq. 1,

$$C_{ij} = \sum_{n \in N_{ij}} \frac{1}{(T_n - 1)} \bullet \frac{A_{all}}{A_i + A_j} \bullet \frac{\min\{A_i, A_j\}}{\max\{A_i, A_j\}} \quad \text{(eq. 1)}$$

where A_i and A_j are the areas of modules M_i and M_j respectively, A_{all} is the total area of all modules, T_n is the number of pins in net n, and N_{ij} is the set of nets connecting module M_i and M_j. In this definition, conventional connectivity is biased according to the module sizes and difference in module size.

3.3 Clustering Algorithm

For simplicity of explanation, we call each of the excluded modules a cluster (a single module cluster). We specify the maximum number of modules in a cluster. Two clusters with the highest connectivity are combined, and this operation is repeated. Fig. 5 shows the clustering algorithm.

4 Cluster Placement

In this section, we describe the cluster placement method. Possible shapes for each cluster are enumerated before cluster placement.

4.1 Enumeration of Cluster Shapes

We enumerate cluster shapes using a constructive method. Lengauer et al. [2] proposed a dynamic programming method to compute a shape function for a set of modules. Based on this method, we obtain a shape function that includes all possible slicing floorplans of a cluster (Fig. 6). As a result, the cluster has various potential

M is a set of modules $\{M_1, M_2, \ldots, M_n\}$
A_i is area of module M_i
A_{max} is the area threshold for clustering
P_i is a cluster i, which consists of a set of modules
N_{max} is the maximum number of modules in a cluster
C_{ij} is a connectivity between cluster P_i and P_j
/* selection of clustering candidates */
foreach module M_i in ***M*** **do**
  $P_i = \{M_i\}$
  if ($A_i < A_{max}$) **then**
    flag(P_i) = T
  else
    flag(P_i) = F
  end if
end do
/* clustering */
while (**exists** cluster pair (P_i, P_j) such that
  flag(P_i) == T && flag(P_j) == T && $|P_i| + |P_j| <= N_{max}$
) **do**
  Update connectivity between clusters.
  Choose (P_m, P_n) with highest connectivity.
  Create $P_l = P_m$ ¾P_n, and remove P_m and P_n.
end do

Fig. 5 Clustering algorithm

shapes, and this flexibility improves the area optimization of cluster placement. The calculation takes only a few seconds if the number of modules is less than 10.

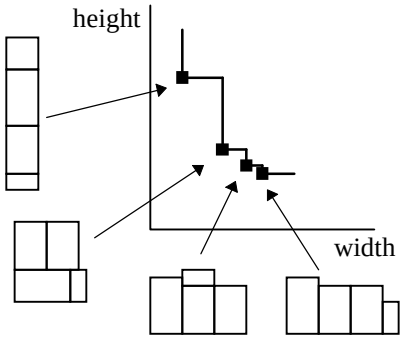


Fig. 6 Possible placements and shape function of a cluster

4.2 Iterative Improvement for Cluster Placement

For iterative improvement, we use a simulated annealing algorithm. Wong et al. applied three operations to a *normalized Polish expression* for modifying the relative placement [9].

[M1] Swap two adjacent operands
[M2] Complement some chain of non-zero length
[M3] Swap two adjacent operand and operator

We use a modified **[M1]** operation.

[M1'] Swap any two operands

We extend their approach for module restructuring. If we replace a part of an expression with a new operand, we

can handle a set of modules as a new module. We denote the operands as A, B, C and the operators as *, +. The operands A and B represent the modules that can be merged. We define three merge operations.

[m1] Replace the sequence (A B [*,+]) with (C)
[m2] Replace the sequence (A [*,+] B) with (C)
[m3] Replace the sequence ([*,+] A B) with (C)

We also define three split operations which are the inverse of the merge operations.

[s1] Replace (C) with the sequence (A B [*,+])
[s2] Replace (C) with the sequence (A [*,+] B)
[s3] Replace (C) with the sequence ([*,+] A B)

The operand C must be a merged module which was created from A and B by an earlier merge operation. If the balloting property [9] of the normalized Polish expression is invalid after [s1] or [s3], then the operation is canceled. Fig. 7 illustrates floorplan modifications based on these module restructuring operations.

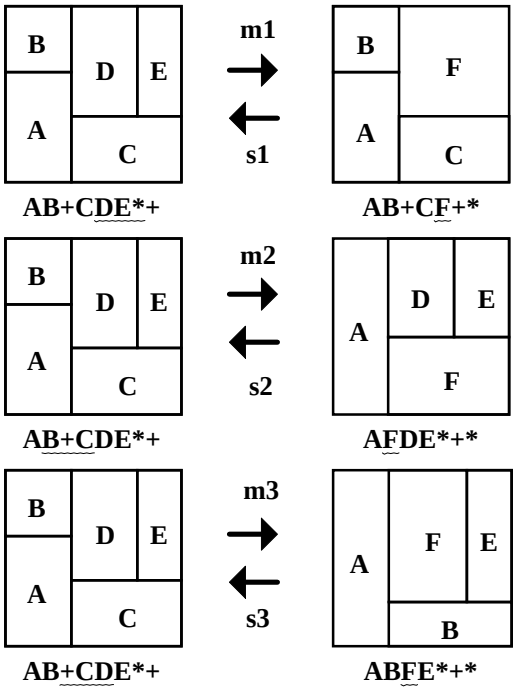


Fig. 7 Examples of module restructuring operations

During each iteration, one of the above operations is randomly applied to the expression, then a shape function of the floorplan is calculated. All locations and shapes of operands are determined for the shape with the minimum area floorplan, and the wire length is obtained. We use the same cost function as used in [9].

5 Module Placement

In the previous stage, the location and orientation of modules are not fixed yet. The purpose of this stage is to determine these subject to reducing the wire length.

5.1 Relative Placement in a Cluster

Because the cluster placement stage has optimized the area and wire length among clusters, in this stage we focus on the wire length within clusters. After cluster placement, we build a slicing tree based on the cluster shape. All information for building this has been already obtained during the enumeration of cluster shapes as described in section 4.1. If a cluster has *n* modules, then there are 2^{n-1} placements whose shapes are same, and we can choose the placement with the minimum wire length. We enumerate the placements by exchanging left and right children of each operand node of the slicing tree (Fig. 8).

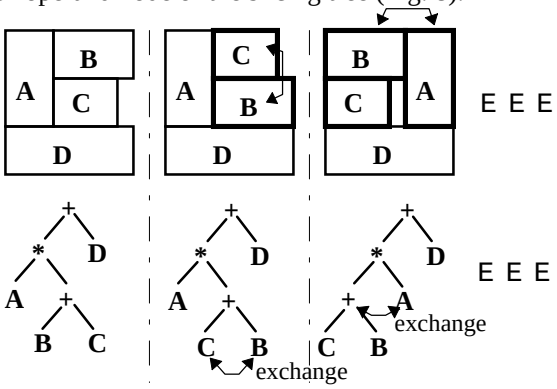


Fig. 8 Enumeration of placement in a cluster

5.2 Module Orientation

Finally we determine the module orientations. A cluster has four possible orientations which represent same relative placement. They are (1) no reflection, (2) reflection about the X axis, (3) reflection about the Y axis, and (4) reflection about both the X and Y axis (Fig. 9). Similarly, each module has 4 possible orientations.

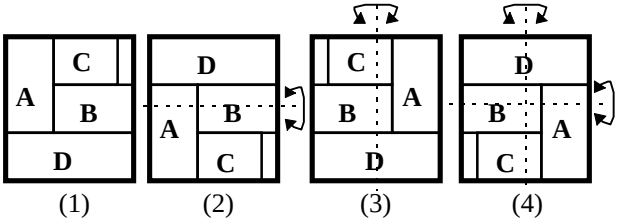


Fig. 9 Possible orientations of a cluster

We determine module orientations in a step by step manner. For each cluster, we determine the cluster orientation, and then determine the orientation of the modules in the cluster.

At the evaluation of cluster orientation, we assume that pins are at the center of their module since the module orientation is still flexible. We evaluate four cluster orientations based on the wire length, then we fix module placement in a cluster. At the evaluation of module orientation, we calculate the wire length with exact pin locations. We show this algorithm in Fig. 10.

```
Initialize pin locations to center of their cluster
foreach cluster P_i do
   /* determination of cluster orientation */
   foreach cluster orientation do
      foreach module M_j in P_i do
         Update pin locations to the center of module M_j.
      end do
      Calculate wire length associated with P_i.
   end do
   Select the best orientation.
   Update module locations in P_i.
   foreach module M_j in P_i do
      /* determination of module orientation */
      foreach module orientation do
         Update pin locations of M_j exactly.
         Calculate wire length associated with M_j.
      end do
      Select the best orientation.
   end do
end do
```

Fig. 10 Algorithm for determining module orientations

6 Experimental Results

We implemented our method, and applied it to the largest MCNC floorplan benchmark example, ami49, and an industrial example.

6.1 Results for Benchmark Data ami49

In the case of ami49, all the modules are rigid, so we only applied partial clustering. The threshold of module size for clustering was twice the average size of the modules, and the number of modules in a cluster was limited to 7. The target aspect ratio of chip was 1.

At cluster placement, the total number of clusters was 14, which contained 7 single module clusters.

Initially, we evaluated the effect of partial clustering. Table 1 shows the results with and without partial clustering. We estimate the wire length with half perimeter of net bounding boxes. In the table, area does not include wiring area. Run time is on a Sun Sparc Station 20. The result shows that partial clustering is effective for both area and wire length optimization, besides being effective in reducing computational time. We show the plot of this result in Fig. 11.

	Width (˚)	Height (˚)	Area (mm^2)	Wire length (mm)	Time (sec)
W.O. clustering	6090	6258	38.11	1030	370.9
With clustering	5824	6440	37.58	723.9	56.3

Table 1

We compared the above results with a Sequence-Pair approach [12] which generates a non-slicing floorplan. Table 2 shows the results. The result for SP was obtained from the authors of [12]. In this case, area includes the estimated wiring area. The estimation method and wire spacing rules are the same as SP's. The spacing rules between wires are 7 micron in both horizontal and vertical directions, which are different from the exact benchmark rules. Our result is smaller than the result for SP, even though we only search slicing floorplans and the solution space is much smaller than SP's.

	Width (˚)	Height (˚)	Area (mm^2)	Wiring area (mm^2)
Ours	6288.4	6796.6	42.72	7.28 (1.00)
SP	6482.0	6925.0	44.89	9.45 (1.30)

Table 2

We also compared our results with the routing result for branch-and-bound (BB) method [13]. This is shown in Table 3. In [13], modules are partitioned before the branch-and-bound operation due to the limitation of the applicable number of modules.

Our routing result is obtained by our channel routing program under the exact design rules of the benchmark. In this case, we assign routing area around modules in proportion to the number of pins, because our placement on the estimated cost is too dense for routing. Our method also generates better results in terms of area. Our wire length is longer than BB, but aspect ratios of the chip are too different for comparison. We show the plot of the routing result in Fig. 12.

	Width (˚)	Height (˚)	Area (mm^2)	Wiring Area (mm^2)	Wire length (mm)
Ours	7108.0	7001.8	49.77	14.33(1.00)	1141
BB	4692.0	10976.0	51.49	16.05(1.12)	1021

Table 3

6.2 Results for an industrial example

Next, we show the effect of module restructuring. We compare the routing result from our algorithm with those from manual design using the industrial data. The initial number of modules is 19, and 10 modules can be merged among 13 flexible modules. This time we did not use partial clustering because the number of modules was small. The results are shown in Table 4. We obtained a better result for area than the manual design. Wire length between modules is also better than the manual design, but this is difficult to compare because the numbers of modules are different. We show the plots of this result in Fig. 13.

	Width (˚)	Height (˚)	Area (mm^2)	Number of modules (at the final floorplan)
Ours	9825.7	9403.1	92.39	11
Manual	9846.6	10391.7	102.32	13

Table 4

7 Conclusion

In this paper, we propose a hybrid floorplanning method based on partial clustering and module restructuring. Experimental results demonstrate the effectiveness of our method. They prove that the hybrid approach improves not only the computational time but also the quality of the resulting floorplan. We also show that the slicing structure still has high flexibility for optimizing floorplans.

References

[1] W. M. Dai and E. S. Kuh, "Simultaneous Floor Planning and Global Routing for Hierarchical Building-Block Layout", IEEE Trans. Computer-Aided Design, vol. CAD-6, no. 5, pp. 828-837, 1987.

[2] T. Lengauer and R. Müler, "Robust and Accurate Hierarchical Floorplanning with Integrated Global Wiring", IEEE Trans. Computer-Aided Design, , no. 6, pp. 802-809, 1993.

[3] P. S. Dasgupta, S. S. Kolay,and B. B. Bhattacharya, "A Unified Approach to Topology Generation and Area Optimization of General Floorplans", in IEEE International Conf. on Computer Aided Design, pp. 712-715, 1995.

[4] P. Pan, W. Shi, and C. L. Liu, "Area Minimization for Hierarchical Floorplans", in IEEE International Conf. on Computer Aided Design, pp. 436-440, 1994.

[5] K. Tani, S. Tsukiyama, I. Shirakawa, and H. Ariyoshi, "Area-efficient drawings of rectangular duals for VLSI floorplan", in Proc. Int. Symp. on Circuits and Systems, pp. 1545-1548, 1988.

[6] S. S. Kolay and B. B. Bhattacharya, "Canonical Embedding of Rectangular Duals with Applications to VLSI Floorplanning", in Proc. 29th ACM/IEEE Design Automation Conf., pp. 69-74, 1992.

[7] T.-C. Wang and D. F. Wong, "Optimal Floorplan Area Optimization", IEEE Trans. Computer-Aided Design, vol. 11, no. 8, pp. 992-1002, 1992.

[8] P. Pan and C. L. Liu, "Area Minimization for Floorplans", IEEE Trans. Computer-Aided Design, vol. 14, no. 1, pp. 123-132, 1995.

[9] D. F. Wong and C. L. Liu, "A New Algorithm for Floorplan Design", in Proc. 23rd ACM/IEEE Design Automation Conf., pp. 101-107, 1986.

[10] R. H. J. M. Otten, "Automatic Floorplan Design", in Proc. 19th ACM/IEEE Design Automation Conf., pp. 261-267, 1982.

[11] J. P. Cohoon, S. U. Hegde, W. N. Martin, and D. S. Richards, "Distributed Genetic Algorithms for the Floorplan Design Problem", IEEE Trans. Computer-Aided Design, vol. 10, no 4, pp. 483-492, 1991.

[12] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-Packing-Based Module Placement", in IEEE International Conf. on Computer Aided Design, pp. 472-479, 1995.

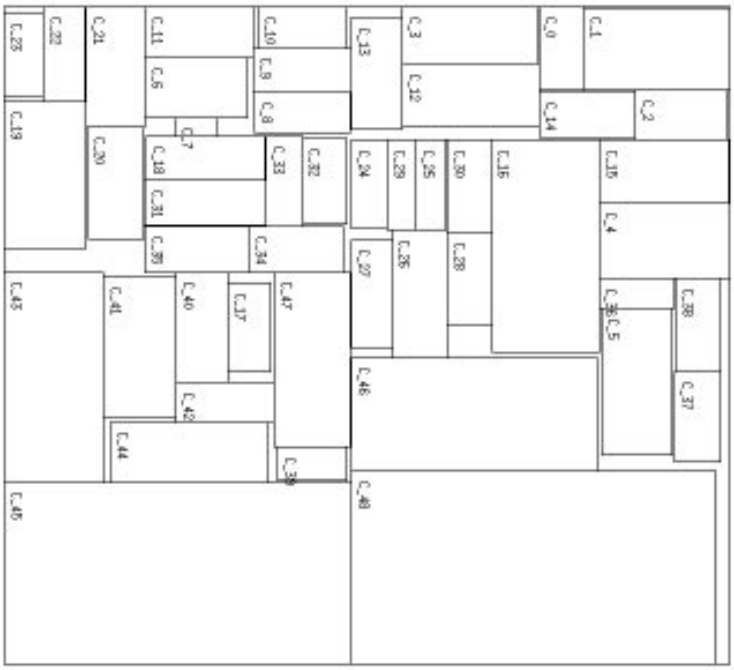
[13] H. Onodera, Y. Taniguchi, and K. Tamaru, "Branch-and-Bound Placement for Building Block Layout", in Proc. 28th ACM/IEEE Design Automation Conf., pp. 433-439, 1991.

Fig. 11 Result of ami49

Fig. 12 Routing result of ami49

merged modules

Fig. 13 Routing result of an industrial example


C_0
C_1
C_2
C_3
C_4
C_5
C_6
C_7
C_8
C_9
C_10
C_11
C_12
C_13
C_14
C_15
C_16
C_17
C_18
C_19
C_20
C_21
C_22
C_23
C_24
C_25
C_26
C_27
C_28
C_29
C_30
C_31
C_32
C_33
C_34
C_35
C_36
C_37
C_38
C_39
C_40
C_41
C_42
C_43
C_44
C_45
C_46
C_47
C_48