Layout Compaction

ELEC 5402

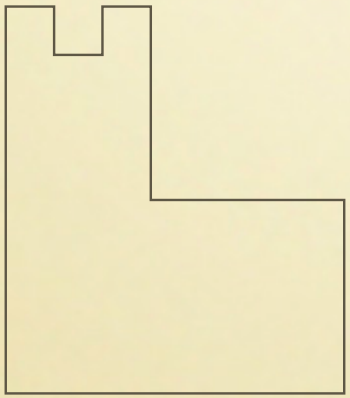
Pavan Gunupudi

Dept. of Electronics, Carleton University

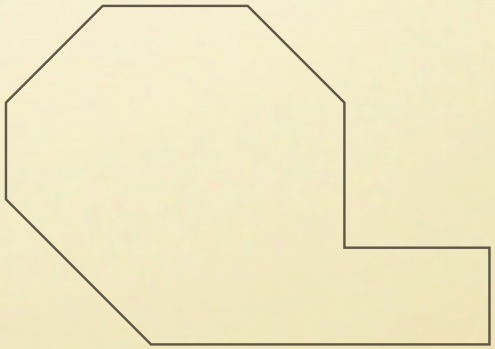
January 10, 2012

Introduction

- General goal is to minimize layout area
- Layout is a collection of polygons
- Generally we have rectilinear polygons
- Some technologies allow polygons to have 45-degree segments

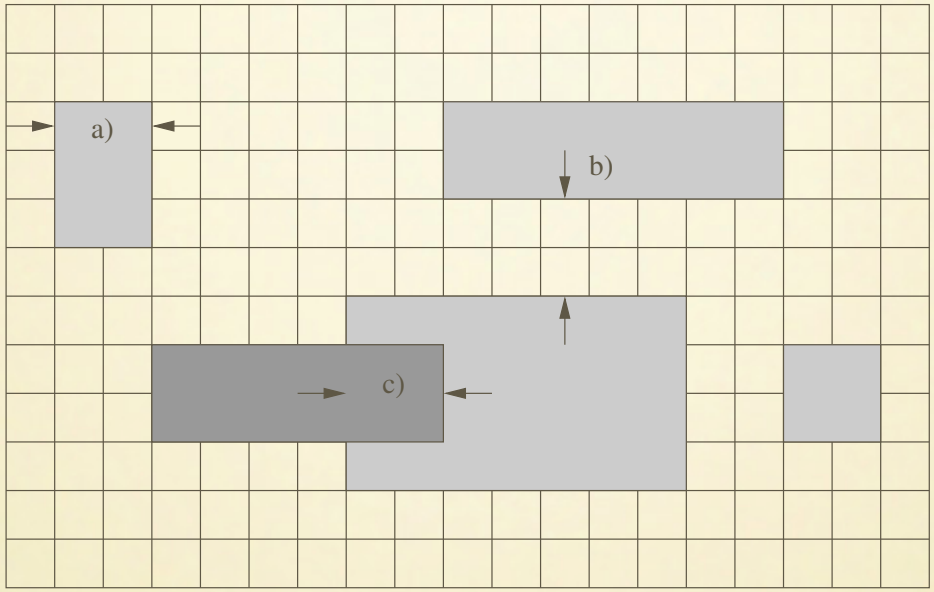


a)
Rectilinear Polygon



b)
45 degree segments

Design Rules



a) Minimum width b) Minimum separation c) Minimum overlap

- Can only place rectangles on grid points
- Constraints imposed through design rules
- Constraints for polygons on same layer and different layers
- Expressed as minimum/maximum distance rules

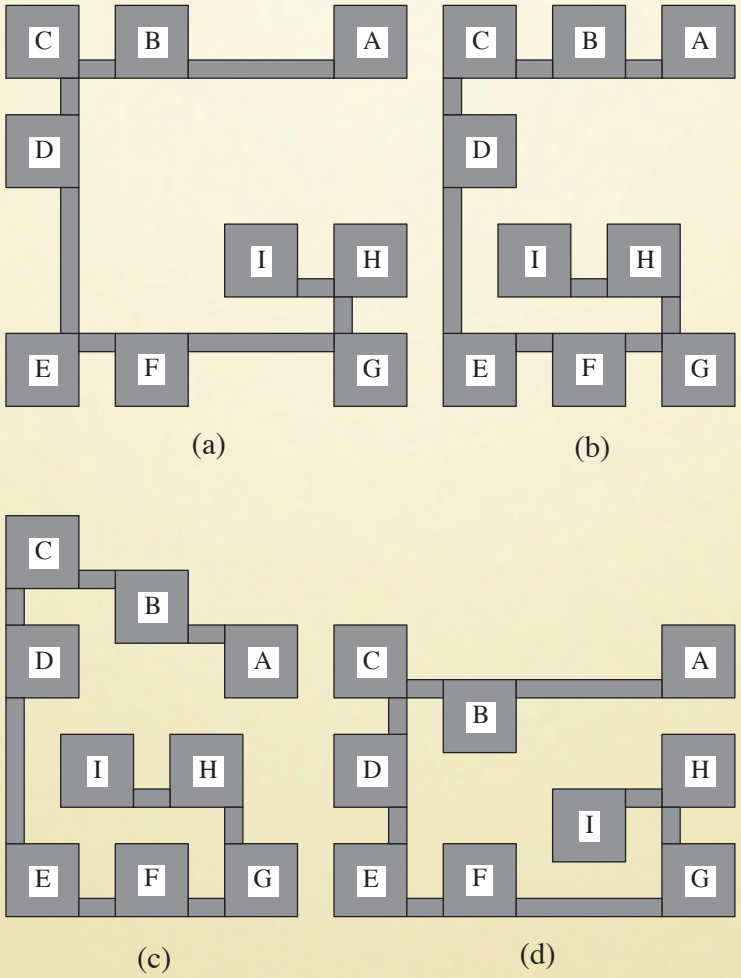
Applications of Layout Compaction

- Removing redundant area from geometric layout
- Adapting geometric layout to new technologies
  - If technology changes, design rules change
  - geometric layout has to be converted to symbolic layout and then reconverted to geometric layout with new design rules
- Correcting small design rule errors
- Converting symbolic layout to geometric layout

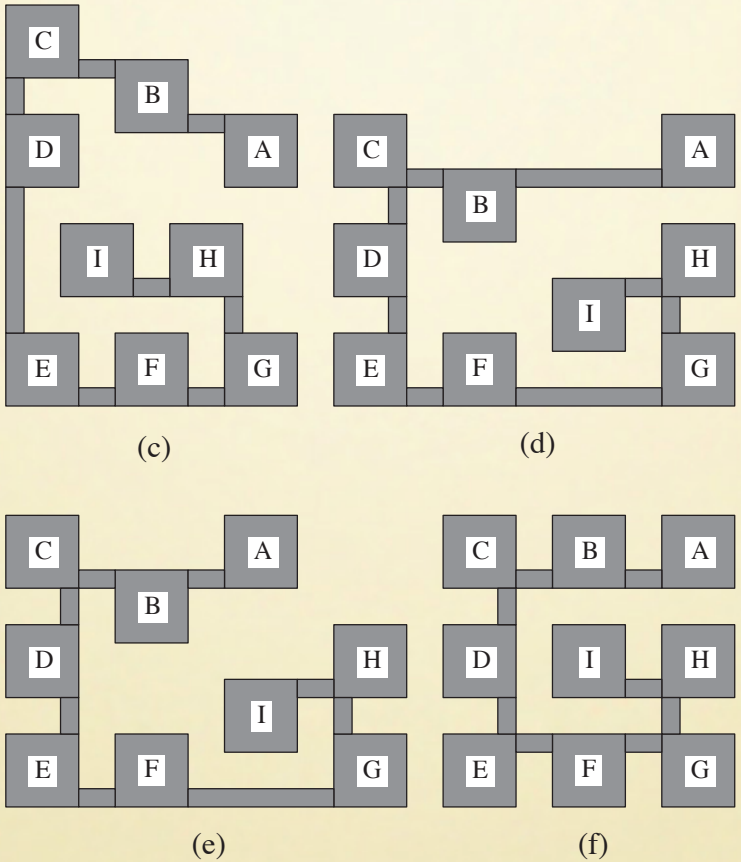
Problem Formulation

- Layout is a collection of rectangles
- Two groups of rectangles: rigid and stretchable
- Rigid: e.g. Transistors, contact cuts etc.
- Stretchable: e.g. Wires (not width but length)
- Layout compaction is a 2-D problem
- 2-D layout compaction is NP-Complete (heuristics needed)
- 1-D layout compaction is in P
- Repeated 1-D layout compaction in each dimension is a valuable *heuristic* for 2-D layout compaction

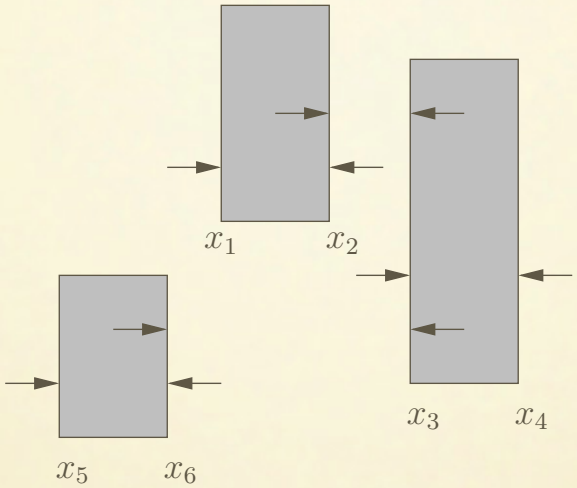
2-D, 1-D Layout Compaction



2-D, 1-D Layout Compaction



Graph Formulation

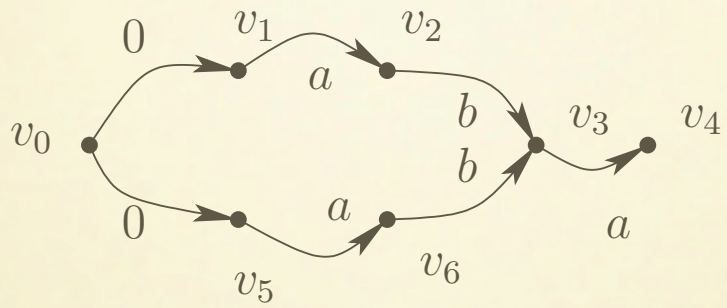


- Rigid rectangle - one variable
- Stretchable rectangle - two variables
- Minimum distance rule: $x_j - x_i \geq d_{ij}$

$$x_2 - x_1 \geq a; \qquad x_3 - x_2 \geq b; \qquad x_3 - x_6 \geq b$$

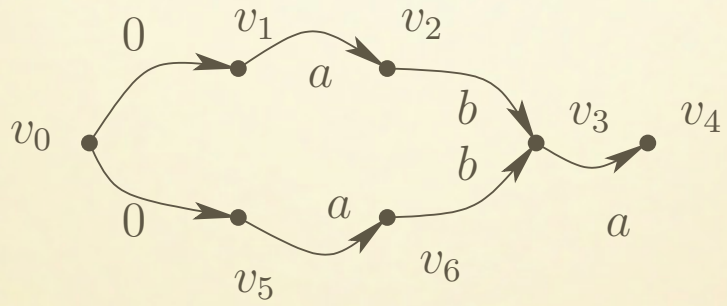
$$x_6 - x_5 \geq a; \qquad x_4 - x_3 \geq a$$

Constraint Graph



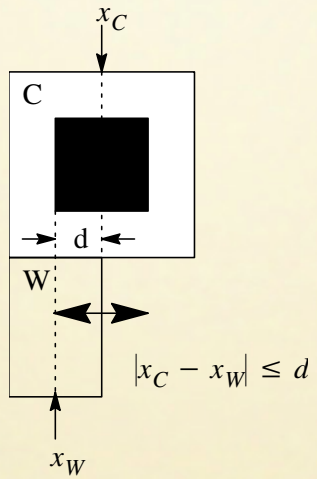
- Vertices of the graph $v_i => x_i$ (source vertex v_0)
- Edges (branches) (v_i, v_j) with weight $w((v_i, v_j)) = d_{ij}$ for each inequality $x_j - x_i \geq d_{ij}$
- The graph can be denoted as $G(V, E)$; V is the set of vertices and E is the set of edges

Constraint Graph - Solution



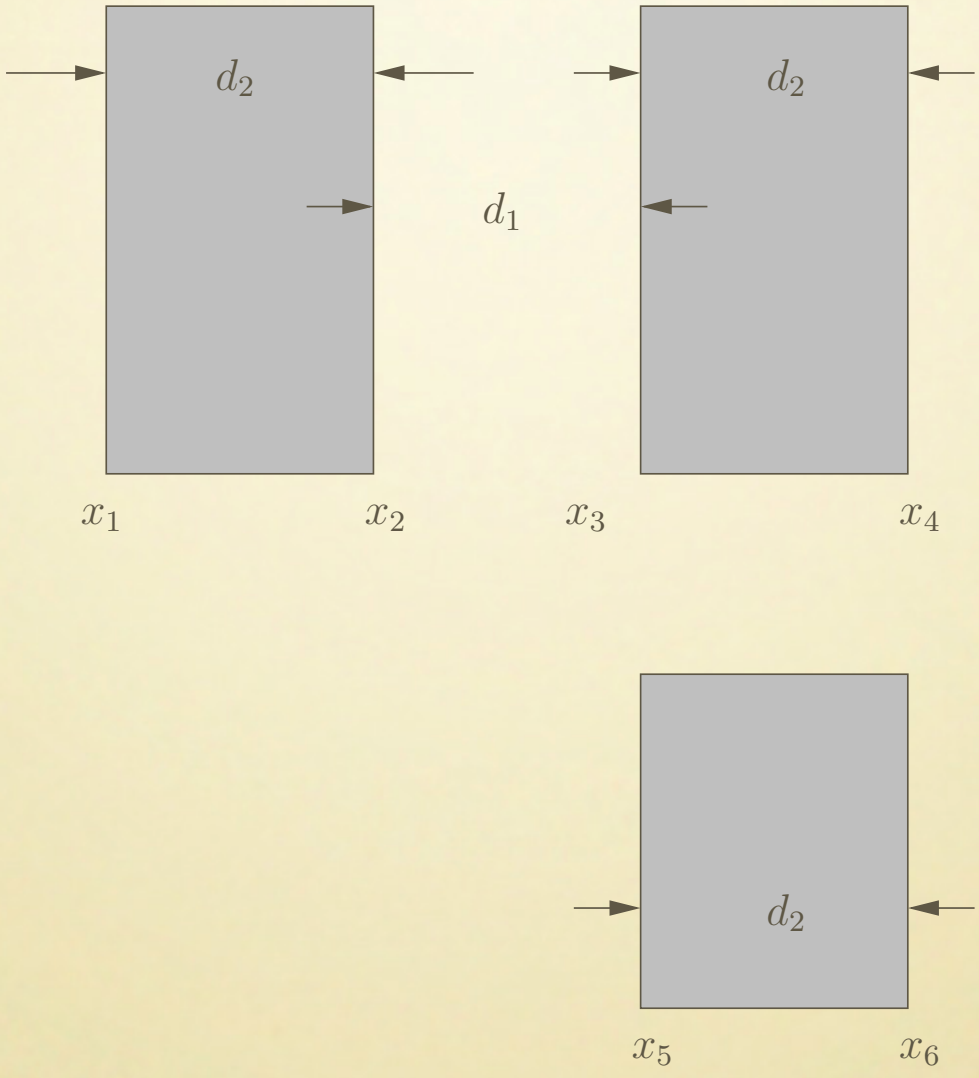
- Length of the longest path from v_0 to v_i gives the minimal x-coordinate x_i associated with the vertex v_i
- By taking the longest path to v_i we make sure that all inequalities in which x_i participates are satisfied

Maximum-Distance Constraints



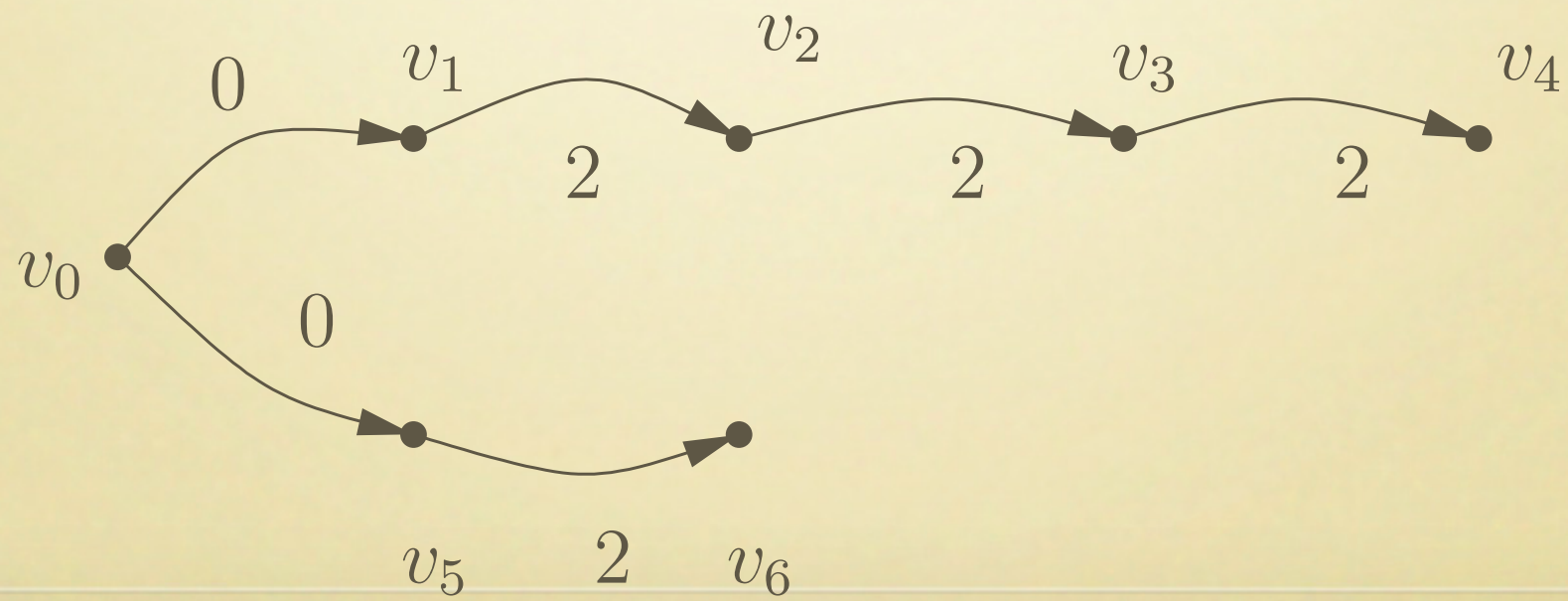
- Written as $x_C - x_W \geq -d$ and $x_W - x_C \geq -d$
- Leads to cycles; solution still longest path

Example 1

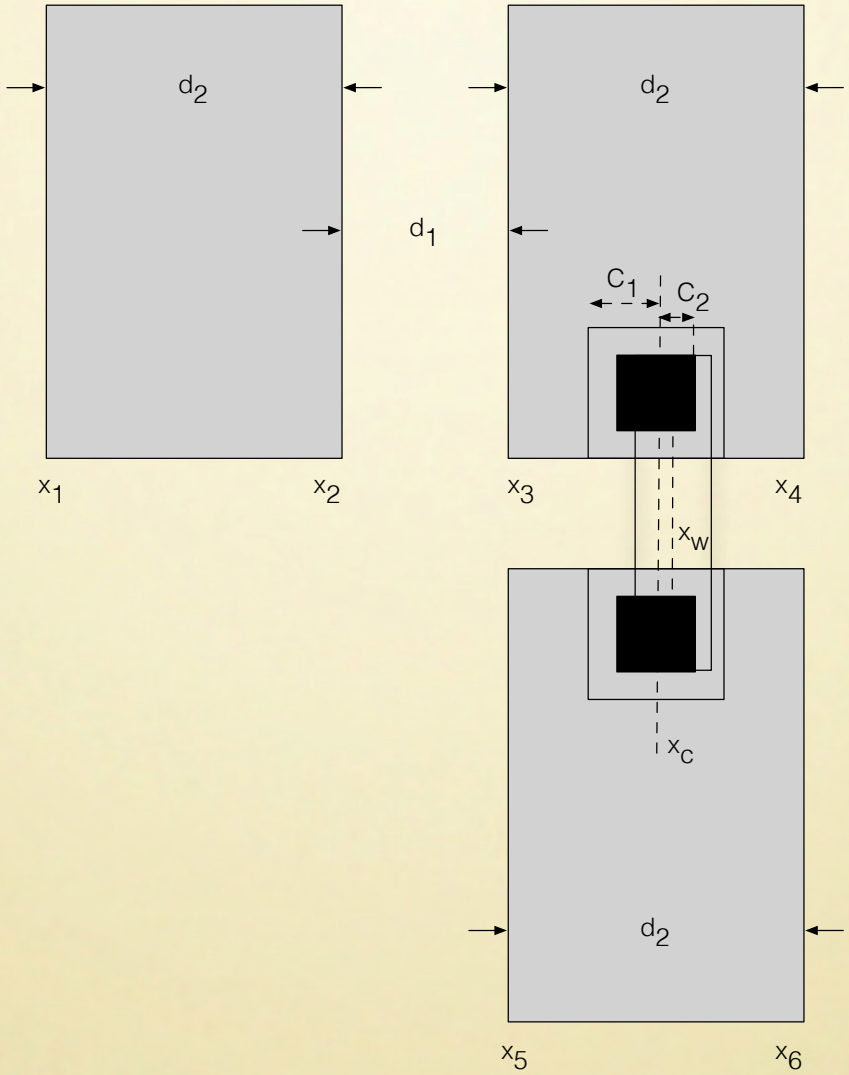


Example 1

$$x_3 - x_2 \geq d_1 = 2$$
$$x_2 - x_1 \geq d_2 = 2$$
$$x_4 - x_3 \geq d_2 = 2$$
$$x_6 - x_5 \geq d_2 = 2$$



Example 2

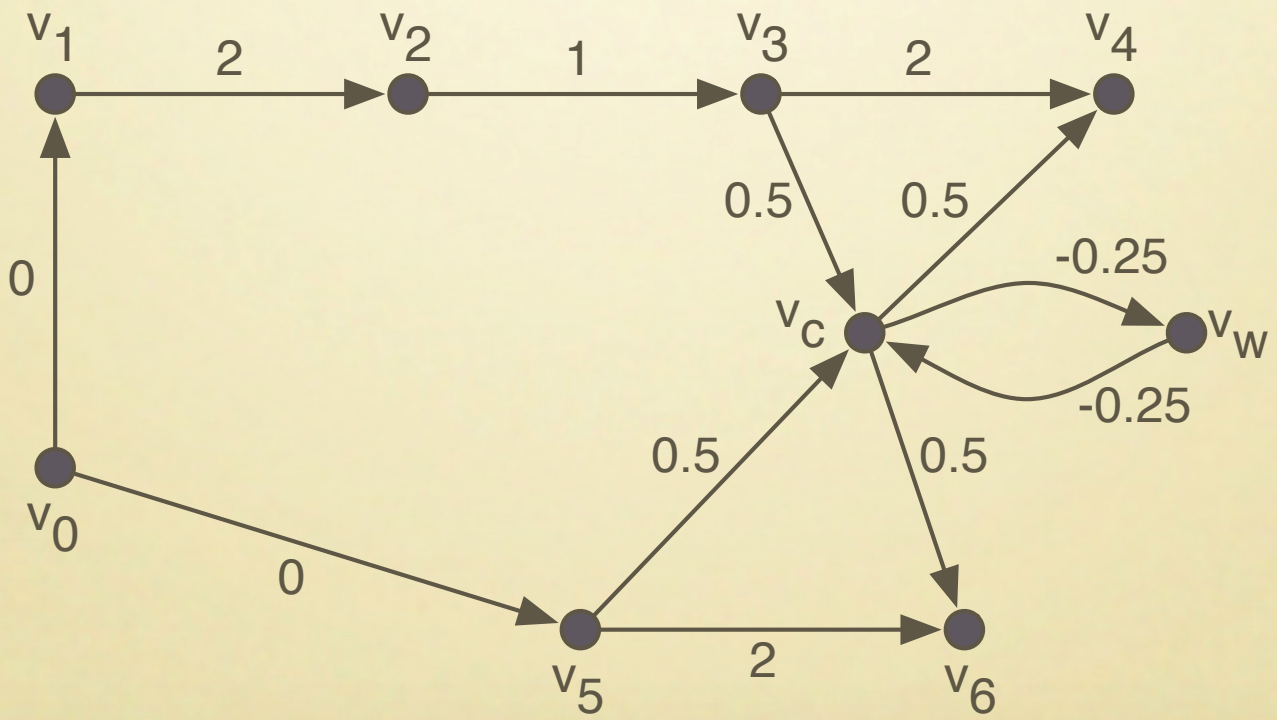


Example 2

$$x_3 - x_2 \geq d_1 = 1; \quad x_2 - x_1 \geq d_2 = 2 \quad x_4 - x_3 \geq d_2 = 2$$

$$x_6 - x_5 \geq d_2 = 2; \quad |x_w - x_c| \leq C_2 = 0.25; \quad x_c - x_3 \geq C_1 = 0.5$$

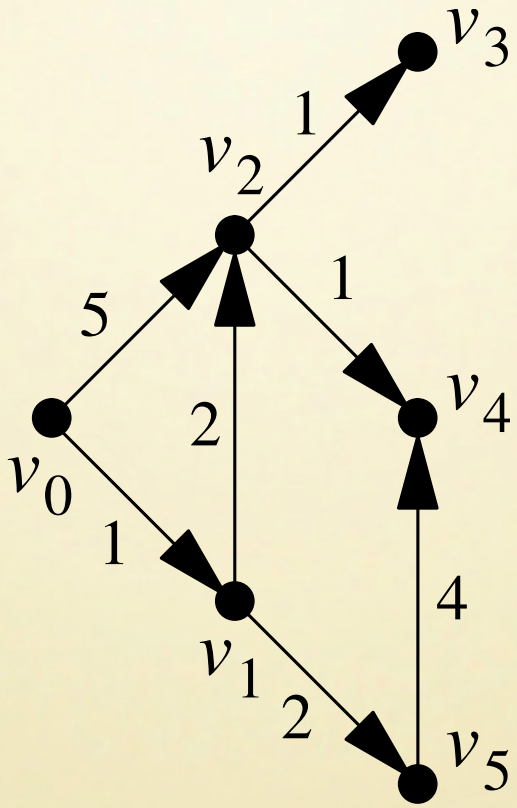
$$x_4 - x_c \geq C_1 = 0.5; \quad x_c - x_5 \geq C_1 = 0.5; \quad x_6 - x_c \geq C_1 = 0.5$$



Longest-Path Algorithm for DAGs

- Applicable only to directed acyclic graphs (DAGs)
- Set Q contains a list of all vertices v_i for which the longest distance from v_0 is known.
- Initially only $v_0 \in Q$; Gradually other vertices will be added.
- Once the vertex is "processed" it is removed from Q
- A variable p_i is associated with each vertex v_i to keep track of the vertices incident on v_i that still have to be processed

Example



LPA - Example

Q	p_1	p_2	p_3	p_4	p_5	x_1	x_2	x_3	x_4	x_5
ϕ	1	2	1	2	1	0	0	0	0	0
$\{v_0\}$	0	1	1	2	1	1	5	0	0	0
$\{v_1\}$	0	0	1	2	0	1	5	0	0	3
$\{v_2, v_5\}$	0	0	0	1	0	1	5	6	6	3
$\{v_3, v_5\}$	0	0	0	1	0	1	5	6	6	3
$\{v_5\}$	0	0	0	0	0	1	5	6	7	3
$\{v_4\}$	0	0	0	0	0	1	5	6	7	3

LPA - Pseudocode

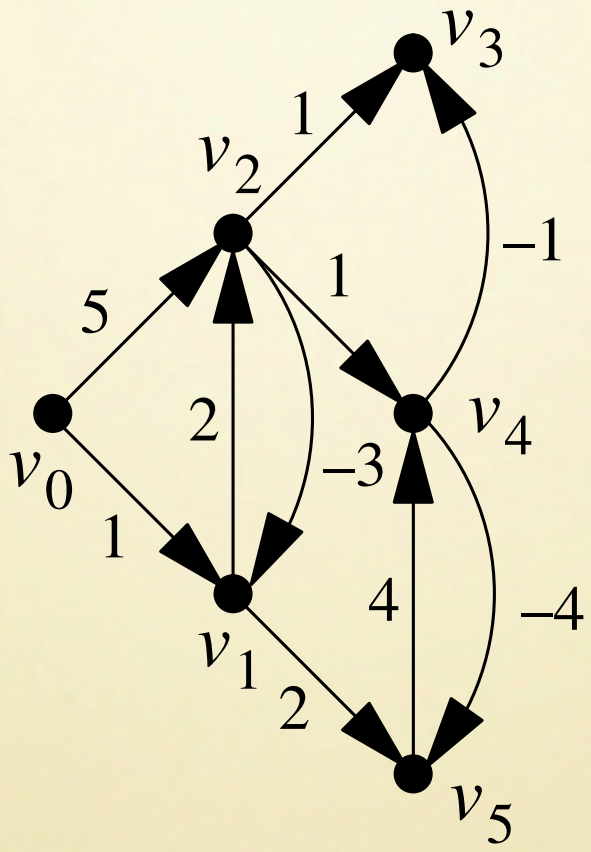
longest-path(G)
{
  for ($i \leftarrow 1; i \leq n; i \leftarrow i + 1$)
    $p_i \leftarrow$ "in-degree of v_i";
  $Q \leftarrow \{v_0\}$;
  while ($Q \neq \emptyset$) {
    $v_i \leftarrow$ "any element from Q";
    $Q \leftarrow Q \setminus \{v_i\}$;
    for each v_j "such that" $(v_i, v_j) \in E$ {
      $x_j \leftarrow \max(x_j, x_i + d_{ij})$;
      $p_j \leftarrow p_j - 1$;
      if ($p_j \leq 0$)
        $Q \leftarrow Q \cup \{v_j\}$;
    }
  }
}

main ()
{
  for ($i \leftarrow 0; i \leq n; i \leftarrow i + 1$)
    $x_i \leftarrow 0$;
  longest-path(G);
}

Directed graphs with cycles

- The previous algorithm only works for acyclic graphs
- Two cases of cyclic graphs
- Negative cycles: Sum of edges in a cycle is negative
- Positive cycles: Sum of edges in a cycle is positive
  - Finding longest path for positive cycles is NP-hard
  - But positive cycle in a layout means conflicting constraints
  - Such a layout is over-constrained
  - Detecting positive cycles is enough
- Two algorithms to calculate longest path for negative cyclic graphs
  - Liao-Wong algorithm
  - Bellman-Form algorithm

Example



Liao-Wong Algorithm

- All edges are partitioned into *forward* edges E_f and *backward* edges E_b
- E_f - Minimum inequality constraints
- E_b - Maximum inequality constraints
- Idea is to start with graph with only E_f edges
- Use the DAG longest path algorithm on it
- Add one edge from E_b, call the DAG longest path algorithm again
- Iterate until all the edges in E_b are added

Solution - LW

Step	x_1	x_2	x_3	x_4	x_5
Initialize	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
Forward 1	1	5	6	7	3
Backward 1	2	5	6	7	3
Forward 2	2	5	6	8	4
Backward 2	2	5	7	8	4
Forward 3	2	5	7	8	4
Backward 3	2	5	7	8	4

LW - Pseudocode

```
count ← 0;
for (i ← 1; i ≤ n; i ← i + 1)
  x_i ← −∞;
x_0 ← 0;

do { flag ← 0;
     longest-path(G_f);
     for each (v_i, v_j) ∈ E_b
       if (x_j < x_i + d_ij) {
         x_j ← x_i + d_ij;
         flag ← 1;
       }
     count ← count +1;
     if (count > |E_b| && flag)
       error("positive cycle")
}
while (flag);
```

Bellman-Ford Algorithm

- This does not discriminate between forward and backward edges
- This algorithm goes through several iterations until it converges to the longest paths
- First we start with a set S_1 containing the source vertex
- Update distances to all the vertices that edges of this vertex go to: add these new vertices to S_1 and remove the source vertex
- Repeat this process with the vertices present in set S_1
- Continue until convergence is attained
- If convergence does not occur in n iterations (n is the number of vertices in the graph then it indicates the presence of positive cycles

Solution - BF

S_1	x_1	x_2	x_3	x_4	x_5
"not initialized"	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$\{v_0\}$	1	5	$-\infty$	$-\infty$	$-\infty$
$\{v_1, v_2\}$	2	5	6	6	3
$\{v_1, v_3, v_4, v_5\}$	2	5	6	7	4
$\{v_4, v_5\}$	2	5	6	8	4
$\{v_4\}$	2	5	7	8	4
$\{v_3\}$	2	5	7	8	4

BF - Pseudocode

for ($i \leftarrow 1; i \leq n; i \leftarrow i + 1$)
  $x_i \leftarrow -\infty$;
$x_0 \leftarrow 0$;
count $\leftarrow 0$;
$S_1 \leftarrow \{v_0\}$;
$S_2 \leftarrow \emptyset$;
while (count $\leq n$ && $S_1 \neq \emptyset$) {
  for each $v_i \in S_1$
    for each v_j "such that" $(v_i, v_j) \in E$
      if ($x_j < x_i + d_{ij}$) {
        $x_j \leftarrow x_i + d_{ij}$;
        $S_2 \leftarrow S_2 \cup \{v_j\}$
      }
  $S_1 \leftarrow S_2$;
  $S_2 \leftarrow \emptyset$;
  count $\leftarrow$ count + 1;
}
if (count $> n$)
  error("positive cycle");