Consider the "clique representation" of a netlist by means of an edge-weighted graph $G(V, E)$ as discussed in Section 7.1. The weight of an edge $(i, j) \in E$ is given by $\gamma_{ij}$ ($\gamma_{ij} = 0$ when $(i, j) \notin E$). Then the center of gravity $(x_i^g, y_i^g)$ of a cell $i$ is defined as:

$$x_i^g = \frac{\sum_j \gamma_{ij} x_j}{\sum_j \gamma_{ij}}$$

$$y_i^g = \frac{\sum_j \gamma_{ij} y_j}{\sum_j \gamma_{ij}}$$

So, the center of gravity is found by computing the weighted average of the coordinates of those cells connected to $i$. An example of a perturbation is then to move a cell to a legal position close to its center of gravity and if there is another cell at that position to move that cell to some empty location or to its own center of gravity (which creates a chain of moves until the last cell in the chain is put in an empty location).

## 7.5 Partitioning

The direct motivation for paying attention here to the partitioning problem originates from min-cut placement. However, partitioning is itself an important problem in the field of VLSI design automation. It shows up e.g. when a large circuit has to be implemented with multiple chips and the number of pins on the IC packages necessary for interchip communication should be minimized.

Many versions of the partitioning problem exist and many algorithms for each version. A famous algorithm was published by Kernighan and Lin in 1970. It will be discussed in the next section. The section on Bibliographic Notes at the end of this chapter provides pointers to other algorithms.

### 7.5.1 *The Kernighan-Lin Partitioning Algorithm*

The model assumed by the algorithm is as follows: there is an edge-weighted undirected graph $G(V, E)$; the graph has $2n$ vertices ($|V| = 2n$); an edge $(a, b) \in E$ has a weight $\gamma_{ab}$; if $(a, b) \notin E$, $\gamma_{ab} = 0$. The problem is to find two sets $A$ and $B$, subject to $A \cup B = V$, $A \cap B = \emptyset$, and $|A| = |B| = n$, which minimizes the *cut cost* defined as follows:

$$\sum_{(a,b) \in A \times B} \gamma_{ab}$$

In other words, the goal is to minimize the total weight of the edges cut by the partitioning of $V$ into the sets $A$ and $B$. Note that the algorithm assumes that the clique model has been used for the representation of nets (see Section 7.1).

what does it mean

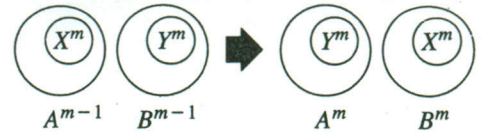



**Figure 7.10** The interchange of subsets in the Kernighan-Lin algorithm.

The first thing to remark is that the problem is NP-complete and that the algorithm proposed by Kernighan and Lin is a heuristic that turns out to be rather successful. The principle of the algorithm is to start with an initial partition consisting of the sets $A^0$ and $B^0$ which, in general, will not have a minimal cut cost. In an iterative process, subsets of both sets are isolated and interchanged. In iteration number $m$, the set isolated from $A^{m-1}$ will be denoted by $X^m$ and the set isolated from $B^{m-1}$ will be denoted by $Y^m$. The new sets, $A^m$ and $B^m$ are then obtained as follows:

$$A^m = (A^{m-1} \setminus X^m) \cup Y^m$$

$$B^m = (B^{m-1} \setminus Y^m) \cup X^m$$

This idea is illustrated in Figure 7.10. The iteration goes on until no improvement in the cut cost is possible.

An important issue in the algorithm is the construction of the sets $X^m$ and $Y^m$. Note that for any nonoptimal partition, there are subsets $X$ and $Y$ that will lead to an optimal partition in one interchange step. The difficulty of the problem arises, of course, from the fact that these subsets cannot be identified easily. Therefore, the algorithm makes an attempt to find suitable subsets, interchanges them and then tries to make a new attempt, until the attempt does not lead to an improvement of the cut cost. In this context, each exchange of subsets is called a *pass*. The total number of passes needed turns out not to be dependent on the problem size $n$: most examples reported in the literature do not need more than 4.

The construction of the sets $X^m$ and $Y^m$ is based on *external* and *internal* costs for vertices in the sets $A^{m-1}$ and $B^{m-1}$. The external cost $E_a$ of $a \in A^{m-1}$ is defined as follows:

$$E_a = \sum_{y \in B^{m-1}} \gamma_{ay}, \quad a \in A^{m-1}$$

So, the external cost for vertex $a \in A^{m-1}$ is a measure for the *pull* that the vertex experiences from the vertices in $B^{m-1}$. In a similar way, the external cost $E_b$ for a vertex $b \in B^{m-1}$ and the internal costs $I_a$ and $I_b$ can be defined:

$$E_b = \sum_{x \in A^{m-1}} \gamma_{bx}, \quad b \in B^{m-1}$$

$$I_a = \sum_{x \in A^{m-1}} \gamma_{ax}, \quad a \in A^{m-1}$$

$$I_b = \sum_{y \in B^{m-1}} \gamma_{by}, \; b \in B^{m-1}$$

The difference between internal and external costs gives an indication about desirability to move the vertex: a positive value shows that the vertex should be moved to the opposite set, a negative value shows a preference to keep the vertex its current set. The differences for the vertices in both sets are given by the variables $D_a$ and $D_b$:

$$\begin{aligned} D_a &= E_a - I_a, \; a \in A^{m-1} \\ D_b &= E_b - I_b, \; b \in B^{m-1} \end{aligned}$$

Now the gain in the cut cost, $\Delta$, resulting from the interchange of two vertices can be expressed as:

$$\Delta = D_a + D_b - 2\gamma_{ab}, \; a \in A^{m-1}, \; b \in B^{m-1}$$

The last term is a correction for a possible edge between $a$ and $b$, which will continue to cross the cut after swapping the vertices.

The pseudo-code code of the Kernighan-Lin algorithm is shown in Figure 7.11. As mentioned earlier, each execution of the outer loop is a new pass. In the inner loop, the `for` loop with iteration variable $i$, the subsets to be interchanged are constructed element by element. In each iteration of this loop, the pair $(a_i, b_i) \in A^{m-1} \times B^{m-1}$ giving the best improvement for the cut cost is selected. The vertices are then "locked", meaning that they cannot be selected once more in the inner loop. They are candidates to be included in the subsets. Actually, it is pretended that they have already been interchanged, and the differences between external and internal costs of the unlocked vertices are therefore updated (check the correctness of the expressions that update the values of $D_x$ and $D_y$).

It is important to realize that the best cut cost improvement leading to the selection of a pair $(a_i, b_i)$ may be negative. Once all vertices have been locked, the pairs are investigated in the order of selection: the actual subsets to be interchanged correspond to the sequence of pairs (starting with $i = 1$) giving the best improvement. So, pairs in the sequence may have negative cost improvements as long as the pairs following them compensate for it. Such a situation would e.g. occur when the exchange of two clusters of tightly connected vertices results in an improvement, while the exchange of individual vertices from each cluster does not improve the cut cost.

The algorithm will be illustrated using the example graph given in Figure 7.12. The graph consists of the vertices $v_1$ to $v_8$. The initial partition consists of the two sets $\{v_2, v_3, v_6, v_7\}$ and $\{v_1, v_4, v_5, v_8\}$ with cut cost 14 as shown in Figure 7.12(a). The evolution of the algorithm in the first pass is shown in Figure 7.13. Each row in the figure corresponds to a value of the loop variable $i$ as is shown in the first column. The values under a column headed by a vertex name $v$ show the subsequent values of the variable $D_v$ (the difference between the external cost $E_v$ and the internal cost

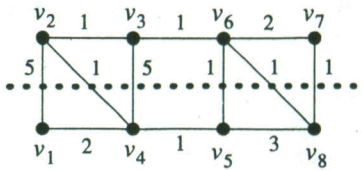
```
initialize(A^0, B^0);
m ← 1;
do { for each a ∈ A^{m-1}
       "compute D_a";
     for each b ∈ B^{m-1}
       "compute D_b";
     for (i ← 1; i ≤ n; i ← i + 1) {
       "find unlocked vertices a_i ∈ A^{m-1}, b_i ∈ B^{m-1} such that
        Δ_i = D_{a_i} + D_{b_i} − 2γ_{a_i b_i} is maximal";
       "lock a_i and b_i";
       for each "unlocked" x ∈ A^{m-1}
         D_x ← D_x + 2γ_{x a_i} − 2γ_{x b_i};
       for each "unlocked" y ∈ B^{m-1}
         D_y ← D_y − 2γ_{y a_i} + 2γ_{y b_i};
     }
     "find a k such that Σ_{i=1}^{k} Δ_i is maximal";
     G ← Σ_{i=1}^{k} Δ_i;
     if (G > 0) {
       X^m ← {a_1, ..., a_k};
       Y^m ← {b_1, ..., b_k};
       A^m ← (A^{m-1} \ X^m) ∪ Y^m;
       B^m ← (B^{m-1} \ Y^m) ∪ X^m;
       "unlock all vertices in A^m and B^m";
       m ← m + 1
     }
   }
while (G > 0);
```

**Figure 7.11** The pseudo-code description of the Kernighan-Lin algorithm.

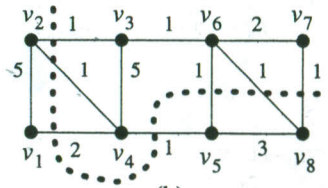
The variables that are locked at a specific stage of the inner loop are underlined the value $\Delta_i$ corresponding to the selected pair is given in the last column.

Once the inner loop has been traversed, the value $k$ has to be determined such the sum $\sum_1^k \Delta_i$ is maximal. In this case, the value of $k$ is 1 and the subsets of ces to be exchanged are $\{v_2\}$ and $\{v_4\}$ (note that $k = 3$ is optimal as well). This to the new sets $A^1 = \{v_3, v_4, v_6, v_7\}$ and $B^1 = \{v_1, v_2, v_5, v_8\}$ as is shown in re 7.12(b). The cut cost has now decreased from 14 to 8.

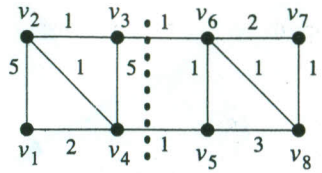
The evolution of the variables in the second pass is shown in Figure 7.14. The clusion of the second iteration is that the maximal gain corresponds to the e of $k = 2$, which means that the vertex subsets to be exchanged are $\{v_3, v_4\}$ $\{v_5, v_8\}$. The exchange of these subsets leads to $A^2 = \{v_1, v_2, v_3, v_4\}$ and $= \{v_5, v_6, v_7, v_8\}$ which is the optimal solution for the example. The cut cost equals 2. Clearly, a third pass will not lead to an improvement and the algorithm stop. The final partition is illustrated in Figure 7.12(c). Note that the Kernighan-algorithm will not always find the optimal solution: it is just a powerful heuristic.




(a)




(b)




(c)

**Figure 7.12** Different steps in the application of the Kernighan-Lin algorithm: the initial partition (a), the situation after the first iteration (b) and the final solution (c).

| | $A^0$ | | | | $B^0$ | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $i$ | $v_2$ | $v_3$ | $v_6$ | $v_7$ | $v_1$ | $v_4$ | $v_5$ | $v_8$ | $\Delta_i$ |
| 1 | 5 | 3 | −1 | −1 | 3 | 3 | −1 | −3 | 6 |
| 2 | | −5 | −1 | −1 | −3 | | −1 | −1 | −2 |
| 3 | | −5 | 1 | | −3 | | | 3 | 2 |
| 4 | | −3 | | | −3 | | | | −6 |

**Figure 7.13** The first pass of the Kernighan-Lin algorithm applied to the graph of Figure 7.12.

| | $A^1$ | | | | $B^1$ | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $i$ | $v_3$ | $v_4$ | $v_6$ | $v_7$ | $v_1$ | $v_2$ | $v_5$ | $v_8$ | $\Delta_i$ |
| 1 | −5 | −1 | −1 | −1 | −3 | −3 | −1 | −1 | −2 |
| 2 | 5 | | −3 | −3 | −7 | −5 | 3 | | 8 |
| 3 | | | −3 | −3 | −7 | −7 | | | −10 |
| 4 | | | | 1 | | 3 | | | 4 |

**Figure 7.14** The second pass of the Kernighan-Lin algorithm applied to the graph of Figure 7.12.



time complexity of the algorithm is determined by the inner loop that is executed $n$ times (remember that the number of executions of the outer loop was problem-dependent). Finding the best pair of vertices to be locked next requires pairwise comparison of elements each with at most $n$ elements. Therefore, the number of comparisons is $\mathcal{O}(n^2)$ and the total time complexity becomes $\mathcal{O}(n^3)$. In typical situations (but not in the worst case) sorting the elements of $A^{m-1}$ and $B^{m-1}$ according to their difference values will limit the number of comparisons to the first few elements of the sorted lists. As sorting can be done in $\mathcal{O}(n \log n)$ time, this would lead to an overall time complexity of $\mathcal{O}(n^2 \log n)$. A better time complexity can be achieved by the use of more sophisticated data structures and a modified search strategy (see also the Bibliographic Notes).

Note that the Kernighan-Lin algorithm can be seen as a local search with variable neighborhood (see Section 5.5): the number of elements exchanged between the sets $A^{m-1}$ and $B^{m-1}$ depends on the feasible solution that one is visiting.

## 5 Bibliographic Notes

Many of the textbooks that deal with physical design automation, such as [Len90], [Wol93], [Sai95] and [Sar96], pay quite some attention to the placement problem. In addition, general information on the placement problem can be obtained through the review papers [Got86], [Bra87] and [Pre88].

Hypergraph circuit representation as presented in this chapter is not so often discussed in the literature, as it is quite straightforward and because different applications have different requirements. Circuit representations that are e.g. appropriate for placement and partitioning are paid attention to in [Len90] and [Alp95]. What has been presented here is an *internal* data structure, meant to reside in the computer's memory while the algorithm is processing it. A related issue is the *external* representation meant to be stored on file, usually in a human-readable format, that can be shared by many applications. Such a format is EDIF (Electronic Design Interchange Format), that has been standardized by ANSI [Kah92a]. It deals with the notions of cells, ports and nets and many, many more issues that are relevant for real-world CAD.

The wire-length metrics mentioned in the text and some others are listed in [Wol93], [Len90] and [Sai95].

The term *logistic signal* (for power and clock wires) was coined by Spaanenburg [Spa85]. An example of a min-cut algorithm for placement is given in [Bre77]. Examples of clustering algorithms can be found in [Ake82].

As reported in [Sun95] simulated annealing is probably the best algorithm currently known for placement, especially for standard-cell placement. Already the first publication on simulated annealing as a general-purpose optimization method [Kir83] presented results applied to the placement problem. Also, the introductory paper on simulated annealing [Rut89] uses the placement problem as the main illustration of the method. The Timberwolf system [Sec85] is famous for its good results in standard-cell placement by simulated annealing. The version discussed in [Sun95] directly eliminates overlaps after each move. A "rectangle packing" method based