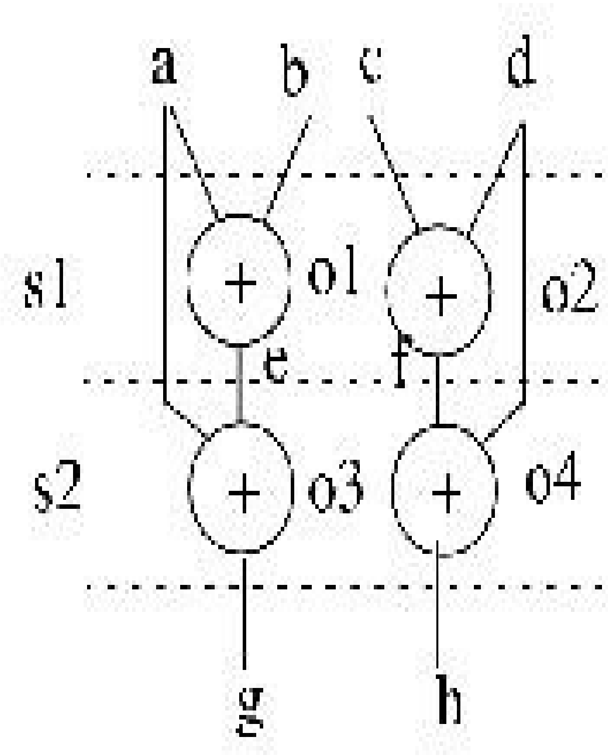
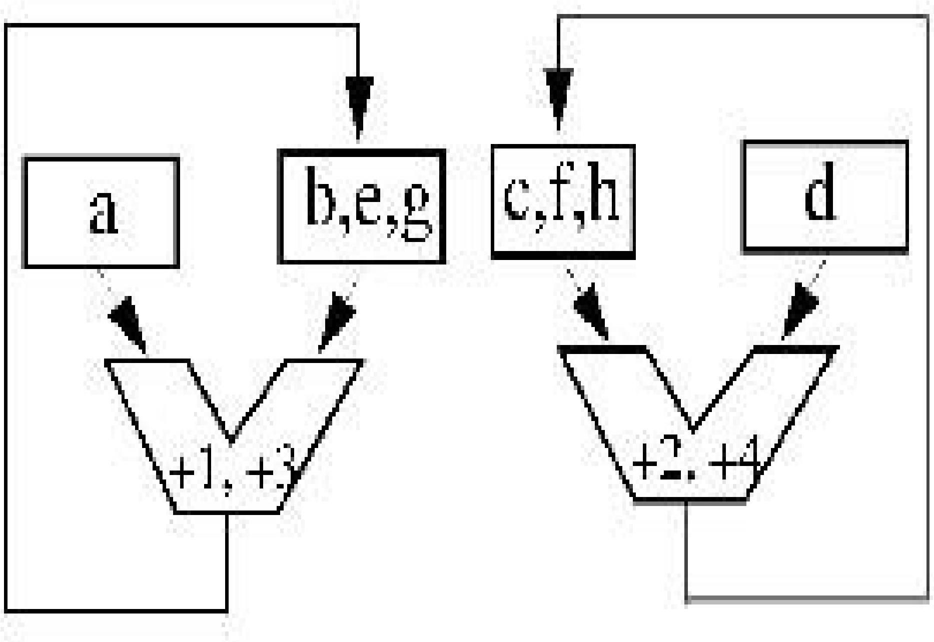
Resource sharing

Allocation and Binding

- Allocation (unit selection) – Determination of the type and number of resources required:
  - Number and types functional units
  - Number and types of storage elements
  - Number and types of busses
- Binding – Assignment to resource instances:
  - Operations to functional unit instances
  - Values to be stored to instances of storage elements
  - Data transfers to bus instances

Allocating and Binding (2)





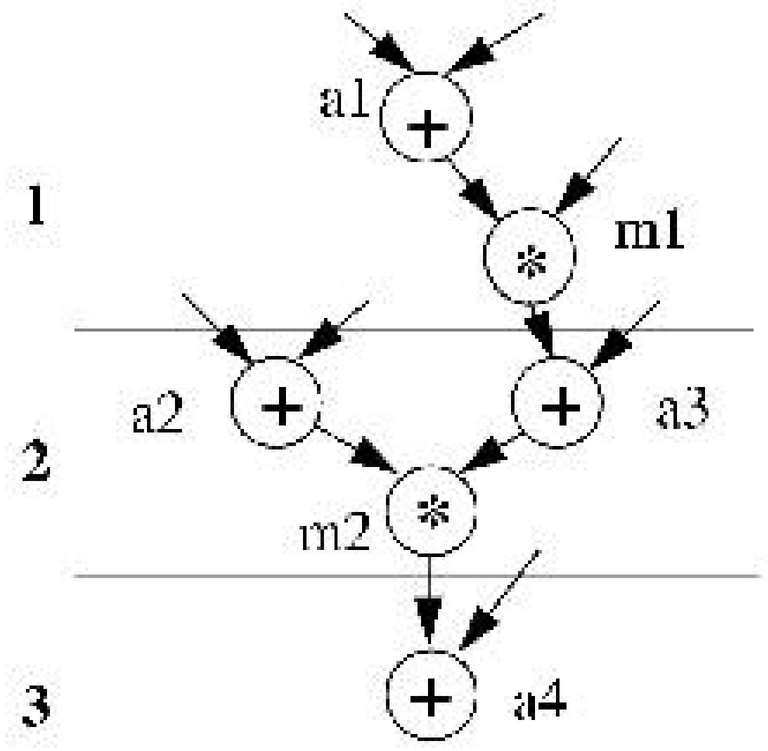
Allocating and Binding (3)

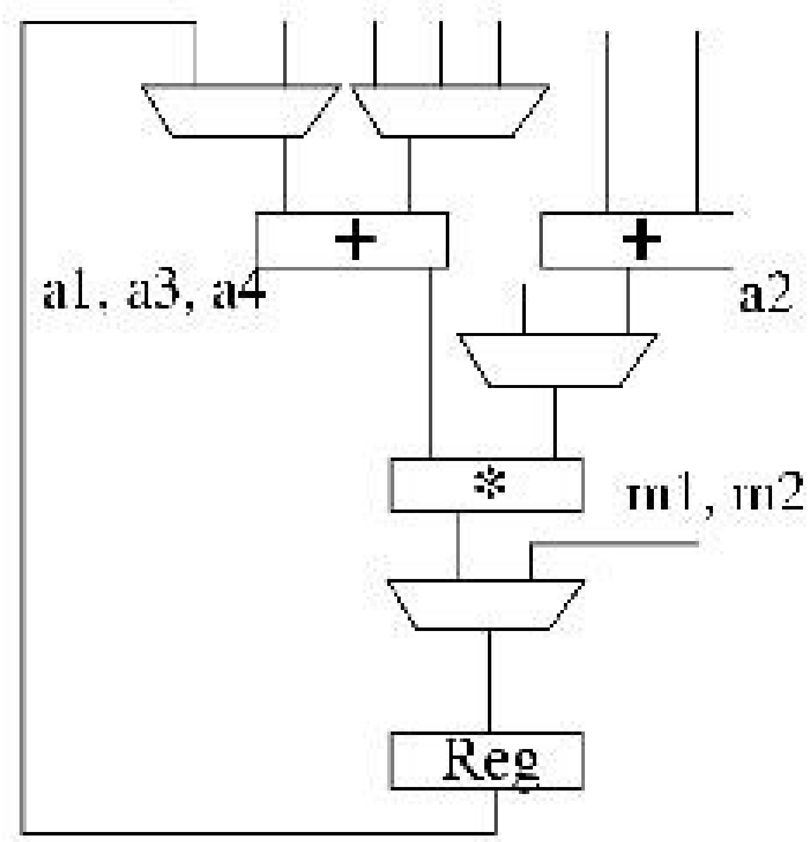
- Optimization goal
  - Minimize total cost of functional units, register, bus driver, and multiplexer
  - Minimize total interconnection length
  - Constraint on critical path delay

Approaches to Allocating/Binding

- Constructive – start with an empty data path and add functional, storage and interconnects as necessary.
  - Greedy algorithms – perform allocation for one control step at a time.
  - Rule-based used to select type and numbers of function units, especially prior to scheduling.

Approaches to Allocating/Binding (2)





Approaches to Allocation/Binding (3)

- Graph-theoretical formulations – sub-tasks are mapped into well-defined problems in graph theory.
  - Clique partitioning.
  - Left-edge algorithm.
  - Graph coloring.

Allocation and binding

- Allocation:
  - Number of resources available
- Binding:
  - Relation between operations and resources
- Sharing:
  - Many-to-one relation
- Optimum binding/sharing:
  - Minimize the resource usage

Optimum sharing problem

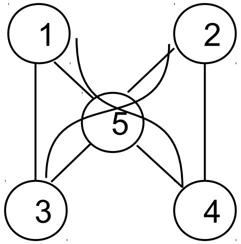
- Scheduled sequencing graphs
  - Operation concurrency well defined
- Consider *operation types* independently
  - Problem decomposition
  - Perform analysis for each resource type

Compatibly and conflicts

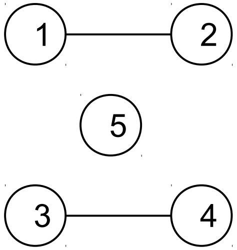
t1	x=a+b y=c+d	1 2
t2	s=x+y t=x-y	3 4
t3	z=a+t	5

- Operation compatibility:
  - Same type
  - Non concurrent
- *Compatibility* graph:
  - Vertices: operations
  - Edges: compatibility relation
- *Conflict* graph:
  - Complement of compatibility graph

Compatibility graph



Conflict graph



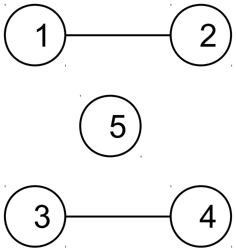
Compatibility and conflicts

- Compatibility graph:
  - Partition the graph into a minimum number of cliques
  - Find clique cover number $k (G_+)$
- Conflict graph:
  - Color the vertices by a minimum number of colors.
  - Find the chromatic number $x (G_-)$
- NP-complete problems:
  - Heuristic algorithms

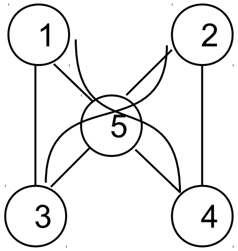
Example

t1	x=a+b	y=c+d	1	2
t2	s=x+y	t=x-y	3	4
t3	z=a+t		5	

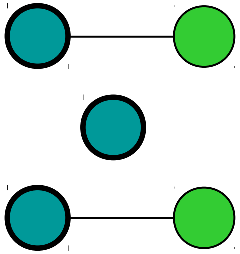
Conflict



Compatibility

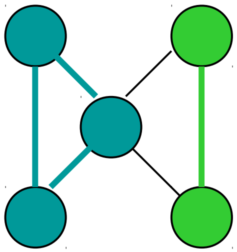


Coloring



ALU1: 1,3,5
ALU2: 2,4

Partitioning



Graph coloring

Graph coloring is a special case of graph labeling; it is an assignment of labels traditionally called "colors" to elements of a graph subject to certain constraints.

- In its simplest form, it is a way of coloring the vertices of a graph such that no two adjacent vertices share the same color; this is called a vertex coloring.
- Similarly, an edge coloring assigns a color to each edge so that no two adjacent edges share the same color. In general, a graph G is k colorable if each vertex can be assigned one of k colors so that adjacent vertices get different colors. The smallest sufficient number of colors is called the chromatic number of G.

Perfect graphs

- *Comparability graph*:
  - Graph $G\ (V, E\ )$ has an orientation $G\ (\ V, F\ )$ with the transitive property

$$(v_i,\ v_j) \in F \quad \text{and} \quad (v_j,\ v_k) \in F \ \rightarrow\ (v_i,\ v_k) \in F$$

A comparability graph is an undirected graph that connects pairs of elements that are comparable to each other in a partial order.

Perfect graphs

Interval graph:

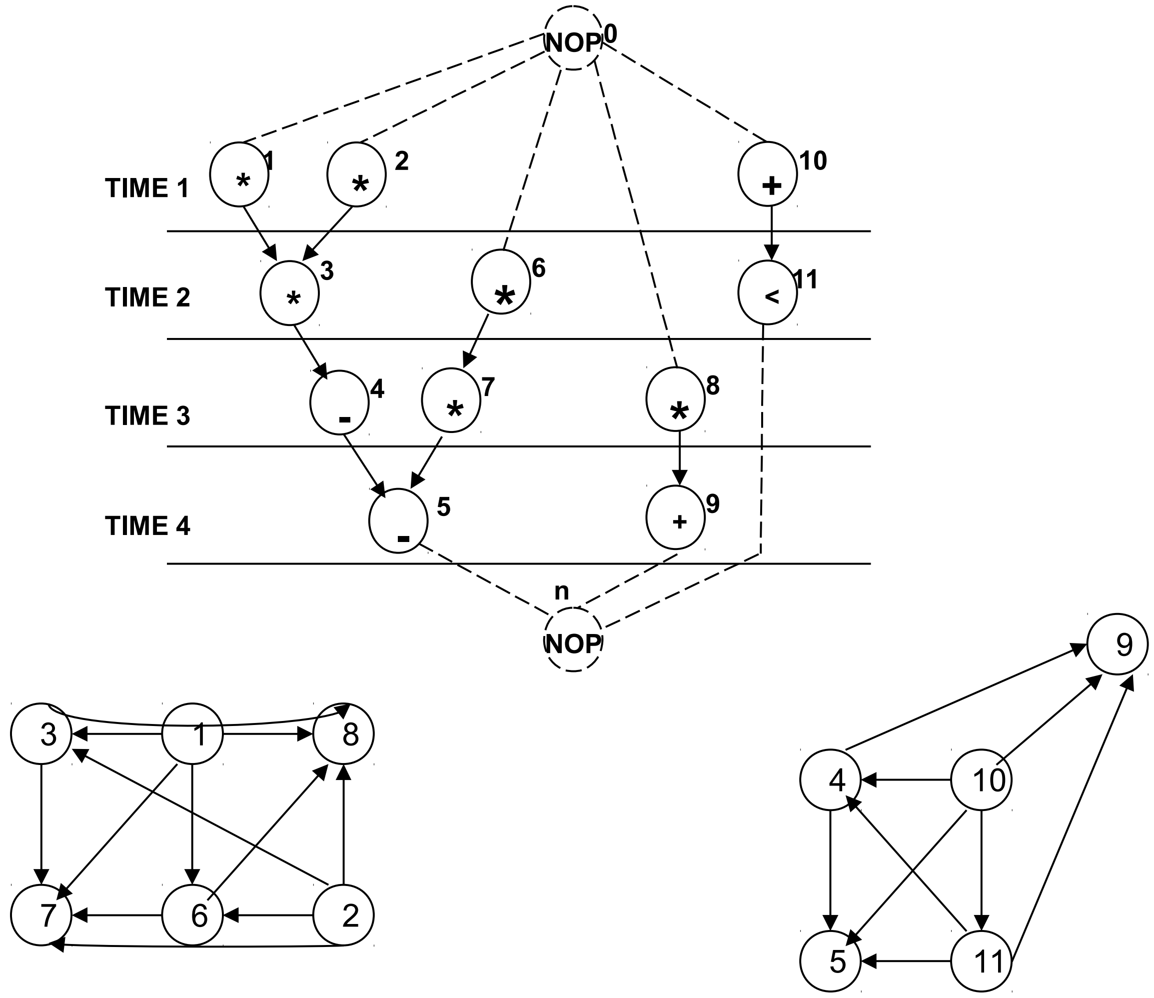
- Vertices correspond to *intervals*
- Edges correspond to interval intersection
- Subset of *chordal* graphs
  - a graph is chordal if each of its cycles of four or more vertices has a chord, which is an edge that is not part of the cycle but connects two vertices of the cycle.

An interval graph is the intersection graph of a multiset of intervals on the real line. It has one vertex for each interval in the set, and an edge between every pair of vertices corresponding to intervals that intersect.

Data-flow graphs
(flat sequencing graphs)

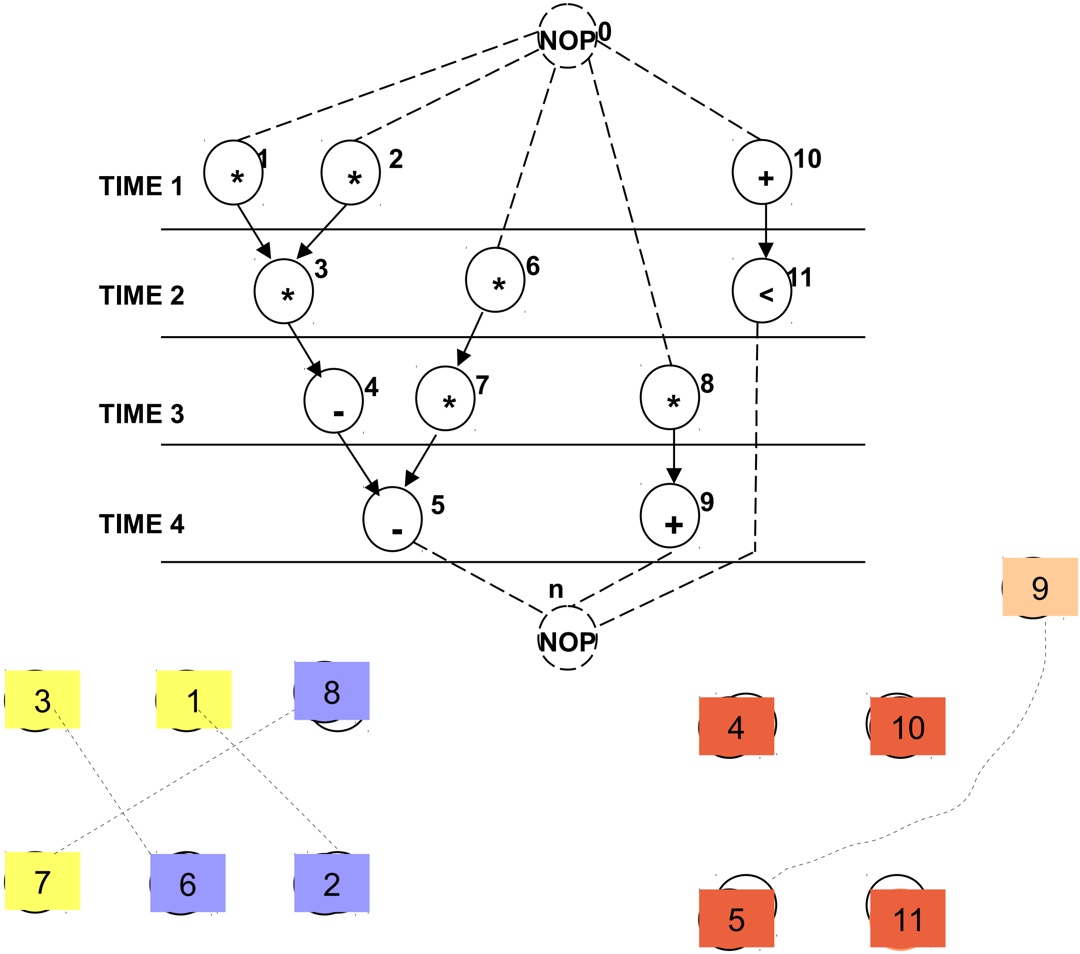
- The compatibility/conflict graphs have special properties:
  - Compatibility
    - Comparability graph
  - Conflict
    - Interval graph
- Polynomial time solutions:
  - Left-edge algorithm

Example 6.2.1



Compatibility graph per resource type

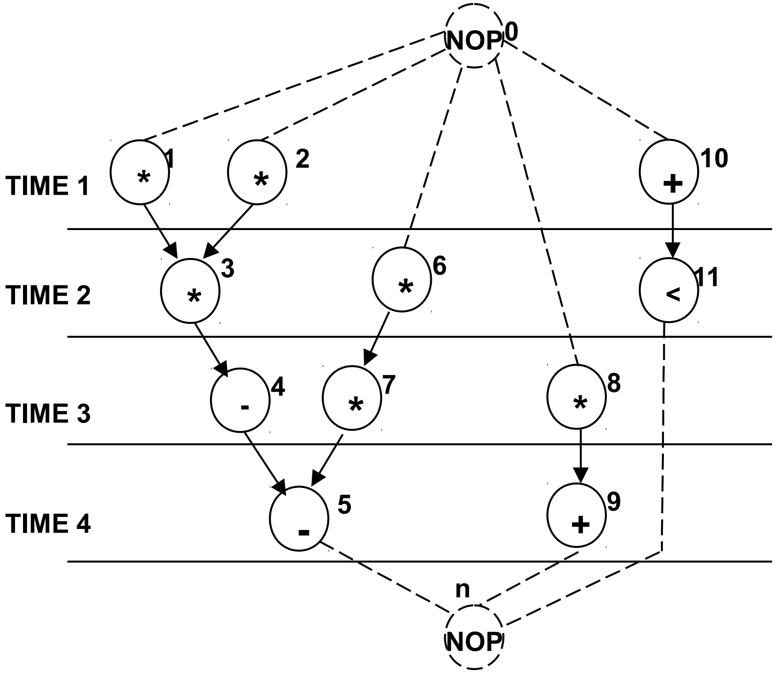
Example 6.2.1b

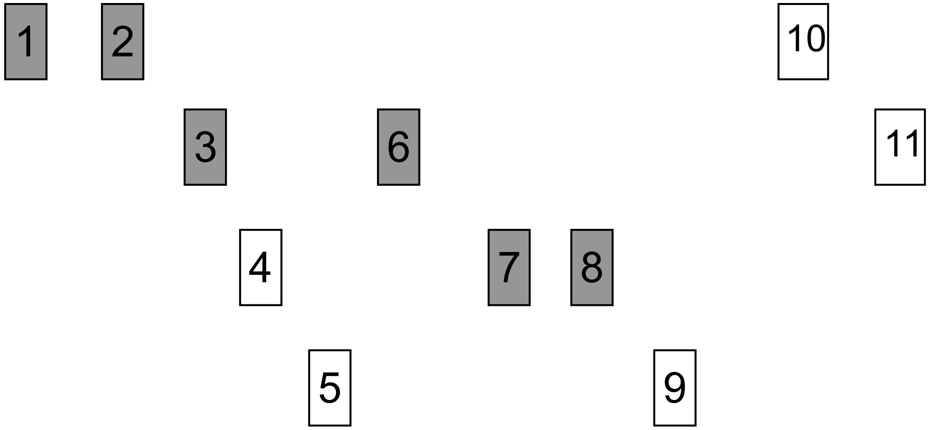


Conflict graph

Example 6.2.4

The set of intervals corresponding to the conflict graphs





Overlapping intervals correspond to edges
In the conflict graph for each type.

MULT = v1,v2 v3,v6 v7,v8
ALU = v5,v9

Left-edge algorithm

- Input:
  - Set of intervals with *left* and *right edge*
  - A set of *colors* (initially one color)
- Rationale:
  - Sort intervals in a *list* by *left* edge
  - Assign non overlapping intervals to first color using the list
  - When possible intervals are exhausted, increase color counter and repeat

ILP formulation of binding

- Boolean variable b_{ir}
  - Operation *i* bound to resource *r*
- Boolean variables x_{il}
  - Operation *i* scheduled to start at step *l*

$$\sum_r b_{ir} = 1 \qquad \text{for all operations } i$$

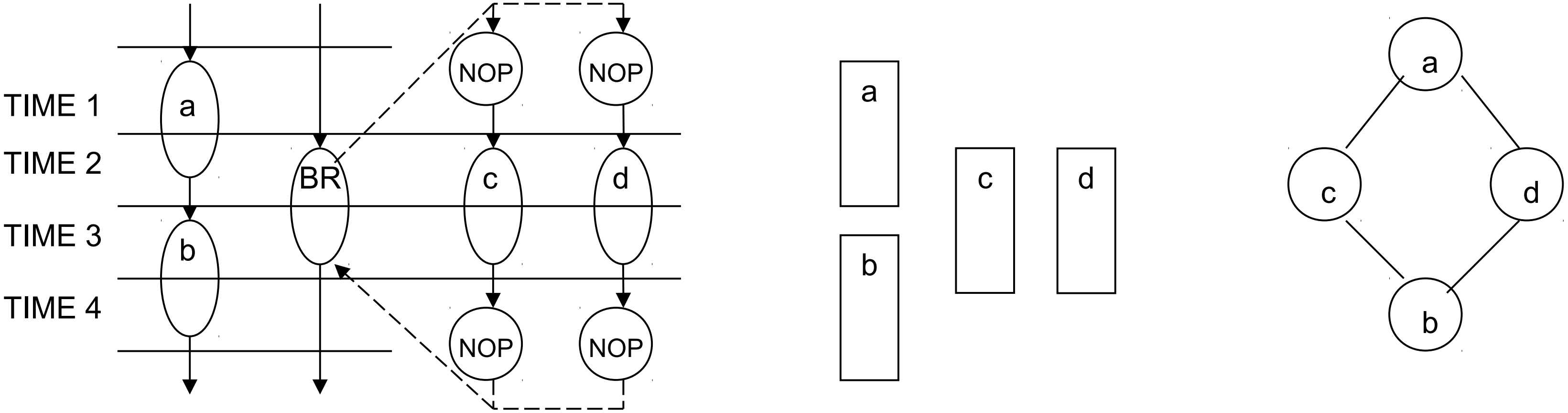
$$\sum_i b_{ir} \sum_{m=l-d_i+1..l} x_{im} \leq 1 \qquad \text{for all steps } l \text{ and resources } r$$

Hierarchical sequencing graphs

- Hierarchical conflict/compatibility graphs:
    - Easy to compute
    - Prevent sharing across hierarchy
- Flatten hierarchy:
    - Bigger graphs
    - Destroy nice properties

Example 6.2.8

Conditional execution. Sequencing graph, execution intervals, Non chordal (!) conflict graph.



(a) (b) (c)

a graph is chordal if each of its cycles of four or more nodes has a chord

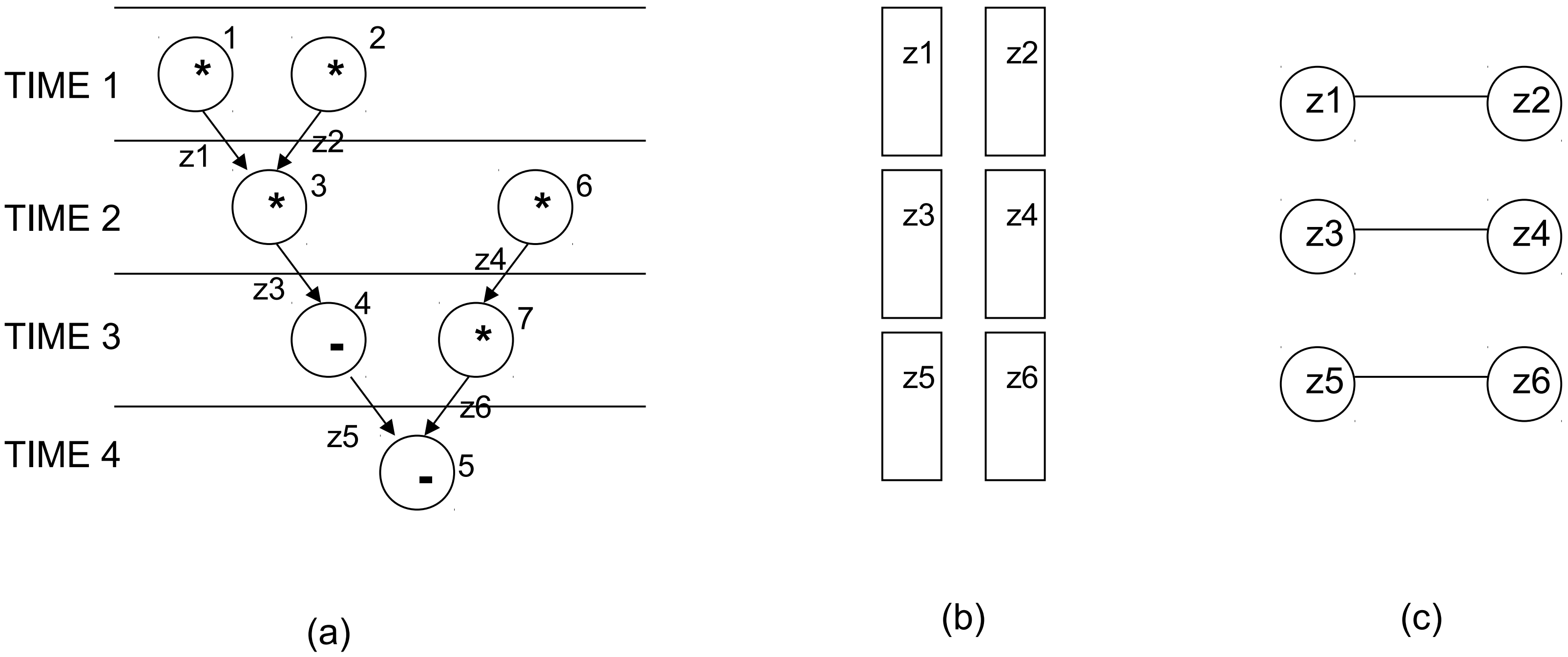
Register binding problem

- Given a schedule:
  - *Lifetime intervals* for variables
  - *Lifetime overlaps*
- Conflict graph (interval graph):
  - Vertices (or nodes) ↔ variables
  - Edges (or links) ↔ overlaps
  - Interval graph
- Compatibility graph (comparability graph):
  - Complement of conflict graph

Register sharing in data-flow graphs

- Given:
  - Variable lifetime conflict graph
- Find:
  - Minimum number of registers storing all the variables
- Key point:
  - Interval graph
    - Left-edge algorithm (polynomial-time complexity)

Example 6.2.9

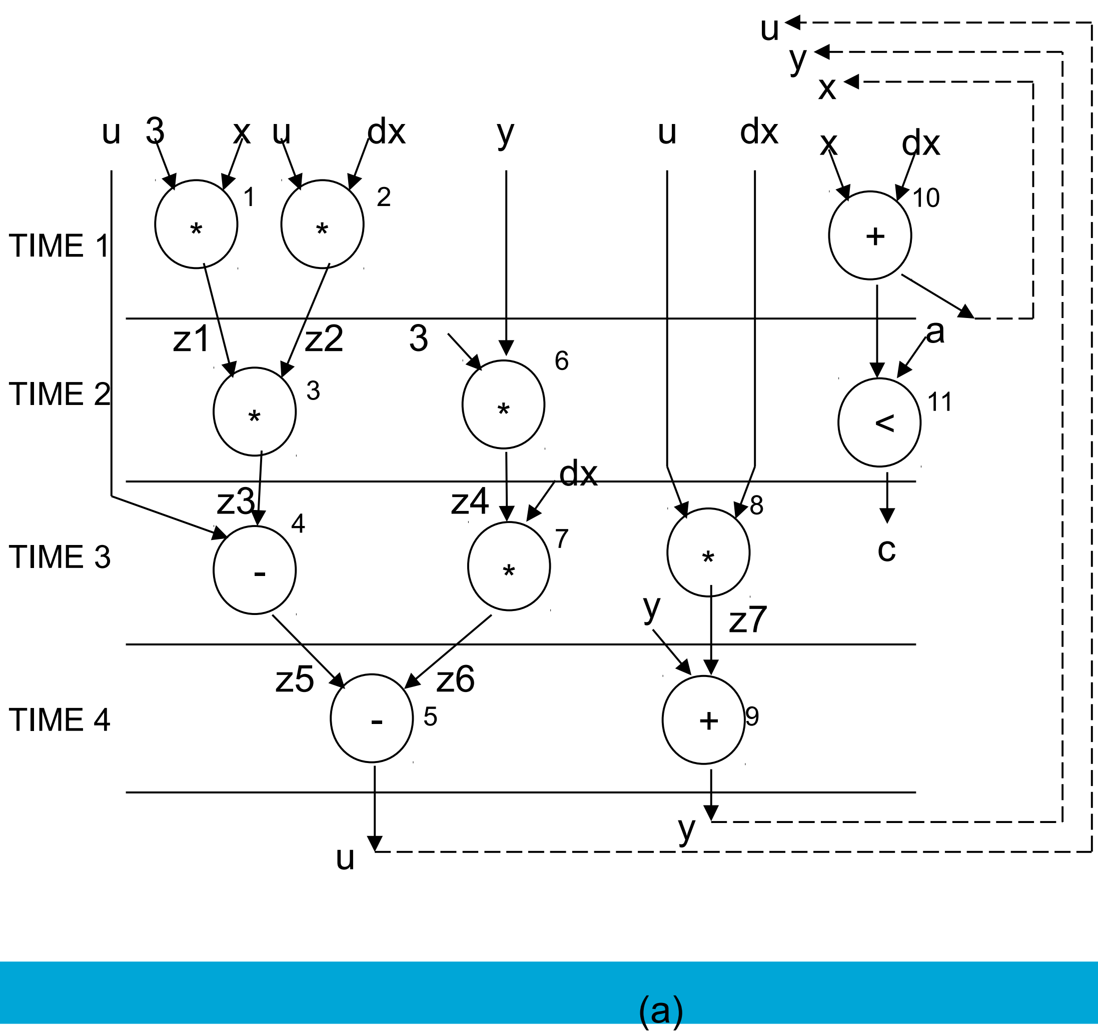


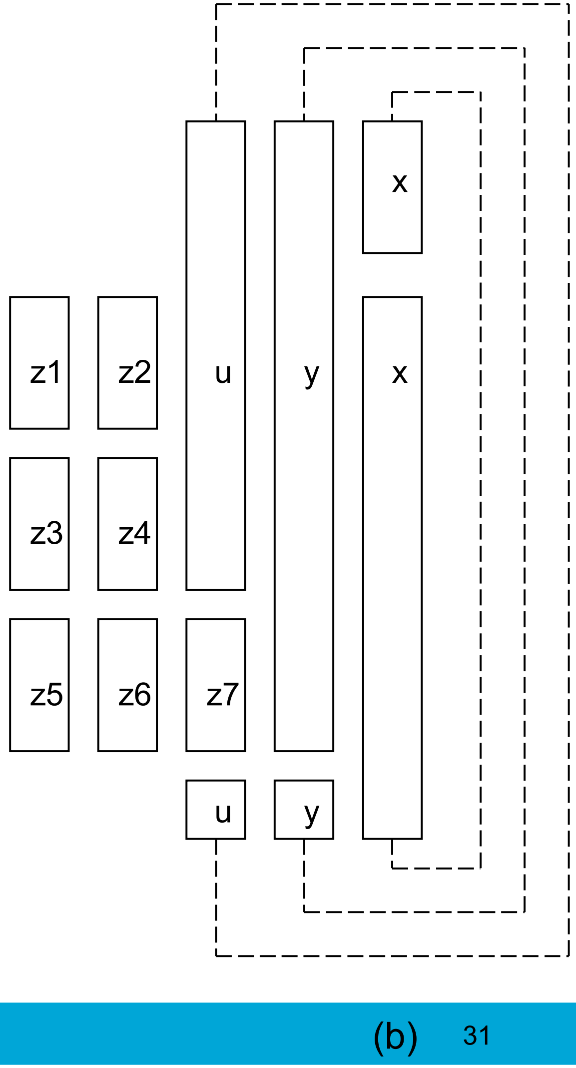
Sharing, conflict graph

Register sharing general case

- Iterative conflicts:
  - Preserve values across iterations
  - Circular-arc conflict graph
    - Coloring is intractable
- Hierarchical graphs:
  - General conflict graphs
    - Coloring is intractable
- Heuristic algorithms

Example 6.2.10





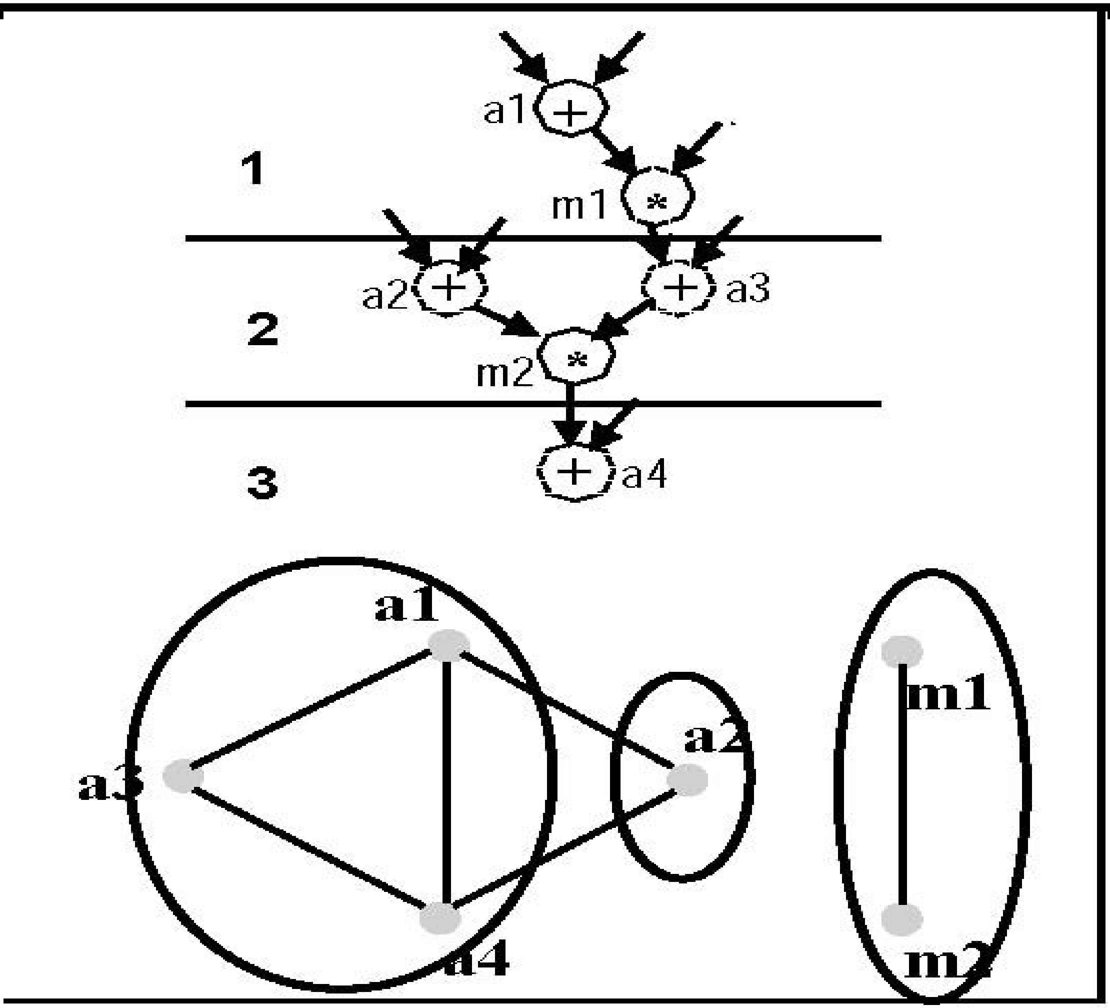
Clique Partitioning

- Let G = (V, E) be an undirected graph with a set V of vertices and a set E of edges.
- A clique is a set of vertices that form a *complete sub graph* of G.
- The problem of partitioning a graph into a minimal number of cliques such that each vertex belongs to exactly one clique is called *clique partitioning*.

Clique Partitioning (2)

- Formulation of *functional unit* allocation as a clique partitioning problem:
  - Each vertex represents an operation.
  - An edge connects two vertices iff:
    - 1. The two operations are scheduled into different control steps, and
    - 2. There exists a functional unit that is capable of carrying out both operations.

A Clique



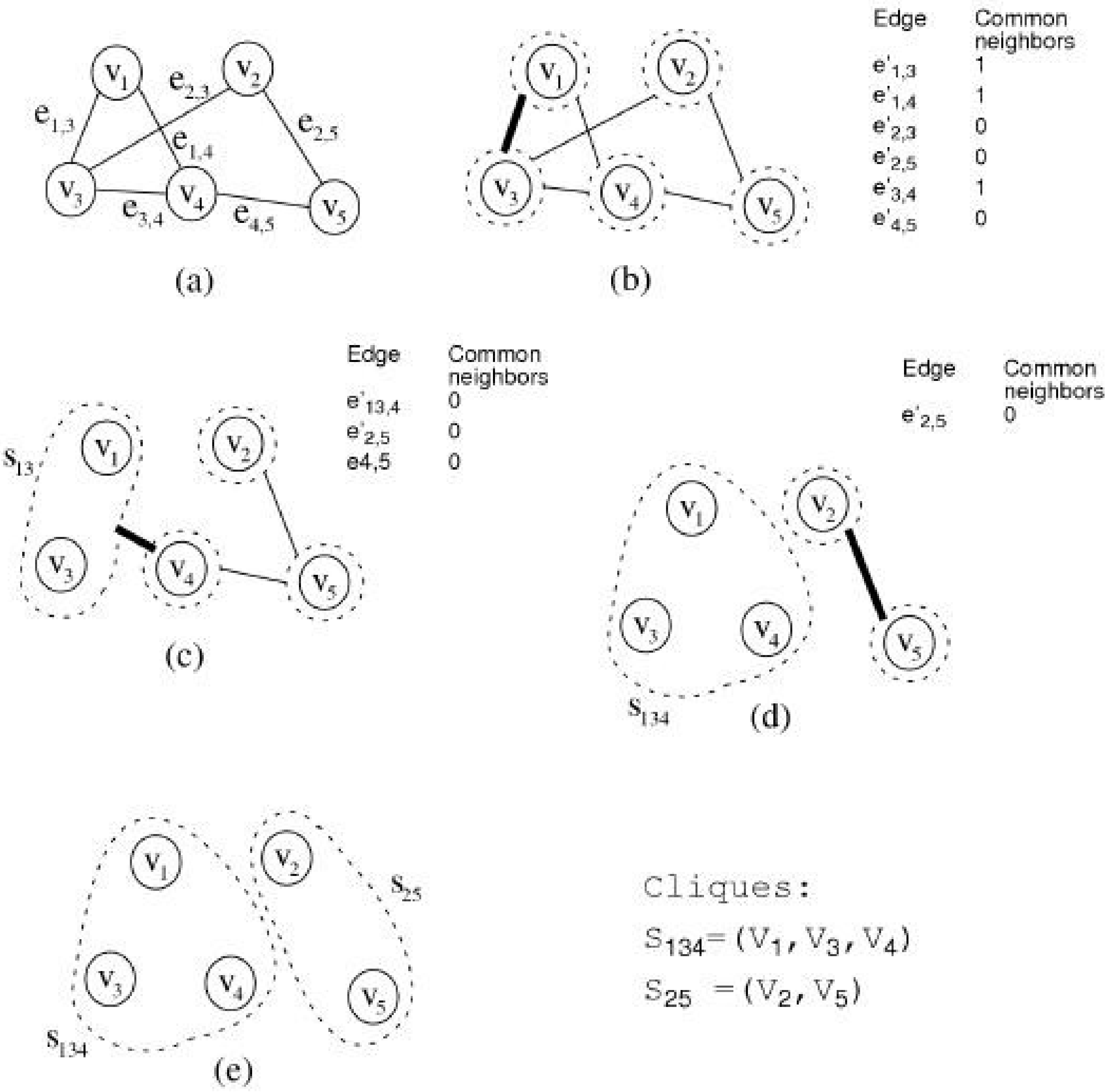
Clique Partitioning (Cont'd)

- Formulation of *storage allocation* as a clique partitioning problem:
  - Each value needed to be stored is mapped to a vertex.
  - Two verticals are connected iff life-time of the two values do not intersect.
- The clique partitioning problem is NP-complete.
- Efficient heuristics have been developed: e.g., Tseng used a polynomial time algorithm which generates very good results.

Tseng’s Algorithm

- A super-graph is derived from the original graph.
- Find two connected super-nodes such that they have the maximum number of common neighbors.
- Merge the two nodes and repeated from the first step, until no more merger can be carried out.

Tse



(a) (b) (c) (d) (e)

Edge	Common neighbors
$e'_{1,3}$	1
$e'_{1,4}$	1
$e'_{2,3}$	0
$e'_{2,5}$	0
$e'_{3,4}$	1
$e'_{4,5}$	0

Edge	Common neighbors
$e'_{13,4}$	0
$e'_{2,5}$	0
e4,5	0

Edge	Common neighbors
$e'_{2,5}$	0

Cliques:

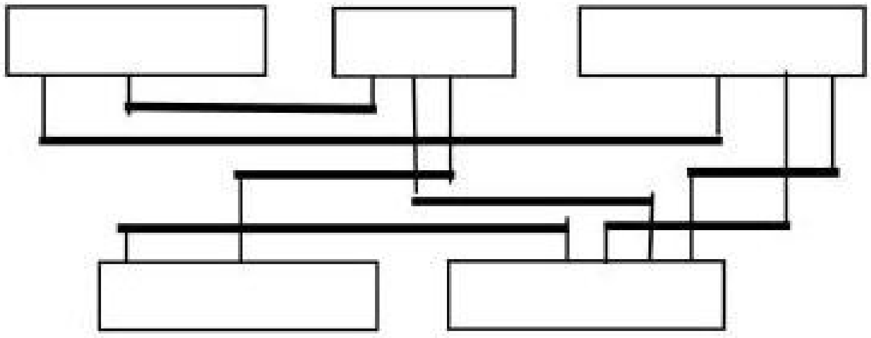
$S_{134} = (V_1, V_3, V_4)$

$S_{25} = (V_2, V_5)$

t

Left-Edge (LE) Algorithm

- The LE algorithm is used in channel routing to minimize the number of tracks used to connect points.



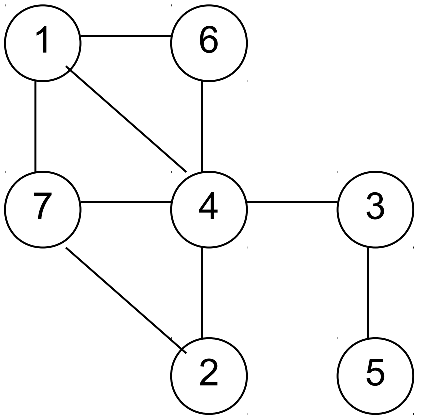
Left-edge algorithm

- Input:
    - Set of intervals with *left* and *right edge*
    - A set of *colors* (initially one color)
- Rationale:
    - Sort intervals in a *list* by *left* edge
    - Assign non overlapping intervals to first color using the list
    - When possible intervals are exhausted, increase color counter and repeat

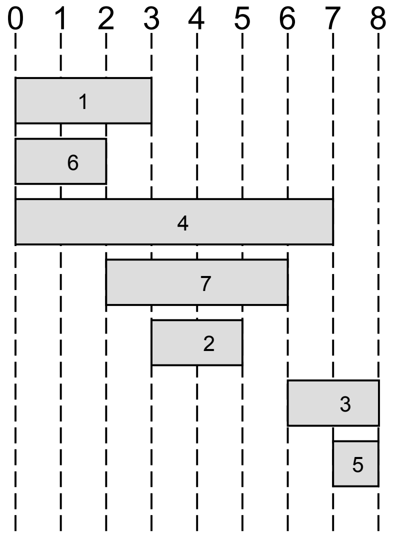
Left-edge algorithm

```
LEFT_EDGE($I$) {
    Sort elements of $I$ in a list $L$ in ascending order of $l_i$;
    $c = 0$;
    while (some interval has not been colored) do {
        $S = \emptyset$;
        $r = 0$;
        while (  exists $s \in L$ such that $l_s > r$) do {
            $s$ = First element in the list $L$ with $l_s > r$;
            $S = S \cup \{s\}$;
            $r = r_s$;
            Delete $s$ from $L$;
        }
        $c = c + 1$;
        Label elements of $S$ with color $c$;
    }
}
```

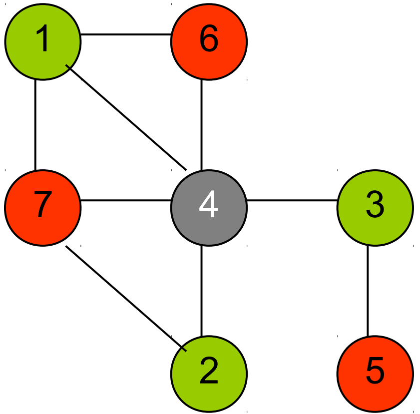
Example

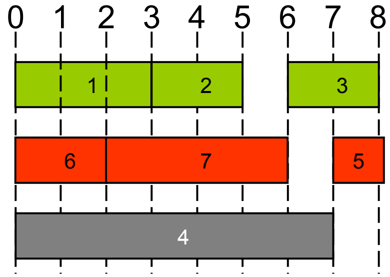


Conflict graph



Intervals



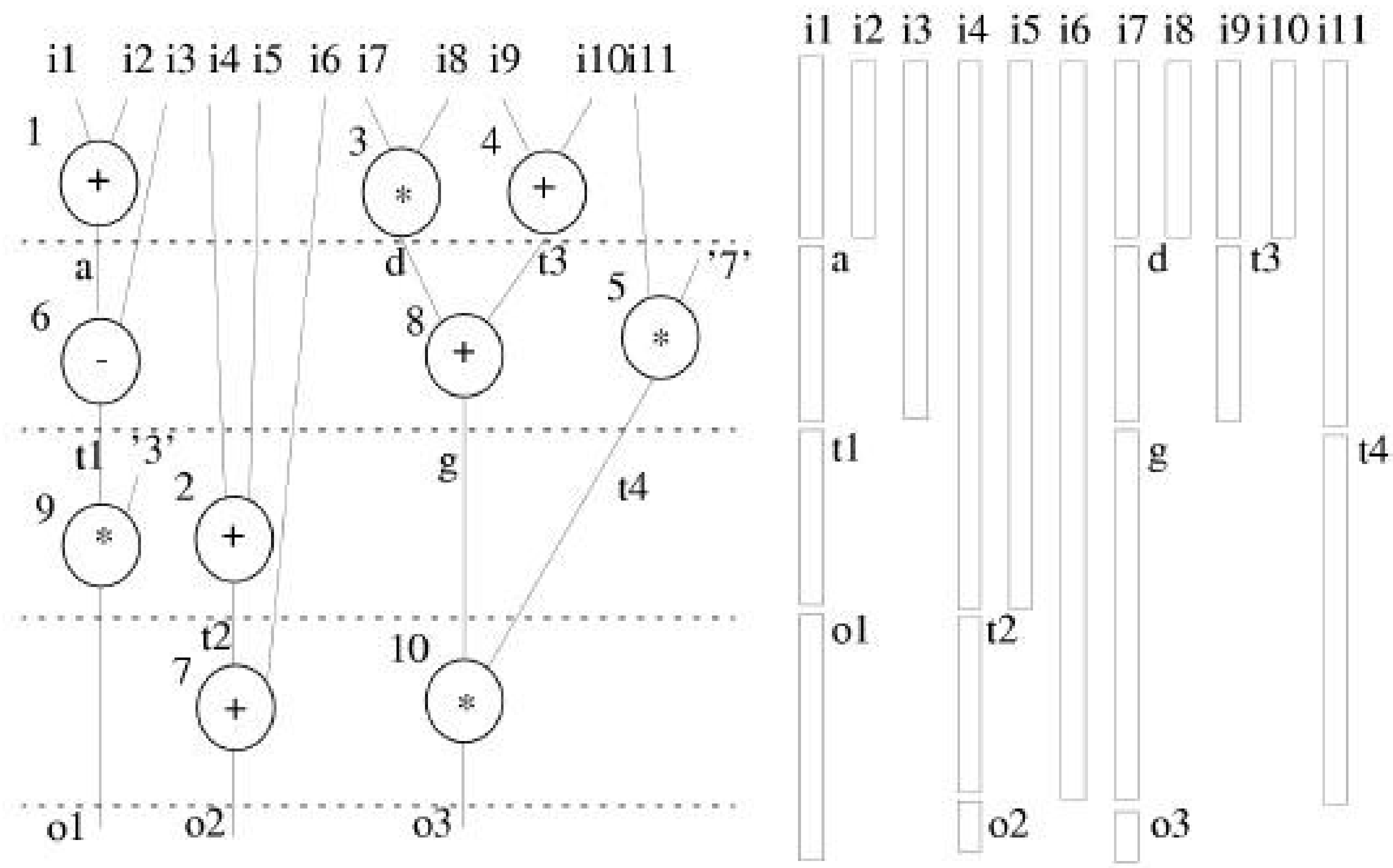


Colored conflict graph

Left-Edge (LE) Algorithm (2)

- The register allocation problem can be solved by the LE algorithm by mapping the birth time of a value to the left edge, and the death time of a value to the right edge of a wire.

Variable Life Times

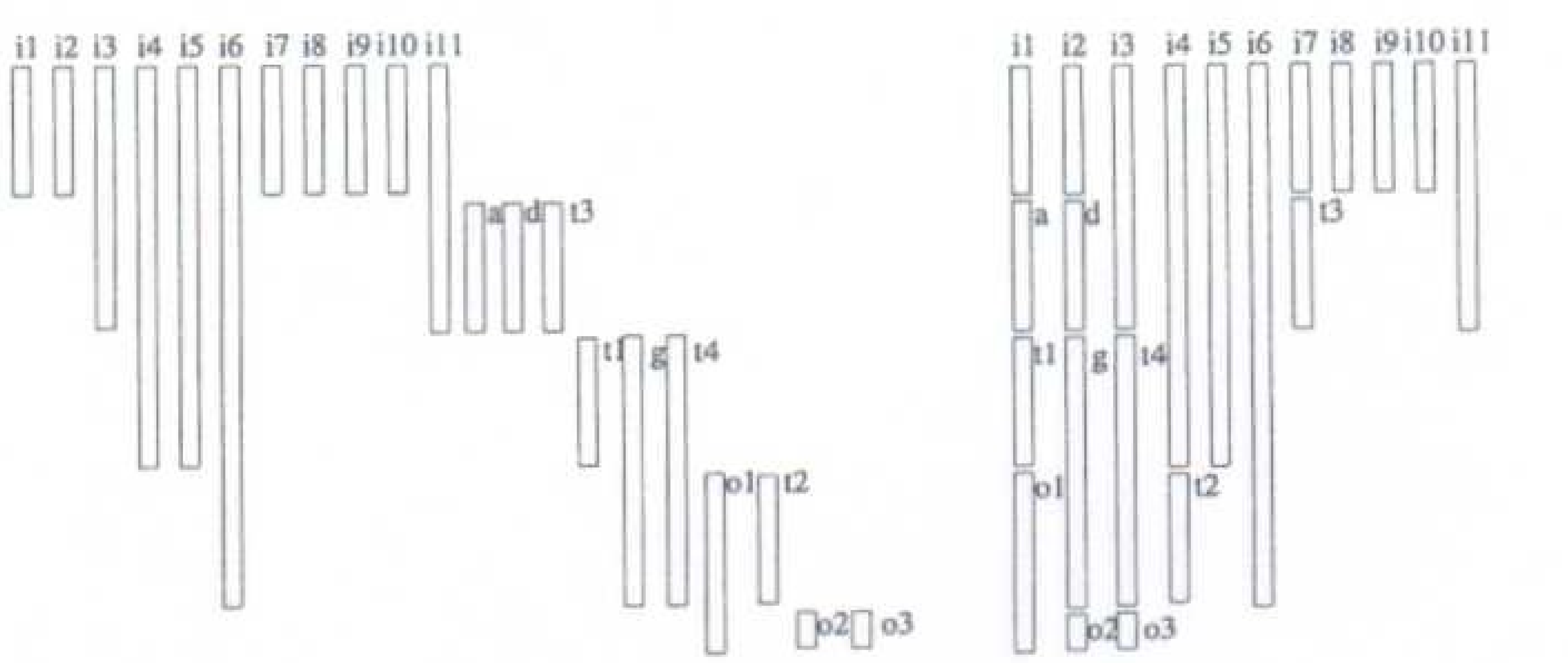


Variable life-times

Left-Edge (LE) Algorithm (Cont'd)

- The algorithm works as follows:
  - The values are sorted in increasing order of their birth time.
  - The first value is assigned to the first register.
  - The list is then scanned for the next value whose birth time is larger than or equal to the death time of the previous value.
  - This value is assigned to the current register.
  - The list is scanned until no more value can shared the same register. A new register will then be introduced.

Left-Edge (LE) Algorithm (Cont'd)



(a) The sorted list of variables

(b) Assignment of variables into registers

Figure 3.14: Applying the left-edge algorithm for register allocation

Left-Edge (LE) Algorithm (Cont'd)

- The algorithm quarantines to allocate the minimum number of registers, but has two disadvantages:
  - Not all life-time table might be interpreted as intersecting intervals on a line.
    - Loop
    - Conditional branches
  - The assignment is neither unique nor necessarily optimal (in terms of minimal number of multiplexers, for example).

Summary

- Resource sharing is reducible to vertex coloring or to clique covering:
  - Simple for flat graphs
  - Intractable, but still easy in practice, for other graphs
  - Resource sharing has several extensions:
    - Module selection
- Data path design and control synthesis are conceptually simple but still important steps
  - Generated data path is an interconnection of blocks
  - Control is one or more finite-state machines

TSENG's Algorithm

```
TSENG( G+(V. E. W) I {
 while (E # 0) do {
   lw = max w;   /* largest edge weight */
   E' = {{u,. u,) t E such that w,, = I w ) :
   G'+(V'. E'. W'] = subgraph of G+(V. E. W) induced by E':
   while (E' #0) do (
     Select I",. q) t E' such that ir, and t j have the most neighbors in common:
     C = (",.Uj):
     Delete edges ( s r, I if ( u i . L?) -$ E' VCI :I V':
     Delete venen ~j from V';
while (one venex adjacent td v, in G'+(V', E'. W')) do I
Select t8k such that ( v , , uk) E E' and r., and L S h~a ve the
mosr neighbors in common;
C = C u { " i ) ;
Delete edges {>'I. u,) if ("1. ui) E' Vu, t V';
Delete venex uk fmm V';
1
Save clique C in the clique list;
I
Delete the vertices in the clique list from V;
I
I
ALGORITHM 6.3.1
```