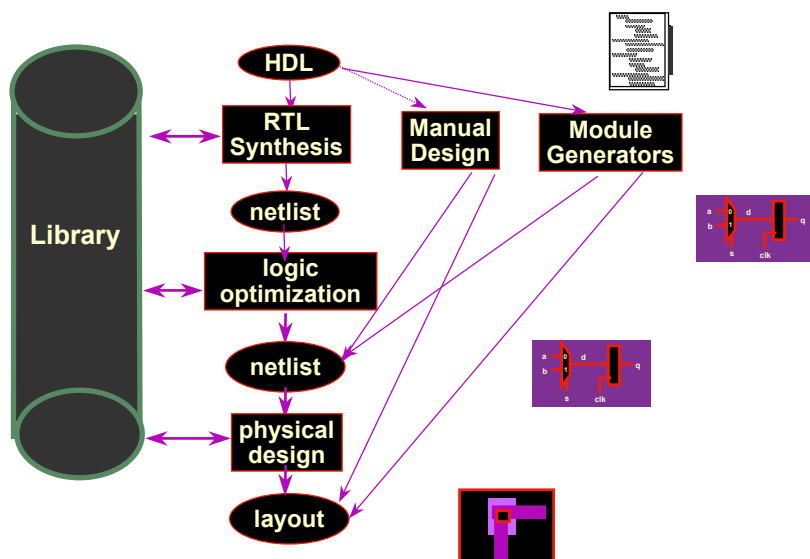


Layout Compaction

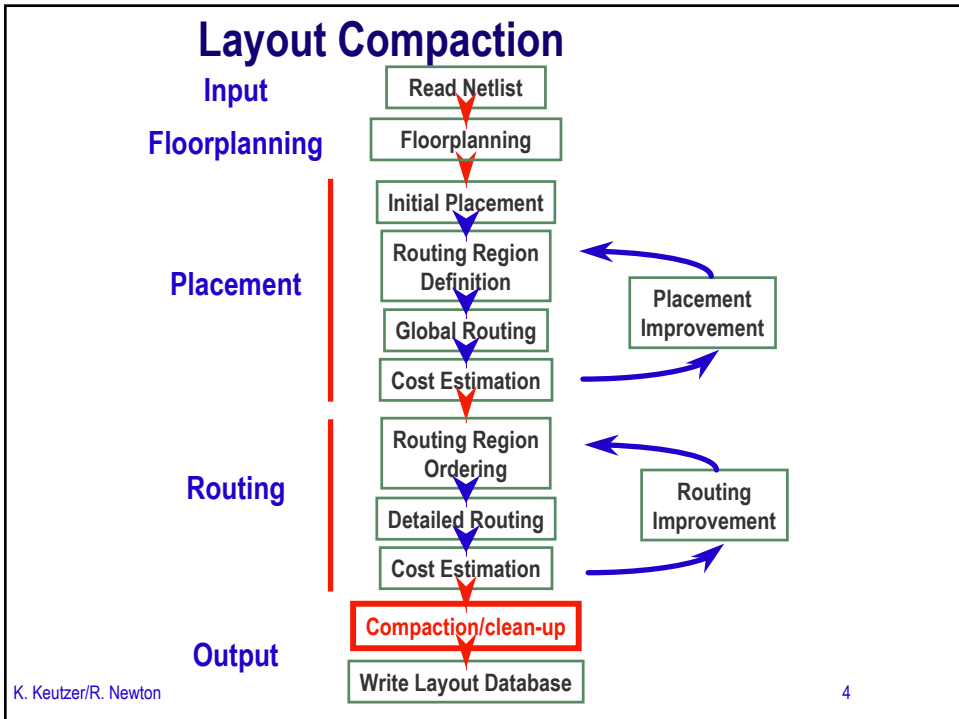
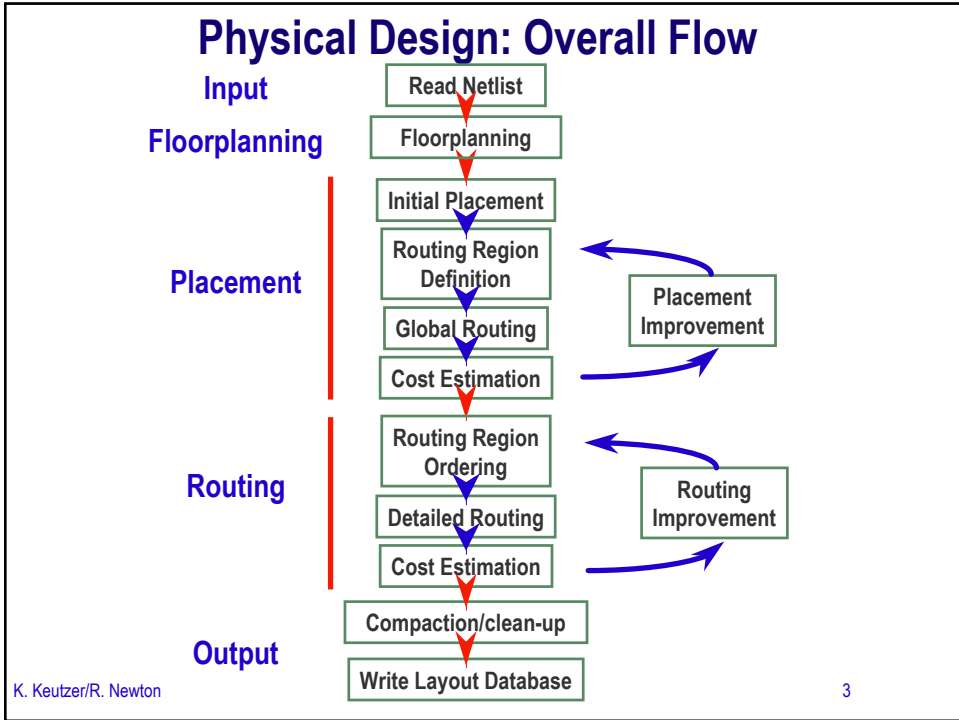
Prof. Kurt Keutzer
Prof. A. R. Newton
UC Berkeley
Prof. M. Orshansky
UCB → U of Texas

RTL Design Flow



K. Keutzer/R. Newton

2



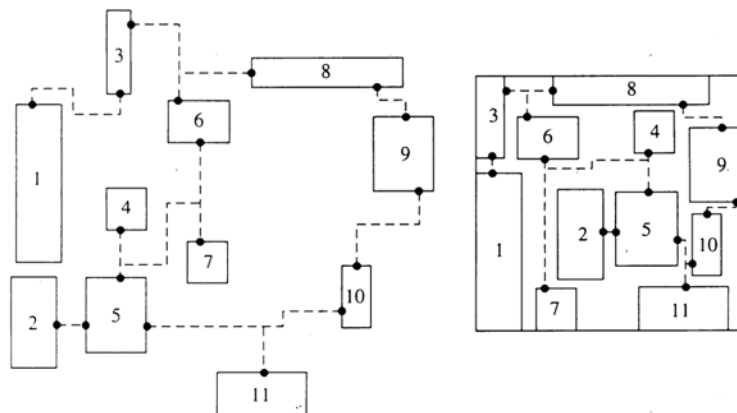
Compaction: Introduction

- ◆ After P&R, the layout is functionally complete
- ◆ Some vacant space may still be present in the layout
 - Due to non-optimality of P&R
- ◆ Compaction = removing vacant space
 - Improves cost, performance, and yield
- ◆ Key for high-performance full-custom layouts
- ◆ Standard cells – only channel heights may be minimized
 - But channel compactors are near-optimal

K. Keutzer/R. Newton

5

Layout Compaction



K. Keutzer/R. Newton

6

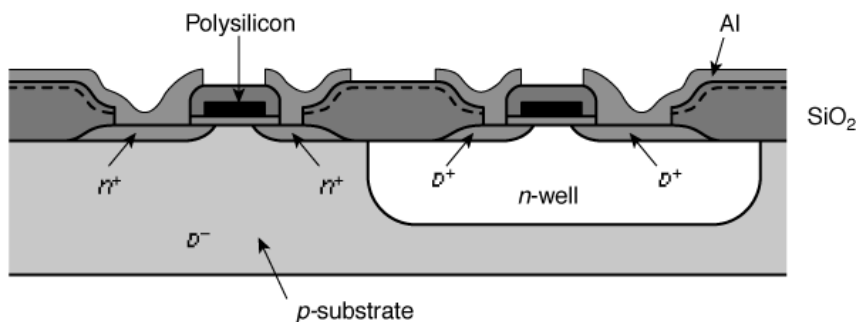
Compaction: Introduction

- ◆ Compaction tries to minimize total layout area while
 - Retaining speed
 - Respecting violating design rules and designer-specified constraints
- ◆ Three ways to minimize the layout area
 - Reducing inter-feature space
 - ◆ Check spacing design rules
 - Reducing feature size
 - ◆ Check size rules
 - Reshaping features
 - ◆ Electrical connectivity must be preserved

K. Keutzer/R. Newton

7

Cross-Section of CMOS Technology



Jan Rabaey

K. Keutzer/R. Newton

8

Design Rules










- ◆ Interface between designer and process engineer
- ◆ Guidelines for constructing process masks
- ◆ Unit dimension: Minimum line width
 - scalable design rules: lambda parameter
 - absolute dimensions (micron rules)

Jan Rabaey

K. Keutzer/R. Newton

9

CMOS Process Layers

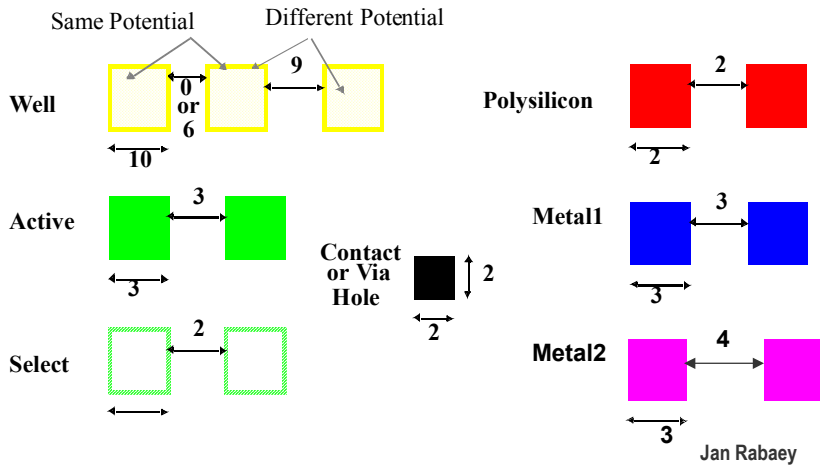
Layer	Color	Representation
Well (p,n)	Yellow	
Active Area (n+,p+)	Green	
Select (p+,n+)	Green	
Polysilicon	Red	
Metal1	Blue	
Metal2	Magenta	
Contact To Poly	Black	
Contact To Diffusion	Black	
Via	Black	

Jan Rabaey

K. Keutzer/R. Newton

10

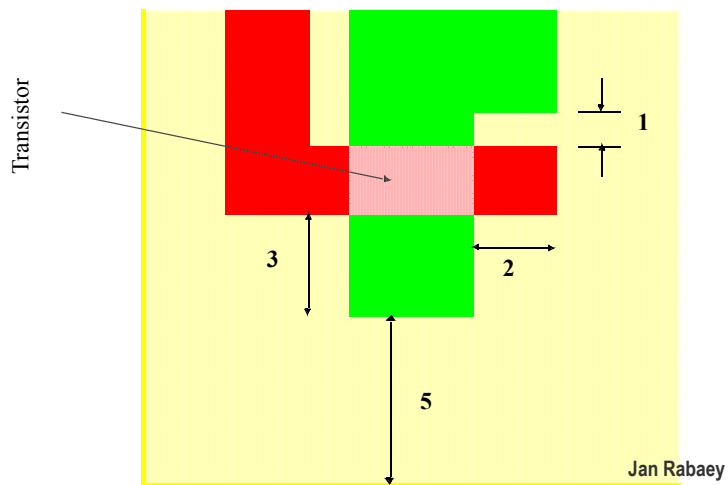
Intra-Layer Design Rules



K. Keutzer/R. Newton

11

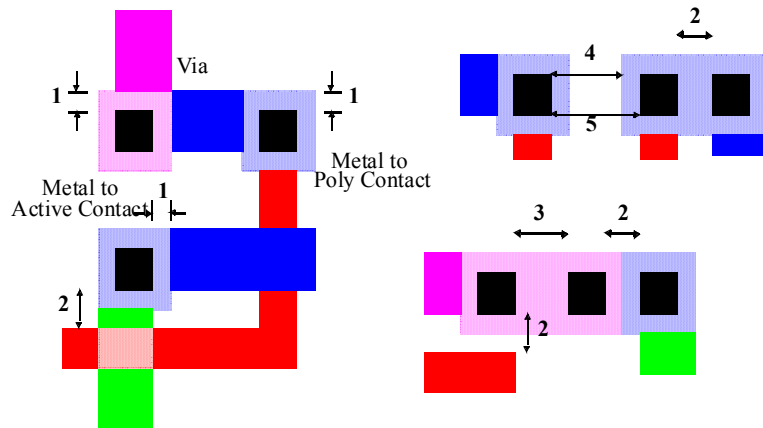
Transistor Layout



K. Keutzer/R. Newton

12

Via's and Contacts



K. Keutzer/R. Newton

Jan Rabaey

13

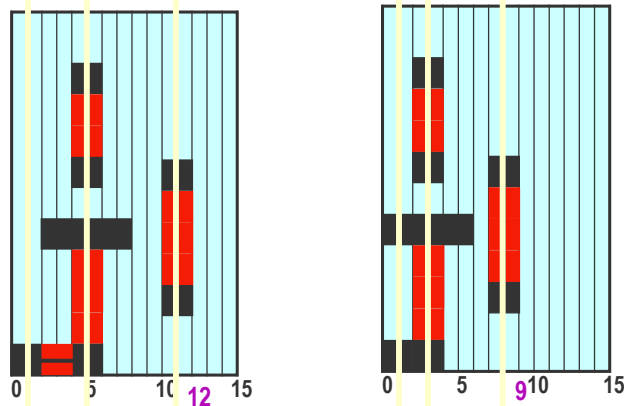
Different Approaches to Compaction

- ◆ One dimensional vs. two-dimensional compaction
- ◆ 1-D compaction
 - Components moved only in x- or y-direction
 - Efficient algorithms available
- ◆ 2-D compaction
 - Components may be moved both in x- and y-direction
 - More effective compaction
 - NP-hard
 - ◆ Determining how x and y should interact to reduce area is hard!
- ◆ Historical interest: Constraint-graph based compaction vs. virtual grid based compaction
 - Virtual grid methods are fast and simple. Results in larger area.

K. Keutzer/R. Newton

14

Historical interest: Virtual Grid Compaction

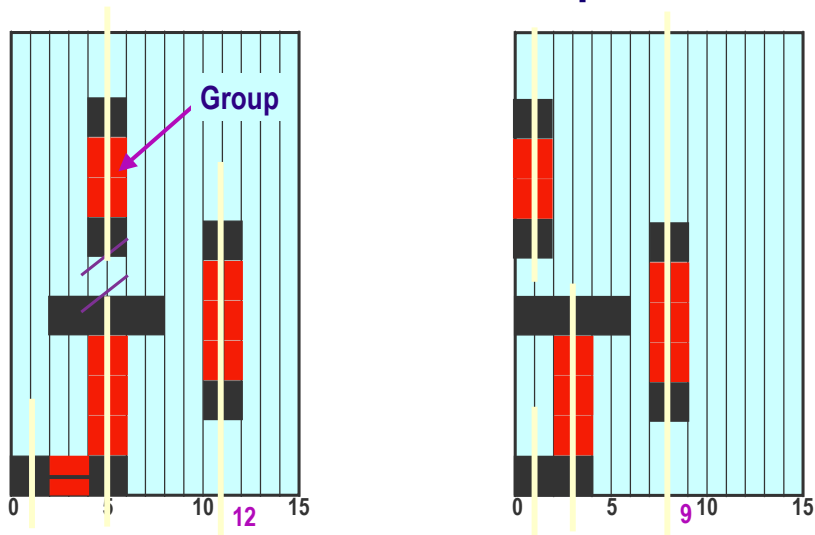


- ◆ Every feature is assumed to lie on a virtual grid line
- ◆ Required to stay at grid locations during compaction – no distortion of topology
- ◆ Compact by finding min possible spacing between each adjacent pair of grids
 - Min spacing is given by worst-case design-rule for any feature on the grid
- ◆ Advantage: algorithm is simple and fast

K. Keutzer/R. Newton

15

Constraint-based Compaction



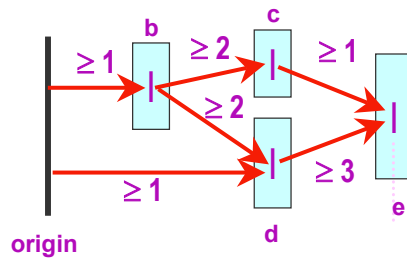
Early tools: Floss, Cabbage, SLIP, SLIM, Sticks

K. Keutzer/R. Newton

16

Layout Constraints

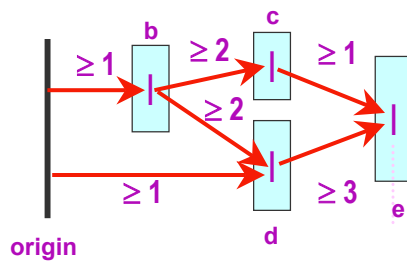
- ◆ For a given layout instance, all features may be described by a set of placement constraints
- ◆ These layout constraints are imposed by design rules
 - Min spacing (separation) design rules



K. Keutzer/R. Newton

17

Expressing constraints



$$b \geq \text{origin} + 1 \quad c \geq b + 2$$

$$d \geq \text{origin} + 1 \quad d \geq b + 2$$

$$e \geq c + 1 \quad e \geq d + 3$$

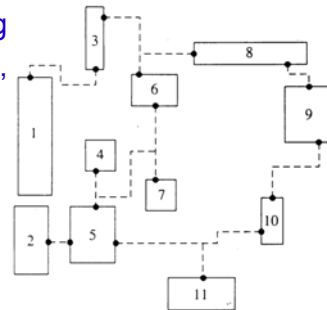
- ◆ Assuming a left-to-right ordering, can capture constraints by non-equalities
- ◆ Example: if c is to the right of b , and they have to be 2 units apart: $c \geq b + 2$

K. Keutzer/R. Newton

18

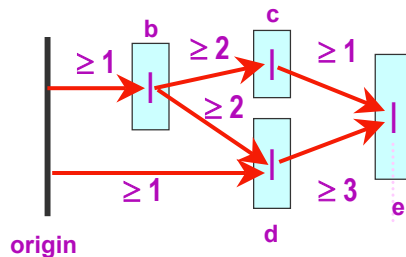
Constraint Graph Generation

- ◆ Constraint graph generation is the most time-consuming part of constraint-based compaction, approached naively $O(n^2)$
 - In the worst case, there may be a design rule between every shape of layout
- ◆ In reality only a small local subset of constraints are needed
- ◆ First generate connectivity (grouping)
- ◆ In generating separation constraints, non-redundant constraints
- ◆ Shadow-propagation algorithm
- ◆ Scan-line algorithm



K. Keutzer/R. Newton

How do we solve these constraints?



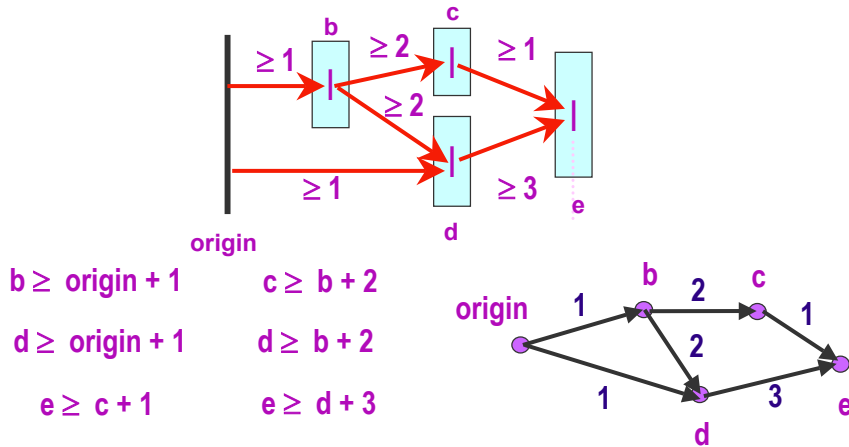
$$\begin{array}{ll}
 b \geq \text{origin} + 1 & c \geq b + 2 \\
 d \geq \text{origin} + 1 & d \geq b + 2 \\
 e \geq c + 1 & e \geq d + 3
 \end{array}$$

- ◆ What is a mathematically efficient way to solve these constraints?

K. Keutzer/R. Newton

20

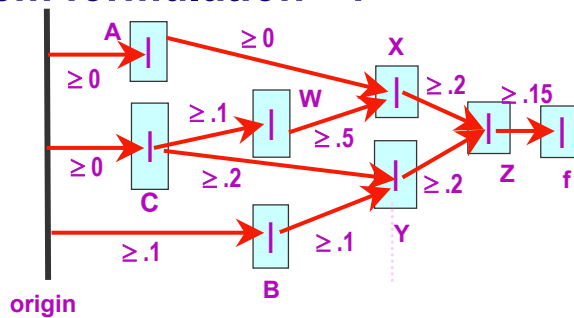
Use graphical structure



◆ We can construct a constraint graph to capture layout constraints

Problem formulation - 1

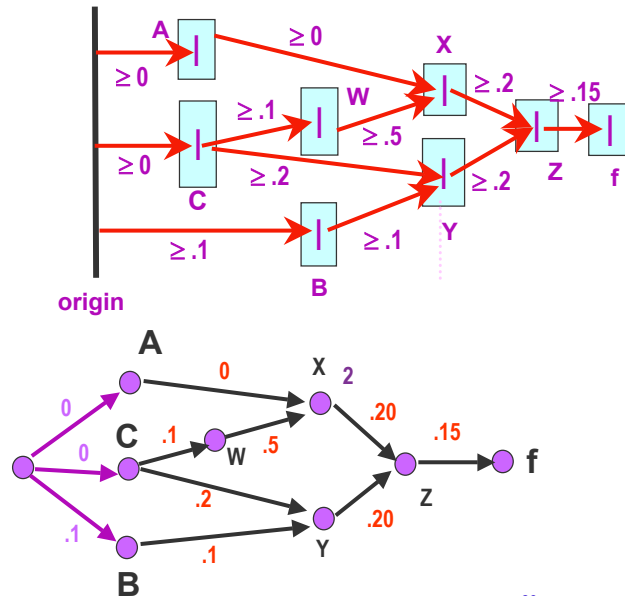
- ◆ Use a labeled *directed* graph
- ◆ $G = \langle V, E \rangle$
- ◆ Vertices layout objects
- ◆ Edges represent constraints between vertices
- ◆ Labels represent constraint values
- ◆ Now what do we do with this?



Hint: What problem does this remind you of?

Problem formulation - 1

- ◆ Use a labeled *directed* graph
- ◆ $G = \langle V, E \rangle$
- ◆ *Vertices* layout objects
- ◆ *Edges* represent constraints between vertices
- ◆ *Labels* represent constraint values
- ◆ Now what do we do with this?



K. Keutzer/R. Newton

23

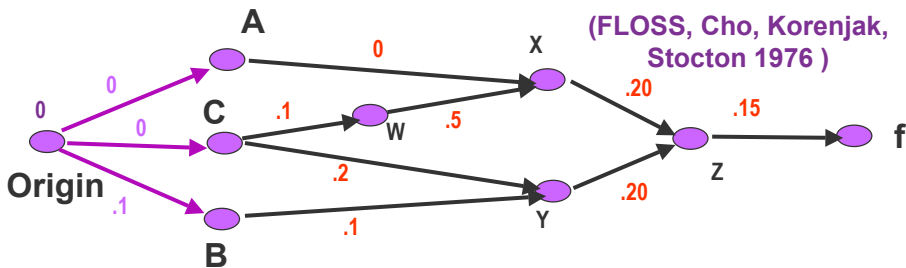
Problem formulation - 2

- ◆ Goal of 1-D compaction is to generate a minimum width layout
- ◆ Determination of minimum width is equivalent to solving a longest path problem
- ◆ The longest path from source to a vertex is the coordinate of the vertex
- ◆ In practical layouts, the constraints graphs are very local
 - ◆ Most edges represent very local constraints in the layout
- ◆ Theoretically, run time is $O(|V|+|E|)$
- ◆ Practically, run time is close to linear in $|V|$, the size of the layout!

K. Keutzer/R. Newton

24

Problem formulation - 3



Compute the longest path in a graph $G = \langle V, E, constraints, Origin \rangle$ (*constraints* is a set of labels, *Origin* is the super-source of the DAG)

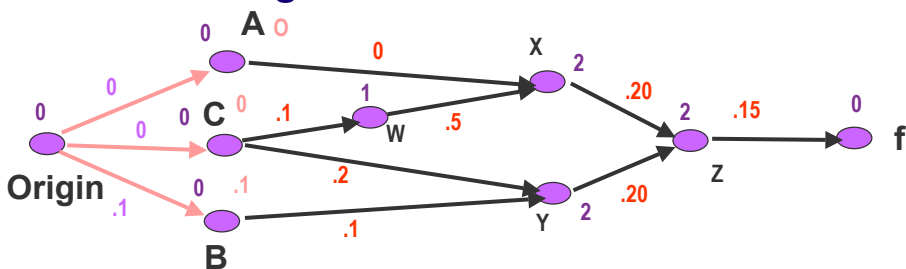
```

Forward-prop(W){
  for each vertex v in W
    for each edge <v,w> from v
      value(w) = max(value(w), value(v) + value(w) + constraint(<v,w>))
      if all incoming edges of w have been traversed
        add w to W
}
Longest path(G)
  Forward_prop(Origin)
}
    
```

K. Keutzer/R. Newton

25

Algorithm Execution -1



Compute the longest path in a graph $G = \langle V, E, constraints, Origin \rangle$ (*constraints* is a set of labels, *Origin* is the super-source of the DAG)

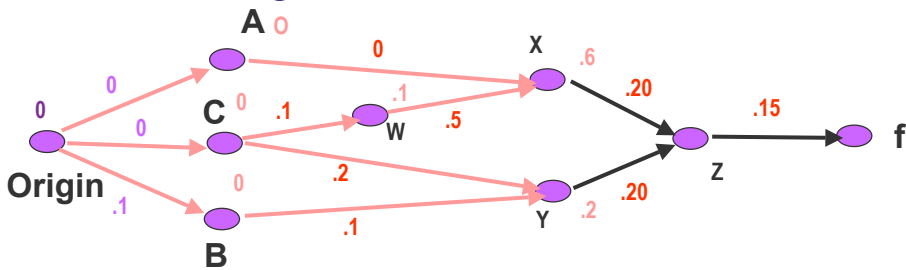
```

Forward-prop(W){
  for each vertex v in W
    for each edge <v,w> from v
      value(w) = max(value(w), value(v) + value(w) + constraint(<v,w>))
      if all incoming edges of w have been traversed
        add w to W
}
Longest path(G)
  Forward_prop(Origin)
}
    
```

K. Keutzer/R. Newton

26

Algorithm Execution - 2



Compute the longest path in a graph $G = \langle V, E, \text{constraints}, \text{Origin} \rangle$ (*constraints* is a set of labels, *Origin* is the super-source of the DAG)

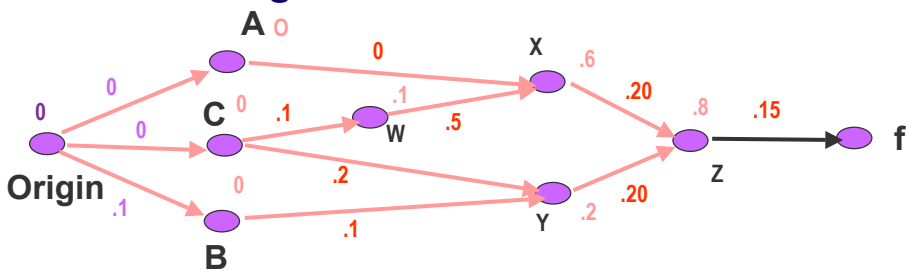
```

Forward-prop(W){
  for each vertex v in W
    for each edge <v,w> from v
      value(w) = max(value(w), value(v) + value(w) + constraint(<v,w>))
    if all incoming edges of w have been traversed
      add w to W
}
Longest path(G)
  Forward_prop(Origin)
}
    
```

K. Keutzer/R. Newton

27

Algorithm Execution - 3



Compute the longest path in a graph $G = \langle V, E, \text{constraints}, \text{Origin} \rangle$ (*constraints* is a set of labels, *Origin* is the super-source of the DAG)

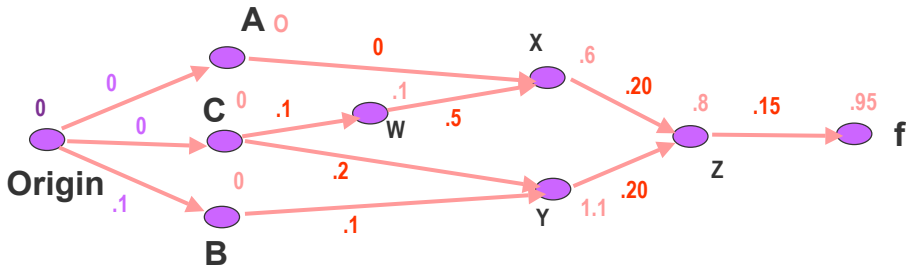
```

Forward-prop(W){
  for each vertex v in W
    for each edge <v,w> from v
      value(w) = max(value(w), value(v) + value(w) + constraint(<v,w>))
    if all incoming edges of w have been traversed
      add w to W
}
Longest path(G)
  Forward_prop(Origin)
}
    
```

K. Keutzer/R. Newton

28

Algorithm Execution - 4



Compute the longest path in a graph $G = \langle V, E, constraints, Origin \rangle$ (*constraints* is a set of labels, *Origin* is the super-source of the DAG)

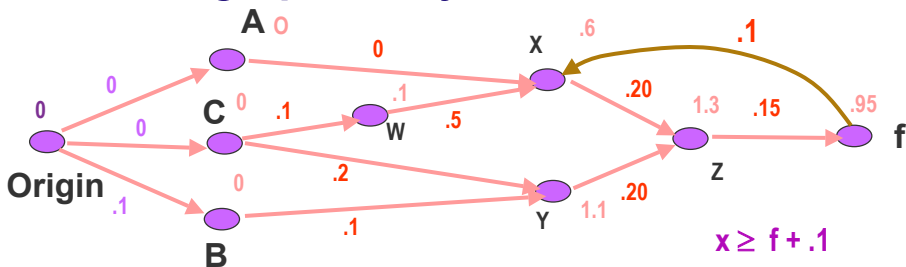
```

Forward-prop(W){
  for each vertex v in W
    for each edge <v,w> from v
      value(w) = max(value(w), value(v) + value(w) + constraint(<v,w>))
      if all incoming edges of w have been traversed
        add w to W
    }
  Longest path(G)
  Forward_prop(Origin)
}
  
```

K. Keutzer/R. Newton

29

Does graph always have a solution?

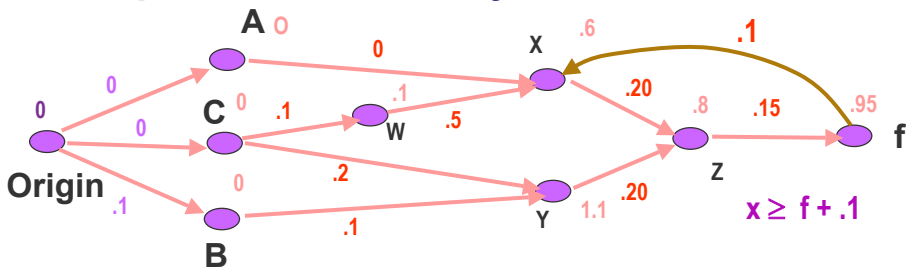


- ◆ What if we added this edge?

K. Keutzer/R. Newton

30

Graph does not always have a solution



Check for cycles in a graph $G = \langle V, E, constraints, Origin \rangle$ (*Origin* is the super-source of the *graph*)

```

depth_first_search(v){
  if visited(v) == TRUE
  then
    return "ERROR, CYCLE IN GRAPH WITH PATH THROUGH v"
  else
    visited(v) = TRUE
    for each edge <v,w> from v
      depth_first_search( w)
}
check_for_cycles(G){
  depth_first_search(Origin)
}

```

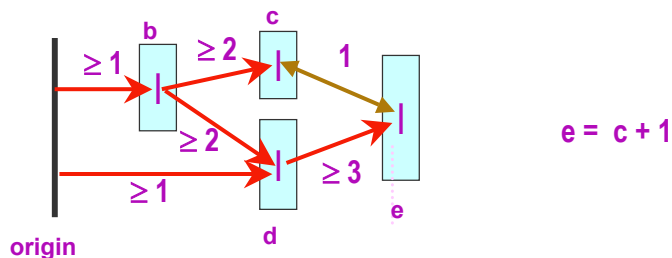
Algorithm breaks down because of cycles

K. Keutzer/R. Newton

31

First Elaboration - Equality Constraints -1

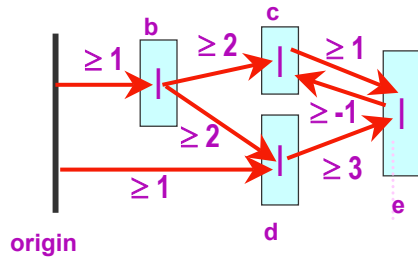
- ◆ Inequality constraints capture min spacing design rules
- ◆ Need also to capture *grouping constraints*
 - Features from same circuit component
 - Need to be moved together
- ◆ Grouping constraints are described by *equality constraints*



K. Keutzer/R. Newton

32

First Elaboration - Equality Constraints -2



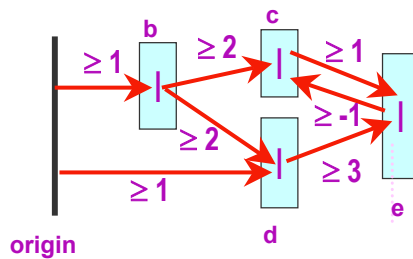
$$e = c + 1$$

$$\text{So what?} == \begin{aligned} e &\geq c + 1 \\ c &\geq e - 1 \end{aligned}$$

K. Keutzer/R. Newton

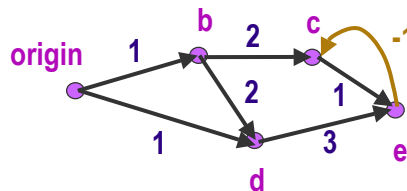
33

Reflecting Equality Constraints



$$e \geq c + 1$$

$$c \geq e - 1$$

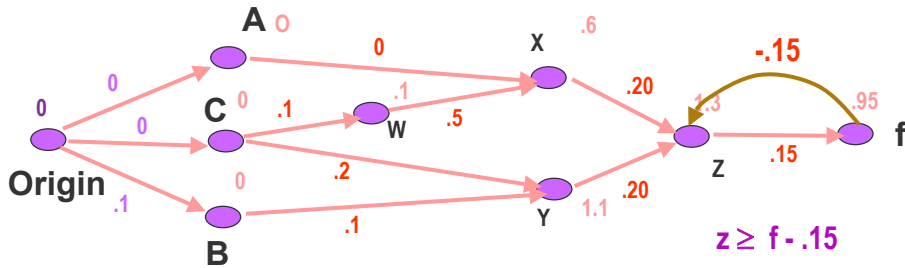


What challenges does this pose to our algorithm?

K. Keutzer/R. Newton

34

Dealing With Legitimate Cycles

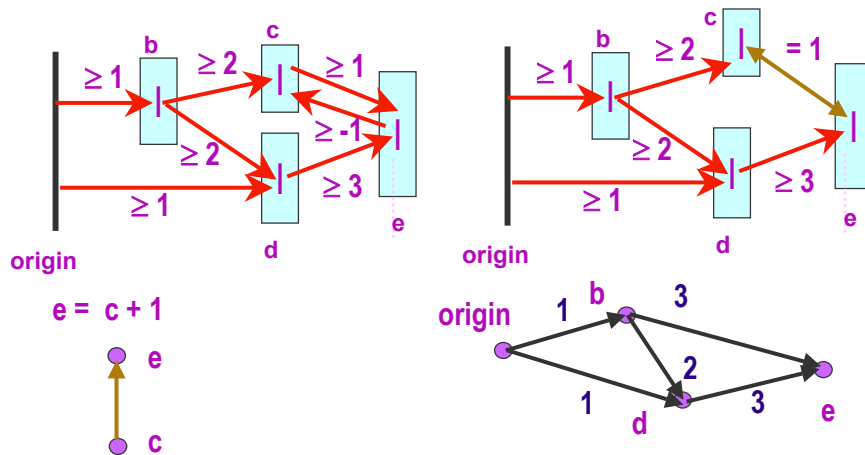


- ◆ Can we use the longest path algorithm on graphs with legitimate negative-edge cycles?
- ◆ Topological longest path algorithm can't handle them.
- ◆ Can use a Bellman / Moore algorithm for general graphs
 - Run time is $O(|V|*|E|)$
 - Run time of topological longest path algorithm on DAG is $O(|V|+|E|)$

K. Keutzer/R. Newton

35

Alternative Approach to Equality Constraints

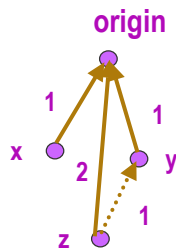


Handle equality constraints independently
 Build independent graph for equality constraints
 Express inequality constraints between *designated representatives*

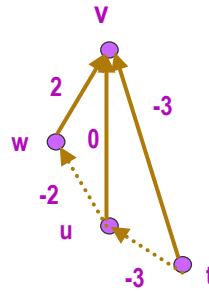
K. Keutzer/R. Newton

36

Handling Equality Constraints



$$\begin{aligned} \text{origin} &= x + 1 \\ \text{origin} &= y + 1 \\ z &= y + 1 \\ \text{origin} &= z + 2 \end{aligned}$$



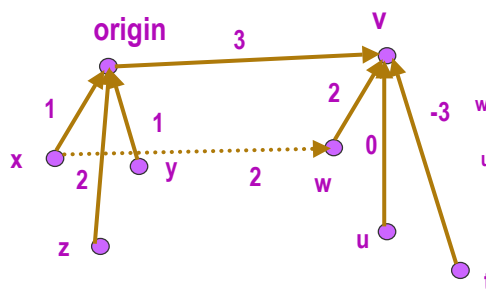
$$\begin{aligned} v &= w + 2 \\ w &= u - 2 \text{ i.e. } u = w + 2 \\ u &= t - 3 \text{ i.e. } t = u + 3 \end{aligned}$$

K. Keutzer/R. Newton

37

Handling Equality Constraints

$$\begin{aligned} \text{origin} &= x + 1 \\ \text{origin} &= y + 1 \\ y &= z + 1 \end{aligned}$$



$$\begin{aligned} v &= w + 2 \\ w &= u - 2 \text{ or } u = w + 2 \\ u &= t - 3 \text{ or } t = u + 3 \end{aligned}$$

Generally, this is known as the union-find algorithm

$$\begin{aligned} w &= x + 2 \\ w(v - 2) &= x(\text{origin} - 1) + 2 \\ v - 2 &= \text{origin} - 1 + 2 \\ v &= \text{origin} + 3 \end{aligned}$$

K. Keutzer/R. Newton

38

Constraint-Based Compaction Approach: Overview

Build constraint graph

if equality constraint

add to equality constraint graph

if inequality constraint

find *distinguished representatives* of each vertex in constraint

add constraint between *distinguished representatives*

Check for cycles in inequality constraint graph

If cycles exist terminate with error

Solve equality constraint graph

Solve inequality constraint graph

K. Keutzer/R. Newton

39

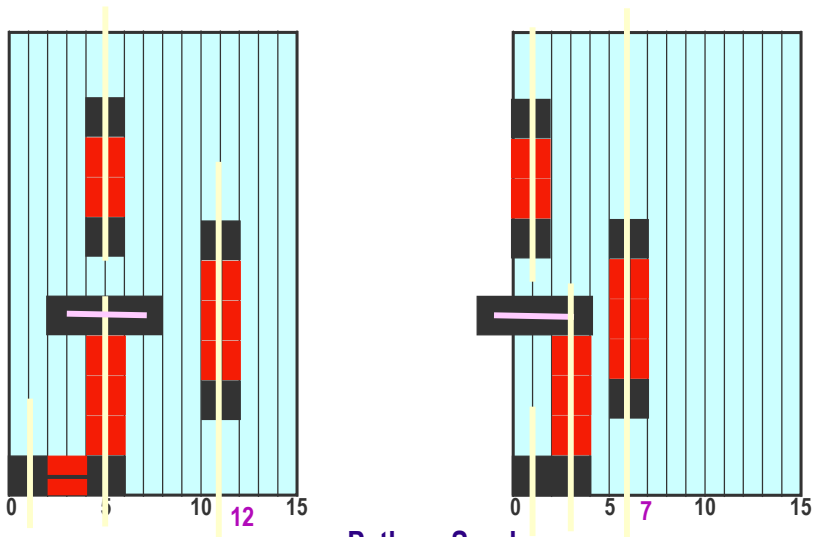
Compaction Enhancements

- ◆ 1-D constraint-based compaction problem can be formulated optimally and computationally efficiently
- ◆ In real circuits what we want is often more complex than can be captured in simple linear inequalities of the form:
 - $e \geq c + 1$
- ◆ Or equalities of the form:
 - $u = t - 3$
- ◆ For example:
 - Wirelength minimization
 - Spacing, slack distribution
 - Jog introduction
- ◆ Improving area minimization using:
 - 1 ½ D compaction
 - 2D compaction

K. Keutzer/R. Newton

40

Enhancements: Sliding/Spacing Terminals

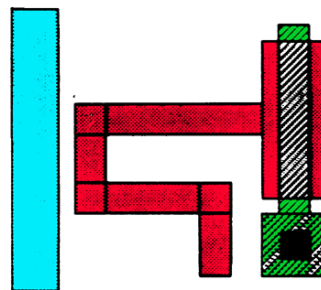
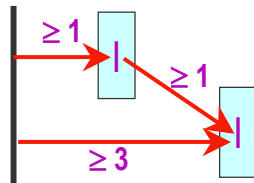


Python, Sparks

(requires use of upper as well as lower-bound constraints)

K. Keutzer/R. Newton

Wire-Length Minimization

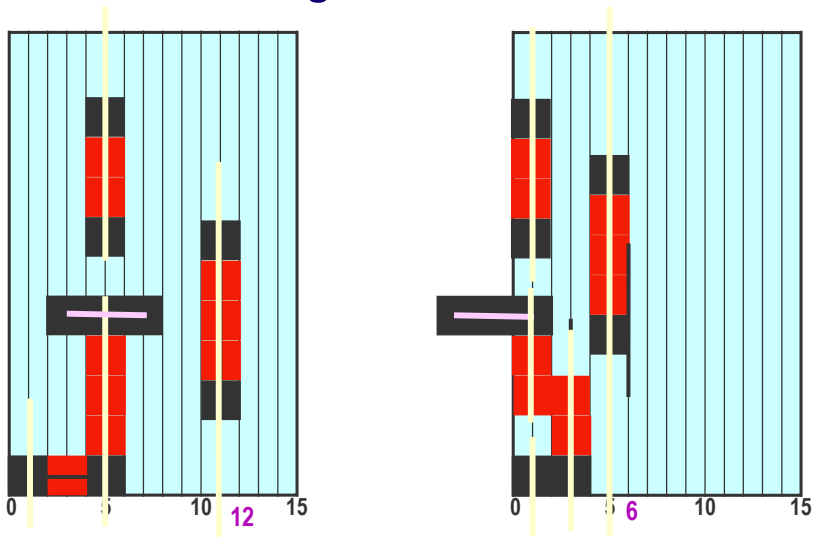


- ◆ Features not on the critical path will be pulled towards a layout edge because they are given their minimal legal spacing
- ◆ May lead to increased wire length and slower circuit performance
- ◆ Can re-distribute 'slack' (the available empty space) to the features not on the critical path

K. Keutzer/R. Newton

42

Jogs in "Wires"

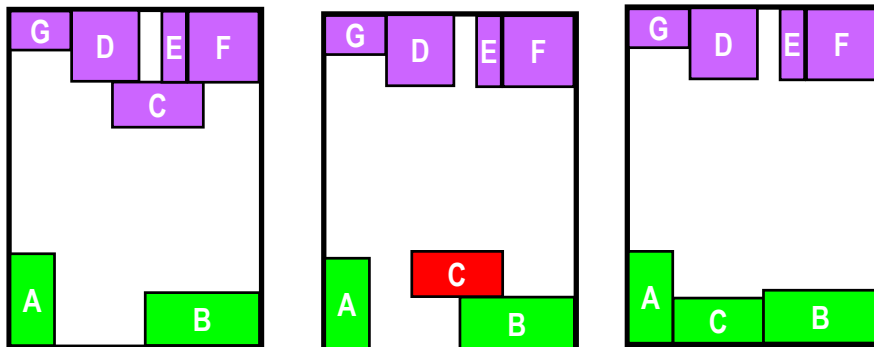


Cabbage, SLIP, Dumbo

K. Keutzer/R. Newton

43

One-and-a-half Dimensional Compaction



C compacted up

C compacted down

If we could just bump C over

K. Keutzer/R. Newton

44

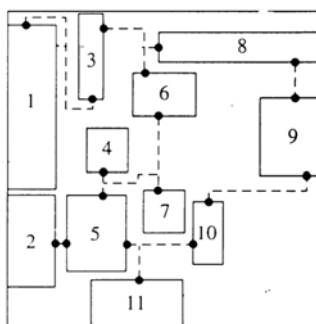
One-and-a-half Dimensional Compaction

- ◆ Key idea: provide enough lateral movements to blocks during compaction to resolve interferences
- ◆ Algorithm starts with a partially compacted layout (two applications of 1-D compaction)
- ◆ Maintain two lists – floor and ceiling
- ◆ Floor is a list of blocks visible from the top, ceiling is the list of blocks visible from the bottom
- ◆ Select the lowest block in the ceiling and move it to the floor *maximizing the gap between floor and ceiling*.
- ◆ Continue until all blocks have been moved from ceiling to floor.

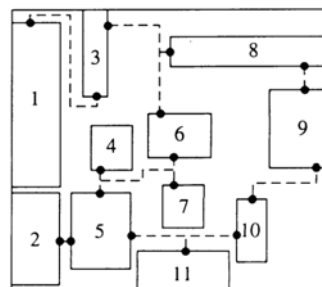
K. Keutzer/R. Newton

45

1-Dimensional Compaction in 2D



X then Y 1D Compaction

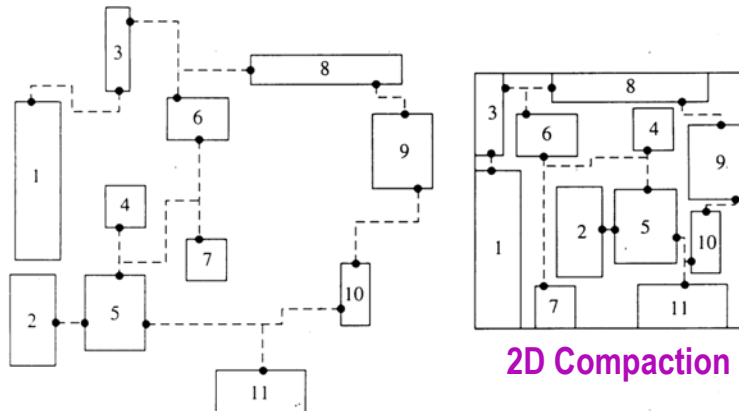


Y then X 1D Compaction

K. Keutzer/R. Newton

46

True Two-Dimensional Compaction

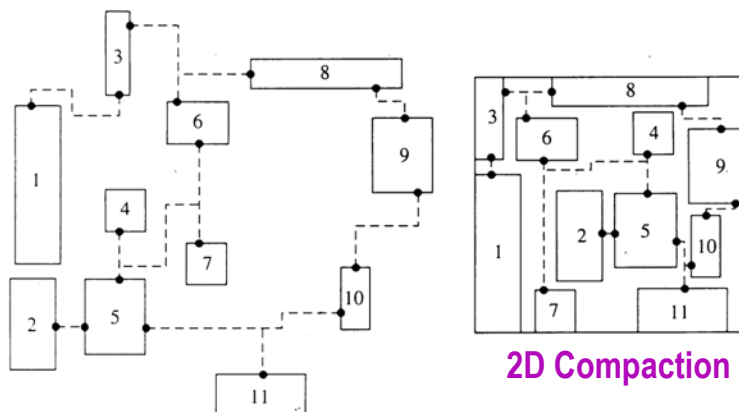


- ◆ Two dimensional compaction is NP Hard (C. K. Wong, 1984)
- ◆ Choosing how two dimensions should interact to produce optimal is hard
- ◆ Can formulate as integer-linear programming problem
 - Worst-case complexity is exponential

K. Keutzer/R. Newton

47

What makes 2-D compaction hard?

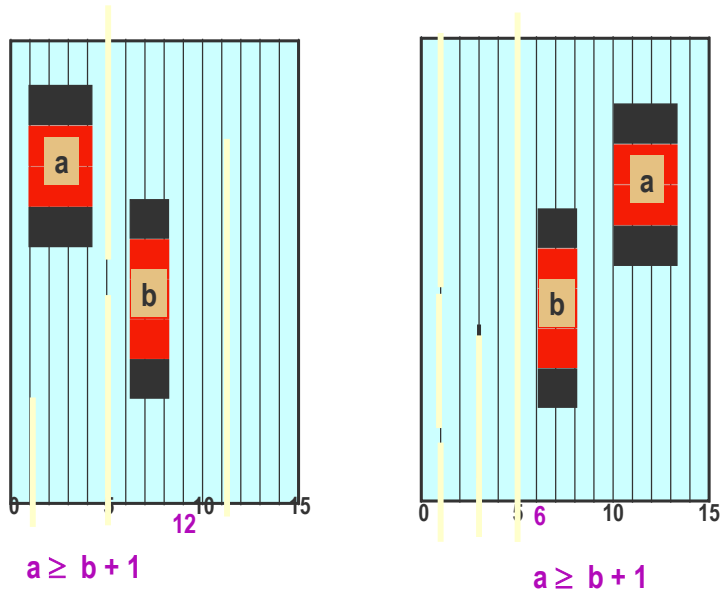


- ◆ Two dimensional compaction is NP Hard (C. K. Wong, 1984)
- ◆ Choosing how two dimensions should interact to produce optimal is hard
- ◆ Can formulate as integer-linear programming problem
 - Worst-case complexity is exponential

K. Keutzer/R. Newton

48

Choices



K. Keutzer/R. Newton

49

Background Material

Handout:

Chapter 2: entitled *Images of Algorithms and Techniques for VLSI Layout Synthesis*, Kluwer Academic Publishers, 1989.
pages 6 - 30..

Algorithmic background:

Introduction to Algorithms, T. Cormen, C. Lesierson, R. Rivest,
The MIT Press, Second Printing, 1996.

depth-first search 477-485

shortest paths 514-578 (probably overkill for our purposes)

union-find algorithm (disjoint forest implementation 448)

K. Keutzer/R. Newton

50