VLSI layout synthesis

Citation for published version (APA):
Aarts, E. H. L., Laarhoven, van, P. J. M., Liu, C. L., & Pan, P. (1997). VLSI layout synthesis. In E. H. L. Aarts, & J. K. Lenstra (Eds.), *Local search in combinatorial optimization* (pp. 415-440). Wiley-Interscience.

Document status and date:
Published: 01/01/1997

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 04. Oct. 2023

Local Search in Combinatorial Optimization
Edited by E. Aarts and J. K. Lenstra
© 1997 John Wiley & Sons Ltd.

12

VLSI layout synthesis

Emile H. L. Aarts
Philips Research Laboratories, Eindhoven

Peter J. M. van Laarhoven
Eindhoven University of Technology

C. L. Liu
University of Illinois at Urbana-Champaign

Peichen Pan
Clarkson University, Potsdam

1 LAYOUT SYNTHESIS

Layout synthesis in VLSI chip design refers to the process of transforming the structural specification of a circuit in the form of modules and interconnects to detailed geometrical data and processing information for chip production. The two main tasks in layout synthesis are allocating spaces to the modules to be placed on a chip, and interconnecting points on the modules, called terminals, according to a prescribed netlist. Each net in the netlist specifies a set of terminals that must be electrically connected. The objective in layout synthesis is usually to minimize the total chip area, though nongeometrical issues such as performance and power consumption have recently been considered. Layout synthesis is an extremely complex problem, impossible to tackle as a whole to arrive at an optimal solution. In practice the problem is broken up into several subproblems which are then solved individually. The solutions to the subproblems are combined to yield a solution to the overall problem. The combined solution to the overall problem is most likely suboptimal since the objective in a subproblems can only partially reflect the overall objective. Actually, most subproblems are still too difficult to be solved to optimality. Heuristic algorithms that seek good but not necessarily optimal solution are employed. Typical subproblems in layout synthesis are partitioning, floorplanning/placement, routing, and compaction.

- *Partitioning* is the process of breaking up a large design into several smaller parts. The usual objective is to minimize the number of nets that have terminals in more than one part. Among the many applications of partitioning in layout synthesis, one is to break up a large design so that each resulting part can be efficiently and effectively managed and optimized.
- *Placement* is the process of placing without overlaps modules of fixed sizes on a two-dimensional surface representing the chip. The purpose is to facilitate routing with an overall objective such as minimizing the total chip area. If the modules have flexible sizes and shapes, one often speaks of *floorplanning*. Placement is often classified according to the targeted design methodology, e.g., gate array placement, standard cell placement, and macro cell placement.
- *Routing* is the process of connecting terminals on modules as specified by the netlist while satisfying both physical constraints and design rules. Due to its complexity, routing is conventionally done in two phases, *global routing* and *detailed routing*. Global routing can be viewed as a routing planning phase. Its purpose is to determine the routing regions that each net is going to pass through. After global routing, detailed routing is carried out in each routing region. Detailed routing produces detailed information for realizing the nets passing through a routing region in terms of tracks and vias. Depending on the types of routing regions or routing resources available, detailed routing can be classified into channel routing, switchbox routing, single-layer routing, two-layout routing, etc.
- *Compaction* is the process of further minimizing the layout area of a design without changing the topological relations of the layout components. Compaction is subject to design rules, which are constraints that specify the relative positions of the components associated with a given IC process technology. Compaction is also used to migrate a layout from one processing technology to another.

For more detailed discussions of layout synthesis and various layout problems, we refer the reader to Lengauer [1990] and Preas & Lorenzetti [1988].

Over the years much effort had been invested in designing efficient and effective algorithms for various layout problems. For some of the early work see Soukup [1981]. For more recent work see Kuh & Ohtsuki [1990]. The success of the existing algorithms was, however, far from satisfactory. On the one hand this was because almost all layout problems are NP-hard [Sahni & Bhatt, 1980], which makes it unlikely that optimal solutions can be found in a reasonable amount of time. Therefore, heuristics for finding approximate solutions are often employed. On the other hand it appeared to be very hard to construct efficient and effective approximation algorithms that could handle the many different aspects in layout problems.

The introduction of generally applicable combinatorial optimization methods, such as simulated annealing, tabu search, and genetic algorithms, initiated a wave of research on their application to layout problems. Local search methods are attractive for their flexibility, which helps them to complex problems such as arise in layout synthesis, problems in which too many factors need to be considered. Also attractive is the trade-off they allow between quality of solutions and computation time. And their relative ease of implementation makes them very popular, too. In this chapter we present some work on applying simulated annealing, tabu search, and genetic algorithms to solve several important layout problems.

2 SIMULATED ANNEALING

Simulated annealing was introduced by Kirkpatrick, Gelatt & Vecchi [1983] and independently by Černý [1985] as a general technique for approximately solving combinatorial optimization problems; see Chapter 4.

Simulated annealing can be viewed as a probabilistic version of conventional local search with the additional ability of moving out of local optima. Conventional local search starts with an initial solution. It then searches for a better solution in the neighborhood of the current solution. If a better solution is found, it is accepted as the new current solution. This process is continued until no better solution is found, which terminates the search process. A conspicuous shortcoming of conventional local search is that it terminates at the very first local optimum. Another shortcoming is that the final solution strongly depends on the choice of the initial solution.

Simulated annealing is also based on the idea of exploring the solution space of a problem by moving around in the neighborhood structure. However, there are several pronounced distinctions between conventional local search and simulated annealing. First, simulated annealing randomly generates a solution in the neighborhood of the current solution instead of searching for a better one. Second, whether the generated solution will be accepted is decided by an acceptance criterion that allows both improvements and, in a probabilistic way, deteriorations in the cost function. Deteriorations are initially accepted with a large probability. The probability is slowly decreased to become zero at the end. Such decrease in acceptance probability is accounted for by a control parameter called *temperature*. The higher the temperature, the higher the probability that deteriorations will be accepted. Allowing deteriorations enables simulated annealing to escape from local optima. Also, with high temperature at the beginning, most solutions are accepted. This makes the final result more or less independent of the choice of the initial solution.

We now give the framework of the homogeneous simulated annealing algorithm; this is the simplest algorithm and it is widely used. The temperature is decreased step by step. For each temperature value, it generates a sequence of solutions of fixed length, say L_t for the tth temperature value c_t, following the generation mechanism and the acceptance criterion. Let f denote the objective function, and $\mathcal{N}(i)$ denote the neighborhood of solution i of an optimization problem. (We always assume the smaller the cost the better the solution.) Suppose i is the current solution. Suppose $j \in \mathcal{N}(i)$ is generated by the generation mechanism. j is accepted as the new current solution with the probability computed according to an *acceptance function.* The most common acceptance function is $\exp(-(f(j)-f(i))^+/c)$, where c is the current temperature, and $(f(j)-f(i))^+ = f(j)-f(i)$ if $f(j)-f(i)>0$, and $(f(j)-f(i))^+=0$ otherwise. There are many other acceptance functions [Nahar, Sahni & Shragowitz, 1985]. They usually have the property that a solution with improved cost is always accepted and a solution with deteriorated cost is more likely to be accepted in high temperature than in low temperature. Another important component is the temperature decrement function, which is usually referred to as the *cooling schedule.* The cooling schedule affects both the quality of the solution and the running time. A commonly used decrement function is $c_{t+1} = \alpha c_t$ where α is a positive constant smaller than but close to 1. For other cooling schedules see Aarts et al. [1985, 1986], Hajek [1988], Huang, Romeo & Sangiovanni-Vincentelli [1986], Lam & Delosme [1988b]. Figure 12.1 shows the general framework of the algorithm. For more advanced topics on simulated annealing, we refer the reader to Aarts & Korst [1989a], Faigle & Schrader [1988], Gidas [1985], Van Laarhoven [1988], Van Laarhoven & Aarts [1987], Lundy & Mees [1986], and Mitra, Romeo & Sangiovanni-Vincentelli [1986].

```
begin SA
  select an initial solution $i_1$;
  select an initial temperature $c_1$;
  $t \leftarrow 1$;
  while the stopping condition is not met do
     count ← 0;
     while count < $L_t$ do
        count ← count + 1;
        $i \leftarrow$ a random solution in $\mathcal{N}(i_t)$;
        $i_t \leftarrow i$ with probability $\exp(-(f(i)-f(i_t))^+/c_t)$;
     end while
     $c_{t+1} \leftarrow \alpha c_t$;
     $i_{t+1} \leftarrow i_t$;
     $t \leftarrow t+1$;
  end while
end SA
```

Figure 12.1 Homogeneous simulated annealing algorithm

Application of the algorithm to an optimization problem requires the following items to be specified: a concise representation of the solutions, a neighborhood structure on the solution space, and the cooling schedule. In practice, the algorithm stops when the improvement on the cost function between two consecutive temperature values is small enough. The final output can be either the last solution or the best solution ever observed; see also Chapter 4.

Simulated annealing has been used to tackle problems in layout synthesis. For an overview see Van Laarhoven & Aarts [1987], and Wong, Leong & Liu [1988]. Simulated annealing is considered to be one of the most effective methods for solving many layout problems in terms of the solution quality. As an example, we mention the placement and routing package Timber Wolf by Sechen & Sangiovanni-Vincentelli [1985]. This package has been improved over the years [Sechen, 1985b; Sechen & Lee, 1987; Sechen & Sangiovanni-Vincentelli, 1986; Sun & Sechen, 1993] and is widely used.

We will now discuss the application of simulated annealing to four layout problems: placement, floorplanning, channel routing, and compaction.

2.1 Placement

Here we present a simulated annealing algorithm for the sea-of-gates placement problem. The problem is to assign the modules to the points of a rectangular grid such that the estimated total wire length is minimized.

Let $G=(V,E)$ denote the circuit, where V represents the set of modules and E the set of nets (two terminal nets are assumed). Edges are weighted. A weight on an edge represents the connectivity between the two modules. Let $w(u,v)$ denote the weight of edge $\{u,v\}$. A *placement* is an assignment of the modules to the points of an $N \times M$ rectangular grid, with $NM \geqslant |V|$. Furthermore, let $x_i(u)$ and $y_i(u)$ denote the x and y coordinates, respectively, of module u in placement i.

The application of simulated annealing to this problem is straightforward and can be carried out as follows:

- The solution space is given by the set of all possible placements.
- The cost of placement i is chosen to be

$$f(i)=\sum_{\{u,v\}\in E} w(u,v)\left\{(x_i(v)-x_i(u))^2+(y_i(v)-y_i(u))^2\right\}. \qquad (1)$$

- The neighborhood of a placement is generated by exchanging the modules assigned to two grid points. The difference in cost between two neighboring solution i and j obtained by exchanging modules u and v in i then equals

$$\begin{aligned}\Delta f &= f(j)-f(i)\\ &= \sum_{\{u,z\}\in E_u} w(u,z)\left\{(x_j(z)-x_j(u))^2+(y_j(z)-y_j(u))^2\right\}\\ &\quad -\sum_{\{u,z\}\in E_u} w(u,z)\left\{(x_i(z)-x_i(u))^2+(y_i(z)-y_i(u))^2\right\}\end{aligned}$$

$$+ \sum_{\{v,z\}\in E_v} w(v,z)\{(x_j(z)-x_j(v))^2+(y_j(z)-y_j(v))^2\}$$
$$- \sum_{\{v,z\}\in E_v} w(v,z)\{(x_i(z)-x_i(v))^2+(y_i(z)-y_i(v))^2\}, \qquad (2)$$

where E_u and E_v denote the sets of edges incident on u and v, respectively, and

$$x_j(u)=x_i(v),$$
$$y_j(u)=y_i(v),$$
$$y_j(z)=y_i(z) \text{ and } y_j(z)=y_i(z), \quad \text{for all } z \neq u, v.$$

Calculation of (2) can be done in $O(m)$ time, whereas calculation of (1) requires $O(m^2)$ time, where $m = N \times M$. Hence, (2) allows more efficient calculation of cost differences.

Experiments were conducted to compare the performance of simulated annealing with that of iterative improvement; the results are shown in Table 12.1. Column n lists the numbers of modules for the problems. The table gives the minimum cost W_{opt}, the cost of a random placement W_{r}, the cost of the placement obtained by iterative improvement W_{ii}, and by simulated annealing W_{sa}. The corresponding running times in seconds are t_{ii} and t_{sa}, respectively. The costs for simulated annealing are the averages over five runs with different initial solutions. From the results we conclude that the running times of simulated annealing become prohibitive for the larger problem instances (up to 15 hours for problem p_7). Furthermore, simulated annealing yields substantially better results than iterative improvement, whereas the running time of iterative improvement is only three to four times smaller than for simulated annealing. Considering the quality of the final results and the computation times, it suggests that simulated annealing also outperforms time-equivalent multistart iterative improvement, i.e., iterative improvement repeated a number of times with different initial solutions such that the total running time is about equal to that of simulated annealing.

Table 12.1 Results of sea-of-gates placement by simulated annealing and iterative improvement for seven problems

Problem	n	W_{opt}	W_{r}	W_{ii}	t_{ii}	W_{sa}	t_{sa}
p_1	400	760	81284	7584	202	1256	872
p_2	625	1200	216445	14429	479	2261	2348
p_3	900	1740	454897	23728	1343	3150	5581
p_4	1225	2380	846986	34835	4567	4013	11182
p_5	1600	3120	1465858	56078	9936	5836	20008
p_6	2025	3960	2362427	86753	15289	6793	32483
p_7	2500	4900	4091694	142055	18478	8845	48899

For other placement algorithms based on simulated annealing, see De Bont et al. [1988], Greene & Supowit [1984], Grover [1986], Jepsen & Gelatt [1983], Kim et al. [1993], Kraus & Mlynski [1991], Mallela & Grover [1988], Nagahara et al. [1989], Sechen [1988b], Sechen & Lee [1987], Sechen & Sangiovanni-Vincentelli [1985], Siarry, Bergonzi & Dreyfus [1987], and Sun & Sechen [1993].

One drawback of simulated annealing algorithms is that they are very time-consuming. To alleviate this problem, a number of parallel simulated annealing placement algorithms were proposed: Banerjee & Jones [1986], Casotto, Romeo & Sangiovanni-Vincentelli [1987], Darema-Rogers, Kirkpatrick & Norton [1987], Kravitz & Rutenbar [1987], RaviKumar & Patnaik [1987], Rose et al. [1986], and Wong & Fiebrich [1989].

2.2 Floorplanning

In the floorplanning problem we are given n modules with aspect ratios (height/width) that may vary within limits determined by *shape constraints*. Let $(A_1, r_1, s_1), (A_2, r_2, s_2), \ldots,$ and (A_n, r_n, s_n) be the triples specifying the area A_t and the bounds r_t and s_t of the shape constraints of module t. Furthermore, for $t = 1, \ldots, n$, let h_t and w_t denote the respective height and width of module t. Then it must be the case that

$$w_t h_t = A_t \quad \text{and} \quad r_t \leqslant \frac{h_t}{w_t} \leqslant s_t \quad \text{for} \quad t = 1, \ldots, n.$$

The objective is to place the modules in a rectangular floorplan such that they do not overlap, their aspect ratios satisfy the shape constraints, and a weighted sum of the estimated total wire length and area of the floorplan is minimized.

Wong & Liu [1986] proposed a floorplanning algorithm based on simulated annealing. They restrict themselves to slicing floorplans. A *slicing floorplan* is a rectangle dissection obtained by recursively cutting rectangles into two rectangles (Figure 12.2). A slicing floorplan can be represented by a binary tree, called a *slicing tree*, which describes the way that the floorplan is constructed. The leaves of the tree correspond to the modules. The internal nodes correspond to the cut lines. Figure 12.3 shows the slicing tree for the slicing floorplan in Figure 12.2. If we associate operator $+$ to a horizontal cutline and $\star$ to a vertical cutline in a slicing tree, it becomes an expression tree with the modules as operands. The *Polish expression* of the tree is used to represent the floorplan by Wong & Liu [1987]. For the floorplan in Figure 12.2, $12\star3\star4+6\star578+\star+$ is the corresponding expression, which can be obtained by a postorder traversal of the slicing tree in Figure 12.3.

The simulated annealing algorithm is specified in the following way:

- The solution space is given by the set of the Polish expressions on the modules.
- The cost of solution i is defined as

$$f(i) = A(i) + \lambda W(i),$$

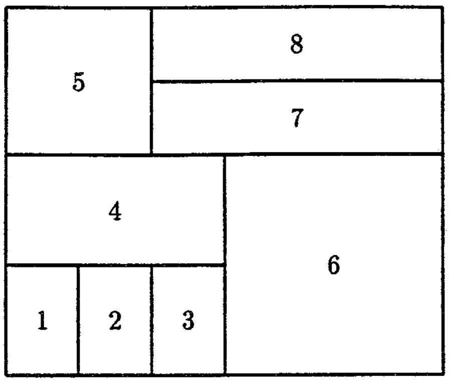


Figure 12.2 A slicing floorplan

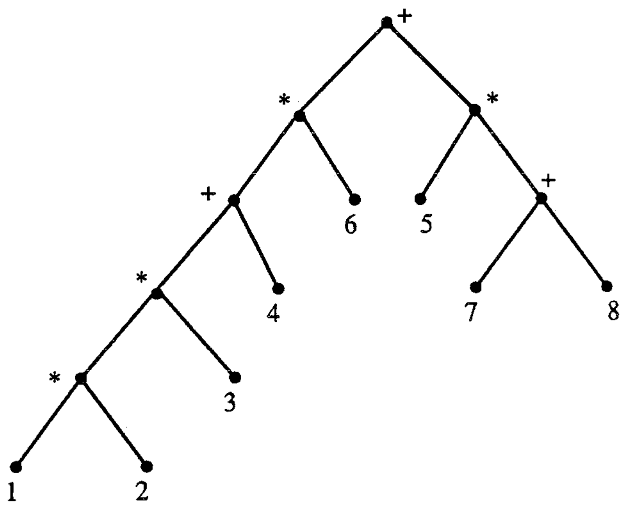


Figure 12.3 The slicing tree of the floorplan in Figure 12.2

where $\lambda \in \mathbb{R}^+$ is a weighting factor. The total area $A(i)$ of the floorplan can be obtained by running the area minimization algorithm by Otten [1982] to determine the shape constraint of the floorplan, then choosing the point with minimum area on the shape constraint of the floorplan. Once the shape of the floorplan is fixed, the shapes of all modules can also be determined in a top-down fashion. This also gives the total estimated length of connections $W(i)$. For each connection between modules u and v, its length is estimated by the distance between the centers of modules u and v in the current floorplan.

- The neighborhood is generated by one of the following techniques:

  - exchanging two consecutive operands in the current Polish expression;
  - interchanging $\star$ and $+$ in a subsequence of operators (e.g., a change of $\star + \star + \star$ into $+ \star + \star +$);
  - exchanging an operand with an adjacent operator in the current expression.

Table 12.2 Results of floorplanning by simulated annealing for eight problems

Problem	n	A_0	A_r	W_r	A_{sa}	W_{sa}
p_1	15	137.08	491.62	106.02	137.86	34.97
p_2	20	198.88	745.35	258.84	202.17	80.49
p_3	20	197.15	808.72	423.90	198.98	197.4
p_4	25	244.68	1 165.7	398.21	245.43	209.7
p_5	25	238.15	1 026.6	576.71	244.63	151.9
p_6	30	333.92	1 549.6	1 023.2	340.15	294.9
p_7	30	314.45	1 476.9	1 095.7	319.40	429.2
p_8	40	407.44	1 934.2	1 002.2	422.95	265.8

Efficient ways for calculating the differences in cost incrementally are also given by Wong & Liu [1986].

The algorithm was applied to a number of randomly generated problems, varying in size from 15 to 40 modules; see Table 12.2. Column n lists the numbers of modules for the problems. Table 12.2 gives the total area A_0 of the modules, the area A_r and wire length W_r of a random floorplan, and the area A_{sa} and wire length W_{sa} of the floorplan obtained by simulated annealing. CPU times on a PYRAMID computer range from 1 to 13 minutes. For all problems the area of the final floorplan obtained by simulated annealing is less than 4% larger than the total area of all modules (a theoretical lower bound for the global minimum).

For other floorplanning algorithms based on simulated annealing see Koakutsu, Sugai & Hirata [1992], Otten & Van Ginneken [1984], Saha, Mohan & Raghunathan [1989], Sechen [1988a], and Wong & Liu [1987].

2.3 Channel routing

In the (two-layer) channel routing problem we are given two sets of l terminals, $T = \{t_1, t_2, \ldots, t_l\}$ and $B = \{b_1, b_2, \ldots, b_l\}$, along the top and bottom boundaries of a rectangular grid representing the routing region. Each terminal is assigned a number which is an integer in $\{1, 2, \ldots, m\}$. The terminals with the same number form a net. The objective is to route the nets in such a way that the number of horizontal tracks in the channel is minimized. Figure 12.4 shows two solutions to a 3-net problem. Note that nets are not allowed to overlap. In this section we present two routing approaches based on simulated annealing.

The first algorithm is due to Leong, Wong & Liu [1985]. It is assumed that a net cannot change tracks; doglegs are not allowed. A *vertical constraint graph* $G_c = (V, A)$ can be constructed by associating each net with a vertex and defining an arc from u to v if the horizontal segment of net u must be placed above that of net v. (For the problem in Figure 12.4 there is an arc from 2 to 3.) Further-

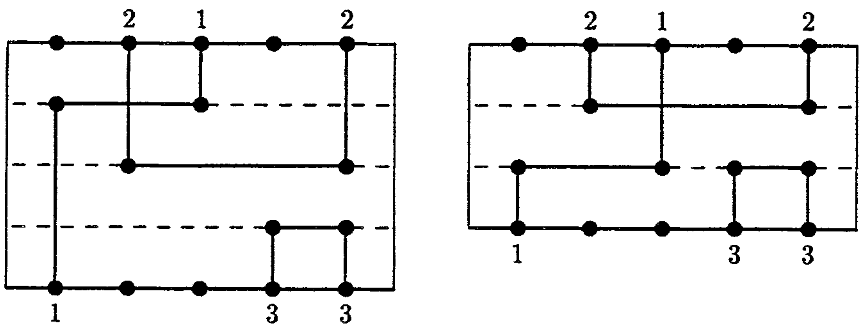


Figure 12.4 Two solutions of a 3-net

more, there is a *horizontal constraint* between vertices u and v if the intervals defined by the leftmost and rightmost terminals of nets u and v overlap. For the problem in Figure 12.4 there are horizontal constraints between 1 and 2 and between 2 and 3.

A partition $i = \{V_1, \ldots, V_m\}$ of the vertex set V induces a second graph G_i as follows: each subset V_p is associated with a vertex $\hat{v}_p$ in G_i, and there is an arc from $\hat{v}_p$ to $\hat{v}_q$ if there is an arc from some $v_r \in V_p$ to some $v_s \in V_q$ in G_c. A partition i is said to be *valid* if G_i is acyclic and there are no horizontal constraints between any two vertices in the same subset. For the problem in Figure 12.4, both $\{\{1\}, \{2\}, \{3\}\}$ and $\{\{1, 3\}, \{2\}\}$ are valid partitions. Leong, Wong & Liu [1985] showed that there is a one-to-one correspondence between the valid partitions of the nets and the solutions of the routing problem. Moreover, the number of subsets in a partition corresponds to the number of horizontal tracks in the routing solution.

Simulated annealing is applied in the following way:

- The solution space is given by the set of all valid partitions.
- The cost of partition i is chosen to be

$$f(i) = W(i)^2 + \lambda_1 P(i)^2 + \lambda_2 U(i),$$

  where $W(i)$ is the number of subsets of i. The second and third terms are added to the cost function to obtain distinct cost values for solutions with the same number of subsets; $P(i)$ (the length of a longest path in G_i) is a lower bound on the number of horizontal tracks needed for all partitions obtained from i by the second type of transitions [Leong, Wong & Liu, 1985], and $U(i)$ is a measure of the amount of unoccupied track, summed over all $W(i)$ tracks. λ_1 and λ_2 are weighting factors, which are experimentally determined.
- The neighborhood is generated by one of the following techniques:

  – exchanging two vertices belonging to different subsets in the current partition;
  – moving a vertex from one subset to another;
  – creating a new subset by separating a vertex from a subset.

Each newly generated partition should be checked to see whether it is valid. A number of ways for carrying out these checks efficiently were given by Leong, Wong & Liu [1985]. Furthermore, the authors present efficient strategies for calculating the differences in cost.

The algorithm was applied to a number of well-known problems from the literature as well as randomly generated problems. For the problems from the literature, including Deutsch's difficult problem [Deutsch, 1976], the algorithm found the same solutions as those obtained by traditional algorithms [Rivest & Fiduccia, 1982; Yoshimura & Kuh, 1982], but using considerably more computation time. On the randomly generated problems, it usually outperformed other routing algorithms, but the improvement was usually small.

Different simulated annealing approaches to the routing problem are proposed by Acan & Unver [1992] and Zargham [1992]. Unlike the previously discussed approach, these ones assume as fixed channel width; a net can change tracks. The objective is then to find a feasible routing (meaning that there is no overlap among different nets) with minimum total wire length and minimum number of vias. By minimizing the total wire length and the number of vias, the performance of the circuits can be increased and the manufacturing cost reduced.

The application of simulated annealing is carried out as follows:

- The solution space is given by the set of all possible ways of connecting the nets, including infeasible ones in which there are overlaps among different nets.
- The cost function is chosen to be

$$f(i) = \lambda_1 O(i) + \lambda_2 W(i) + \lambda_3 V(i),$$

  where $O(i)$, $W(i)$, and $V(i)$ are the total amount of overlaps, total wire length, and total number of vias of solution i, respectively. λ_1, λ_2, and λ_3 are weight factors. To stress the importance of the feasibility of a solution, λ_1 is usually chosen to be considerably larger than λ_2 and λ_3.
- The neighborhood is generated by one of the following techniques:
  - using the multiterminal Lee algorithm [1961] to modify the current routing of a net;
  - reshaping the current routing of a net by reassigning its wire segments to layers; the purpose here is to change the number of vias.

This approach can handle the switchbox routing problem as well. Another advantage of this approach is that doglegs can be introduced automatically.

All channel routing benchmarks in the literature are tested on this algorithm. Good improvements in terms of the wire length and the number of vias are observed [Acan & Unver, 1992].

For other simulated annealing routers see Leong & Liu [1987], Rossi [1990], and Vecchi & Kirkpatrick [1983].

2.4 Compaction

In the compaction problem we are given a layout, consisting of a number of (layout) components such as modules and wires. A module usually has fixed dimensions; a wire has a fixed width but is lengthwise stretchable and shrinkable. If a wire is connected to a module, it can slide along the boundary of that module but it cannot turn around a corner. In other words, topological relations exist between modules and wires. The problem is to minimize the area of the bounding rectangle while preserving these relations.

An example is shown in Figure 12.5(a); an optimally compacted layout is shown in Figure 12.5(d). The compacted layout has shrunk some of the wires to zero length, e.g., for three of the five wires connecting modules 1 and 3. And some wires have a negative length, shifting modules 4 and 5 from a position to the left of modules 6 and 7 in the initial layout to a position to the right in the compacted layout.

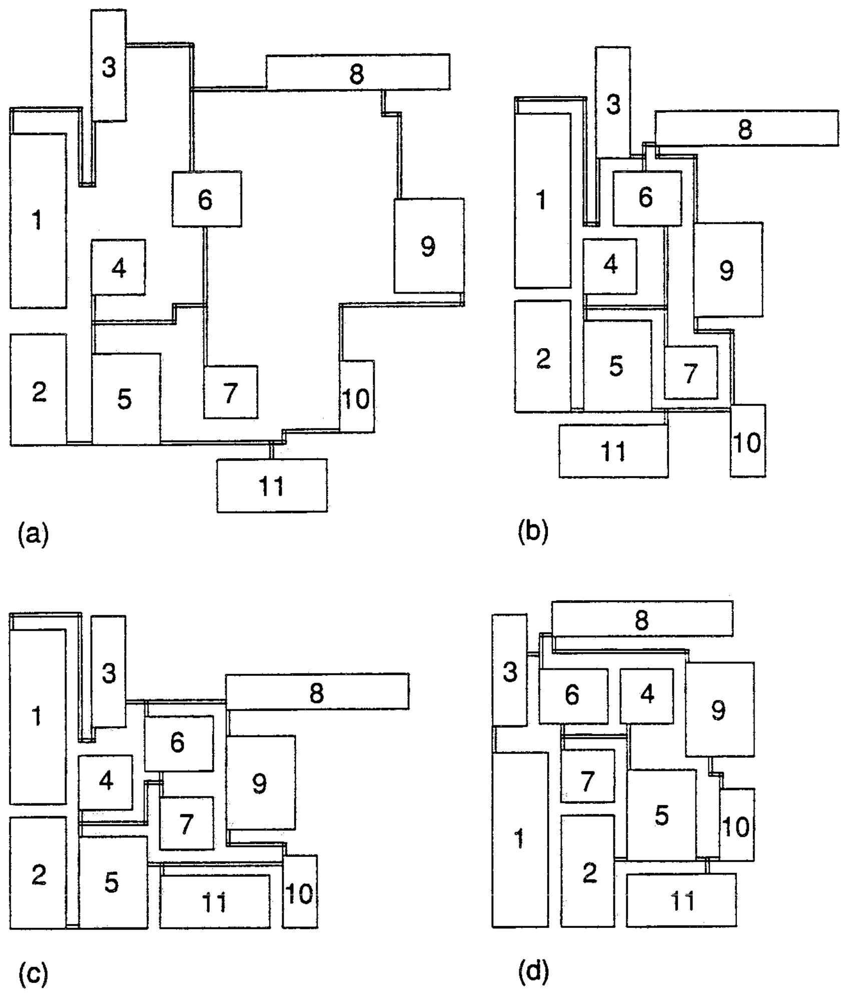


Figure 12.5 An instance of a compaction problem: (a) initial solution, (b) solution obtained after x-compaction followed by y-compaction, (c) solution obtained after y-compaction followed by x-compaction, (d) solution obtained after two-dimensional compaction

For component u there is a pair of variables x_u and y_u, which are the coordinates of either the center point of a module or one of the endpoints of the centerline of a wire. The topological relations can be expressed as (in)equalities to be satisfied by these variables. Following Schalg, Liao & Wong [1983], these inequalities are called the *base constraints*. For a more detailed description the reader is referred to Schalg, Liao & Wong [1983].

A pair of components is called a *free pair* if they are two modules not connected by wires (e.g., modules 1 and 2 in Figure 12.5), or a module and a wire that is not connected to that module. The topological relation between a free pair of components may vary from solution to solution; consider the relation between modules 1 and 2 in the initial and compacted layouts. For each free pair the variables have to satisfy certain inequalities in order to prevent modules and wires from overlapping each other. More precisely, for each free pair (u, v) one of four possible inequalities must be satisfied, so that u is either to the left, to the right, on top, or below v. We call these inequalities *distance constraints*. The number of free pairs can be substantially reduced by applying *pruning rules* [Hsieh, Leong & Liu, 1987]. For example, if modules u and v are connected by a single wire and modules v and w by a single wire in the same dimension, the relative positions of the pairs (u, v), (v, w), and (u, w) are fixed.

A solution of the problem can now be found by choosing a set of distance constraints, one for each free pair of components, and computing the area of the bounding rectangle of the resulting layout. Thus, after applying the pruning rules, if there are m free pairs, the number of solutions is 4^m. The number of *feasible* solutions is much smaller, since many choices will give rise to conflicts. The area of the bounding rectangle can be computed by constructing two directed graphs, one for the x-dimension and one for the y-dimension. The nodes of a graph correspond to the variables (in one dimension), the arcs to the inequalities (base and distance constraints) to be satisfied by the variables: An inequality $x_u + 2 \leqslant x_v$ produces an arc from node u to node v in the x-graph with weight 2. The length of the longest path in the x-graph (y-graph) then equals the dimension of the bounding rectangle in x-dimension (y-dimension). Furthermore, the feasibility of a solution is equivalent to the requirement that neither of the graphs contain positive cycles.

To illustrate these concepts, we return to Figure 12.5. In Figure 12.5(a) the distance constraint chosen for the free pair (1, 2) is apparently '1 on top of 2', creating arc (2, 1) in the y-graph. In the solution shown in Figure 12.5(d) the choice is '1 to the left of 2', creating arc (1, 2) in the x-graph. In the compacted layout, the longest path in the x-graph connects nodes 3, 6, 4, and 9; in the y-graph it connects nodes 11, 10, 9, and 8.

Simulated annealing can be applied as follows:

- The solution space is given by the 4^m possible choices of the m distance constraints.
- The cost of a solution is given by the product of the length of a longest path in the x-graph and the length of a longest path in the y-graph.

- To generate the neighborhood, Osman [1987] chooses a free pair (u, v) that is on a longest path in the current x-graph or y-graph, then replaces its current distance constraint by a new distance constraint.

This algorithm was applied to five problems, consisting of 6, 11, 15, 39, and 73 modules, and 12, 24, 31, 63, and 122 wires. The first two problems are adapted from Schalg, Liao & Wong [1983], the third problem is from Osman [1987], the fourth and fifth are adapted from Hsieh, Leong & Liu [1987]. Results were compared with those obtained by two other algorithms: an optimization algorithm, based on *branch-and-bound* techniques [Schalg, Liao & Wong, 1983], and an approximation algorithm based on *one-dimensional compaction.* For the second problem, whose initial layout is shown in Figure 12.5(a), the result of x-compaction followed by y-compaction is shown in Figure 12.5(b); y-compaction followed by x-compaction leads to the layout shown in Figure 12.5(c). The areas of the bounding rectangles of these layouts are 63.2% and 45.9% larger, respectively, than the area of the bounding rectangle of the optimally compacted layout shown in Figure 12.5(d). The performance of simulated annealing for each of the five problems is reported in Table 12.3. Column n gives the total number of layout components in each problem. The algorithms were implemented on a VAX 8650 computer.

Table 12.3 gives the average cost of the best solution $\bar{A}_{\text{best}}$, the average running time in seconds $\bar{t}$, and standard deviations σ_A and σ_t, respectively, obtained from five runs of the simulated annealing algorithm. The table also includes the results obtained by the branch-and-bound algorithm A_{opt} as well as its running time t, and the results obtained by applying two consecutive one-dimensional compactions: x followed by y, A_{x-y}, and y followed by x, A_{y-x}. No running times are mentioned for the approximation algorithm, since they are virtually negligible. For Problem 3 the branch-and-bound algorithm takes at least 3 days of CPU time, and 672 is the conjectured optimal value; Problems 4 and 5 are too large to be solved by the branch-and-bound algorithm.

From these results we conclude that, in the trade-off between CPU time and quality of final solutions, simulated annealing comes out very well. It was able to find high-quality solutions (as opposed to the one-dimensional approach) in

Table 12.3 Results of compaction by simulated annealing for five problems

		Simulated annealing				Branch-and-bound		1D compaction	
Problem	n	$\bar{A}_{\text{best}}$	σ_A	$\bar{t}$	σ_t	A_{opt}	t	A_{x-y}	A_{y-x}
1	18	114.0	0.0	4.8	3.7	114.0	0.2	126.0	142.5
2	35	3 557.8	0.0	164.9	19.8	3 557.8	643.0	5 808.0	5 192.3
3	46	685.2	10.8	36.4	4.6	672.0	>3 days	1 015.0	1 200.0
4	102	1 736.0	0.0	100.4	16.4			2 016.0	2 067.0
5	195	3 372.0	24.0	440.2	158.4			4 032.0	4 187.0

reasonable amounts of CPU time (as opposed to the branch-and-bound approach). On the other hand, if one were to deal with instances consisting of say 500 modules and wires, one still would have to resort to one-dimensional compaction, because the running times for simulated annealing would be prohibitive.

For other compaction approaches based on simulated annealing, we refer the reader to Mosteller [1986] and Zeestraten [1985].

Simulated annealing has been applied to other layout problems. For applications to partitioning see Chatterjee & Hartley [1990], Chung & Rao [1986], Sugai & Hirata [1991], and Tao et al. [1992]. Wong, Leong & Liu [1986] proposed a PLA folding algorithm based on simulated annealing.

3 TABU SEARCH

Tabu search is another general local search procedure. Though some of the ideas in tabu search were proposed over a period of time, tabu search was introduced in its present form by Glover [1989, 1990] and has since been applied to solve combinatorial optimization problems arising in many different areas [Glover et al., 1993].

Unlike simulated annealing, which is a memoryless search method, tabu search accumulates information during the search process and stores the information in memory. This information is used either to encourage or discourage the generation of certain types of solutions. Like simulated annealing, tabu search may accept a solution inferior to the current one.

Though accepting inferior solutions is necessary for local search methods to jump out of local optima, it may lead to cycling if the search repeatedly returns to a local optimum previously visited. The most important feature of tabu search is to use information stored in memory to forbid the generation of certain types of solutions. The conditions that characterize the forbidden solutions are called *tabu conditions*. Yet a solution that violates one or more of the tabu conditions can be accepted if its aspiration value, computed according to an *aspiration function*, exceeds a certain value. Both the list of tabu conditions and the aspiration function are dynamically updated as the algorithm proceeds.

The main step in tabu search consists of generating the best solution in the neighborhood of the current solution subject to the tabu conditions (which can, however, be overridden according to the aspiration value of a solution), and replacing the current solution by the new solution. We now set up the general framework of tabu search. Let $\mathscr{S}$ denote the solution space and f the objective function of an optimization problem. Let $\mathscr{N}(i)$ denote the neighborhood of solution i. Let T denote the tabu list and A the aspiration function. Figure 12.6 shows the general framework of tabu search.

Application of tabu search to an optimization problem has four key components: neighborhood structure, tabu list, aspiration function, and stopping condition; see also Chapter 5.

```
begin TABU
  select an initial solution $i \in \mathcal{S}$;
  initialize tabu list $T$ and aspiration function $A$;
  while the stopping condition is not met do
    compute the best $i'$ in $\mathcal{N}(i)$
      subject to tabu list $T$ and aspiration function $A$;
    $i \leftarrow i'$;
    update tabu list $T$;
    update aspiration function $A$;
  end while
end TABU
```

Figure 12.6 Tabu search paradigm

Tabu search has been applied to the solution of several layout problems. We will now discuss its application to two of them, placement and graph partitioning.

3.1 Partitioning

Let $G = (V, E)$ be a weighted graph with an even the number of vertices, where V is the set of modules and E is the set of nets. Let $w(u, v)$ denote the weight of edge $\{u, v\}$ in E. (As a convention, if $\{u, v\}$ is not in E, we let $w(u, v) = 0$.) The bipartitioning problem asks for a division of V into two equal halves, U and W, so as to minimize the sum of the weights of the edges with one vertex in U and the other in W. This problem has various applications in design automation of VLSI circuits. Here we describe a tabu search approach to graph partitioning due to Tao et al. [1992].

The solution space consists of all equal-size bipartitions of V. This is,

$$\mathcal{S} = \{(U, W) \mid U \cup W = V, U \cap W = \emptyset, |U| = |W|\}.$$

For $i = (U, W)$, the value of the objective function is

$$f(i) = \sum_{u \in U,\, v \in W} w(u, v).$$

For $i = (U, W)$, $\mathcal{N}(i)$, the neighborhood of i is defined as follows. The vertices in U are sorted according to the linear order $\prec$ such that for u_1, $u_2 \in U$, $u_1 \prec u_2$ if

$$f((U - \{u_1\}, W \cup \{u_1\})) \leqslant f((U - \{u_2\}, W \cup \{u_2\})).$$

In other words, the decrement in the value of the cut when u_1 is moved from U to W is larger than or equal to the corresponding decrement when u_2 is moved from U to W. The vertices in W are sorted in a similar way.

Let $u_1, u_2, \ldots, u_K$ denote the K smallest vertices in U, according to the linear ordering $\prec$, and $v_1, v_2, \ldots, v_K$ the K smallest vertices in W. For $i = (U, W)$, we define

$$\mathcal{N}(i) = \{(U', W') \mid U' = (U - \{u_p\}) \cup \{v_q\},\ W' = (W - \{v_q\}) \cup \{u_p\},\ \text{where } p, q \leqslant K\}.$$

In other words, (U', W') can be obtained from (U, W) by exchanging two vertices from the sets $\{u_1, u_2, \ldots, u_K\}$ and $\{v_1, v_2, \ldots, v_K\}$. Thus, the number of neighboring solutions of i is K^2.

The tabu list is simply a record of the r most recent exchanges for a positive integer r. To be specific, if $u_1 \in U$ and $v_1 \in W$ are selected to be exchanged to yield a new solution, the pair (v_1, u_1) will be added to the tabu list unless it is already there. (That u_1 and v_1 are selected when (v_1, u_1) is in the tabu list is possible because a tabu condition can be overridden by the aspiration function.) In the case that (v_1, u_1) is to be added to the tabu list, the rth exchange will be removed if there is one.

The aspiration function A is defined to be a function from $\mathbb{R}$ to $\mathbb{R}$. Specifically, $A(c)$ is one less than the cost of the best solution that has been reached from a solution of cost c. Intuitively, it means the tabu conditions will be overruled if a new solution has a cost lower than any other solution that has been reached from a solution with the current cost. Consequently, in each iteration, the aspiration function is updated as follows:

```
if $A(f(i')) > f(i)$ then $A(f(i')) \leftarrow f(i) - 1$
else
if $A(f(i)) > f(i')$ then $A(f(i)) \leftarrow f(i') - 1$.
```

The algorithm will stop if no improvement is observed for a certain number, of iterations.

We now turn to multiway partitioning, which is the problem of partitioning the vertices of a graph into several mutually exclusive subsets so that the sum of the weights of the edges crossing the subsets is minimized under the condition that the vertex weights are distributed evenly among the subsets. Let $G = (V, E)$. To each vertex $v \in V$, a vertex weight $d(v)$ is assigned. To each edge $\{u, v) \in E$, an edge weight $w(u, v)$ is assigned. Let $\mathcal{S}$ denote the set of all feasible solutions. For $i \in \mathcal{S}, i = (V_1, V_2, \ldots, V_m)$ where $V_1 \cup V_2 \cup \cdots \cup V_m = V$ such that $V_p \cap V_q = \varnothing$ for $p \neq q$.

The cost of i, $f(i)$, is the sum of the $w(u, v)$ such that $u \in V_p$ and $v \in V_q$ for $p \neq q$. We want to search for a solution which minimizes the value of $f(i)$ subject to the constraint that

$$\sum_{1 \leqslant p < q \leqslant m} \left(\sum_{u \in V_p} d(u) - \sum_{v \in V_q} d(v) \right)^2 \tag{3}$$

is minimum. We now transform the problem so that the weights on vertices in this constraint are removed. Given $G = (V, E)$, we construct a complete graph $G^* = (V, E^*)$ such that there is an edge $\{u, v\}$ between every two vertices u and v in

V. For each edge $\{u, v\}$ in E^*, its edge weight $w'(u, v)$ is defined as follows:

$$w'(u,v) = \begin{cases} -d(u)\,d(v)R + w(u,v) & \text{if } (u,v)\in E, \\ -d(u)\,d(v)R & \text{if } (u,v)\notin E, \end{cases}$$

where R is a positive real constant that is larger than the sum of the edge weights of all the edges in E. For a partition i of G^* (a partition of G as well), its cost, the sum of the weights of the edges crossing the subsets, is

$$f'(i) = -R \sum_{1\leqslant p<q\leqslant m} \left(\sum_{u\in V_p} d(u)\right)\left(\sum_{v\in V_q} d(v)\right) + f(i).$$

It can be shown that minimizing $f'(i)$ is equivalent to minimizing $f(i)$ under the constraint that (3) is minimum. Namely, solving the transformed problem is the same as solving the original problem. Thus, f' is chosen to be the cost function.

The neighborhood of solution i consists of two parts, $S_{1,i}$ and $S_{2,i}$, which are defined as follows. Let $i = (V_1, V_2, \ldots, V_m)$. We define

$$S_{1,i} = \{(V_1, V_2, \ldots, V_t - \{u\}, \ldots, V_j \cup \{u\}, \ldots, V_m) \mid u \in V_t\}.$$

In other words, a solution is in $S_{1,i}$ if it can be obtained by moving a vertex from one subset to another subset.

Let u be a vertex in V_t. Suppose that moving u from V_t to any one of $V_1, V_2, \ldots, V_{t-1}, V_{t+1}, \ldots, V_j, \ldots, V_m$ will yield a solution of cost larger than $f(i)$. Let V_j be the subset into which u is moved that will lead to the smallest increase in the cost. We define

$$S_{2,i} = \{(V_1, V_2, \ldots, V_t - \{u\} \cup \{v\}, \ldots, V_j - \{v\} \cup \{u\}, \ldots, V_m) \mid u \in V_t, v \in V_j\}.$$

In other words, we shall only exchange u with every vertex v in V_j but not with other vertices in other subsets. $\mathcal{N}(i)$ is simply the union of $S_{1,i}$ and $S_{2,i}$.

The tabu list, aspiration function, and stopping condition are all the same as for bipartitioning.

For other partitioning algorithms based on tabu search, see Areibi & Vannelli [1993] and Lim & Chee [1991].

3.2 Placement

Tabu search has also been applied to placement [Lim, Chee & Wu, 1991; Song & Vannelli, 1992]. Here we briefly describe the placement algorithm proposed by Song & Vannelli [1992]. This algorithm is applicable to both sea-of-gates and gate array design styles.

An estimate of the total wire length is used as the cost of a placement. The wire length of a net is estimated by the half-perimeter of the bounding box of the terminals of the net in the placement. The estimate of the total wire length is simply the sum of the estimated wire lengths of all the nets.

Let i be a placement and u be a module. We define $N_i(\delta, u)$ to be the set of modules whose x- and y-coordinates are within distance δ from that of u, i.e.,

$$N_i(\delta, u) = \{v \text{ is a module } \mid |x_i(u) - x_i(v)| \leqslant \delta \text{ and } |y_i(u) - y_i(v)| \leqslant \delta\},$$

where $x_i(u)$ and $y_i(u)$ are the respective x- and y-coordinates of module u in placement i. To define the neighboring placements of i, a module is selected randomly. Suppose that module u is selected. The neighborhood of i, $\mathcal{N}(i)$, is then defined to be the set of placements that can be obtained from i by exchanging module u with any one of the modules in $N_i(\delta, u)$.

The tabu list is the list of the r most recent module exchanges for some control parameter r which is a positive integer. The exchanges in the tabu list are ordered from the oldest to the most recent. The current exchange will be added to the tabu list if it is not there. The addition of the current exchange will cause the rth exchange (the oldest) in the list to be dropped. It was observed that the aspiration function has almost no influence on the behavior of the algorithm, so it was actually not used.

The stopping condition is as follows. An upper limit is set on the total number of iterations between improvements. If the limit is reached without encountering an improvement after the previous improvement, the algorithm stops.

By using a constructive placement algorithm to generate the initial placement, the results were comparable to those obtained using TimberWolfSC5.4, a placement package based on simulated annealing, but the running time was considerably shorter.

4 GENETIC ALGORITHMS

Genetic algorithms have their origin in biological processes. Species in the natural world undergo a process of slow evolution; they gradually adapt themselves to the environment through mutation and mixing of genes. Good characteristics gradually emerge and bad characteristics gradually disappear.

A genetic algorithm maintains a *population* of solutions to the problem at hand and allows them to evolve through successive generations. Crossover and mutation are used to create the next generation from the present generation. A *crossover* operation combines two (or sometimes more) solutions to yield a new solution; a *mutation* operation perturbs an individual solution. The solutions to be included in the next generation are then selected, according to their *fitness* values, from the set of solutions comprising of the current generation and the newly generated solutions. Crossover and the selection criteria form the main optimization step in genetic algorithms. The assumption here is that good partial solutions (genes) to a problem are combined during crossover to form better solutions. Mutation is the main operation for avoiding local optima. A good tutorial on genetic algorithms is Goldberg [1989].

A constant number of solutions with the highest fitness values are typically selected so that the population is maintained at a fixed size for each generation.

```
begin GEN
  set up the first generation;
  for ($t \leftarrow 1$ to $max\_of\_generation$) do
      select a mating pool from the current generation;
      crossover solutions in the mating pool to generate
      a children pool;
      mutate the solutions in the children pool and current
      generation;
      select solutions to form the new current generation;
  end for
end GEN
```

Figure 12.7 Genetic algorithm paradigm

The process is terminated after a certain number of generations. Either the best remaining solution or the best solution ever observed is reported. Figure 12.7 is a high-level description of a genetic algorithm; see also Chapter 6.

Genetic algorithms have been applied to many layout problems; we present two examples: floorplanning and improvement of layouts.

4.1 Floorplanning

We present a genetic algorithm for floorplan design due to Cohoon et al. [1991]. The solution space consists of all slicing floorplans with n modules. As in the simulated annealing algorithm of Wong & Liu [1987] (Section 2.2), Polish expressions are used to represent the solutions. Four crossover operations are introduced. They are defined as follows.

- CO_1: One child is produced. The child is constructed by carrying out a one-for-one replacement of the operators in parent 1 by the operators in parents 2 in the order they appear in parent 2. Figure 12.8 shows an example.
- CO_2: One child is produced. The child is constructed by carrying out a one-for-one replacement of the operands in parent 1 by the operands in parent 2 in the order they appear in parent 2. Figure 12.9 shows an example.
- CO_3: One child is produced. A subtree in parent 1 is first identified. The child is constructed by reordering the operands in parent 1 that are not in the subtree; the reordering is made according to the order of the operands in parent 2. Figure 12.10 shows an example.
- CO_4: Two children are produced. Two subtrees of the same size are identified in parent 1 and parent 2, respectively. Child 1 is constructed as follows. The subtree in parent 1 is replaced by the subtree in parent 2. The remaining operators of parent 1 are retained in their original positions. The operands in parent 1 that do not appear in the subtree from parent 2 will retain their order and occupy the positions for operands in parent 1 that are not occupied by

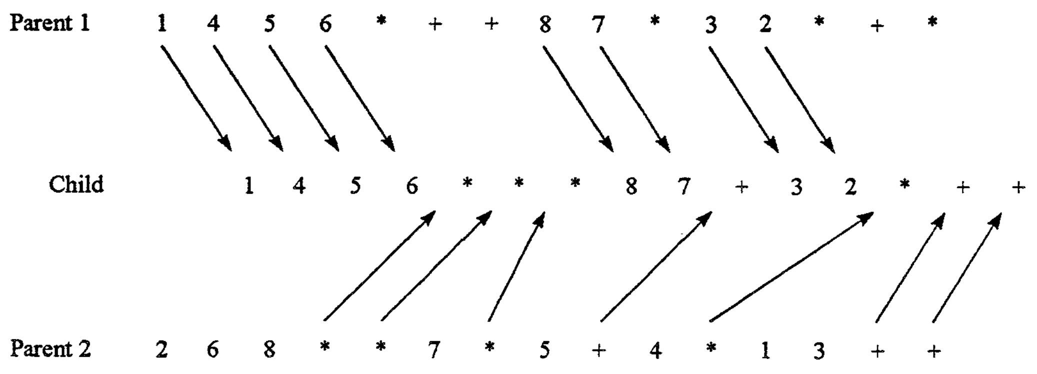

Parent 1 1 4 5 6 * + + 8 7 * 3 2 * + *
Child 1 4 5 6 * * * 8 7 + 3 2 * + +
Parent 2 2 6 8 * * 7 * 5 + 4 * 1 3 + +

Figure 12.8 Illustration of crossover operator CO_1

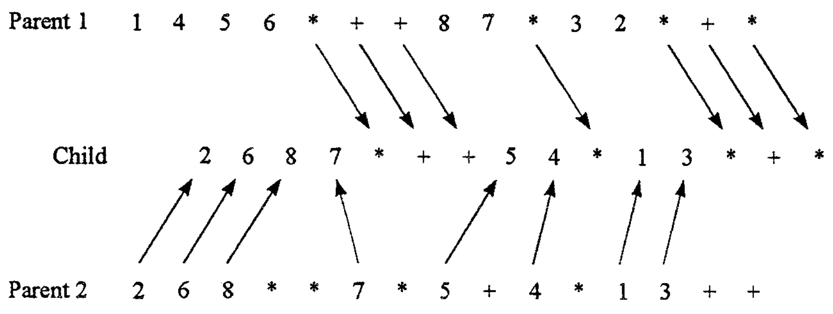

Parent 1 1 4 5 6 * + + 8 7 * 3 2 * + *
Child 2 6 8 7 * + + 5 4 * 1 3 * + *
Parent 2 2 6 8 * * 7 * 5 + 4 * 1 3 + +

Figure 12.9 Illustrations of crossover operator CO_2

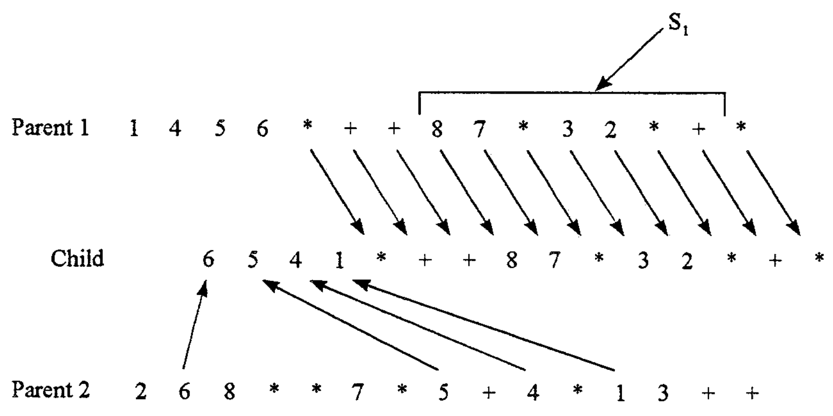
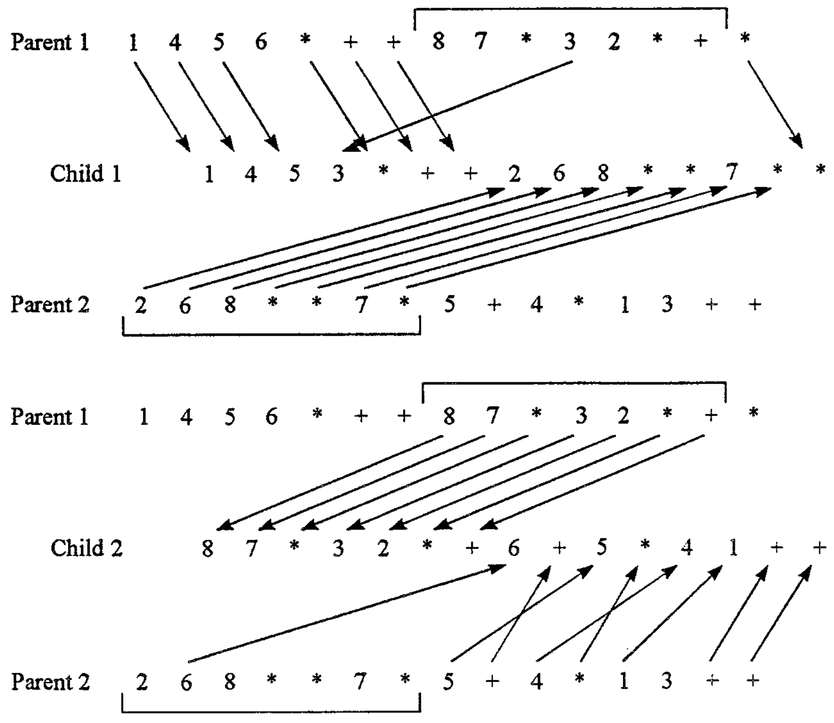

S1
Parent 1 1 4 5 6 * + + 8 7 * 3 2 * + *
Child 6 5 4 1 * + + 8 7 * 3 2 * + *
Parent 2 2 6 8 * * 7 * 5 + 4 * 1 3 + +

Figure 12.10 Illustration of crossover operator CO_3

operands in the subtree from parent 2. Child 2 is constructed in the same way except that the roles of parent 1 and parent 2 are interchanged. Figure 12.11 shows an example.

The mutation operators take a single individual and modify it in a localized manner. The operations defined by Wong & Liu [1987] in their simulated



Parent 1 1 4 5 6 * + + 8 7 * 3 2 * + *
Child 1 1 4 5 3 * + + 2 6 8 * * 7 * *
Parent 2 2 6 8 * * 7 * 5 + 4 * 1 3 + +
Parent 1 1 4 5 6 * + + 8 7 * 3 2 * + *
Child 2 8 7 * 3 2 * + 6 + 5 * 4 1 + +
Parent 2 2 6 8 * * 7 * 5 + 4 * 1 3 + +

Figure 12.11 Illustration of crossover operator CO_4

annealing algorithm are chosen as the mutation operators, namely, M_1: swapping two adjacent operands, M_2: switching a sequence of adjacent operators, and M_3: swapping an operator and a neighboring operand.

A score is given for every solution in the current population. The score of a solution is the cost function defined in the simulated annealing floorplanning algorithm by Wong & Liu [1987]. Let μ denote the mean and σ denote the standard deviation of the distribution of the scores of the solutions in the current population. For each solution i in the population, its fitness value is determined according to the formula

$$fitness(i) = \frac{\mu - score(i) + \alpha\sigma}{2\alpha\sigma}$$

where α is a constant parameter. Since we shall keep the population size constant, the solutions with the n highest fitness values will survive. Experiments with the algorithm showed very encouraging results.

4.2 Global improvement of macro cell layouts

In most macro cell floorplanning systems the shapes (widths and heights) of the modules and the total wire length are used as control parameters, e.g. the algorithm of Wong & Liu [1987]. Since no actual routing is carried out during the floorplanning stage, total wire length is often estimated very roughly.

The area of a layout consists of two parts: the area occupied by the modules and the area needed for routing. Given a floorplan for a set of modules with fixed shapes, a placement can be obtained by placing the modules according to the floorplan and leaving enough routing area between every pair of neighboring modules. The routing area cannot be determined precisely without actually completing the routing. However, given the pin positions on the boundary of each module and the global routes, channel density gives a fairly good estimation of the channel width. The density of a channel can be changed by laterally shifting the modules on the boundaries of the channel [Tammassia & Tollis, 1988; Lengauer, 1990]. We now present a genetic algorithm for improving a layout generated by a floorplanning system due to Glasmacher, Hess & Zimmermann [1991].

The problem can be formally defined as follows. Given a set of rectangular modules $\{m_1, \ldots m_n\}$ with fixed width and height and fixed pin positions, a floorplan for the modules, and a global routing, determine a placement of the modules with minimum total area for a given aspect ratio such that the separation between any two neighboring modules is larger than or equal to the density of the channel formed by the two modules. We further assume that the floorplans are slicing. Figure 12.12 shows an example of the problem with six modules and the corresponding slicing floorplan.

A solution to the global improvement problem consists of one gene for each cutline and the genes are arranged in a fixed linear order as a table. A gene specifies which two modules on opposite sides of the cutline are chosen as well as the offset between the two modules in the corresponding placement. The offset of two modules may be measured by, say, the difference between the coordinates of

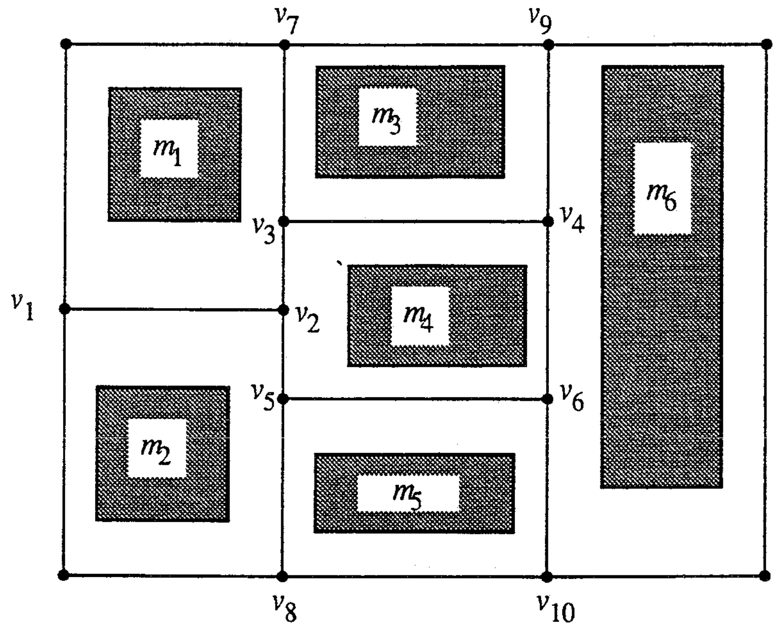


Figure 12.12 An example of the problem with six modules

gene	1	2	3	4	5
cutline	(v_1, v_2)	(v_3, v_4)	(v_5, v_6)	(v_7, v_8)	(v_9, v_{10})
module pair	m_1, m_2	m_3, m_4	m_4, m_5	m_1, m_3	m_3, m_6
offset	2	1	2	1	1

Figure 12.13 A solution to the example in Figure 12.12

the lower left corners of the two modules in the direction of the cutline. Figure 12.13 shows a solution to the problem in Figure 12.12. Gene 1 corresponds to the horizontal cutline (v_1, v_2). The only pair of modules on opposite sides of (v_1, v_2) is m_1 and m_2, so this pair is chosen. The difference in the horizontal coordinates of the lower left corners of m_1 and m_2 is 2 units as specified by the offset value in column 1. For cutline (v_7, v_8) (gene 4), on the left side either m_1 or m_2 can be chosen; on the right side either m_3, m_4, or m_5 can be chosen. For the solution in Figure 12.13 the chosen pair is m_1 and m_3. Since (v_7, v_8) is vertical, the difference in the vertical coordinates of the lower left corners of m_1 and m_3 is 1 unit, as specified by the offset value in column 5.

Given a solution, we now describe how to determine the corresponding placement. The algorithm works bottom-up in the slicing tree. A cutline is processed only after its two children have been processed. To process a cutline, shift the two partial placements on opposite sides of the cutline so that the two chosen modules on opposite sides have the offset specified by the solution. The offsets between any other pair of neighboring modules along the cutline are then determined as well. Next the minimum separation requirement between any pair of neighboring modules along the cutline is determined by computing the density of the channel formed by the two modules. The two partial placements are now moved in the direction perpendicular to the cutline until the minimum separation between any two modules on opposite sides of the cutline is satisified. The final placement is generated after the root of the slicing tree is processed.

The fitness value of a solution i is a combination of three measures:

- the aspect ratio $AR(i)$ of the enclosing rectangle;
- the area $AP(i)$ of the polygon enclosing all the modules, excluding the routing area on the boundary of the layout;
- the area $AB(i)$ of the bounding box.

Let i be a solution and ARg be the given aspect ratio. The following formula is used to compute the fitness value of a solution:

$$fitness(i) = x(i)\left(\frac{A}{AP(i)} + \frac{A}{AB(i)}\right),$$

Parent 1

gene	1	2	3	4	5
cutline	(v_1, v_2)	(v_3, v_4)	(v_5, v_6)	(v_7, v_8)	(v_9, v_{10})
module pair	m_1, m_2	m_3, m_4	m_4, m_5	m_1, m_3	m_3, m_6
offset	2	1	2	1	1

Parent 2

gene	1	2	3	4	5
cutline	(v_1, v_2)	(v_3, v_4)	(v_5, v_6)	(v_7, v_8)	(v_9, v_{10})
module pair	m_1, m_2	m_3, m_4	m_4, m_5	m_2, m_5	m_4, m_6
offset	2	1	1	1	2

Child 1

gene	1	2	3	4	5
cutline	(v_1, v_2)	(v_3, v_4)	(v_5, v_6)	(v_7, v_8)	(v_9, v_{10})
module pair	m_1, m_2	m_3, m_4	m_4, m_5	m_1, m_3	m_4, m_6
offset	2	1	2	1	2

Child 2

gene	1	2	3	4	5
cutline	(v_1, v_2)	(v_3, v_4)	(v_5, v_6)	(v_7, v_8)	(v_9, v_{10})
module pair	m_1, m_2	m_3, m_4	m_4, m_5	m_2, m_5	m_3, m_6
offset	2	1	1	1	1

Figure 12.14 Illustration of the crossover operation

where A is the total area of the modules and

$$x(i) = \begin{cases} ARg/AR(i) & \text{if } ARg < AR(i), \\ AR(i)/ARg & \text{otherwise.} \end{cases}$$

To cross two solutions, a cutline is selected randomly. Two children (solutions) are generated by concatenating the first segment of one solution and the second segment of the other, and vice versa. Figure 12.14 shows an example of the crossover operation. In this example, cut line (v_9, v_{10}) (gene 5) is selected to cross the two parents. Child 1 is formed by concatenating genes 1 to 4 of parent 1 and gene 5 of parent 2; similarly child 2 is formed by concatenating genes 1 to 4 of parent 2 and gene 5 of parent 1.

Mutation of a solution is performed by modifying the genes. Whether a gene will be modified or not is determined by a given probability. A gene is modified by randomly selecting two modules on opposite sides of the cutline together with a random offset for the two modules.

The initial population of solutions is generated by randomly selecting two adjacent modules on opposite sides of each cutline, and an offset is assigned to them randomly. The mating pool is set up by selecting the solutions from the current generation of solutions at random, favoring those with high fitness values. Note that a solution may be selected several times. The size of the mating pool is chosen to be the number of current solutions. To cross the solutions in the mating pool, two solutions are selected at random and removed from the pool. Crossover will take place with a given probability and the two resulting solutions are added to the child pool. If crossover did not take place, the two solutions themselves will be added to the child pool. The next generation is produced by choosing 25% of the fittest solutions in the current generation and another 25% of the remaining solutions randomly. And 25% of the fittest solutions in the mutated child pool are chosen together with another 25% randomly chosen solutions among the remaining ones. Therefore, the number of solutions in each generation is held fixed. An upper bound on the number of generations is set. The final placement is the one corresponding to the solution with the best fitness value in the last generation. Good experimental results were observed.

There are many other results on applying genetic algorithms to the solution of layout problems. Here we list some of them. For partitioning see Chandrasekharam, Subhramanian & Chaudhury [1993], Jin & Chan [1992], and Saab & Rao [1989]. For floorplanning/placement see Chan, Mazumder & Shahookar [1991], Cohoon & Paris [1987], Esbensen [1992], Glasmacher & Zimmermann [1992], Kling & Banerjee [1987, 1991], Mohan & Mazumder [1992], and Shahookar & Mazumder [1990]. For routing see Geraci et al. [1990], Lin, Hsu & Tsai [1989], and Liu, Sakamoto & Shimamoto [1993]. Fourman [1985] proposed a genetic algorithm for compaction. Hill [1993] applied genetic algorithms to the gate sizing problem.