Chapter 2: Partitioning

Sadiq M. Sait & Habib Youssef

King Fahd University of Petroleum & Minerals
College of Computer Sciences & Engineering
Department of Computer Engineering

September 2003

Introduction

- Introduction to Partitioning
- Problem Definition
- Cost Function and Constraints
- Approaches to Partitioning
  1. Kernighan-Lin Heuristic
  2. Fiduccia-Mattheyses heuristic
  3. Simulated Annealing

Partitioning

- Partitioning is assignment of logic components to physical packages.
  1. Circuit is too large to be placed on a single chip.
  2. I/O pin limitations.
- Relationship between the number of gates and the number of I/O pins is estimated by Rent‘s rule,

$$IO = tG^r$$

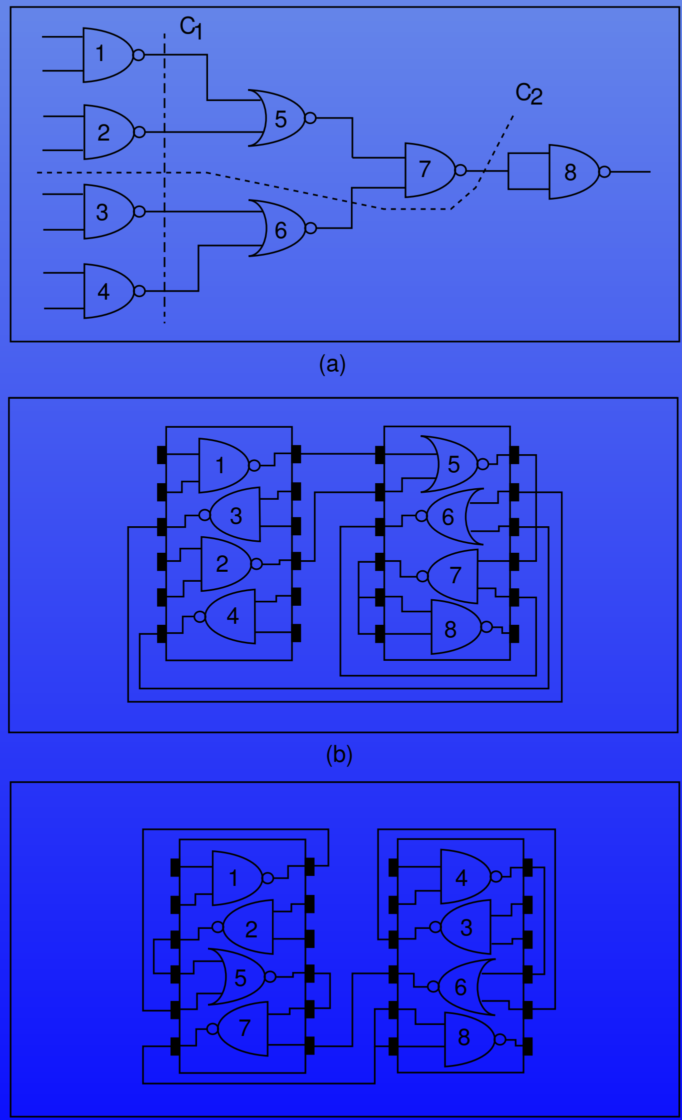
where:
IO: number of I/O pins,
t: number of terminals per gate,
G: the number of gates in the circuit, and
r is Rent’s exponent ($0 < r < 1$).

Partitioning - contd

- A large pin count increases dramatically the cost of packaging the circuit.
- The number of I/O pins must correspond to one of the standard packaging technologies - 12, 40, 128, 256 etc.
- When it becomes necessary to split a circuit across packages, care must be exercised to minimize cross-package interconnections. because off-chip wires are undesirable.

  1. Electrical signals travel slower along wires external to the chip.
  2. Off-chip wires take up area on a PCB and reduce reliability. Printed wiring and plated-through holes are both likely sources of trouble.
  3. Finally, since off-chip wires must originate and terminate into I/O pins, more off-chip wires essentially mean more I/O pins.

Partitioning - examples



(a)

(b)

(c)

K-way Partitioning

Given:

- A graph $G(V, E)$, where each vertex $v \in V$ has a *size* $s(v)$, and each edge $e \in E$ has a *weight* $w(e)$.

Output:

- A division of the set V into k subsets $V_1, V_2, \cdots, V_k$, such that
  1. an objective function is optimized,
  2. subject to certain constraints.

Constraints

- The cutset of a partition is indicated by ψ and is equal to the set of edges cut by the partition.
- The size of the i^{th} subcircuit is given by

$$S(V_i) \;=\; \sum_{v \in V_i} s(v)$$

where $s(v)$ is the size of a node v (area of the corresponding circuit element).

- Let A_i be the upper bound on the size of i^{th} subcircuit; then,

$$\sum_{v \in V_i} s(v) \leq A_i$$

Constraints - contd

- If it is desirable to divide the circuit into roughly equal sizes then,

$$S(V_i) \;=\; \sum_{v \in V_i} s(v) \leq \lceil \frac{1}{k} \sum_{v \in V} s(v) \rceil = \frac{1}{k} S(V)$$

- If all the circuit elements have the same size, then above equation reduces to:

$$n_i \leq \frac{n}{k}$$

where n_i and n are the number of elements in V_i and in V respectively.

Cost Functions

Minimize External Wiring.

$$Cost = \sum_{e \in \psi} w(e)$$

where $w(e)$ is the cost of edge/connection e.

- Let the partitions be numbered $1, 2, \cdots, k$, and $p(u)$ be the partition number of node u.
- Equivalently, one can write the function $Cost$ as follows:

$$Cost = \sum_{\forall e=(u,v) \& p(u) \neq p(v)} w(e)$$

Two-Way Partitioning

- Given a circuit with $2n$ elements, we wish to generate a balanced two-way partition of the circuit into two subcircuits of n elements each.
- The cost function is the size of the cutset.
- If we do not place the constraint that the partition be *balanced*, the two-way partitioning problem (TWPP) is easy. One simply applies the well known max-flow mincut.
- However, the balance criterion is extremely important in practice and cannot be overlooked. This constraint makes TWPP NP-Complete.

Two-Way Partitioning-contd

- A number of “heuristic” techniques can be used to find a good feasible solution.
  1. Deterministic.
     (a) Kernighan-Lin.
     (b) Fiduccia-Mattheyes.
  2. Non-Deterministic.
     (a) Simulated Annealing.
     (b) Genetic Algorithm.
     (c) Tabu Search.
  3. Constructive vs. Iterative.

Two-Way Partitioning-contd

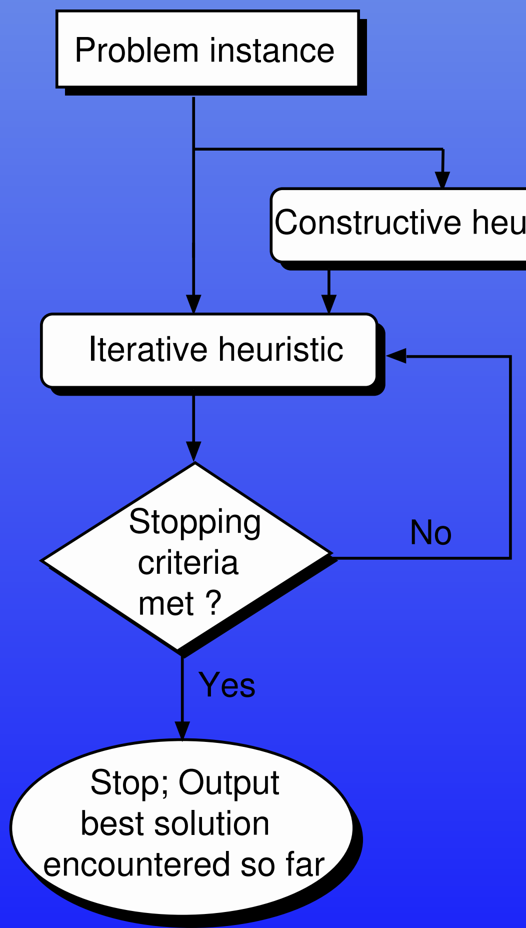


Figure 2: General structure combining constructive and iterative heuristics

Kernighan-Lin Algorithm

- Most popular algorithm for the two-way partitioning problem.
- The algorithm can also be extended to solve more general partitioning problems.
- The problem is characterized by a *connectivity matrix* C. Element c_{ij} represents the sum of weights of the edges connecting elements i and j.
- In TWPP, since the edges have unit weights, c_{ij} simply counts the number of edges connecting i and j.
- The output of the partitioning algorithm is a pair of sets A and B such that $|A| = n = |B|$, and $A \cap B = \emptyset$, and such that the size of the cutset T is minimized.

K-L Algorithm - contd

$$T = \sum_{a \in A, b \in B} c_{ab}$$

- Kernighan-Lin heuristic is an iterative improvement algorithm. It starts from an initial partition (A, B) such that $|A| = n = |B|$, and $A \cap B = \emptyset$.
- How can a given partition be improved?
- Let $P^* = \{A^*, B^*\}$ be the optimum partition and $P = \{A, B\}$ be the current partition.
- Then, in order to attain P^* from P, one has to swap a subset $X \subseteq A$ with a subset $Y \subseteq B$ such that,

  (1) $|X| = |Y|$

  (2) $X = A \cap B^*$

  (3) $Y = A^* \cap B$

K-L Algorithm - contd

- $A^* = (A - X) + Y$ and $B^* = (B - Y) + X$.
- The problem of identifying X and Y is as hard as that of finding $P^* = \{A^*, B^*\}$.

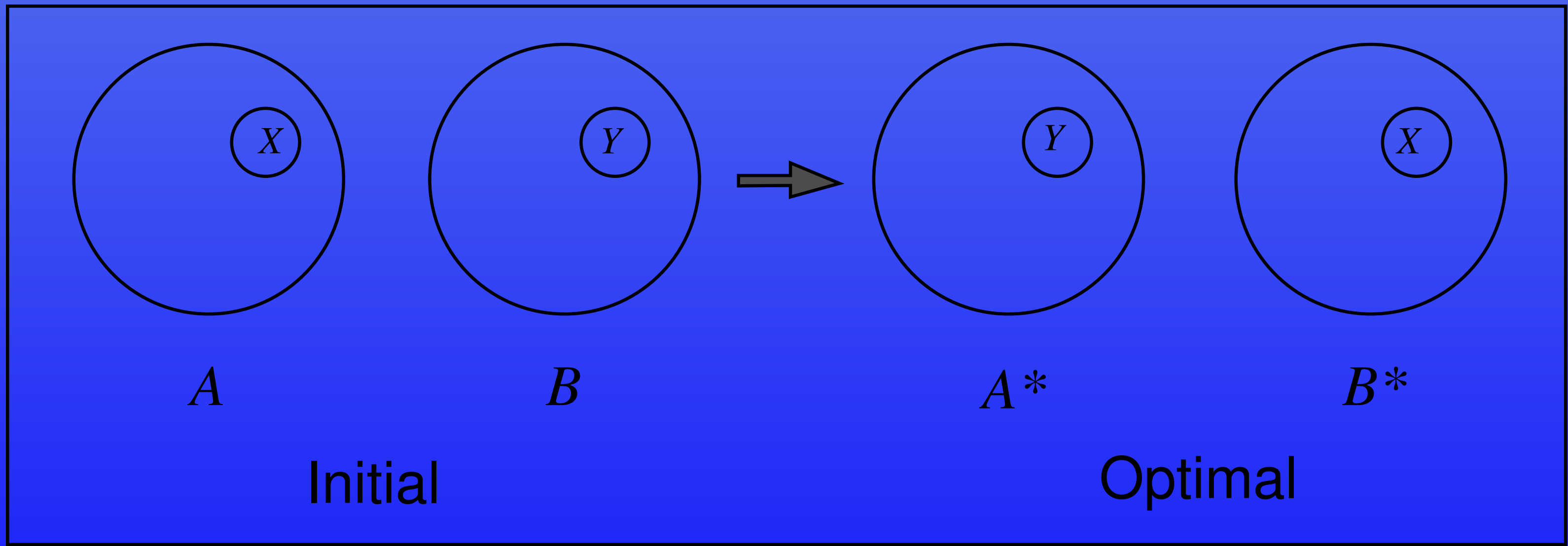


Figure 3: Initial & optimal partitions

Definitions

Definition 1:

Consider any node a in block A. The contribution of node a to the cutset is called the external cost of a and is denoted as E_a, where

$$E_a = \sum_{v \in B} c_{av}$$

Definition 2:

The internal cost I_a of node $a \in A$ is defined as follows.

$$I_a = \sum_{v \in A} c_{av}$$

Definitions

Moving node a from block A to block B would increase the value of the cutset by I_a and decrease it by E_a.

Therefore, the benefit of moving a from A to B is

$$D_a = E_a - I_a$$

Example

Consider the figure with, I_a=2, I_b=3, E_a =3, E_b=1, $D_a = 1$, and $D_b = -2$.

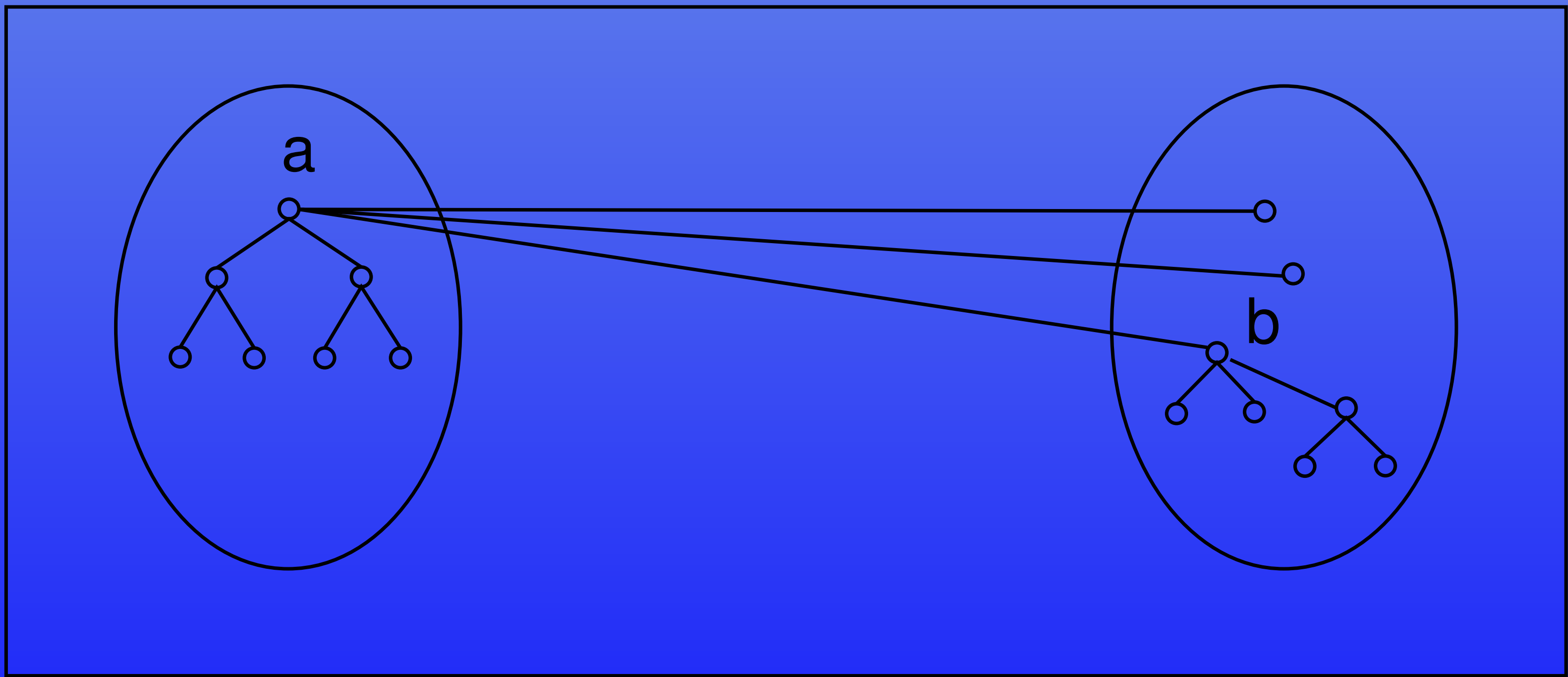


Figure 4: Internal cost versus external costs

Example-contd

- To maintain balanced partition, we must move a node from B to A each time we move a node from A to B.
- The effect of swapping two modules $a \in A$ with $b \in B$ is characterized by the following lemma.

Lemma 1:

- *If two elements $a \in A$ and $b \in B$ are interchanged, the reduction in the cost is given by*

$$g_{ab} = D_a + D_b - 2c_{ab}$$

Proof

- The external cost can be re-written as

$$E_a = c_{ab} + \sum_{v \in B, v \neq b} c_{av}$$

- Therefore,

$$D_a = E_a - I_a = c_{ab} + \sum_{v \in B, v \neq b} c_{av} - I_a$$

- Similarly

$$D_b = E_b - I_b = c_{ab} + \sum_{u \in A, u \neq a} c_{bu} - I_b$$

Proof - contd

- Moving a from A to B reduces the cost by

$$\sum_{v \in B, v \neq b} c_{av} - I_a = D_a - c_{ab}$$

- Moving b from B to A reduces the cost by

$$\sum_{u \in A, u \neq a} c_{bu} - I_b = D_b - c_{ab}$$

- When both moves are carried out, the total cost reduction is given by the sum of above two equations, that is

$$g_{ab} = D_a + D_b - 2c_{ab}$$

Proof - contd

- The following lemma tells us how to update the $D-$values after a swap.

Lemma 2:

- *If two elements $a \in A$ and $b \in B$ are interchanged, then the new $D-$values are given by*

$$D_x^{'} = D_x + 2c_{xa} - 2c_{xb}, \ \ \forall x \in A - \{a\}$$
$$D_y^{'} = D_y + 2c_{yb} - 2c_{ya}, \ \ \forall y \in B - \{b\}$$

- Notice that if a module x is neither connected to a nor to b then $c_{xa} = c_{xb} = 0$, and, $D_x' = D_x$.

Proof - contd

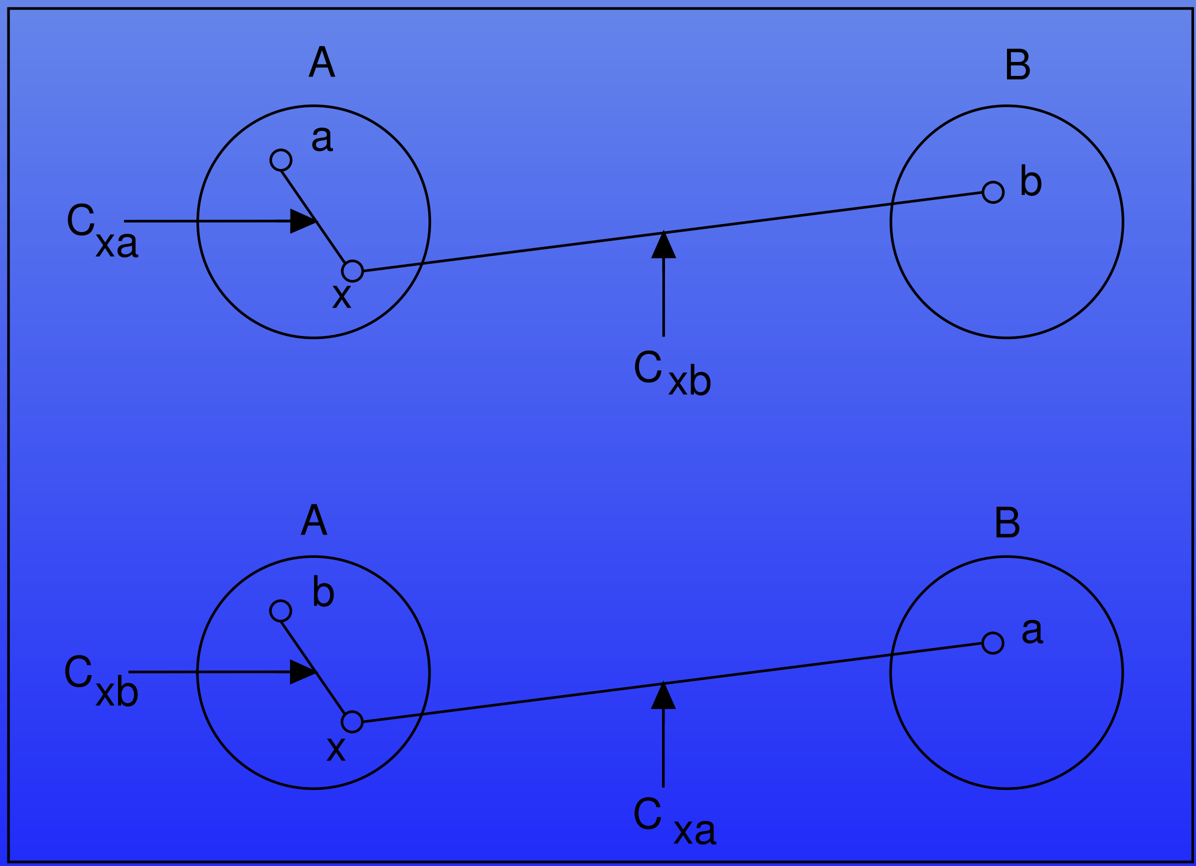


Figure 5: Updating D-Values after an exchange

Proof - contd

- Consider a node $x \in A - \{a\}$. Since b has entered block A, the internal cost of x increases by c_{xb}.
- Similarly, since a has entered the opposite block B, the internal cost of x must be decreased by c_{xa}.
- The new internal cost of x therefore is

$$I_x^{'} = I_x - c_{xa} + c_{xb}$$

Proof - contd

- One can similarly show that the new external cost of x is

$$E_x^{'} = E_x + c_{xa} - c_{xb}$$

- Thus the new $D-$value of $x \in A - \{a\}$ is

$$D_x^{'} = E_x^{'} - I_x^{'} = D_x + 2c_{xa} - 2c_{xb}$$

- Similarly, the new $D-$value of $y \in B - \{b\}$ is

$$D_y^{'} = E_y^{'} - I_y^{'} = D_y + 2c_{yb} - 2c_{ya}$$

Overview of K-L Algorithm:

- Start from an initial partition $\{A, B\}$ of n elements each.
- Use lemmas 1 and 2 together with a greedy procedure to identify two subsets $X \subseteq A$, and $Y \subseteq B$, of equal cardinality, such that when interchanged, the partition cost is improved.
- X and Y may be empty, indicating in that case that the current partition can no longer be improved.

Greedy Procedure-Identify X,Y

1. Compute g_{ab} for all $a \in A$ and $b \in B$.
2. Select the pair (a_1, b_1) with maximum gain g_1 and lock a_1 and b_1.
3. Update the $D-$values of remaining free cells and recompute the gains.
4. Then a second pair (a_2, b_2) with maximum gain g_2 is selected and locked. Hence, the gain of swapping the pair (a_1, b_1) followed by the (a_2, b_2) swap is $G_2 = g_1 + g_2$.
5. Continue selecting $(a_3, b_3), \cdots, (a_i, b_i), \cdots, (a_n, b_n)$ with gains g_3, $\cdots, g_i, \cdots, g_n$.
6. The gain of making the swap of the first k pairs is $G_k = \sum_{i=1}^{k} g_i$. If there is no k such that $G_k > 0$ then the current partition cannot be improved; otherwise choose the k that maximizes G_k, and make the interchange of $\{a_1, a_2, \cdots, a_k\}$ with $\{b_1, b_2, \cdots, b_k\}$ permanent.

Iterative Improvement

- The above improvement procedure constitutes a single pass of the Kernighan-Lin procedure.
- The partition obtained after the i^{th} pass constitutes the initial partition of the $i + 1^{st}$ pass.
- Iterations are terminated when $G_k \leq 0$, that is, no further improvements can be obtained by pairwise swapping.

K-L algorithm for TWPP

Algorithm KL_TWPP
Begin
Step 1. V = set of $2n$ elements; $\{A, B\}$ is initial partition such that
$|A| = |B|$; $A \cap B = \emptyset$; and $A \cup B = V$;
Step 2. Compute D_v for all $v \in V$; $queue \leftarrow \phi$; and $i \leftarrow 1$;
$A' = A$; $B' = B$;
Step 3. Choose $a_i \in A'$, $b_i \in B'$, which maximizes
$g_i = D_{a_i} + D_{b_i} - 2c_{a_i b_i}$;
add the pair (a_i, b_i) to *queue*;
$A^{'} = A^{'} - \{a_i\}$; $B^{'} = B^{'} - \{b_i\}$;
Step 4. **If** A' and B' are both empty **then Goto** Step 5
Else recalculate $D-$values for $A' \cup B'$;
$i \leftarrow i + 1$; **Goto** Step 3;
Step 5. Find k to maximize the partial sum
G=$\sum_{i=1}^{k} g_i$;
If $G > 0$ then
Move $X = \{a_1, \cdots, a_k\}$ to B;
move $Y = \{b_1, \cdots, b_k\}$ to A;
Goto Step 2
Else STOP
EndIf **End**.

Example

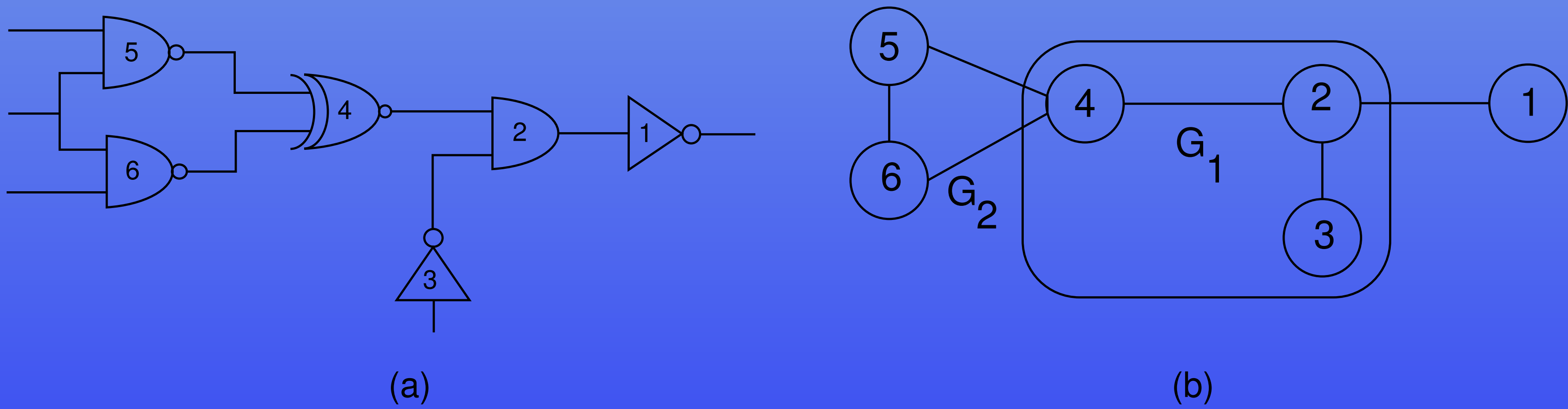


Figure 6: (a) A circuit to be partitioned (b) Its corresponding graph

Example - contd

- Step 1: Initialization.

  Let the initial partition be a random division of vertices into the partition A={2,3,4} and B={1,5,6}.

  $A' = A$ ={2,3,4}, and $B' = B$ ={1,5,6}.

- Step 2: Compute $D-$values.

$$
\begin{aligned}
D_1 &= E_1 - I_1 = 1 - 0 = +1 \\
D_2 &= E_2 - I_2 = 1 - 2 = -1 \\
D_3 &= E_3 - I_3 = 0 - 1 = -1 \\
D_4 &= E_4 - I_4 = 2 - 1 = +1 \\
D_5 &= E_5 - I_5 = 1 - 1 = +0 \\
D_6 &= E_6 - I_6 = 1 - 1 = +0
\end{aligned}
$$

Example - contd

- Step 3: Compute gains.

$$
\begin{aligned}
g_{21} &= D_2 + D_1 - 2c_{21} = (-1) + (+1) - 2(1) = -2 \\
g_{25} &= D_2 + D_5 - 2c_{25} = (-1) + (+0) - 2(0) = -1 \\
g_{26} &= D_2 + D_6 - 2c_{26} = (-1) + (+0) - 2(0) = -1 \\
g_{31} &= D_3 + D_1 - 2c_{31} = (-1) + (+1) - 2(0) = +0 \\
g_{35} &= D_3 + D_5 - 2c_{35} = (-1) + (+0) - 2(0) = -1 \\
g_{36} &= D_3 + D_6 - 2c_{36} = (-1) + (+0) - 2(0) = -1 \\
g_{41} &= D_4 + D_1 - 2c_{41} = (+1) + (+1) - 2(0) = +2 \\
g_{45} &= D_4 + D_5 - 2c_{45} = (+1) + (+0) - 2(1) = -1 \\
g_{46} &= D_4 + D_6 - 2c_{46} = (+1) + (+0) - 2(1) = -1
\end{aligned}
$$

- The largest g value is g_{41}. (a_1, b_1) is $(4, 1)$, the gain $g_{41} = g_1$= 2, and $A' = A' - \{4\}$=$\{2,3\}$, $B' = B' - \{1\} = \{5, 6\}$.

Example - contd

- Both A' and B' are not empty; then we update the $D-$values in the next step and repeat the procedure from Step 3.
- Step 4: Update $D-$values of nodes connected to (4,1).

  The vertices connected to (4,1) are vertex (2) in set A' and vertices (5,6) in set B'. The new $D-$values for vertices of A' and B' are given by

$$D_2^{'} = D_2 + 2c_{24} - 2c_{21} = -1 + 2(1-1) = -1$$
$$D_5^{'} = D_5 + 2c_{51} - 2c_{54} = +0 + 2(0-1) = -2$$
$$D_6^{'} = D_6 + 2c_{61} - 2c_{64} = +0 + 2(0-1) = -2$$

Example - contd

- To repeat Step 3, we assign $D_i = D_i^{'}$ and then recompute the gains:

$$g_{25} = D_2 + D_5 - 2c_{25} = (-1) + (-2) - 2(0) = -3$$
$$g_{26} = D_2 + D_6 - 2c_{26} = (-1) + (-2) - 2(0) = -3$$
$$g_{35} = D_3 + D_5 - 2c_{35} = (-1) + (-2) - 2(0) = -3$$
$$g_{36} = D_3 + D_6 - 2c_{36} = (-1) + (-2) - 2(0) = -3$$

- All the g values are equal, so we arbitrarily choose g_{36}, and hence the pair (a_2, b_2) is $(3, 6)$,

$$g_{36} = g_2 = -3,$$
$$A' = A' - \{3\} = \{2\},$$
$$B' = B' - \{6\} = \{5\}.$$

Example - contd

- The new $D-$values are:

$$D_2^{'} = D_2 + 2c_{23} - 2c_{26} = -1 + 2(1-0) = 1$$
$$D_5^{'} = D_5 + 2c_{56} - 2c_{53} = -2 + 2(1-0) = 0$$

- The corresponding new gain is:
$g_{25} = D_2 + D_5 - 2c_{52} = (+1) + (0) - 2(0) = +1$
- Therefore the last pair (a_3, b_3) is (2,5) and the corresponding gain is $g_{25} = g_3 = +1$.

Example - contd

- Step 5: Determine k.
- We see that $g_1 = +2$, $g_1 + g_2 = -1$, and $g_1 + g_2 + g_3 = 0$.
- The value of k that results in maximum G is 1.
- Therefore, $X = \{a_1\} = \{4\}$ and $Y = \{b_1\} = \{1\}$.
- The new partition that results from moving X to B and Y to A is, $A = \{1, 2, 3\}$ and $B = \{4, 5, 6\}$.
- The entire procedure is repeated again with this new partition as the initial partition.
- Verify that the second iteration of the algorithm is also the last, and that the best solution obtained is $A = \{1, 2, 3\}$ and $B = \{4, 5, 6\}$.

Time Complexity

- Computation of the $D-$values requires $O(n^2)$ time $((O(n)$ for each node).
- It takes constant time to update any $D-$value. We update as many as $(2n-2i)$ $D-$values after swapping the pair (a_i, b_i).
- Therefore the total time spent in updating the $D-$values can be

$$\sum_{i=1}^{n}(2n-2i) = O(n^2)$$

- The pair selection procedure is the most expensive step in the Kernighan-Lin algorithm. If we want to pick (a_i, b_i), there are as many as $(n-i+1)^2$ pairs to choose from leading to an overall complexity of $O(n^3)$.

Time Complexity - contd

- To avoid looking at all pairs, one can proceed as follows.
- Recall that, while selecting (a_i, b_i), we want to maximize $g_i = D_{a_i} + D_{b_i} - 2c_{a_i b_i}$.
- Suppose that we sort the $D-$values in a decreasing order of their magnitudes. Thus, for elements of Block A,

$$D_{a_1} \geq D_{a_2} \geq \cdots \geq D_{a_{(n-i+1)}}$$

- Similarly, for elements of Block B,

$$D_{b_1} \geq D_{b_2} \geq \cdots \geq D_{b_{(n-i+1)}}$$

Time Complexity - contd

- Sorting requires $O(n \log n)$.
- Next, we begin examining D_{a_i} and D_{b_j} pairwise.
- If we come across a pair (D_{a_k}, D_{b_l}) such that $(D_{a_k} + D_{b_l})$ is less than the gain seen so far in this improvement phase, then we do not have to examine any more pairs.
- Hence, if $D_{a_k} + D_{b_l} < g_{ij}$ for some i, j then $g_{kl} < g_{ij}$.
- Since it is almost never required to examine all the pairs (D_{a_i}, D_{b_j}), the overall complexity of selecting a pair (a_i, b_i) is $O(n \log n)$.
- Since n exchange pairs are selected in one pass, the complexity of Step 3 is $O(n^2 \log n)$.

Time Complexity - contd

- Step 5 takes only linear time.
- The complexity of the Kernighan-Lin algorithm is $O(pn^2 \log n)$, where p is the number of iterations of the improvement procedure.
- Experiments on large practical circuits have indicated that p does not increase with n.
- The time complexity of the pair selection step can be improved by scanning the unsorted list of $D-$values and selecting a and b which maximize D_a and D_b. Since this can be done in linear time, the algorithm's time complexity reduces to $O(n^2)$.
- This scheme is suited for sparse matrices where the probability of $c_{ab} > 0$ is small. Of course, this is an approximation of the greedy selection procedure, and may generate a different solution as compared to greedy selection.

Variations of K-L Algorithm

The Kernighan-Lin algorithm may be extended to solve several other cases of the partitioning problem.

Unequal sized blocks. Partitioning of a graph $G = (V, E)$ with $2n$ vertices into two subgraphs of unequal sizes n_1 and n_2, $n_1 + n_2 = 2n$.

1. Divide the set V into two subsets A and B, one containing $MIN(n_1, n_2)$ vertices and the other containing $MAX(n_1, n_2)$ vertices (this division may be done arbitrarily).
2. Apply the algorithm starting from Step 2, but restrict the maximum number of vertices that can be interchanged in one pass to $MIN(n_1, n_2)$.

Another approach

- Another possible approach would be to proceed as follows.
- Assume that $n_1 < n_2$.
- To divide V such that there are *at least* n_1 vertices in block A and *at most* n_2 vertices in block B, the procedure shown below may be used:
  1. Divide the set V into blocks A and B; A containing n_1 vertices and B containing n_2 vertices.
  2. Add $n_2 - n_1$ dummy vertices to block A. *Dummy* vertices have no connections to the original graph.
  3. Apply the algorithm starting from Step 2.
  4. Remove all dummy vertices.

Another approach - contd

Unequal sized elements

- To generate a two-way partition of a graph whose vertices have unequal sizes, we may proceed as follows:
  1. Without loss of generality assume that the smallest element has unit size.
  2. Replace each element of size s with s vertices which are fully connected with edges of infinite weight. (In practice, the weight is set to a very large number M.)
  3. Apply the original Kernighan-Lin algorithm.

$k-$way partition

- Assume that the graph has $k \cdot n$ vertices, $k > 2$, and it is required to generate a $k-$way partition, each with n elements.
  1. Begin with a random partition of k sets of n vertices each.
  2. Apply the two-way partitioning procedure on each pair of partitions.

$k-$way partition - contd

- Pairwise optimality is only a necessary condition for optimality in the $k-$way partitioning problem. Usually a complex interchange of 3 or more items from 3 or more subsets will be required to reduce the pairwise optimal to the global optimal solution.
- Since there are $\binom{k}{2}$ pairs to consider, the time complexity for one pass through all pairs for the $O(n^2)$-*procedure* is $\binom{k}{2}n^2 = O(k^2n^2)$.
- In general, more passes than this will be actually required, because when a particular pair of partitions is optimized, the optimality of these partitions with respect to others may change.

Fiduccia-Mattheyses Heuristic

- Fiduccia-Mattheyses heuristic is an iterative procedure that takes into consideration multipin nets as well as sizes of circuit elements.
- Fiduccia-Mattheyses heuristic is a technique used to find a solution to the following bipartitioning problem:
- *Given a circuit consisting of C cells connected by a set of N nets (where each net connects at least two cells), the problem is to partition circuit C into two blocks A and B such that the number of nets which have cells in both the blocks is minimized and a balance factor r is satisfied.*

Illustration

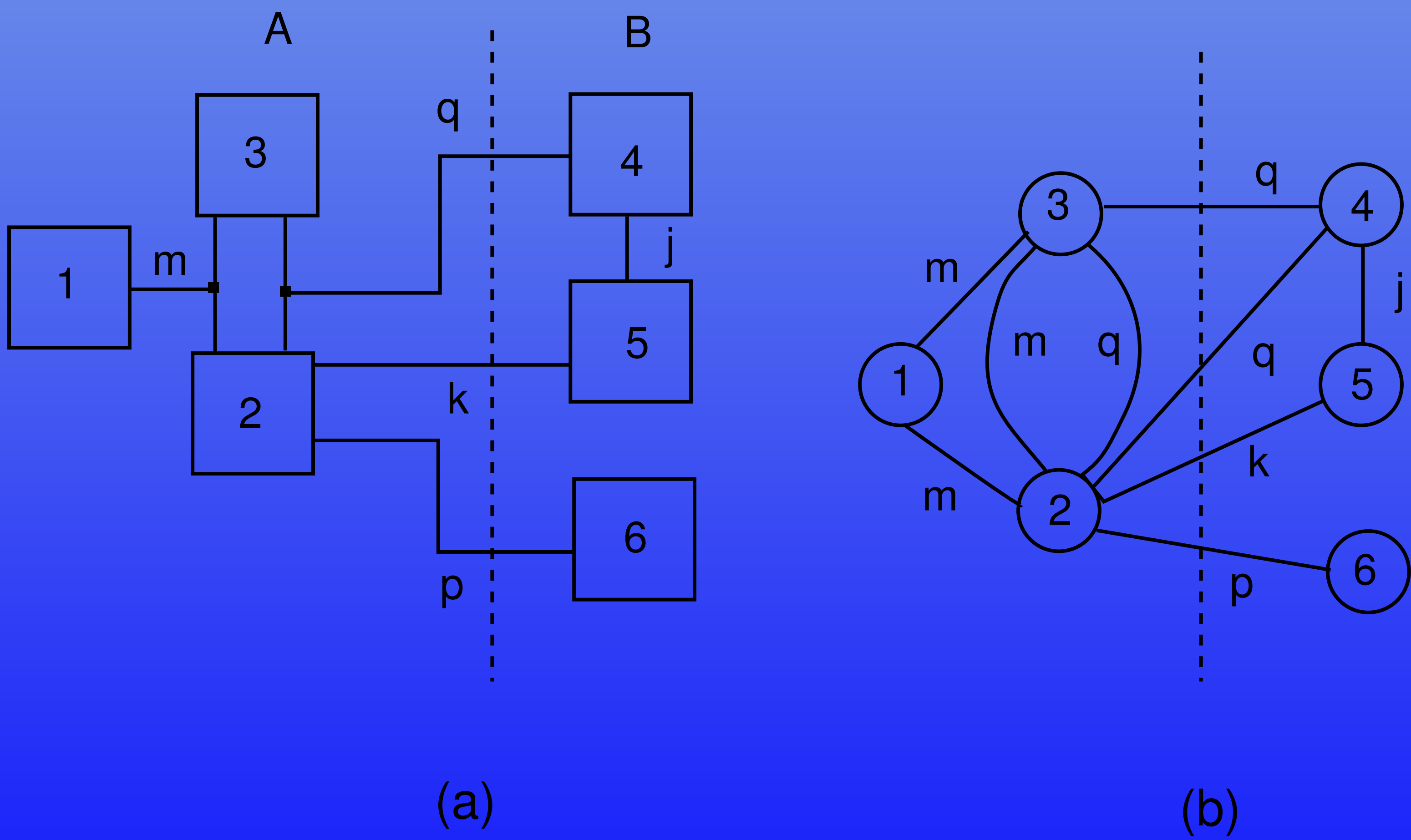


Figure 7: Illustration of (a) Cut of nets. (b) Cut of edges

KL vs. FM heuristics

- Movement of cells
    1. In Kernighan-Lin heuristic, during each pass a **pair** of cells is selected for swapping, one from each block.
    2. In the Fiduccia-Mattheyses heuristic a *single* cell at a time, from either block is selected and considered for movement to the other block.
- Objective of partitioning
    1. Kernighan-Lin heuristic partitions a graph into two blocks such that the cost of $edges$ cut is minimum.
    2. Fiduccia-Mattheyses heuristic aims at reducing the cost of $nets$ cut by the partition.

KL vs. FM heuristics - contd

- Selection of cells
    1. Fiduccia-Mattheyses heuristic is similar to the Kernighan-Lin in the selection of cells. But the gain due to the movement of a single cell from one block to another is computed instead of the gain due to swap of two cells. Once a cell is selected for movement, it is locked for the remainder of that pass. The total number of cells that can change blocks is then given by the best sequence of moves $c_1, c_2, \cdots, c_k$.
    2. In contrast, in Kernighan-Lin the first best k pairs in a pass are swapped.

KL vs. FM heuristics - contd

- Balance criterion?
  1. Kernighan-Lin heuristic can produce imbalanced partition in case cells are of different sizes.
  2. Fiduccia-Mattheyses heuristic is designed to handle imbalance, and it produces a balanced partition with respect to size. The balance factor r (called $ratio$) is user specified and is defined as follows: $r = \frac{|A|}{|A|+|B|}$, where $|A|$ and $|B|$ are the sizes of partitioned blocks A and B.
- Some of the cells can be initially locked to one of the partitions.
- Time complexity of Fiduccia-Mattheyses heuristic is linear. In practice only a very small number of passes are required leading to a fast approximate algorithm for min-cut partitioning.

Definitions

Let $p(j)$ be the number of pins of cell 'j', and $s(j)$ be the size of cell 'j', for $j = 1, 2, \cdots, C$. If V is the set of the C cells, then $|V| = \sum_{i=1}^{C} s(i)$.

"Cutstate of a net" : A net is said to be cut if it has cells in both blocks, and is $uncut$ otherwise. A variable $cutstate$ is used to denote the state of a net.

"Cutset of partition" : The $cutset$ of a partition is the cardinality of the set of all nets with cutstate equal to cut.

"Gain of cell" : The gain $g(i)$ of a cell 'i' is the number of nets by which the cutset would decrease if cell 'i' were to be moved. A cell is $moved$ from its current block (the From_block) to its complementary block (the To_block).

Definitions - contd

"Balance criterion" : To avoid having all cells migrate to one block a balancing criterion is maintained.
A partition (A, B) is balanced iff

(1) $$r \times |V| - s_{max} \leq |A| \leq r \times |V| + s_{max}$$

where $|A| + |B| = |V|$; and $s_{max} = Max[s(i)]$, $i \in A \cup B = V$.

"Base cell" : The cell selected for movement from one block to another is called "*base cell*". It is the cell with maximum gain and the one whose movement will not violate the balance criterion.

Definitions - contd

"Distribution of a net" : Distribution of a net n is a pair $(A(n), B(n))$ where (A, B) is an arbitrary partition, and, $A(n)$ is the number of cells of net n that are in A and $B(n)$ is the number of cells of net n that are in B.

"Critical net" : A net is critical if it has a cell which if moved will change its cutstate. That is, if and only if $A(n)$ is either 0 or 1, or $B(n)$ is either 0 or 1.

Illustration of critical nets

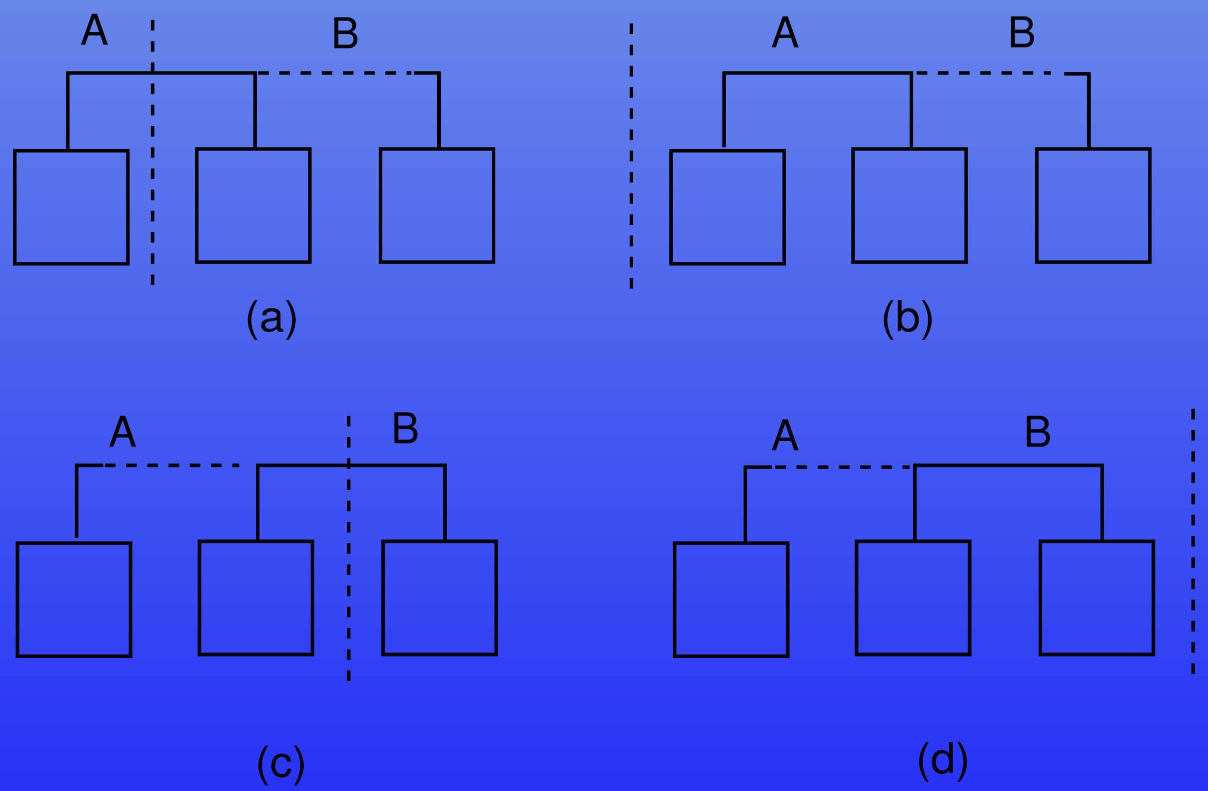


Figure 8: Block to the left of partition is designated as 'A' and to the right as 'B'. (a) $A(n) = 1$ (b) $A(n) = 0$ (c) $B(n) = 1$ (d) $B(n) = 0$

FM Algorithm TWPP

Algorithm FM_TWPP
Begin
Step 1. Compute gains of all cells.
Step 2. i =1. Select 'base cell' and call it c_i;
  If no base cell **Then Exit**;
  A base cell is one which
    (i) has maximum gain;
    (ii) satisfies balance criterion;
     If tie **Then** use Size criterion or Internal connections;
Step 3. Lock cell c_i;
  Update gains of cells of those affected critical nets;
Step 4. **If** free cells $\neq \phi$ **Then** $i = i + 1$; select next base cell;
  If $c_i \neq \phi$ then **Goto** Step 3;
Step 5. Select best sequence of moves $c_1, c_2, \cdots, c_k$
  $(1 \leq k \leq i)$ such that G=$\sum_{j=1}^{k} g_j$ is max;
  If tie **then** choose subset that achieves a superior balance;
  If $G \leq 0$ **Then Exit**;
Step 6. Make all i moves permanent;
  Free all cells; **Goto** Step 1
End.

FM Algorithm TWPP - contd

Step 1.

- The first step consists of computing the gains of all $free$ cells.
- Cells are considered to be free if they are not locked either initially by the user, or after they have been moved during this pass.
- Similar to the Kernighan-Lin algorithm, the effect of the movement of a cell on the cutset is quantified with a gain function.
- Let $F(i)$ and $T(i)$ be the From_block and To_block of cell i.
- The gain $g(i)$ resulting from the movement of cell i from block $F(i)$ to block $T(i)$ is:

$$g(i) = FS(i) - TE(i)$$

- where $FS(i)$ = the number of nets connected to cell i and not connected to any other cell in the From_Block $F(i)$ of cell i.
- and $TE(i)$ = the number of nets that are connected to cell i and not crossing the cut.

Example

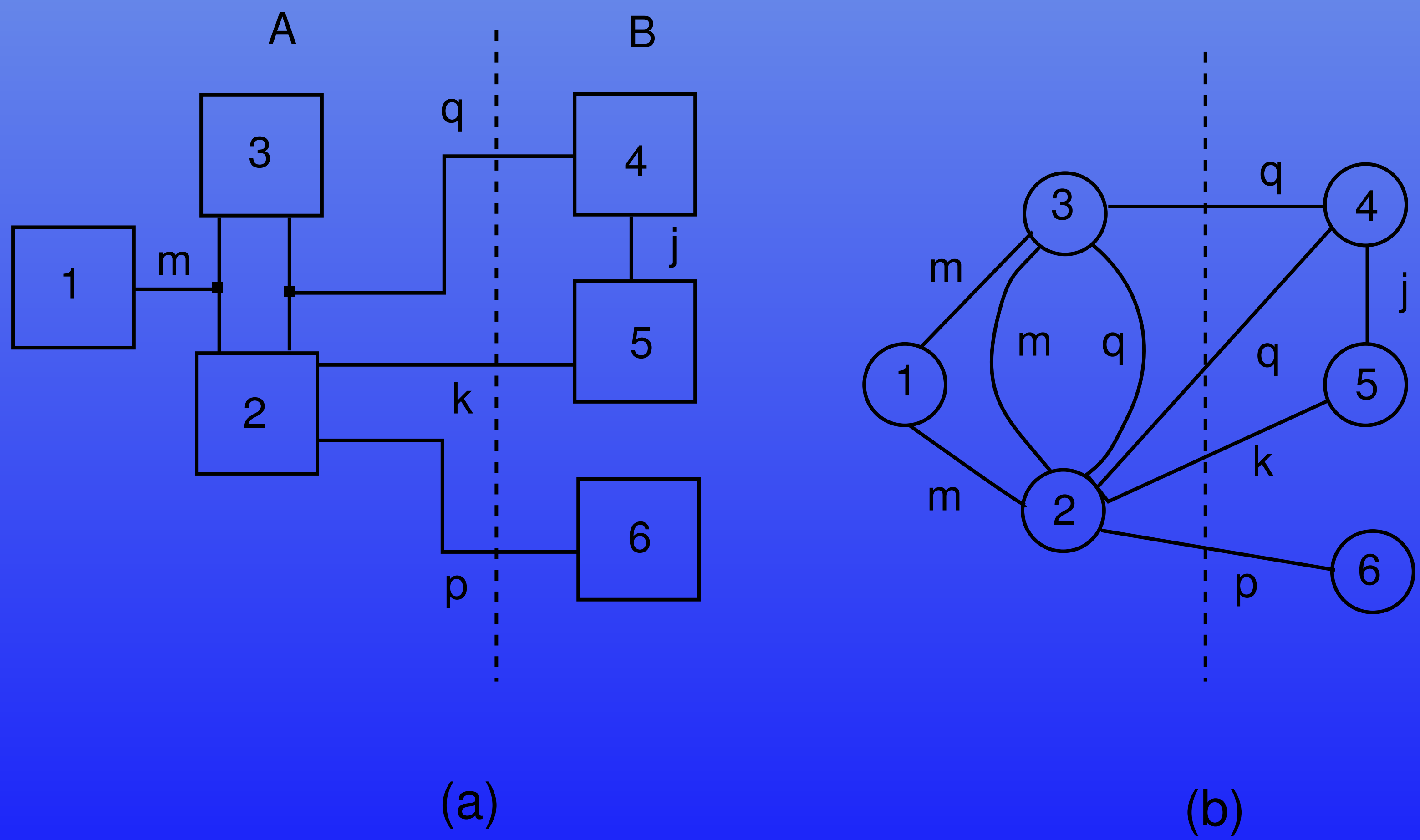


Figure 9: Illustration of (a) Cut of nets. (b) Cut of edges.

Example - contd

Gains of cells

$Cell\ i$	F	T	$FS(i)$	$TE(i)$	$g(i)$
1	A	B	0	1	-1
2	A	B	2	1	+1
3	A	B	0	1	-1
4	B	A	1	1	0
5	B	A	1	1	0
6	B	A	1	0	+1

Example - contd

- Consider cell 2, its From_Block is A and its To_Block is B.
- Nets k, m, p, and q are connected to cell 2 of block A, of these only two nets k and p are **not** connected to any other cell in block A.
- Therefore, by definition, $FS(2)$=2. And $TE(2)$=1 since the only net connected and not crossing the cut is net m.
- Hence $g(2)$=2-1=1. Which means that the number of nets cut will be reduced by 1 (from 3 to 2) if cell 2 were to be moved from A to B.

Example - contd

- Consider cell 4. In Block B, cell 4 has only one net (net j) which is connected to it and also not crossing the cut, therefore TE(4)=1. FS(4)=1 and $g(4)$=1-1=0, that is, no gain.
- Finally consider cell 5. Two nets j and k are connected to cell 5 in block B, but one of them, that is, net k is crossing the cut, while net j is not. Therefore, TE(5) is also 1. (see table of previous slide)
- The above observation can be translated into an efficient procedure to compute the gains of all free cells.

Example - contd

Algorithm Compute_cell_gains.
Begin
  For each free cell 'i' **Do**
    $g(i) \leftarrow 0$;
    $F \leftarrow From\_block$ of cell i;
    $T \leftarrow To\_block$ of cell i;
    For each net 'n' on cell 'i' **Do**
      If $F(n) = 1$ **Then** $g(i) \leftarrow g(i) + 1$;
      (*Cell i is the only cell in the From_Block
       connected to net n.*)
      If $T(n) = 0$ **Then** $g(i) \leftarrow g(i) - 1$
      (* All of the cells connected to net n are
       in the From_Block. *)
    EndFor
  EndFor
End.

Example - contd

- We apply the previous procedure compute the gains of all the free cells of the circuit.
- We first compute the values of $A(n)$ and $B(n)$ (where $A(n)$ and $B(n)$ are the numbers of cells of net n that are in block A and block B respectively). For the given circuit we have,

$$A(j) = 0, A(m) = 3, A(q) = 2, A(k) = 1, A(p) = 1,$$
$$B(j) = 2, B(m) = 0, B(q) = 1, B(k) = 1, B(p) = 1.$$

- For cells in block A we have, the From_block A ($F = A$) and To_block is B ($T = B$). For this configuration we get,

$$F(j) = 0,\ F(m) = 3,\ F(q) = 2,\ F(k) = 1,\ F(p) = 1,$$
$$T(j) = 2,\ T(m) = 0,\ T(q) = 1,\ T(k) = 1,\ T(p) = 1.$$

$F(i)$ is the number of cells of net i in From_block.

Example - contd

- Since only critical nets affect the gains, we are interested only in those values which have,
  - for cells of block A, $A(n) = 1$ and $B(n) = 0$, and
  - for cells of block B, $B(n) = 1$ and $A(n) = 0$.
- Therefore, values of interest for Block A are $F(k) = 1$, $F(p) = 1$, and $T(m) = 0$.
- Now, the application of "Compute_cell_gains" would produce the following:
  - $i = 1$; $F = A$; $T = B$; net on cell 1 is m. Values of interest are $T(m) = 0$;therefore, $g(1) = 0 - 1 = -1$.
  - $i = 2$; $F = A$; $T = B$; nets on cell 2 are m, q, k, and p. Values of interest are $F(k) = 1$; $F(p) = 1$; and $T(m) = 0$; therefore, $g(2) = 2 - 1 = 1$.
  - $i = 3$; $F = A$; $T = B$; nets on cell 3 are m and q, but only $T(m) = 0$; therefore, $g(3) = 0 - 1 = -1$.

Example - contd

Step 2. Selection of ‘base cell

- Having computed the gains of each cell, we now choose the ‘base cell’.
- The base cell is one that has a maximum gain and does not violate the balance criterion.
- If no base cell is found then the procedure stops.

Algorithm Select_cell;
Begin
For each cell with maximum gain
    If moving will create imbalance
        Then discard it
    EndIf
EndFor;
    If neither block has a qualifying cell
    Then Exit
End.

Example - contd

Step 2. Selection of 'base cell' (Cont.):

- When the balance criterion is satisfied then the cell with maximum gain is selected as the base cell.
- In some cases, the gain of the cell is non-positive. However, we still move the cell with the expectation that the move will allow the algorithm to "escape out of a local minimum".
- To avoid migration of all cells to one block, during each move, the balance criterion is maintained.
- The notion of a tolerance factor is used in order to speed up convergence from an unbalanced situation to a balanced one.

Example - contd

- The balance criterion is therefore relaxed to the inequality below:

  $$r \times S(V) - k \times s_{max} \leq S(A) \leq r \times S(V) + k \times s_{max}$$

  where k is an increasing function of the number of free cells.
- Initially k is large and is slowly decreased with each pass until it reduces to unity.
- If more than one cell of maximum gain exists, and all such cells satisfy the balance criterion, then ties may be broken depending on the size, internal connectivity, or any other criterion.

Example - contd

Step 3. Lock cell and update gains:

- After each move the selected cell is locked in its new block for the remainder of the pass.
- Then the gains of cells of affected critical net are updated using the following procedure.

Example - contd

Algorithm Update_Gains;
Begin
(* move base cell and update neighbors' gains *)
$F \leftarrow$ the From_block of base cell; $T \leftarrow$ the To_block of base cell;
Lock the base cell and complement its blocks;
For each net n on base cell **Do**(* check critical nets before the move *)
    If $T(n) = 0$ **Then** increment gains of free cells on net n
    Else If $T(n) = 1$ **Then** decrement gain of the only T cell on net n, if it is free
    EndIf;
    (* update $F(n)$ & $T(n)$ to reflect the move *)
    $F(n) \leftarrow F(n) - 1$; $T(n) \leftarrow T(n) + 1$;
    (* check for critical nets after the move *)
    If $F(n) = 0$ **Then** decrement gains of free cells on net n
    Else If $F(n) = 1$ **Then** increment the gain of the only
        F cell on net n, if it is free
    EndIf
EndFor
End.

Example - contd

Step 4. Select next base cell:

- In this step, if more free cells exist then we search for the next base cell. If found then we go back to Step 3, lock the cell, and repeat the update. If no free cells are found then we move on to Step 5.

Step 5. Select best sequence of moves:

- After all the cells have been considered for movement, as in the case of Kernighan-Lin, the best partition encountered during the pass is taken as the output of the pass. The number of cells to move is given by the value of k which yields maximum positive gain G_k, where $G_k = \sum_{i=1}^{k} g_i$.

Example - contd

Step 6. Make moves permanent:

- Only the cells given by the best sequence, that is, $c_1, c_2, \cdots, c_k$ are permanently moved to their complementary blocks. Then all cells are freed and the procedure is repeated from the beginning.

Another Example

- We would like to apply the remaining steps of the Fiduccia-Mattheyses heuristic to the circuit of previous example to complete one pass.
- Assume that the desired balance factor be 0.4 and the sizes of cells are as follows:
  $s(c_1)$=3, $s(c_2)$=2, $s(c_3)$=4, $s(c_4)$=1, $s(c_5)$=3, and $s(c_6)$=5.
  Solution:
- We have already found that cell c_2 is the candidate with maximum gain.
- c_2 also satisfies the balance criterion.
- Now, for each net n on cell c_2 we find its distribution $F(n)$ and $T(n)$ (that is, the number of cells on net n in the From_block and in the To_block respectively before the move).

Example - contd

Similarly we find $F'(n)$ and $T'(n)$, the number of cells after the move.

	Before Move		After Move	
Net	F	T	F'	T'
k	1	1	0	2
m	3	0	2	1
q	2	1	1	2
p	1	1	0	2

Notice that the change in net distribution to reflect the move is a decrease in $F(n)$ and an increase in $T(n)$.

Example - contd

We now apply the procedure of Step 3 to update the gains of cells and determine the new gains.

1. For each net n on the base cell we check for the critical nets before the move.
2. If $T(n)$ is zero then the gains of all free cells on the net n are incremented.
3. If $T(n)$ is one then the gains of the only T cell on net n is decremented (if the cell is free).

In our case, the selected base cell c_2 is connected to nets k, m, p, and q, and all of them are critical, with $T(m) = 0$, and $T(k) = T(q) = T(p) = 1$.

Example - contd

- Therefore, the gains of the free cells connected to net m (c_1 and c_3) are incremented, while the gains of the free T_cells connected to nets k, p and q (c_5, c_6, and c_4) are decremented.
- These values are tabulated in the first four columns (Gain due to $T(n)$) of the table below.

Gain	due to $T(n)$				due to $F(n)$				Gains	
$Cells$	k	m	q	p	k	m	q	p	Old	New
c_1		1							-1	0
c_3		1					1		-1	1
c_4			-1						0	-1
c_5	-1				-1				0	-2
c_6				-1				-1	1	-1

Example - contd

- We continue with the procedure "*Update_Gains*" and check for the critical nets after the move.
- If $F(n)$ is zero then the gains of all free cells on net n are decremented and if $F(n)$ is one then the gain of the only F cell on net n is incremented, if it is free.
- Since we are looking for the net distribution after the move, we look at the values of F'.
- Here we have $F'(k) = F'(p) = 0$ and $F'(q) = 1$.
- The contribution to gain due to cell 5 on net k and cell 6 on net p is -1, and since cell 3 is the only F cell (cell on From_block), the gain due to it is +1.
- These values are tabulated in the four columns "Gain due to $F(n)$)" of previous table.

Example - contd

- From previous table, the updated gains are obtained.
- The second candidate with maximum gain (say g_2) is cell c_3. This cell also satisfies the balance criterion and therefore is selected and locked.
- We continue the above procedure of selecting the base cell (Step 2) for different values of i.
- Initially A_0={1,2,3}, B_0={4,5,6}. The results are summarized below.

$i = 1$: The cell with maximum gain is c_2. $|A| = 7$. This move satisfies the balance criterion. Maximum gain $g_1 = 1$. Lock cell $\{c_2\}$. A_1={1,3}, B_1={2,4,5,6}.

$i = 2$: Cell with maximum gain is c_3. $|A| = 3$. The move satisfies the balance criterion. Maximum gain $g_2 = 1$. Locked cells are $\{c_2, c_3\}$. A_2={1}, B_2={2,3,4,5,6}.

Example - contd

$i = 3:$ Cell with maximum gain (+1) is c_1. If c_1 is moved then A ={}, B ={1,2,3,4,5,6}. $|A| = 0$. This *does not* satisfy the balance criterion. Cell with next maximum gain is c_6. $|A| = 8$. This cell satisfies the balance criterion. Maximum gain $g_3 = -1$. Locked cells are $\{c_2, c_3, c_6\}$. A_3={1,6}, B_3={2,3,4,5}.

$i = 4:$ Cell with maximum gain is c_1. $|A| = 5$. This satisfies the balance criterion. Maximum gain $g_4 = 1$. Locked cells are $\{c_1, c_2, c_3, c_6\}$. A_4={6}, B_4={1,2,3,4,5}.

$i = 5:$ Cell with maximum gain is c_5. $|A| = 8$. This satisfies the balance criterion. Maximum gain $g_5 = -2$. Locked cells are $\{c_1, c_2, c_3, c_5, c_6\}$. A_5={5,6}, B_5={1,2,3,4}.

$i = 6:$ Cell with maximum gain is c_4. $|A| = 9$. This satisfies the balance criterion. Maximum gain $g_6 = 0$. All cells are locked. A_6={4,5,6}, B_6={1,2,3}.

Example - contd

- Observe that when $i = 3$, cell c_1 is the cell with maximum gain, but since it violates the balance criterion, it is discarded and the next cell (c_6) is selected. When $i = 4$ cell c_1 again is the cell with maximum gain, but this time, since the balance criterion is satisfied, it is selected for movement.
- We now look for k that will maximize G =$\sum_{j=1}^{k} g_j$; $1 \leq k \leq i$. We have a tie with two candidates for k, $k = 2$ and $k = 4$, giving a gain of +2. Since the value of $k = 4$ results in a better balance between partitions, we choose k=4. Therefore we move across partitions the first four cells selected, which are cells c_2, c_3, c_6, and c_1. The final partition is $A =${6}, and $B =${1,2,3,4,5}. The cost of nets cut is reduced from 3 to 1.

Simulated Annealing

- First application of simulated annealing to placement reported by Jepsen and Gelatt.
- It is an *adaptive* heuristic and belongs to the class of *non-deterministic* algorithms. This heuristic was first introduced by Kirkpatrick, Gelatt, and Vecchi in 1983.
- The simulated annealing heuristic derives inspiration from the process of carefully cooling molten metals in order to obtain a good crystal structure.
- In SA, first an initial solution is selected; then a controlled walk through the search space is performed until no sizeable improvement can be made or we run out of time.
- Simulated annealing has *hill-climbing* capability.

Simulated annealing- contd

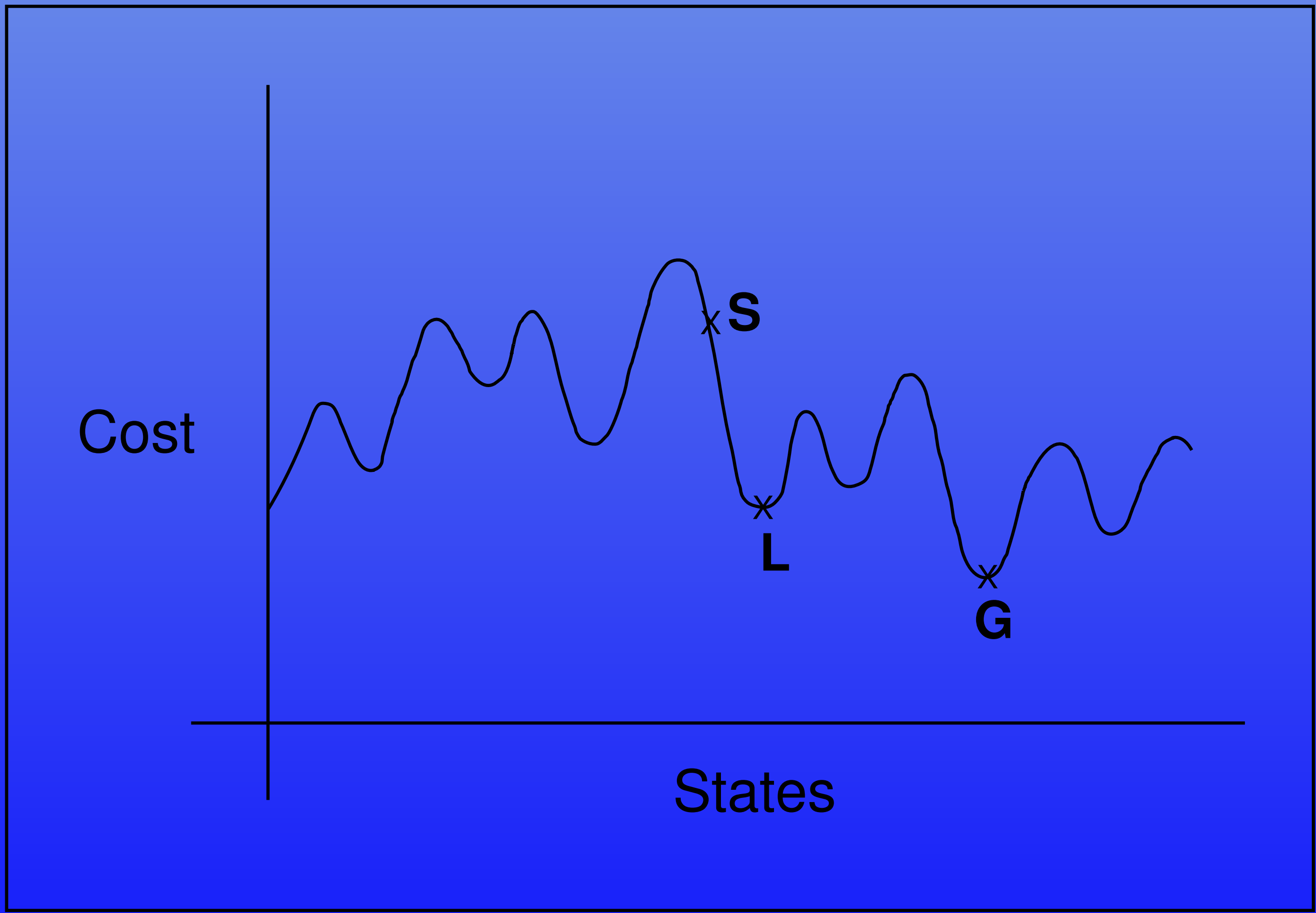


Figure 10: Local vs. Global Optima

Simulated annealing- contd

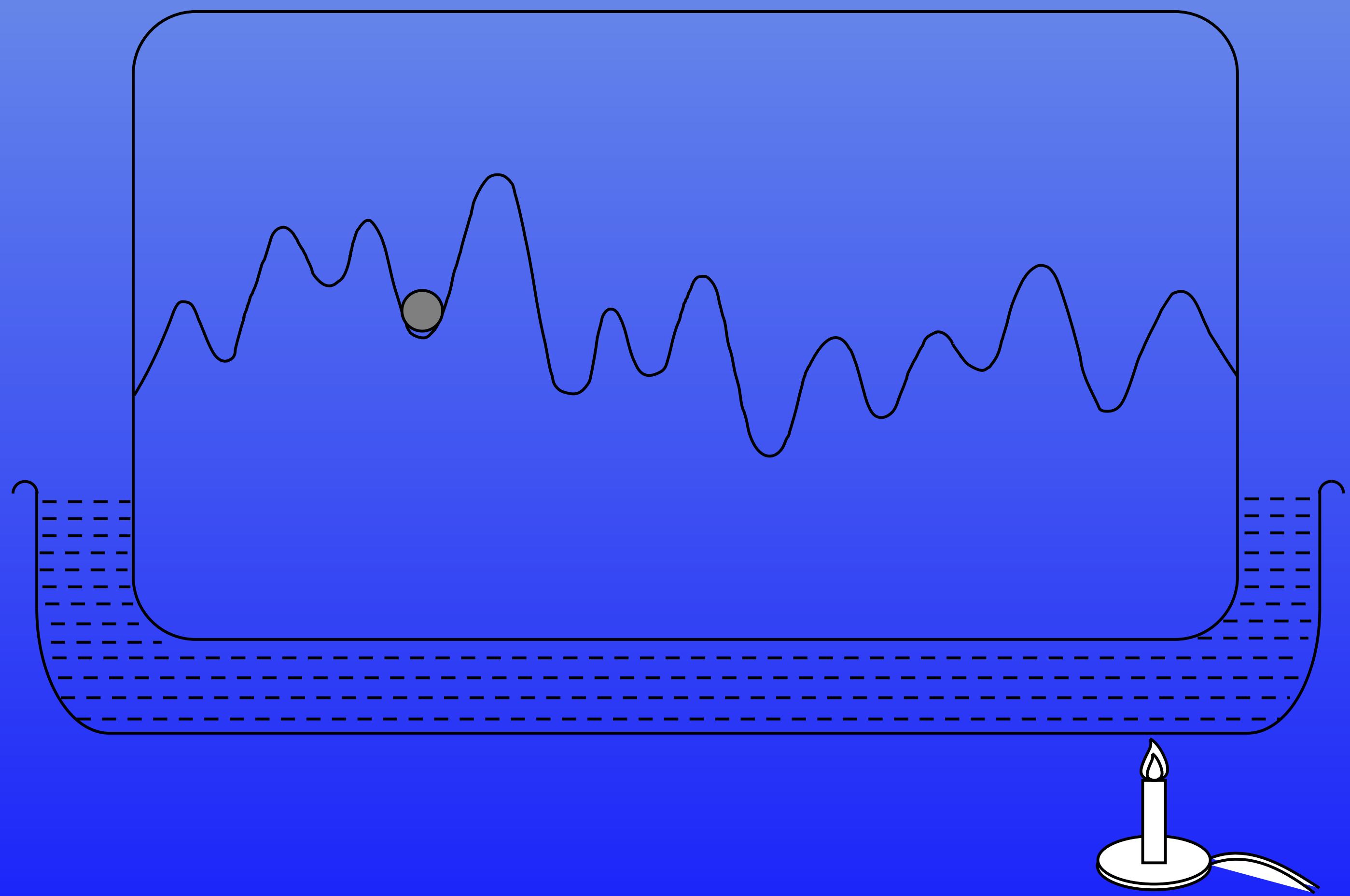


Figure 11: Design space analogous to a hilly terrain

Simulated annealing- Algorithm

Algorithm SA($S_0, T_0, \alpha, \beta, M, Maxtime$);
(*S_0 : the initial solution *)
(*T_0 : the initial temperature *)
(*α : the cooling rate *)
(*β : a constant *)
(*$Maxtime$: max allowed time for annealing *)
(*M : time until the next parameter update *)
begin

$T = T_0$;
$S = S_0$;
$Time = 0$;
repeat
Call $Metropolis(S, T, M)$;
$Time = Time + M$;
$T = \alpha \times T$;
$M = \beta \times M$
until ($Time \geq MaxTime$);
Output Best solution found

End. (**of SA* *)

Simulated annealing- Algorithm

Algorithm Metropolis(S, T, M);
begin
    repeat
        $NewS$=$neighbor(S)$;
        Δh=$(Cost(NewS) - Cost(S))$;
        if $((\Delta h< 0)$ **or** $(random < e^{-\Delta h/T}))$
            then $S = NewS$;
            {accept the solution}
        $M = M - 1$
    until $(M = 0)$
End. (**of Metropolis**).

Simulated annealing- contd

- The core of the algorithm is the *Metropolis* procedure, which simulates the annealing process at a given temperature T.
- The procedure is named after a scientist who devised a similar scheme to simulate a collection of atoms in equilibrium at a given temperature.
- Besides the temperature parameter, *Metropolis* receives as input the current solution S which it improves through local search. *Metropolis* must also be provided with the value M, which is the amount of time for which annealing must be applied at temperature T.
- The procedure *Simulated_annealing* simply invokes *Metropolis* at various (decreasing) temperatures.

Simulated annealing- contd

- Temperature is initialized to a value T_0 at the beginning of the procedure, and is slowly reduced in a geometric progression; the parameter α is used to achieve this cooling. The amount of time spent in annealing at a temperature is gradually *increased* as temperature is lowered. This is done using the parameter $\beta > 1$.
- The variable *Time* keeps track of the time already expended by the heuristic. The annealing procedure halts when *Time* exceeds the allowed time.

Simulated annealing- contd

- To apply the simulated annealing technique we need to be able to:

  (1) generate an initial solution,

  (2) disturb a feasible solution to create another feasible solution,

  (3) evaluate the objective function for these solutions.