Embedded Systems 21

The Partitioning Problem

REVIEW

Definition: The **partitioning problem** is to assign
n **objects** $O=\{o_1, \ldots, o_n\}$ to
m **blocks** (also called **partitions**) $P=\{p_1, \ldots, p_m\}$
such that

- $p_1 \cup p_2 \ldots \cup p_m = O$
- $p_i \cap p_j = \varnothing$ for all $i \neq j$, and
- cost c(*P*) is minimized.

Cost function (Estimated) quality of design, may include

- System price
- Latency
- Power consumption, …

Exact methods: Linear Programming REVIEW

- Binary variables $x_{i,k}$
  - $x_{i,k}$=1: object o_i in block p_k
  - $x_{i,k}$=0: object o_i not in block p_k
- Cost $c_{i,k}$ if object o_i in block p_k
- Integer linear program:

$$x_{i,k} \in \{0,1\} \qquad 1 \le i \le n, 1 \le k \le m$$

$$\sum_{k=1}^{m} x_{i,k} = 1 \qquad 1 \le i \le n$$

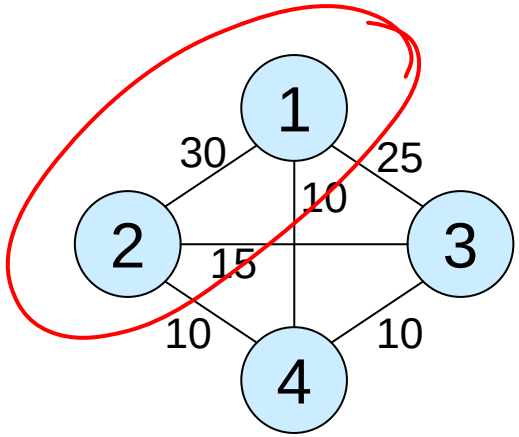
minimize $$\sum_{k=1}^{m} \sum_{i=1}^{n} x_{i,k} \cdot c_{i,k}$$

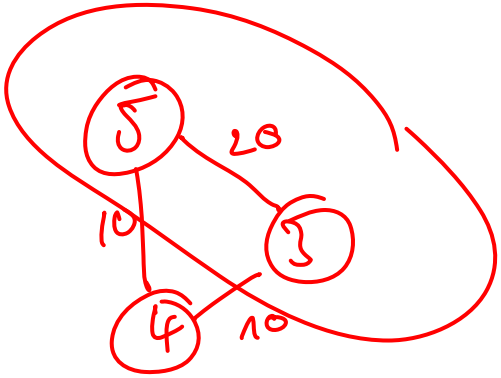
Constructive methods: Hierarchical Clustering

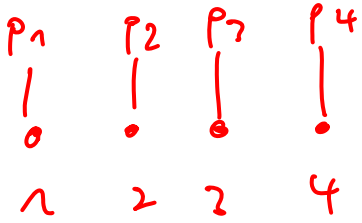
REVIEW

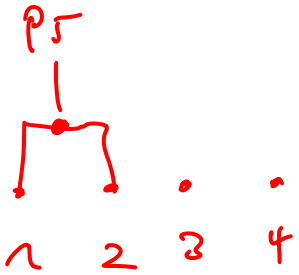
Average closeness;

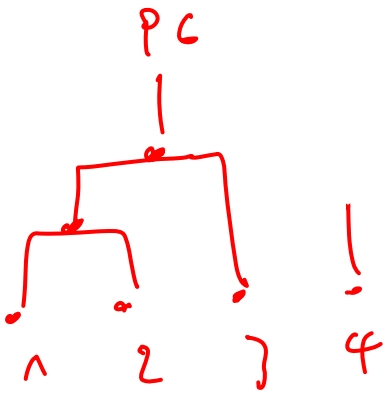
Termination: 2 blocks











Iterative Methods: Kernighan-Lin (K-L)

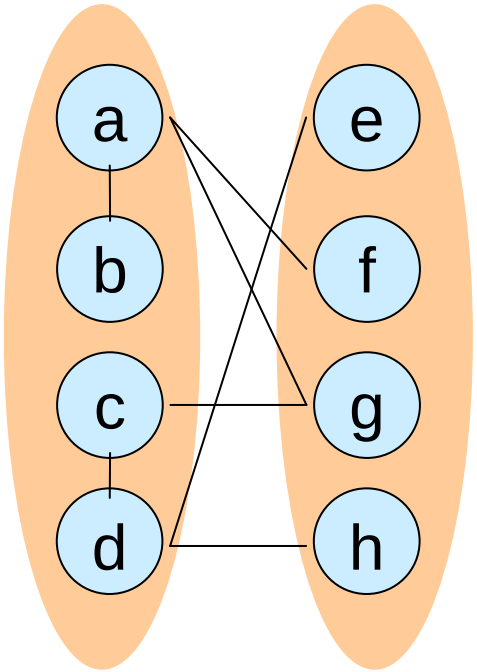
An iterative balanced partitioning (bi-sectioning) heuristic

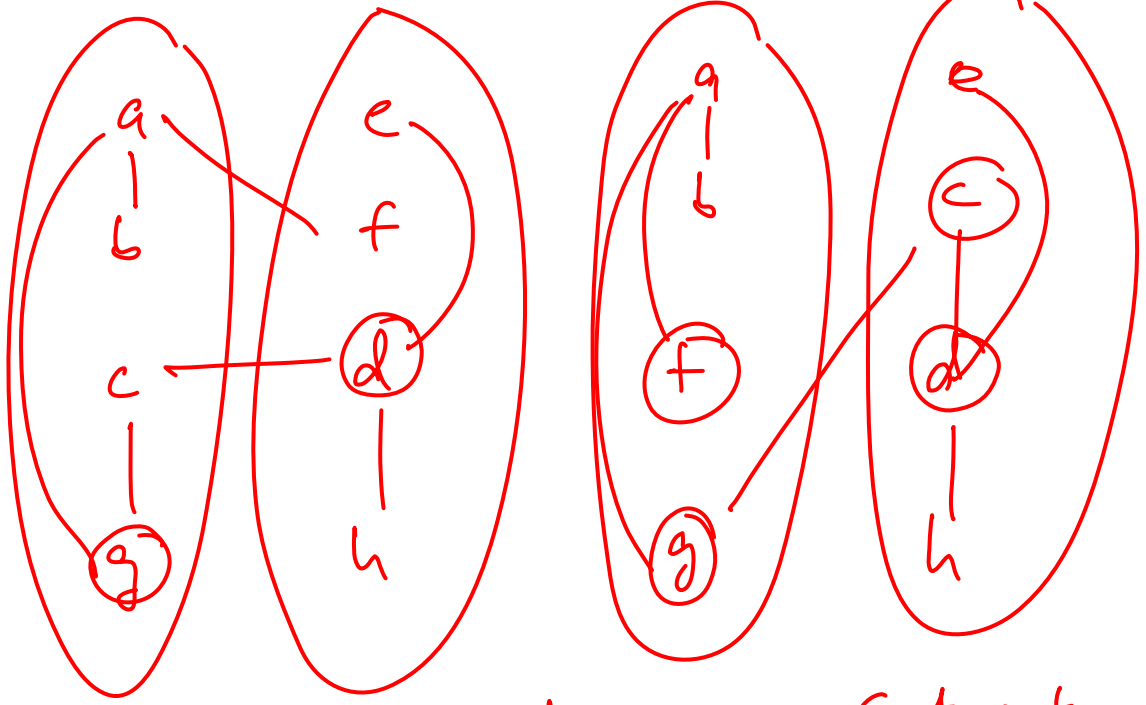
Given: Two sets A and B, such that $|A|=|B|=n$ and $A \cap B = \emptyset$
cost of edge (a,b) in cut: c_{ab}

While the cost keeps decreasing

- Mark all objects as „unlocked“
- While there are unlocked pairs left
  - Select pair of unlocked objects (a,b) which give the largest decrease or the smallest increase in cut size
  - Mark a and b as „locked“
  - Exchange a and b
  - Record resulting partition and cost
- Continue with the partition with least cost

Example

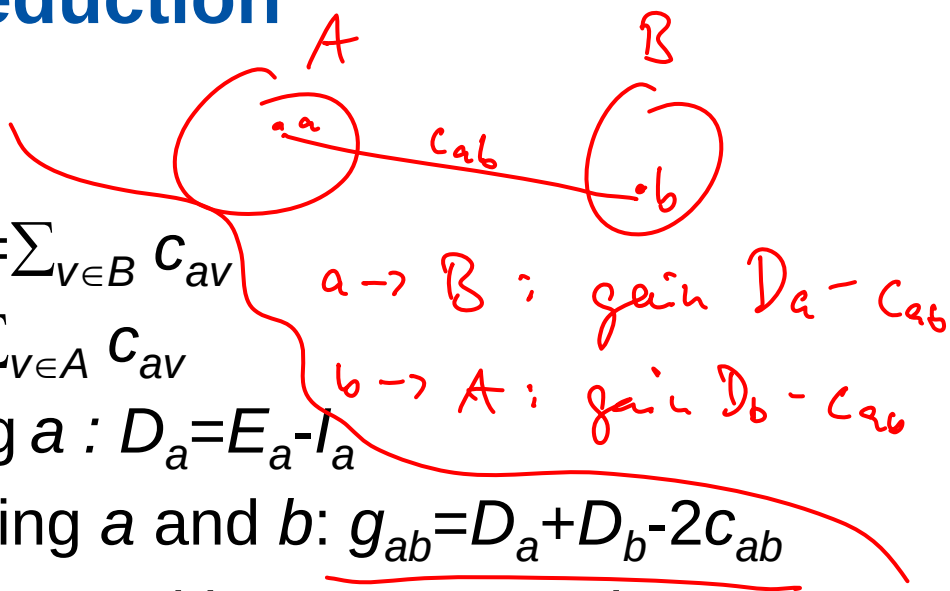




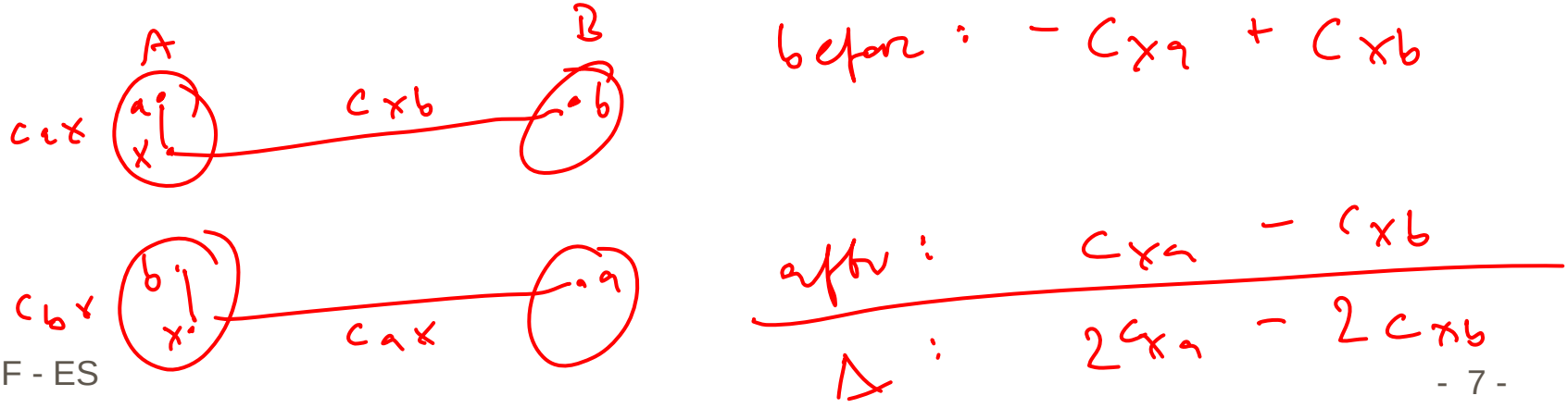
Step # | object pair | cost reduction | Cut cost

			5
0			2
1	{d, g}	3	1
2	{c, f}	1	3
3	{b, h}	-2	5
4	{a, e}	-2	

Computing the cost reduction



- External cost of a$\in A$: $E_a=\sum_{v\in B} c_{av}$
- Internal cost of $a\in A$: $I_a=\sum_{v\in A} c_{av}$
- Cost reduction for moving a : $D_a=E_a-I_a$
- Cost reduction for swapping a and b: $g_{ab}=D_a+D_b-2c_{ab}$
- Update to D-values when a and b are swapped:
  $D'_x = D_x + 2c_{xa} - 2c_{xb}$ for all $x\in A-\{a\}$
  $D'_y = D_y + 2c_{yb} - 2c_{ya}$ for all $y\in B-\{b\}$



Weighted Example

	a	b	c	d	e	f
a	0	1	2	3	2	4
b	1	0	1	4	2	1
c	2	1	0	3	2	1
d	3	4	3	0	4	3
e	2	2	2	4	0	2
f	4	1	1	3	2	0

A={a,b,c}

B={d,e,f}

$I_a = 1+2=3$ $E_a = 3+2+4 = 9$ $D_a = 9-3 = 6$

$I_b = 1+1=2$ $E_b = 4+2+k\ 7$ $D_b = 7-2 = 5$

$g_{ad} = D_a + D_d - 2c_{ad} = 6+3-2\times 3 = 3$

$g_{bf} = 4$ ← MAX

Swap b and f!

Weighted Example

A={a,b,c}

B={d,e,f}

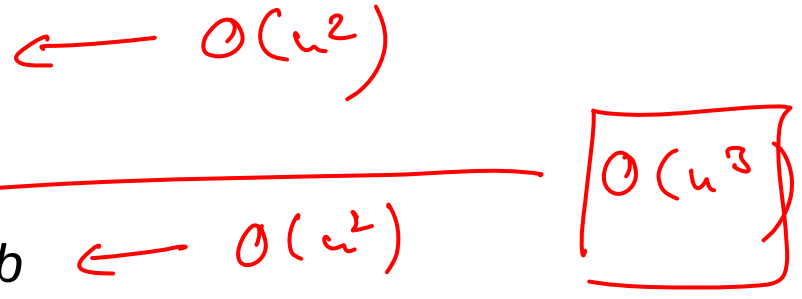
	a	b	c	d	e	f
a	0	1	2	3	2	4
b	1	0	1	4	2	1
c	2	1	0	3	2	1
d	3	4	3	0	4	3
e	2	2	2	4	0	2
f	4	1	1	3	2	0

A
$$\begin{cases} D_a' = D_a + 2c_{ab} - 2c_{af} = 0 \\ D_c' = D_c + 2c_{cb} - 2c_{cf} = 3 \end{cases}$$

B
$$\begin{cases} D_d' = D_d + 2c_{df} - 2c_{ab} = 1 \\ D_e' = D_e + 2c_{ef} - 2c_{eb} = 0 \end{cases}$$

Kernighan-Lin

- **Repeat**
  - Compute D_v für all objects
  - Mark all vertices as unlocked
  - **For** i=1 to $n/2$ **do**
    - Compute g_{ab} for all pairs a,b
    - Pick unlocked a_i,b_i with largest $g_{ab,i}$
    - Mark a_i,b_i as locked
    - Store gain
    - Update D_v für all objects
  - Find k such that $G_k=\sum_{i=1}^{k} g_{ab,i}$ is maximal
  - **If** $G_k>0$, **then** move $a_1,\ldots,a_k$ from A to B and $b_1,\ldots,b_k$ from B to A.
- **Until** $G_k \leq 0$



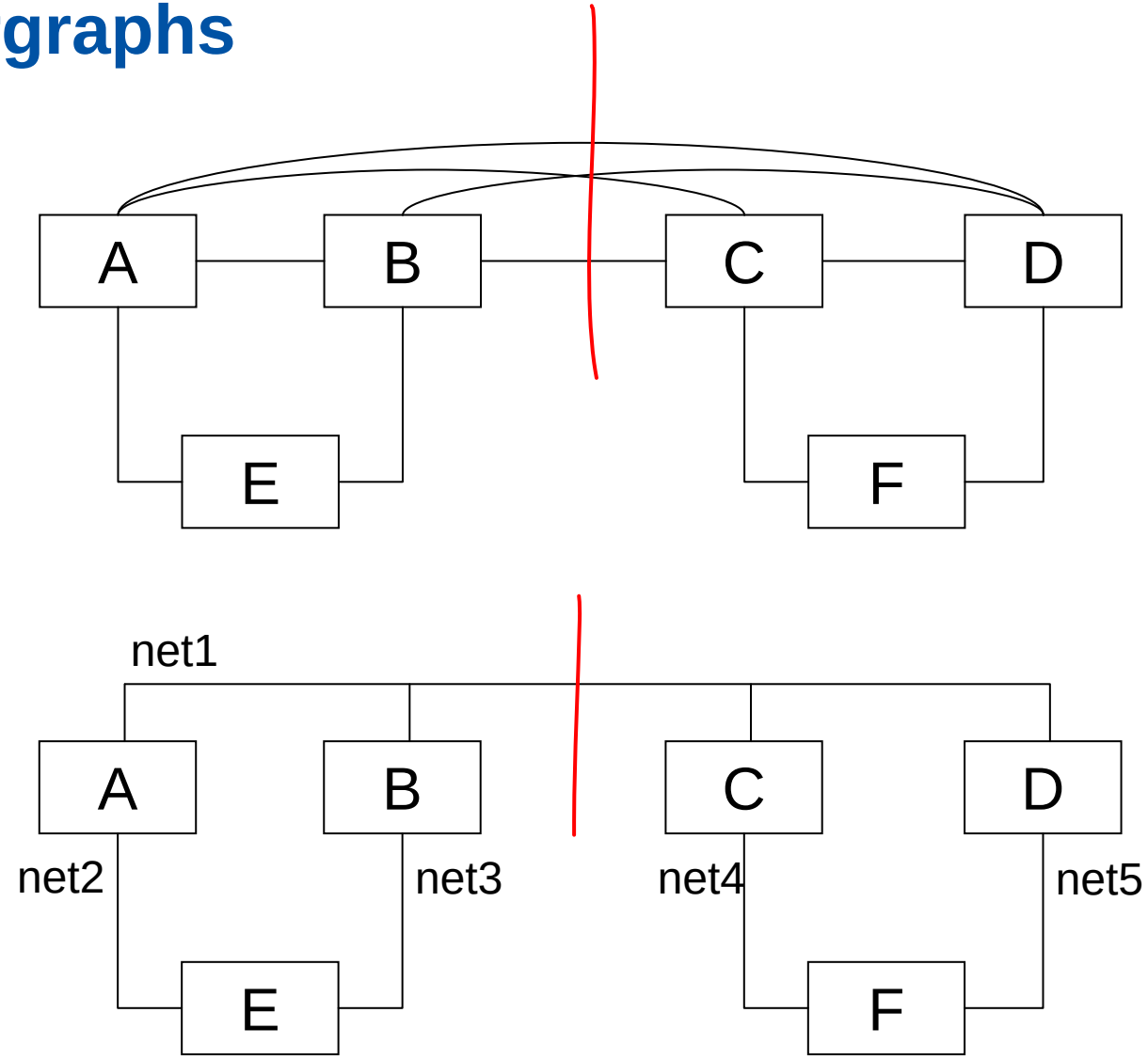
If repeat loop terminates after r iterations
$\Rightarrow O(r n^3)$

"in practice r is small".

Extensions to K-L

- Different block sizes
  - If |A|<|B|, add |B|-|A| dummy objects to A.
    Dummy objects are not connected
  - Apply K-L
  - Remove dummies
- Objects with size > 1
  - Replace each object of size *s* with *s* objects of size 1
    new objects are fully connected with edges of infinite weight
  - Apply K-L
- More than 2 blocks
  - Apply K-L to each pair of blocks

Hypergraphs

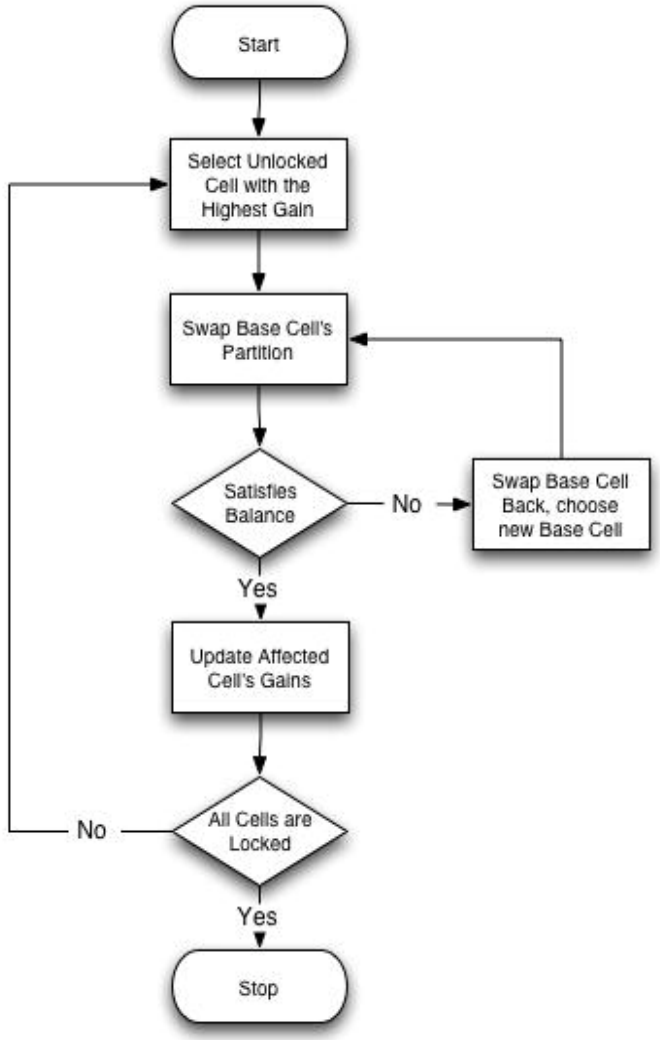


Fiduccia-Mattheyses Heuristic (F-M)

- Objects have size s(o)
- Size of block: sum of size of objects
- **Balanced two-way partition**:
  Given a fraction r, $0 < r < 1$,
  partition a graph into two blocks A and B such that
  $|A| / (|A|+|B|) \approx r$
  and cutset is minimized
- **Linear complexity**

Terminology: object=„cell“, hyperedges=„net“

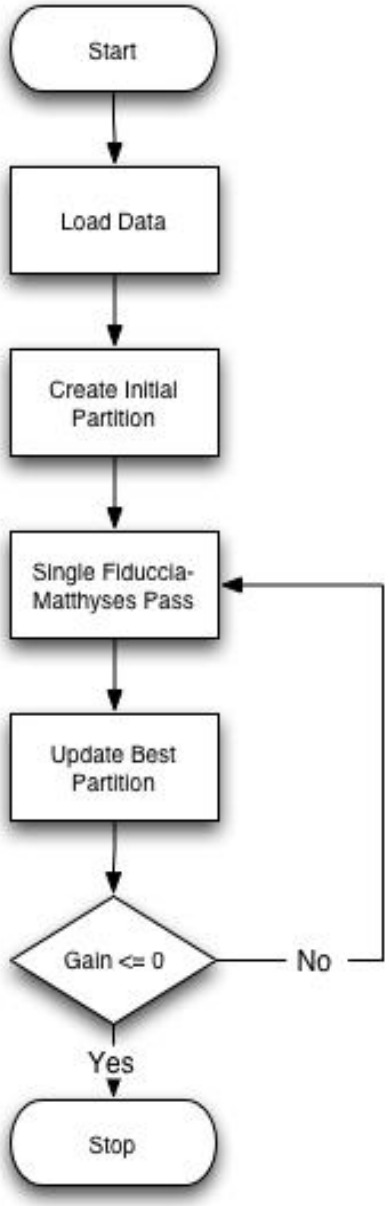
Single pass of the F-M heuristic



- Select the cell with the greatest gains that satisfies balance conditions
- Move the cell and lock it
- Update gains
- Repeat until all cells are locked or will dissatisfy balance conditions

Overall F-M heuristic

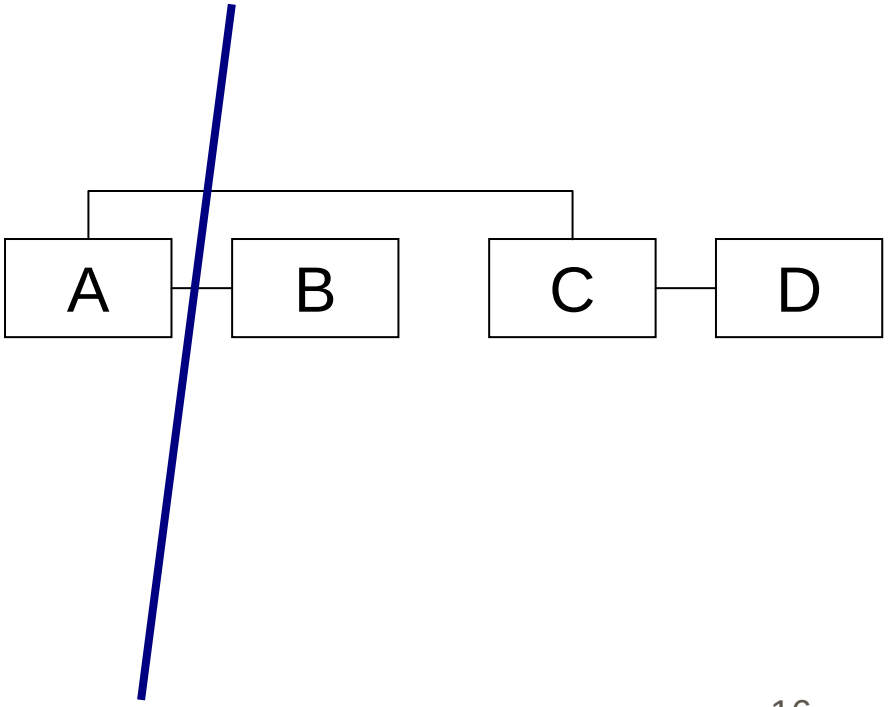
- Create an initial partition
- Execute a pass of the FM heuristic
- Start again using the result partition as the initial partition
- Continue until the resulting gain is no longer greater than zero



Calculating Gain

- g(i) = FS(i) - TE(i)
- FS(i): The number of nets which contain cell i but no other object in the same partition as i
- TE(i): The number of nets that consist only of i and other cells currently in the same partition as i

object	FS	TE	gain
A	2	0	2
B	1	0	1
C	1	1	0
D	0	1	-1



Calculate Gain

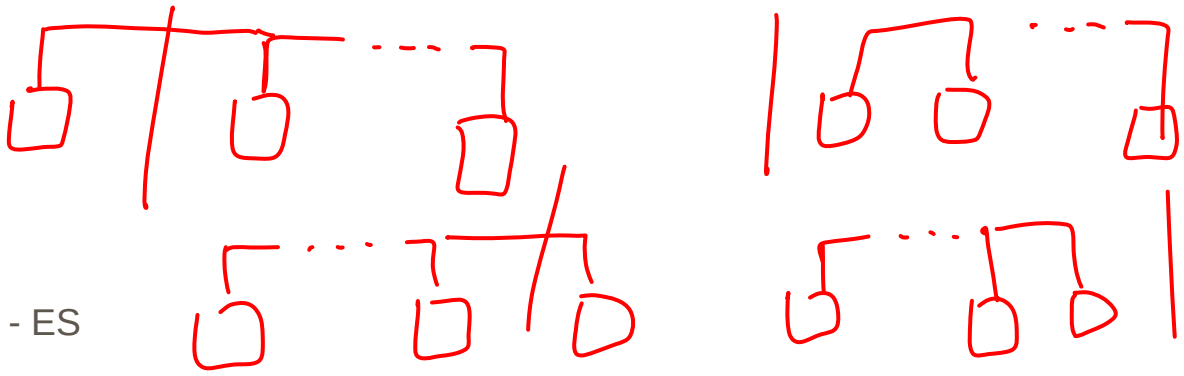
- **For** each unlocked cell i **do**
  - g(i) = 0
  - F = the “from block” of object i
  - T = the “to block” of object i
  - **For** each net n that contains i **do**
    - If F(n) = 1 increment g(i)
    - If T(n) = 0 decrement g(i)

→ linear cost.

Net distribution and critical nets

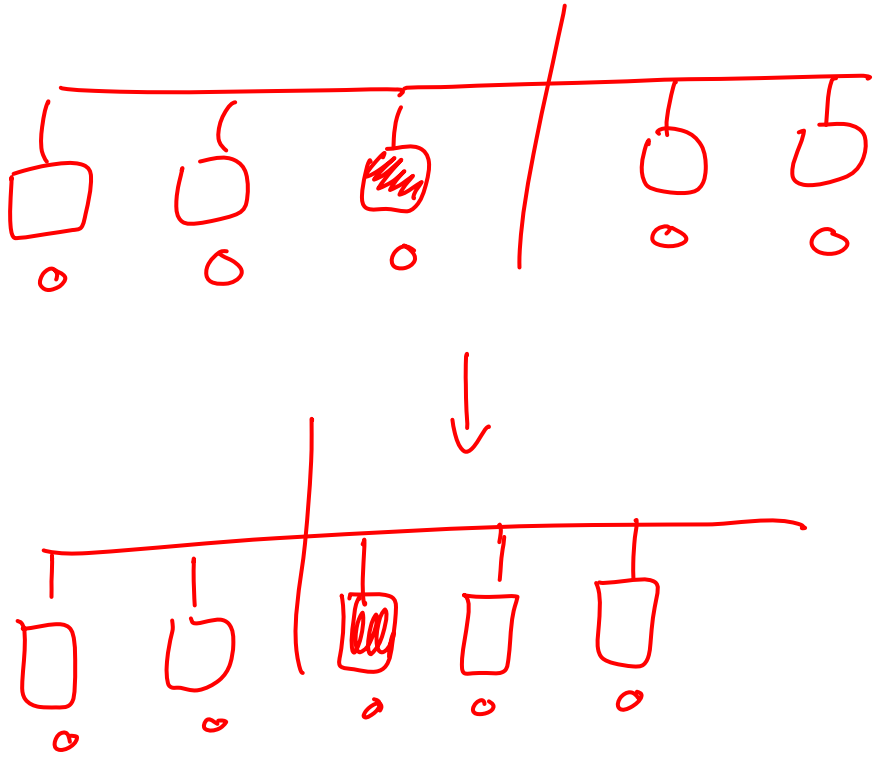
- Distribution of net i:
  (A(i),B(i))=(# of cells in A, # of cells in B)

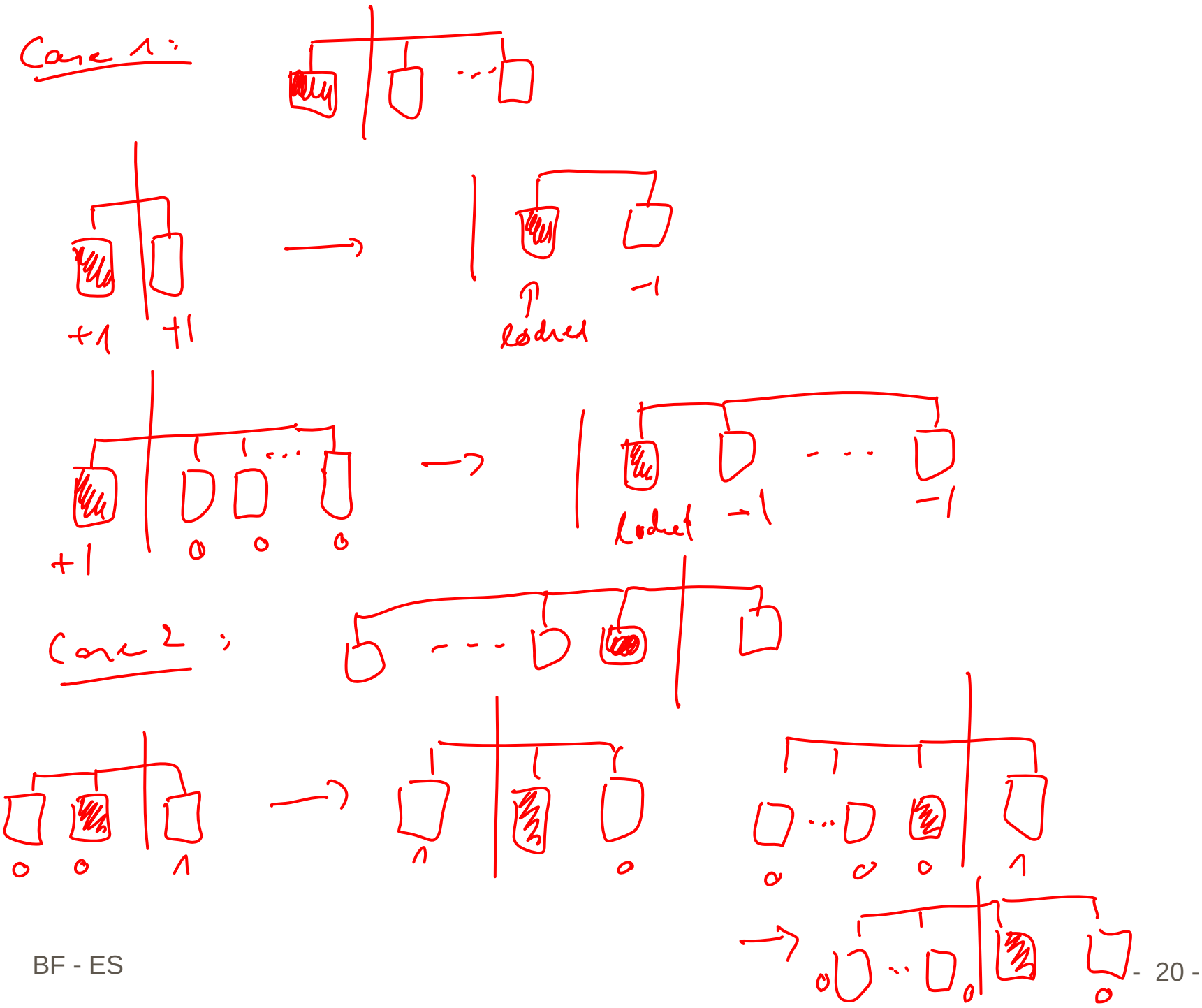
- A net is **critical** if it has an cell that if moved will add or remove the net from the cutset
- Gain of a cell depends only on its critical nets.
- 4 cases: A(i)=0 or 1, B(i)=0 or 1

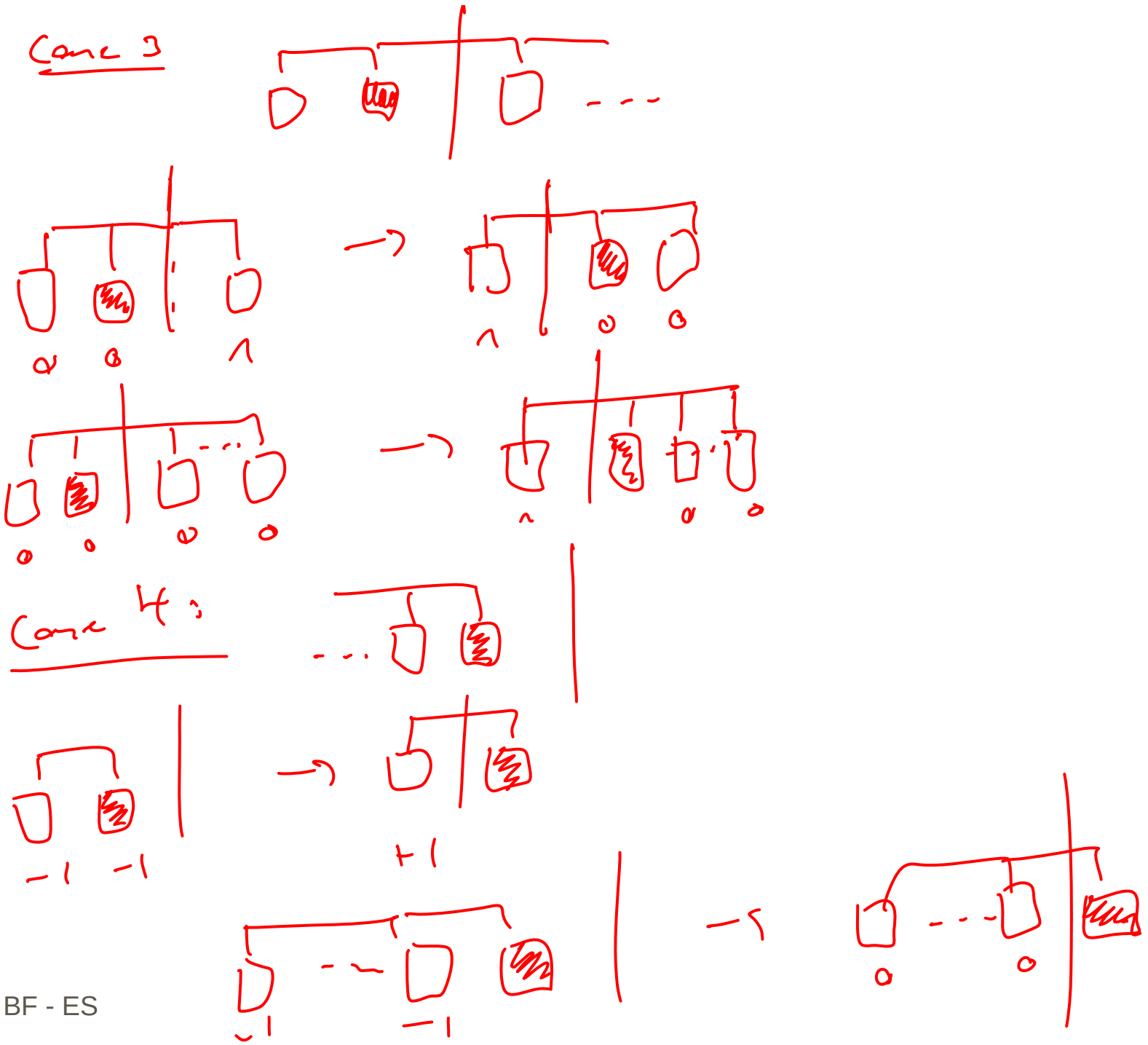


Updating gains

- For the update of the gains, we only need to consider nets that contain the cell selected for movement and that are critical before or after the move.

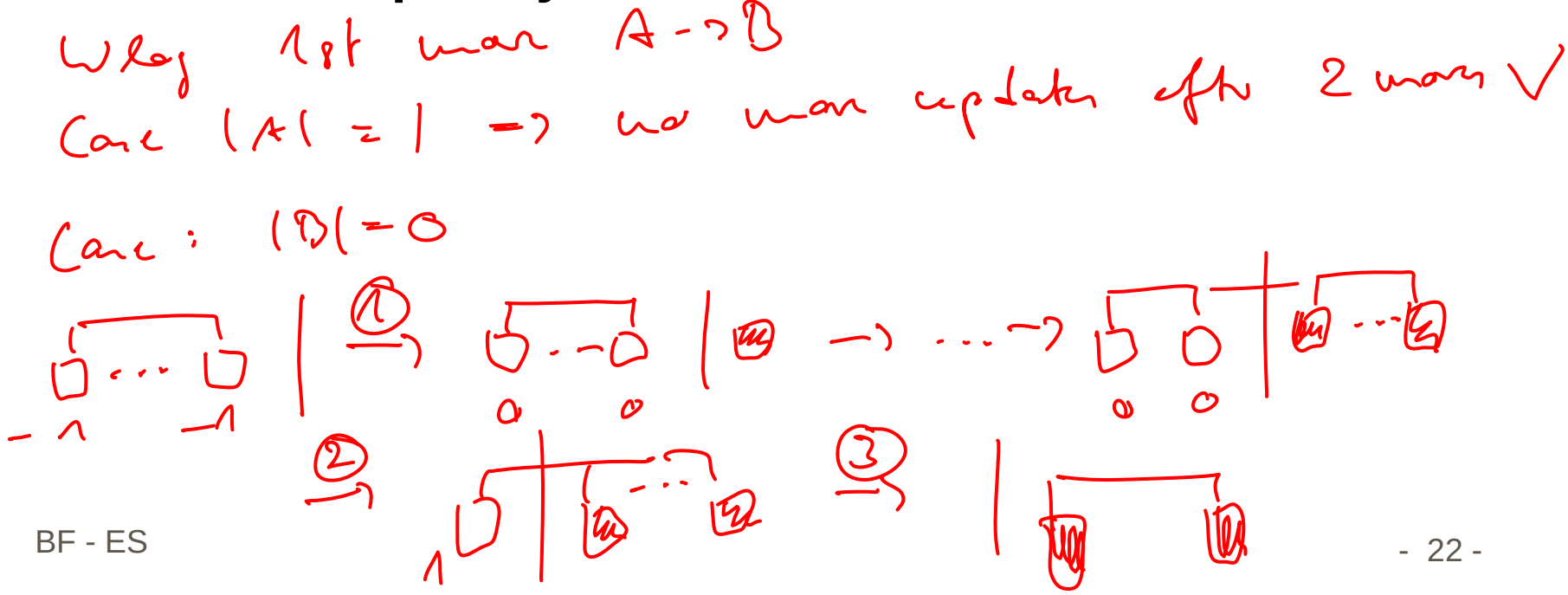



Cas 1 :
+1
+1
-1
Cas 2 :


Cas 3
Cas 4 :

Complexity

- Once a net has some locked cell at both sides, the net will remain in the cut set.
- At most 3 update operations per net during one pass of the algorithm
- **Linear complexity**



Case $|B|=1$

(similar).

Case $|B|>1$

(similar).

Simulated Annealing

- General method for solving combinatorial optimization problems.
- Based the model of slowly cooling crystal liquids.
- Changes leading to a poorer configuration (with respect to some cost function) are accepted with a certain probability.
- This probability is controlled by a temperature parameter: the probability is smaller for smaller temperatures.

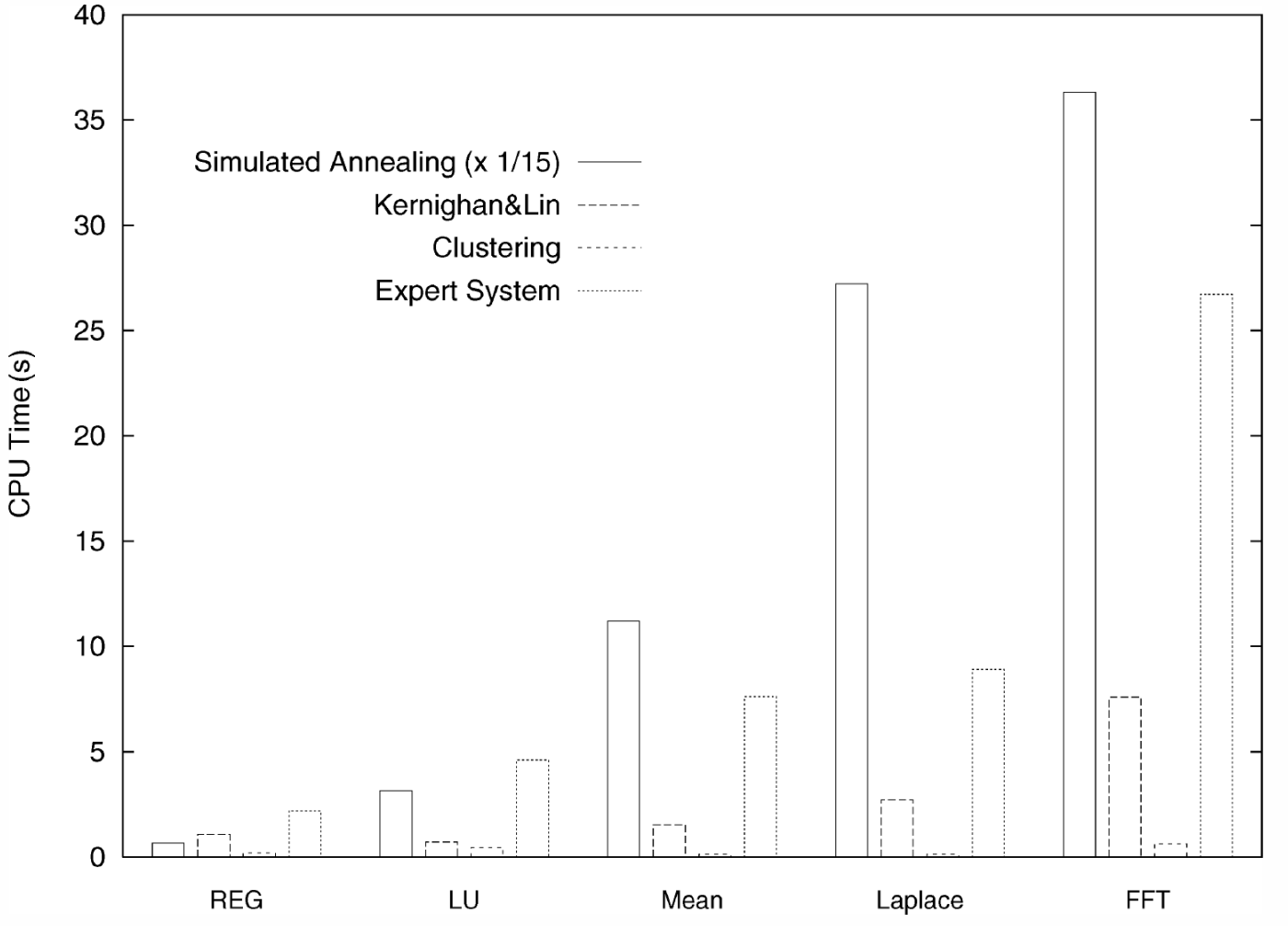
Simulated Annealing Algorithm

```
procedure SimulatedAnnealing;
var i, T: integer;
 begin
   temp := temp_start;
   cost:=c(P);
   while (Frozen()==FALSE) do
    begin
     while (Equilibrium()==FALSE) do
       begin P' := RandomMove(P);
         cost'=c(P')
         deltacost := cost' - cost;
         if (Accept(deltacost, temp)>random[0,1))
         then P=P'; cost=cost'
       end;
      temp:= decreaseTemp(temp)
   end;
end;
```

Simulated Annealing

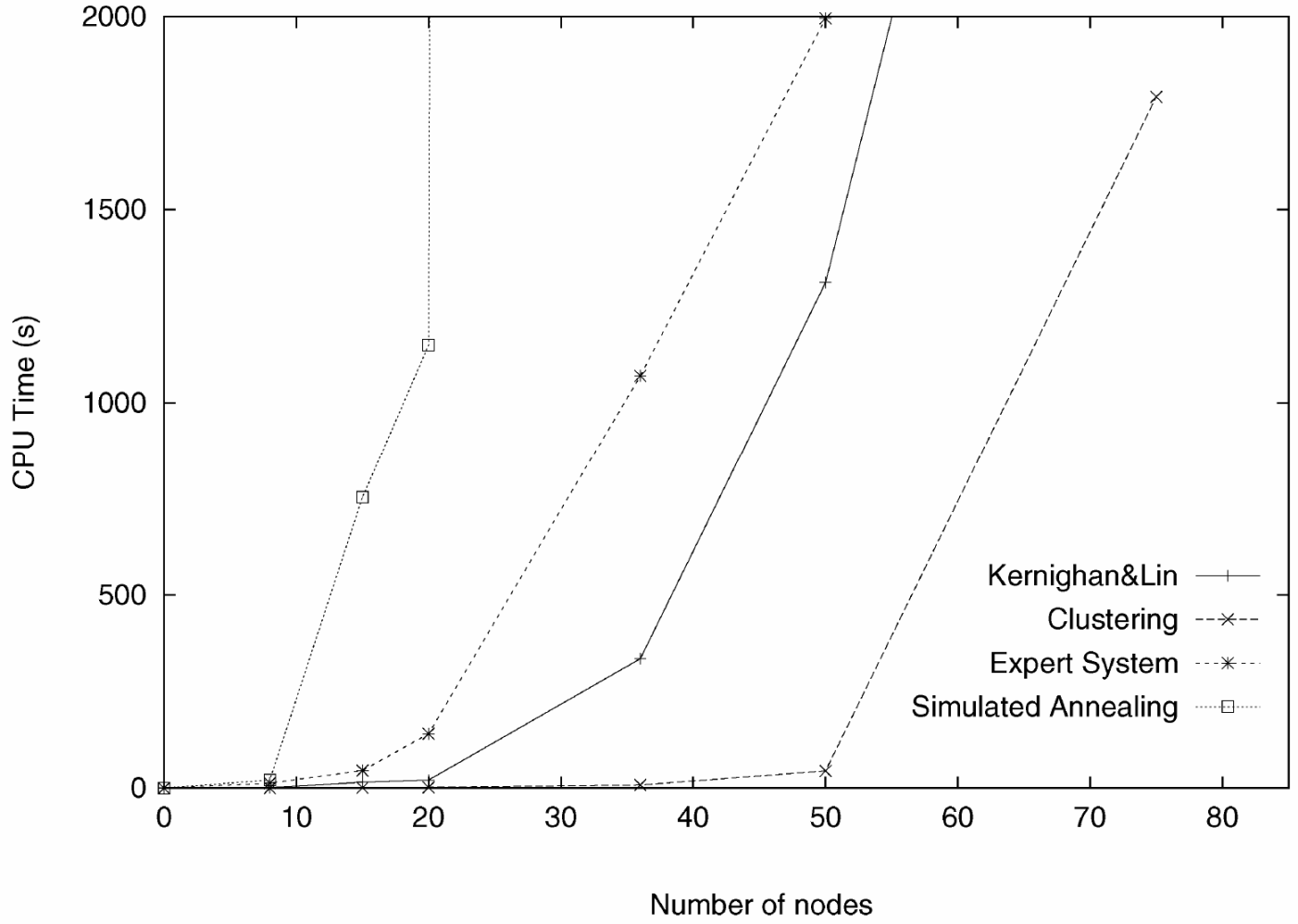
- Annealing schedule: DecreaseTemp(), Frozen()
  - temp_start=1.0
  - temp = $\alpha \cdot$ temp (typical: $0.8 \leq \alpha \leq 0.99$)
  - stop at temp < temp_min or if no more improvement
- Equilibrium:
  - After certain number of iterations or when no more improvement
- Complexity:
  - From exponential to constant, depending on choice of Equilibrium(), DecreaseTemp(), Frozen()
  - The longer the runtime, the better the results
  - Usually functions constructed to obtain polynomial runtime

Comparison of Partitioning Algorithms



López-Vallejo/López (2003):
On the Hardware-Software Partitioning Problem: System Modeling and Partitioning Techniques

Comparison of Partitioning Algorithms



López-Vallejo/López (2003):
On the Hardware-Software Partitioning Problem: System Modeling and Partitioning Techniques

Case Study: YSC (IBM)

- **Yorktown Silicon Compiler:** functional partitioning of hardware
- **Input:** functional description on the level of arithmetic and logical expressions
- **Target:** partitioning to several chips
- **Abstraction level:** functional units of datapaths (ALUs, registers)
- **Method:** hierarchical clustering

$$closeness(p_i, p_j) = \left(\frac{sharedwires(p_i, p_j)}{maxwires} \right)^{c_1} \cdot \left(\frac{maxsize}{\min\{size(p_i), size(p_j)\}} \right)^{c_2} \cdot \left(\frac{maxsize}{size(p_i) + size(p_j)} \right)$$

Case Study: Vulcan (Stanford)

- **Input:** program in HardwareC
  - C extended by a process concept and interprocess communication
  - Specification with constraints (min/max times and data rates)
- **Target:** 1 processor + several hardware components
  - One global bus, one global memory
  - Processor is bus master
- **Abstraction level:** basic blocks
- **Method:** HW-oriented greedy
  all operations with external nondeterminism in HW,
  all operations with internal nondeterminism in SW,
  deterministic operations: HW-oriented greedy
- cost function includes hw cost, memory requirement, performance and synchronization effort

Case Study: Cosyma (Braunschweig)

- **Input:** program in C^x
  - C extended by a process concept and interprocess communication
  - Specification with constraints (min/max times and data rates)
- **Target:** processor + coprocessor
  - Coupled by shared memory
  - Computations on the processor and coprocessor may not overlap in time
- **Abstraction level:** basic blocks
- **Method:** SW-oriented, two loops:
  - **Inner loop:** simulated annealing with cost function that gives the time gain für a HW realization of a block
  - **Outer loop:** synthesis to improve the estimation for the inner loop

Case study: COOL (Dortmund)

Inputs to COOL:

1. Target technology
   - As evaluation is based on either simulation or performance figures provided by code generators / hw synthesizers, it suffices to provide information on how to start the compilers / the hw synthesis, plus their relevant parameters (target technology library, etc.)
2. Design constraints
   - Required throughput, latency, max. memory, max. area, …
3. Required behavior
   - Task graphs for specifying the coordination, plus
   - VHDL descriptions of algorithmic behavior of leaf nodes

Steps of the COOL partitioning algorithm (1)

- **Translation of the behavior into an internal graph model**
- **Translation of the behavior of each node from VHDL into C**
- **Compilation**
  - All C programs compiled for the target processor,
  - Computation of the resulting program size,
  - estimation of the resulting execution time (simulation input data might be required)
- **Synthesis of hardware components**:
  - For each leaf node, application-specific hardware is synthesized.
  - High-level synthesis sufficiently fast (uses RT level rather than gate level synthesis; performance estimates found to be sufficiently precise).

Steps of the COOL partitioning algorithm (2)

- **Flattening of the hierarchy**:
  Granularity used by the designer is maintained.
  Cost and performance information added to the nodes. Information required for partitioning is pre-computed.
    - Can't be precise at that stage, yet sufficiently accurate estimates possible
- **Generating and solving a mathematical model of the optimization problem**:
  Integer programming (IP) model used to encode the optimization problem.
  Optimality of the partitioning coincides with optimality wrt. the cost function (up to approximations in the model due to lacking preciseness of parameter estimation, in particular wrt. communication time)
    - Solved with standard IP packages.

Steps of the COOL partitioning algorithm (3)

- **Iterative improvements & Interface Synthesis**
  Adjacent nodes mapped to the same hardware component are merged.
  After partitioning, the glue logic required for interfacing processors, application-specific hardware and memories is created.