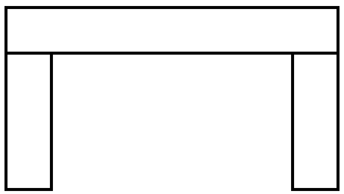
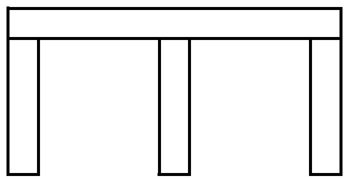
Trade-offs ~~ Optimizations

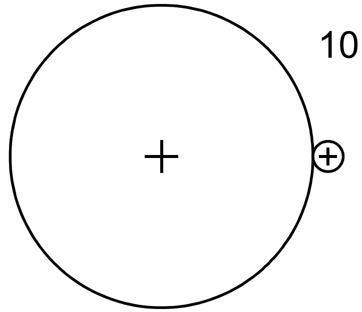
constructions



vs.

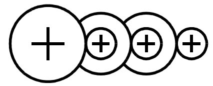


mechanics



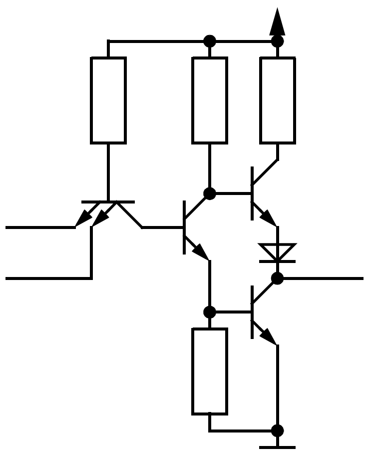
10:1 (200:20)

vs.



10:4:2:1
2.5:1, 2:1, 2:1
(50:20, 40:20, 40:20)

electronics



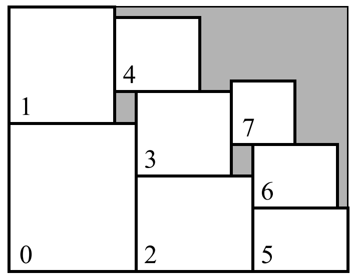
currents, voltages, etc.
of transistors -->
frequency & power

economics

logistics
travelling salesman

scheduling

storage use



Optimization

- An *optimization problem* is problem whose solution can be measured in terms of *cost* (or *objective*) function and such that the cost function attains a maximum or minimum value
- An *algorithm* is a computational procedure that has a set of *inputs* and *outputs*, has a *finite* number of unambiguously defined steps and terminates in a finite number of steps
- An *exact* algorithm always provides the exact solution
- *Approximation* algorithms (*heuristics*) are not guaranteed to find the exact solution in all cases but can provide good approximations
  - local vs. global optima (minimal vs. minimum, maximal vs. maximum)
  - practicality (speed) vs. optimality

Optimization Algorithms

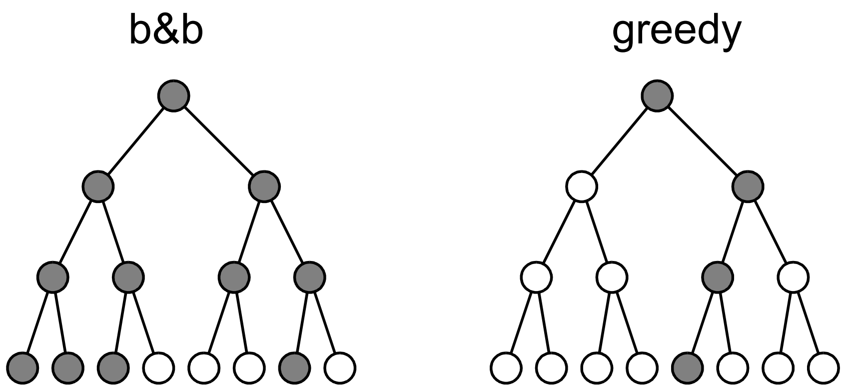
- Fundamental algorithms
  - Linear Program (LP) Integer Linear Program (ILP) Zero-One LP (ZOLP)
  - Branch-and-Bound algorithm
  - Dynamic Programming
  - Greedy algorithm
- Constructive & iterative algorithms
- Hard-computing algorithms
- Soft-computing algorithms
  - simulated annealing, self-organizing maps, etc.

Complexity of Optimization Algorithms

- **Complexity - O(n), O(n^2), O(2^n), etc.**
  - **polynomial complexity - *P***
  - **non-polynomial complexity - *NP***
    can be checked in polynomial time if the answer can be guessed
  - ***P*⊆*NP* or *P*=*NP* is still unsolved!**

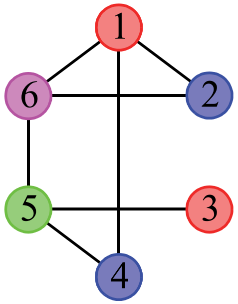
- **Branch and bound method**

- **Greedy method**

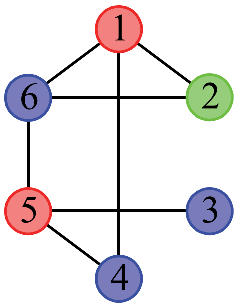


Algorithm – Greedy vs. Branch-and-Bound?

Greedy

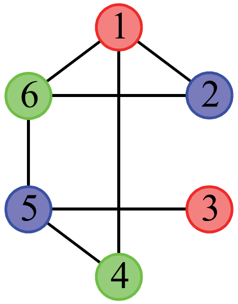


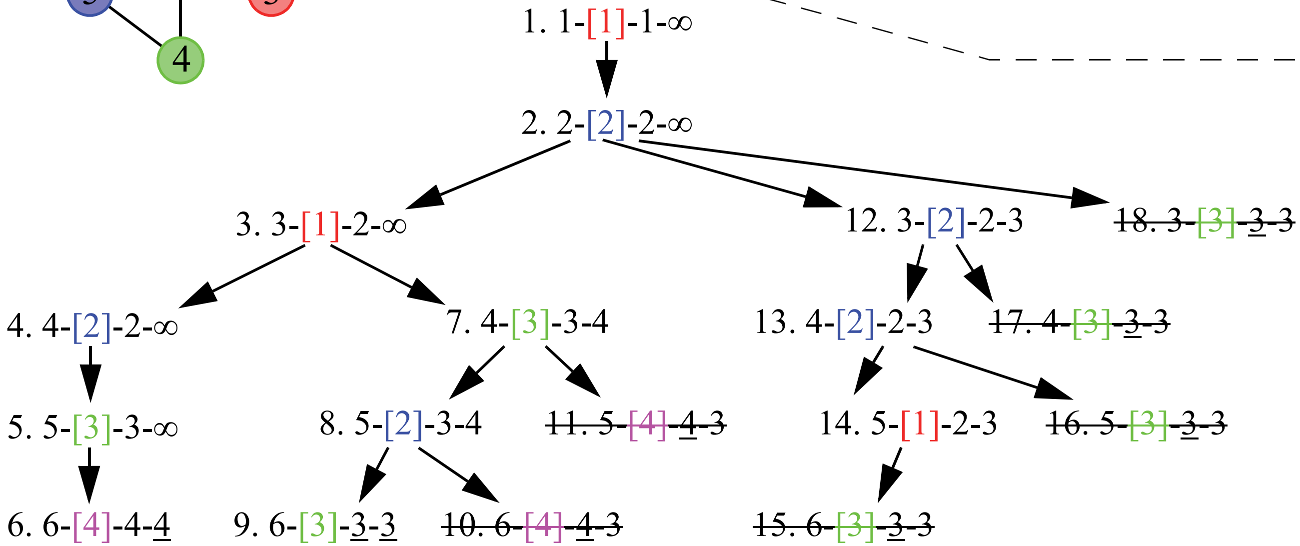
1. 1-[1]-1
2. 2-[2]-2
3. 3-[1]-2
4. 4-[2]-2
5. 5-[3]-3
6. 6-[4]-4



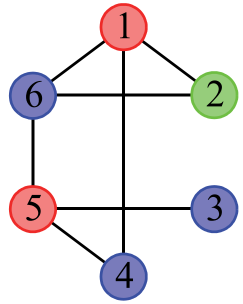
1. 1-[1]-1
2. 5-[1]-1
3. 6-[2]-2
4. 2-[3]-3
5. 4-[2]-3
6. 3-[2]-3

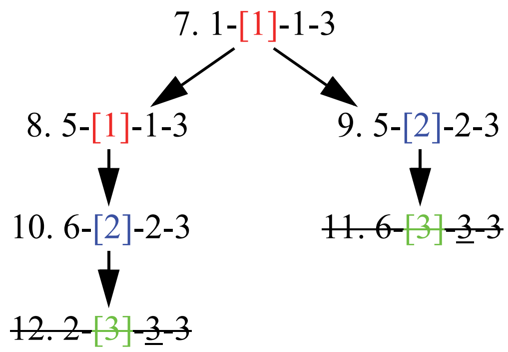
Full search





B&B

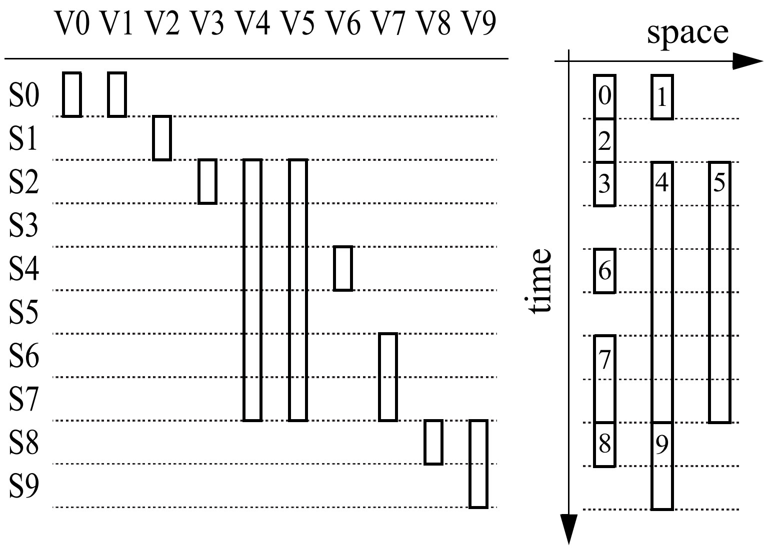




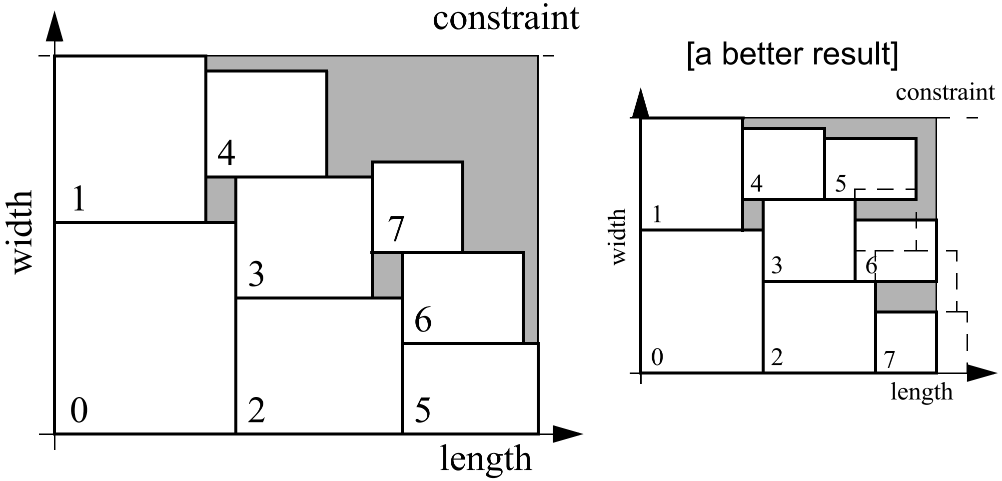
Packing

- **Exact vs. approximate algorithms**
  - **Depends on the complexity of the task!**

1D packing
(interval graph coloring)
[left-edge algorithm]



2D packing
(boxes in a store)
[bottom-left algorithm]



Optimizations in Hardware Design

- **Optimizations at logic level**
  - **thousands of nodes (gates) can exist**
  - **only few possible ways exist how to map an abstract gate onto physical gate from target library**
  - **optimization algorithms can take into account only few of the neighbors**
- **Optimizations at register transfer level (RTL)**
  - **handle hundreds of nodes exist (adders, registers, etc.)**
  - **there are tens of possibilities how to implement a single module**
- **At higher levels, e.g. at system level**
  - **there are only tens of nodes to handle (to optimize)**
  - **there may exist <u>hundreds</u> of ways how to implement a single node**
  - **every possible decision affects much stronger the constraints put onto neighboring nodes thus significantly affecting the quality of the whole design**

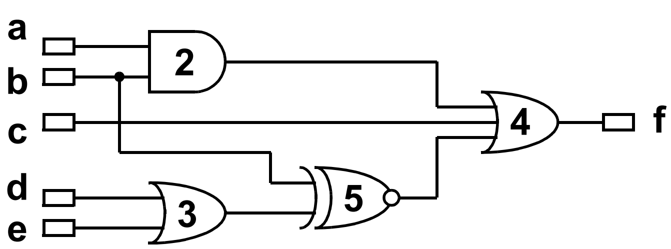
Decisions at Higher Abstraction Levels

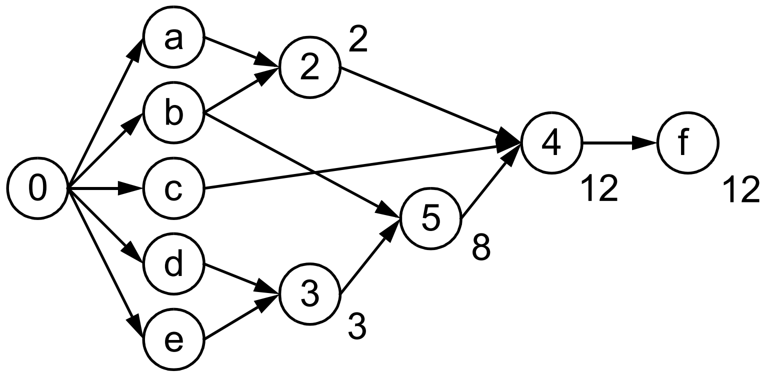
- Two major groups of decisions
  - selection of the right algorithm to solve a subtask
    - making transformations inside the algorithm, e.g. parallel versus sequential execution
    - affect primarily the final architecture of the chip
  - decisions about the data representation
    - e.g. floating point versus fixed point arithmetic, bit-width, precision.

- Selection of a certain algorithm puts additional constraints also onto the data representation

- Selecting a data representation narrows also the number of algorithms available

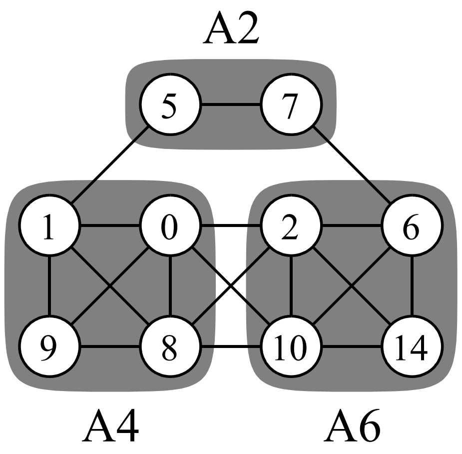
- **Design task #1 – critical path in a network**
  - *longest path* **(acyclic graphs only)**
  - **nodes - logic gates, weight - delay**
  - **edges - connections between gates**



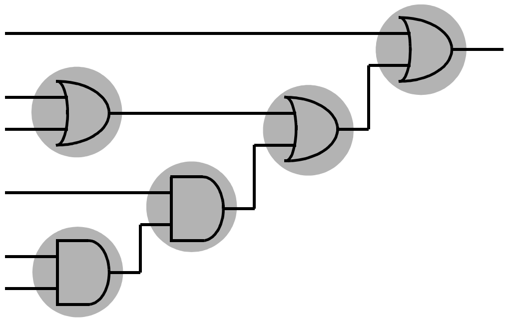


- **Design task #2 – minimizing the number of prime implicants**
  - **finding the *minimal clique cover* of a graph**
  - **node – essential input combination (minterm)**
  - **clique – prime implicant (it is possible to use hyper-edges)**

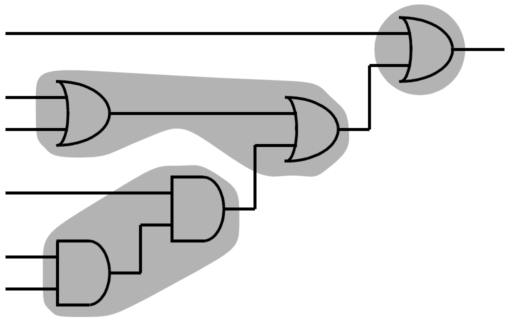
Impl.	0	1	2	5	6	7	8	9	10	14
A1		x		x						
A2				x		x				
A3					x	x				
A4	x	x					x	x		
A5	x		x				x		x	
A6			x		x				x	x



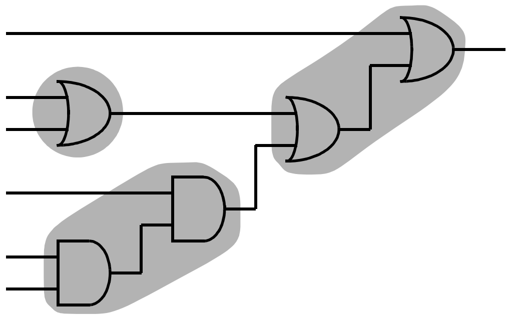
- **Design task #3 – technology mapping**
  - **cheapest** ***clustering***
  - **nodes – logic operations (abstract gates), edges – connections**
  - **cluster – library element (gates)**



2-input
gates

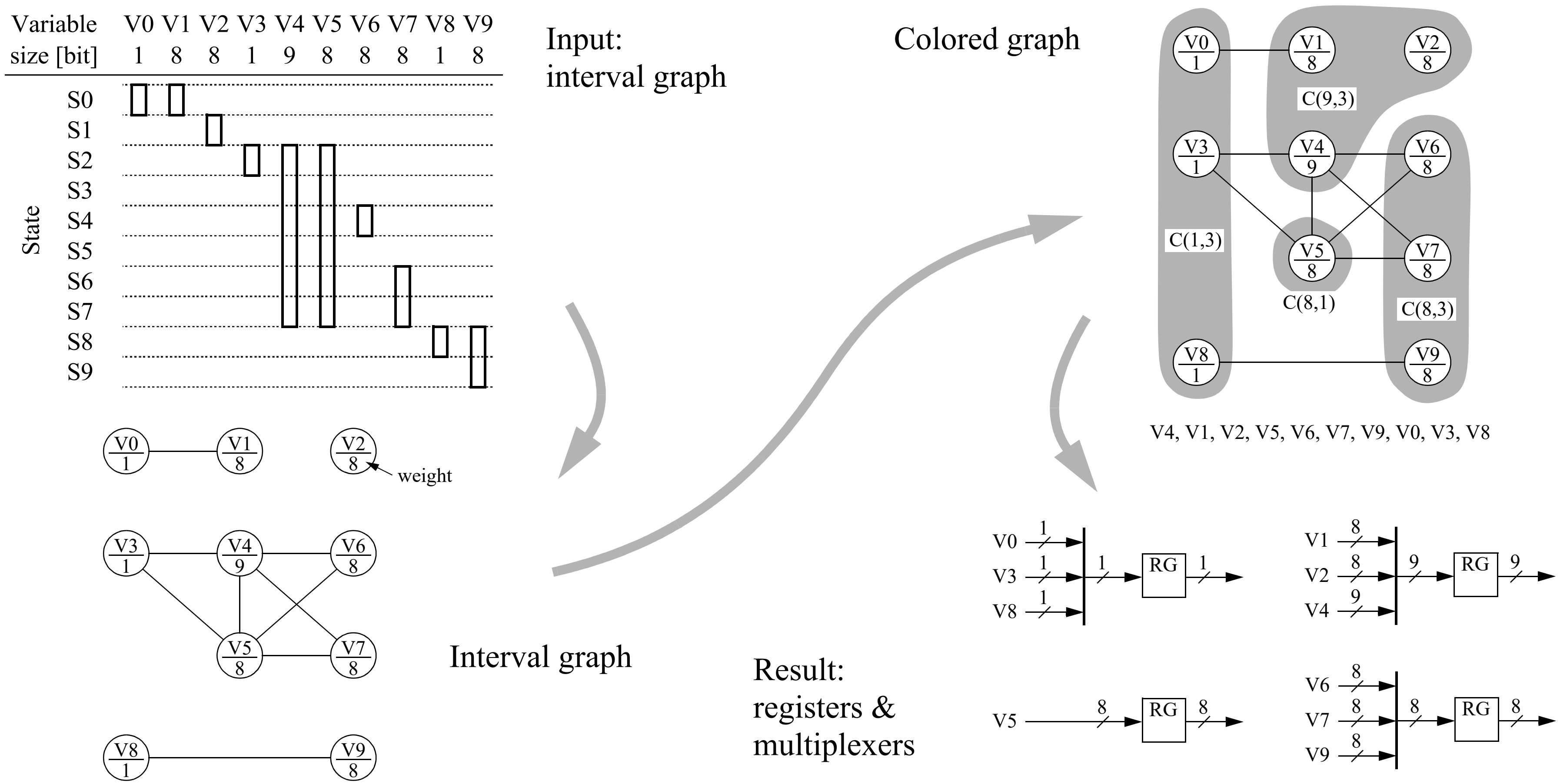


3-input
gates



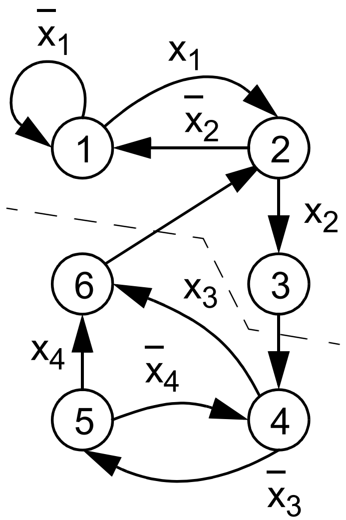
3-input gates
(version - delay?)

- **Design task #4 – register binding**
  - **weighted (interval) graph *coloring***
  - **nodes – variables, edges – variables are used at the same time (intervals overlap)**

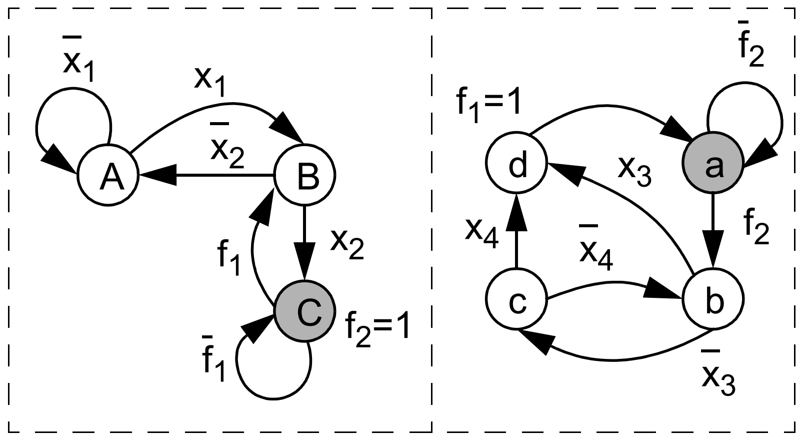


- **Design task #5 – algorithm (state machine) partitioning**
  - **weighted graph *partitioning***
  - **node – state, edge – transition + conditions + frequencies/probabilities**

state machine

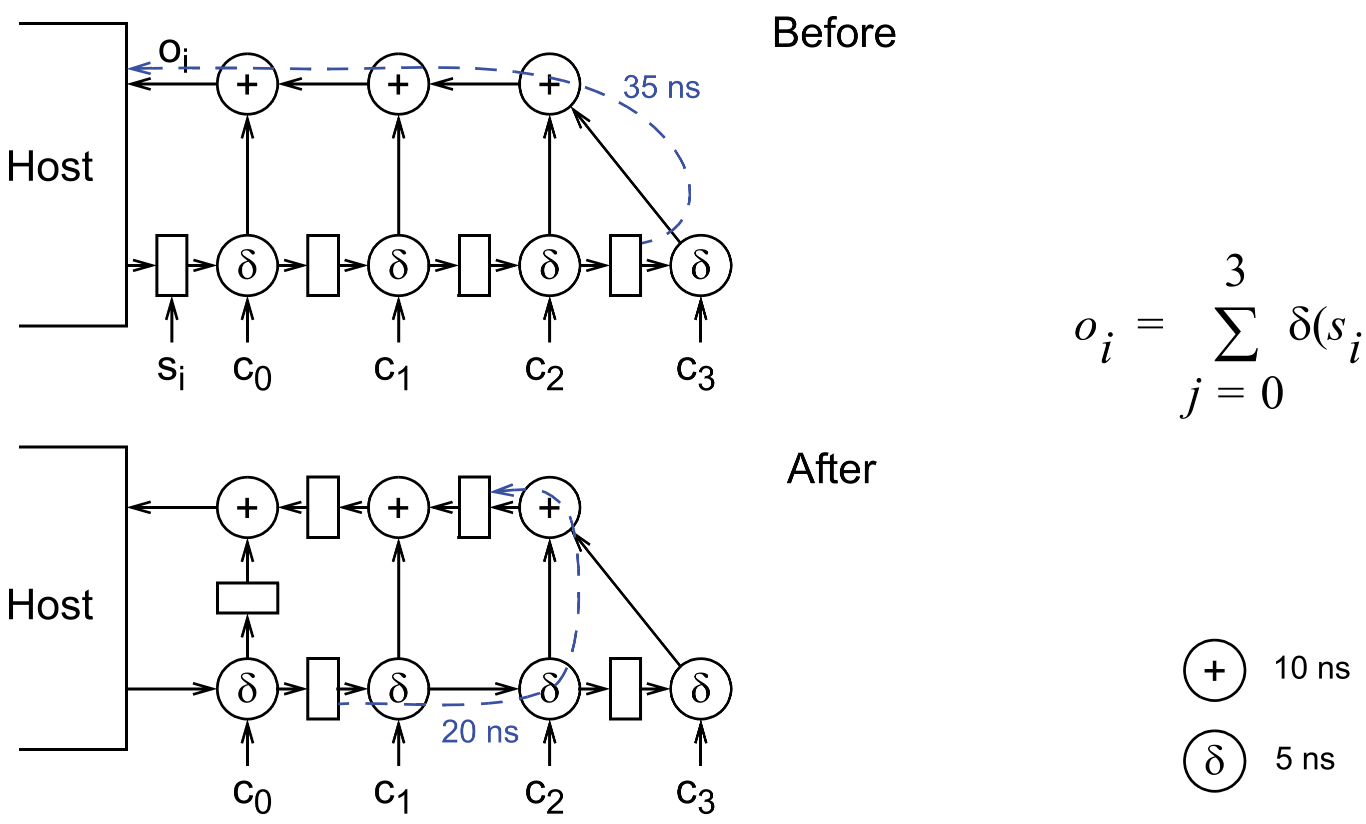


alternately working components



- **Design task #6 – retiming**

Digital correlator



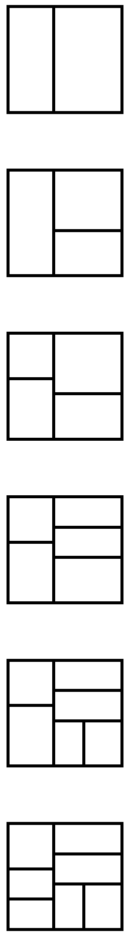
$$o_i = \sum_{j=0}^{3} \delta(s_{i-j}, c_j)$$

The Other Optimization Tasks

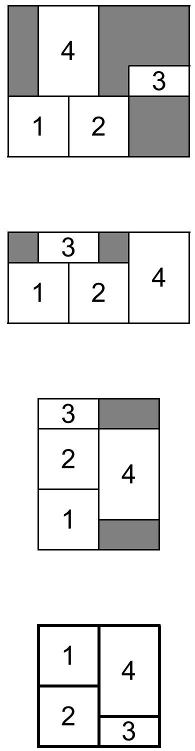
- **System level**
  - **partitioning into sub-modules**
- **Algorithm level**
  - **operation scheduling**
  - **allocation and binding**
- **Register-transfer level**
  - **arithmetic unit architecture selection**
- **Logic level**
  - **multi-level optimization**
  - **encoding**

Physical Level

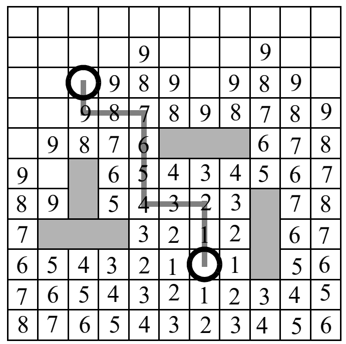
floorplanning



placement



routing

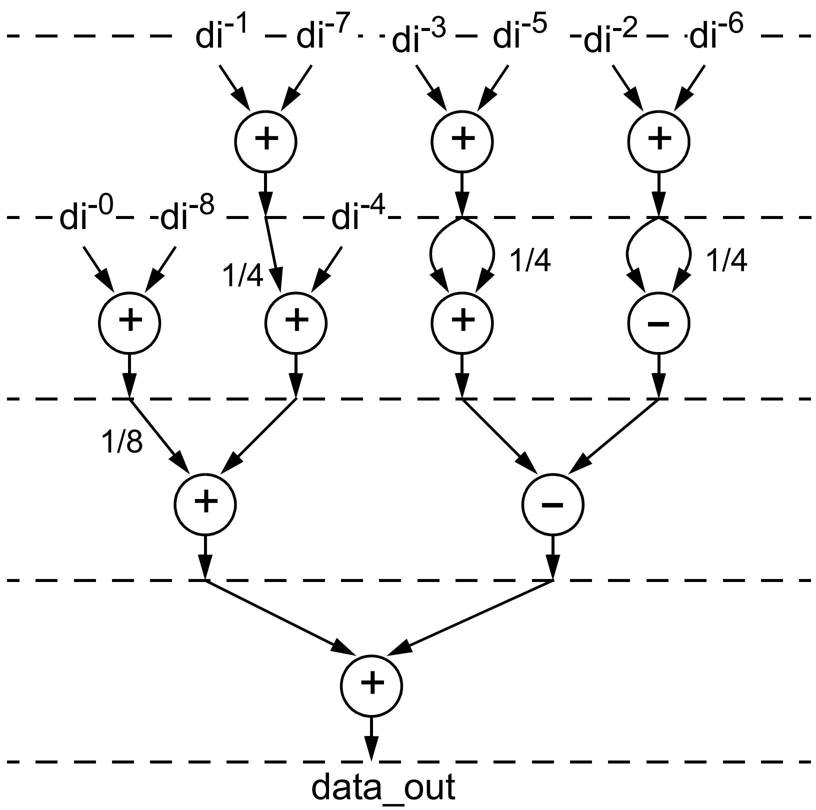


Code Transformations

- High-level synthesis
  - behavioral synthesis / algorithm level synthesis
- allocation, scheduling binding
- FIR filter
- data_out = $0.125*di^{-0} + 0.25*di^{-1} - 0.75*di^{-2} + 1.25*di^{-3} + 1.0*di^{-4} + 1.25*di^{-5} - 0.75*di^{-6} + 0.25*di^{-7} + 0.125*di^{-8}$
- transformations
  - shift-add trees & input swapping

	add #1	add #2	add #3	sub #1
1	$v1=di^{-1}+di^{-7}$	$v2=di^{-2}+di^{-6}$	$v3=di^{-3}+di^{-5}$	
2	$v4=di^{-0}+di^{-8}$	$v5=di^{-4}+v1/4$	v6=v3+v3/4	v7=v2-v2/4
3	v8=v4/8+v5			v9=v6-v7
4	v0=v8+v9			

- 3 add, 1 sub, 4 reg, 12 2-mux

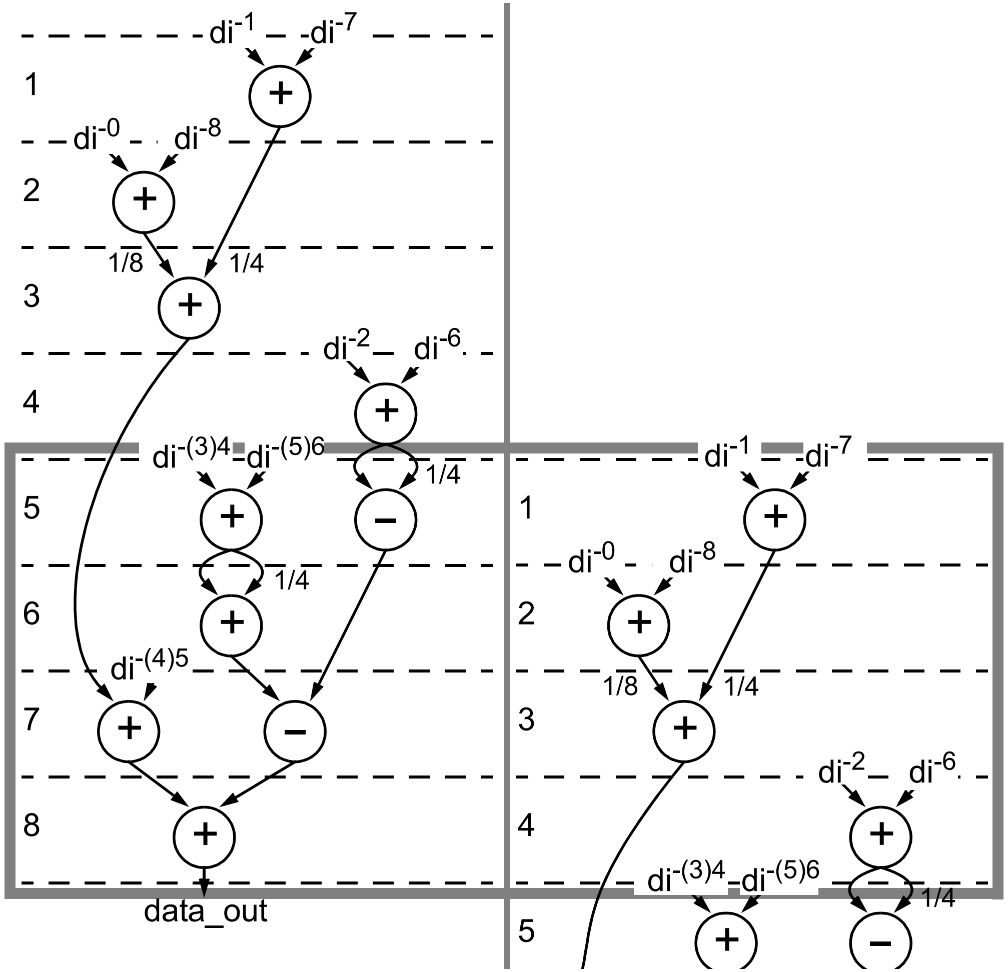


Introducing Pipelining

- **Additional delay at the output!**
  - **scheduling of operations must be analyzed at both stages**
  - **4+4 steps**

	add #1	add #2	sub #1
1	v1=di^{-1}+di^{-7}	v5=di^{-4}+di^{-6}	v6=v4-(v4/4)
2	v2=di^{-0}+di^{-8}	v7=v5+(v5/4)	
3	v3=(v2/8)+(v1/4)	v8=v3+di^{-5}	v9=v7-v6
4	v4=di^{-2}+di^{-6}	v0=v8+v9	

  - **2 add, 1 sub, 5 reg, 15 2-mux**
  - **less FU-s (-1)**
    but more reg-s (+1) & mux-s (+3)

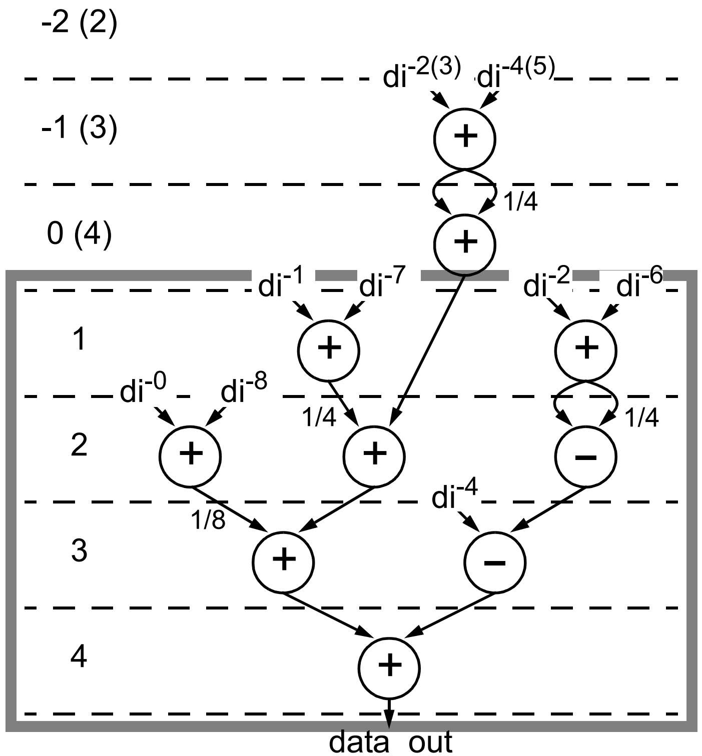


Functional Pipelining

- **Out-of-order execution**
  - **Earlier samples are available!**

	add #1	add #2	sub #1
1	$v1=di^{-1}+di^{-7}$	$v2=di^{-2}+di^{-6}$	
2	$v3=di^{-0}+di^{-8}$	v4=(v1/4)+v9	v5=v2-(v2/4)
3	v6=(v3/8)+v4	$v8=di^{-2}+di^{-4}$	$v7=di^{-4}-v5$
4	v0=v6+v7	v9=v8+(v8/4)	

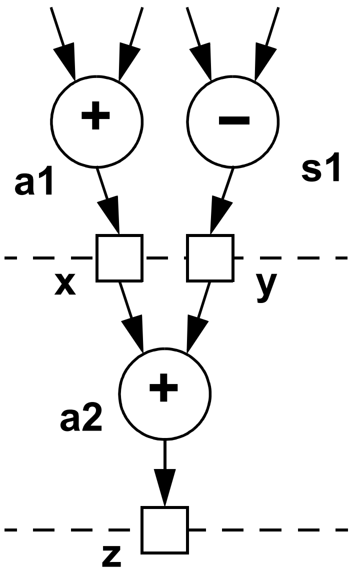
  - **2 add, 1 sub, 3 reg, 14 2-mux**
  - **less FU-s (-1) & reg-s (-1) but more mux-s (+2)**



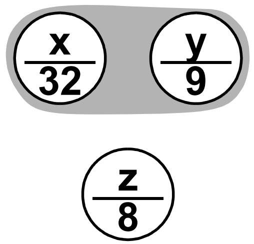
TTÜ1918

Register and Functional Unit Allocation and Binding

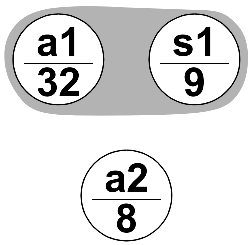
Design Task #4 – Conflict Graph Coloring



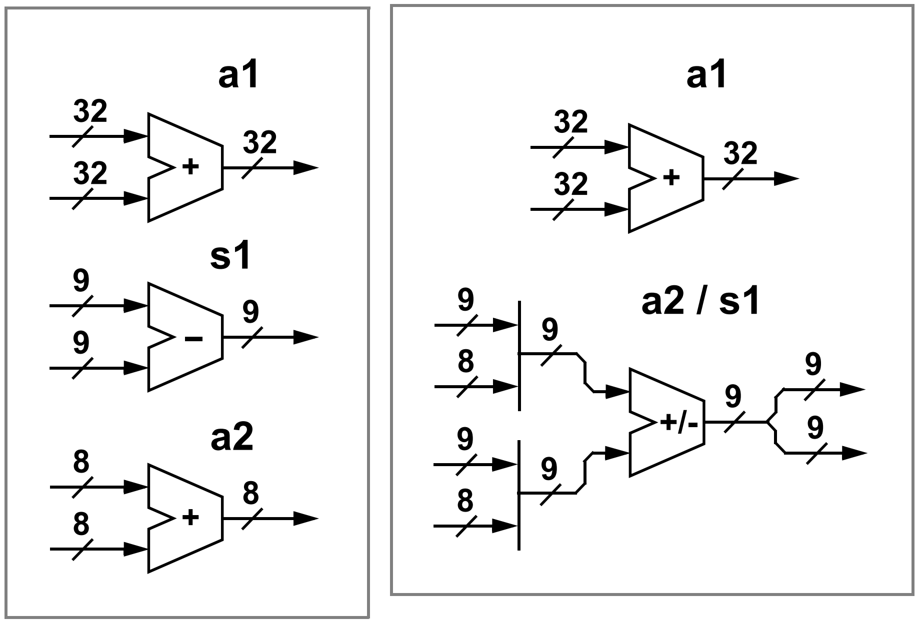
CDFG



registers
(conflict graph)



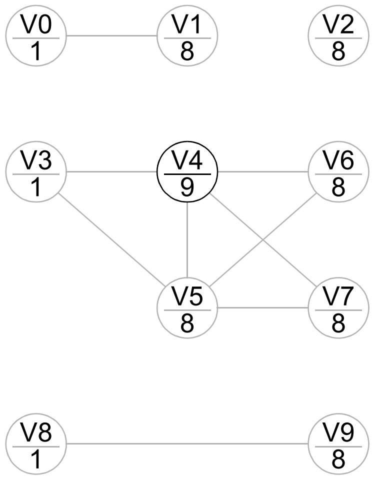
FU-s
(conflict graph)

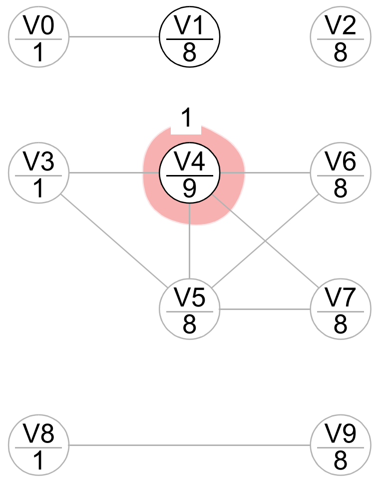


FU allocation and binding - possibilities

Greedy Coloring

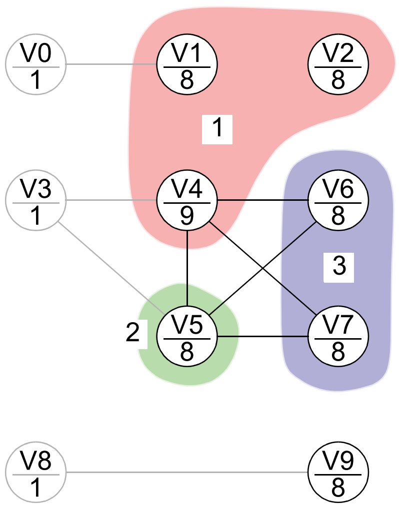
1. Initial.
   V4 - the largest unbound.





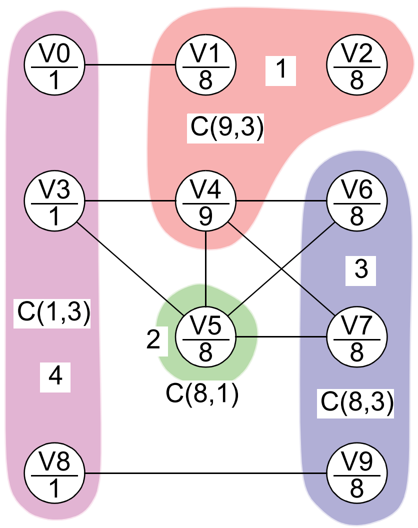
2. **V1** - the largest unbound.
   $C_1(9,2) < C_1(9,1)+C_2(8,1)$
3. **V2** - the largest unbound.
   $C_1(9,3) < C_1(9,2)+C_2(8,1)$
4. **V5** - the largest unbound.
   Creating the 2nd color.
5. **V6** - the largest unbound.
   Creating the 3rd color.
6. **V7** - the largest unbound.
   $C(8,2) < C(8,1) + C(8,1)$

7. **V9** - the largest unbound.
   $C_1(9,3)+C_2(8,1)+C_3(8,3) <$
   $C_1(9,4)+C_2(8,1)+C_3(8,2)$ or
   $C_1(9,3)+C_2(8,2)+C_3(8,2)$ or
   $C_1(9,3)+C_2(8,1)+C_3(8,2)+C_4(8,1)$
8. **V0** - the largest unbound.
   $C_1(9,3)+C_2(8,1)+C_3(8,3)+C_4(1,1)$
   is the cheapest solution -
   multiplexer cost.

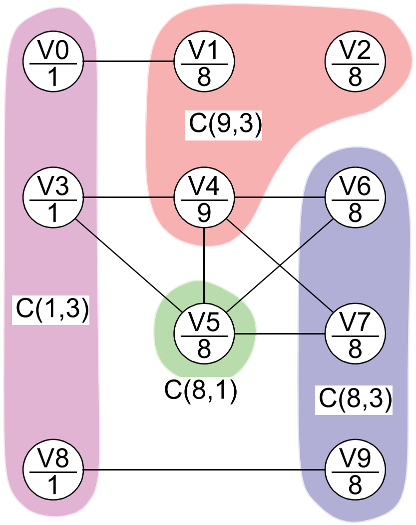


9. **V3** - the largest unbound.
   $C_4(2,1)+\dots$ is the cheapest
10. **V8** - the largest unbound.
    $C_4(3,1)+\dots$ is the cheapest

cost - 278 (66%)



Comparison of Node Selection Heuristics



H1

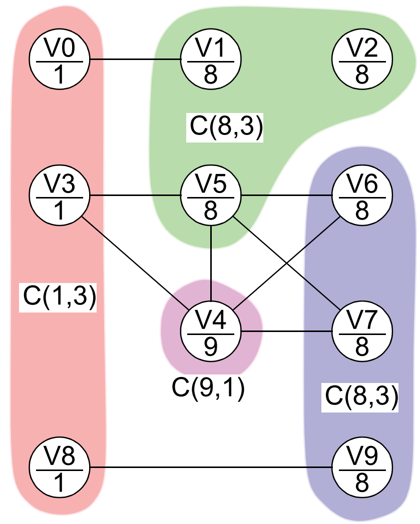
The most expensive nodes first:
V4, V1, V2, V5, V6, V7, V9, V0, V3, V8

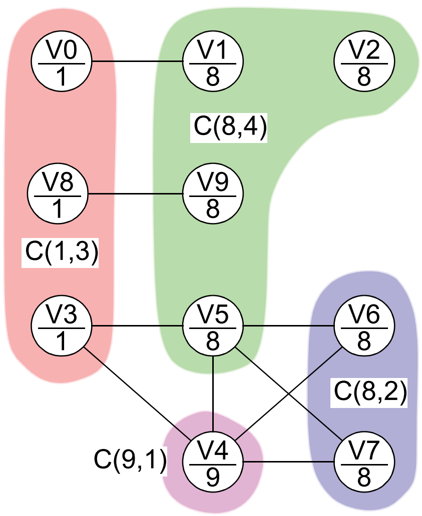
cost - 278 (66%)

H2

The cheapest nodes first:
V0, V3, V8, V1, V2, V5, V6, V7, V9, V4

cost - 273 (65%)





H3

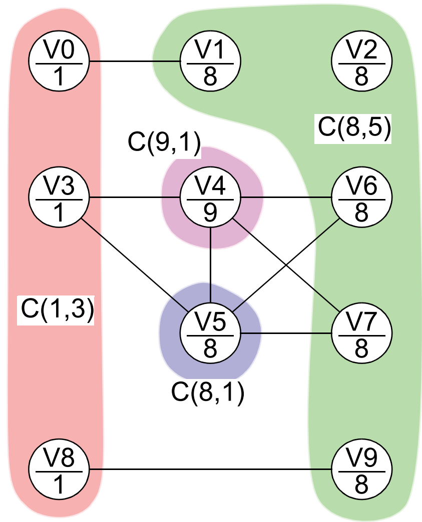
Dynamic selection:
V0, V3, V8, V1, V2, V5, V9, V6, V7, V4

cost - 275 (65%)

H4

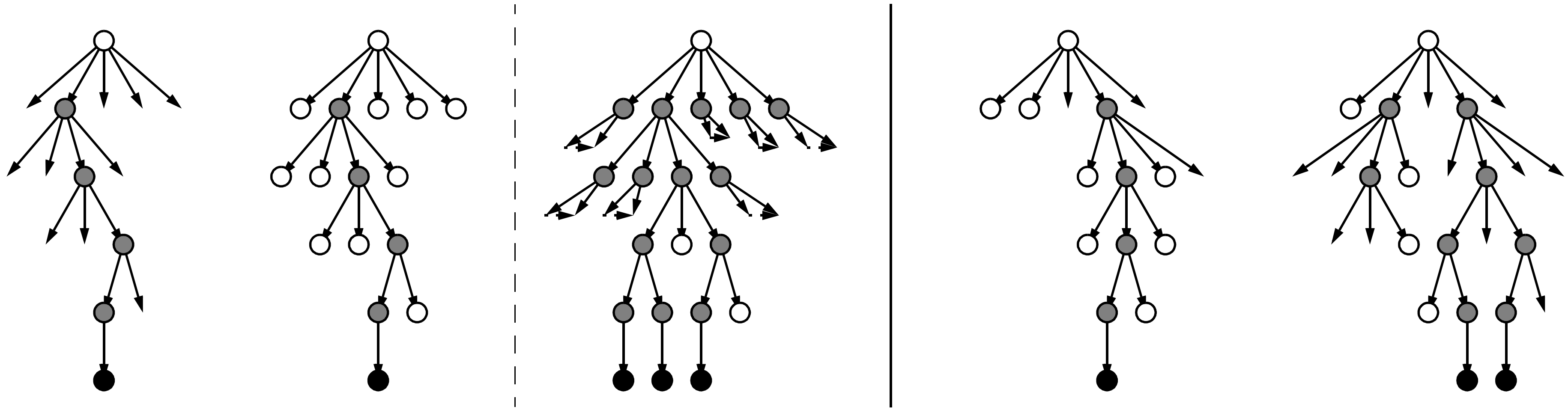
Random selection:
V0, V8, V7, V5, V1, V2, V9, V3, V6, V4

cost - 264 (63%)



Heuristic Optimization Algorithms

"fast but not good enough" "exact but slow" "fast enough and good enough"



static greedy — dynamic greedy — branch-and-bound — extended checking — extended checking and selecting

○ checking ● selecting ● final solution

Basic heuristics

Results of experiments

- **More than 200 tasks from real-life examples**
  - **only 78 did not lead to a global optimum in all cases**

Node selection criterion	Best (of 78)	Minimal (of 78)	Average penalty	Maximum penalty	Average time
H1 (largest)	31	18	2.8%	8.0%	2.8”
H2 (smallest)	21	13	3.8%	10.4%	2.9”
H3 (random)	73	49	1.2%	5.6%	31.5”
H4 (dynamic)	10	5	4.1%	13.3%	185”

- **Coloring of 150 random graphs gave comparable results**

Extending heuristics

Results of experiments

- **Effects of extensions**
  - **123 random graphs (10 to 40 nodes)**
  - **checking: 1 to 8 possible solutions**
  - **storing: 1 to 8 partial solutions**

Selection criterion	**Best**	**Minimal**	**Average penalty**	**Maximum penalty**	**Average time**
H1 (largest)	85	85	1.0%	10.2%	0.1”
check 4, store 4	**93**	**93**	**0.6%**	**10.8%**	**0.9”**
H2 (smallest)	90	89	1.2%	16.9%	0.1”
check 8, store 4	**91**	**91**	**0.9%**	**10.8%**	**1.9”**
H3 (random)	109	107	0.3%	7.5%	3.1”
check 8, store 1	**112**	**109**	**0.2%**	**3.8%**	**35”**