Markovian PERT networks: A new CTMC and benchmark results

Stefan Creemers

(October 22, 2017)

IÉSEG SCHOOL OF MANAGEMENT

KATHOLIEKE UNIVERSITEIT LEUVEN

Agenda

- CTMC of Kulkarni and Adlakha (1986)
- New CTMC
- Comparison of performance for the SRCPSP:
  - CPU times
  - Memory requirements
  - New state-of-the-art results
- Comparison of performance for the SNPV:
  - CPU times
  - Memory requirements
  - New state-of-the-art results
- Conclusion

Agenda

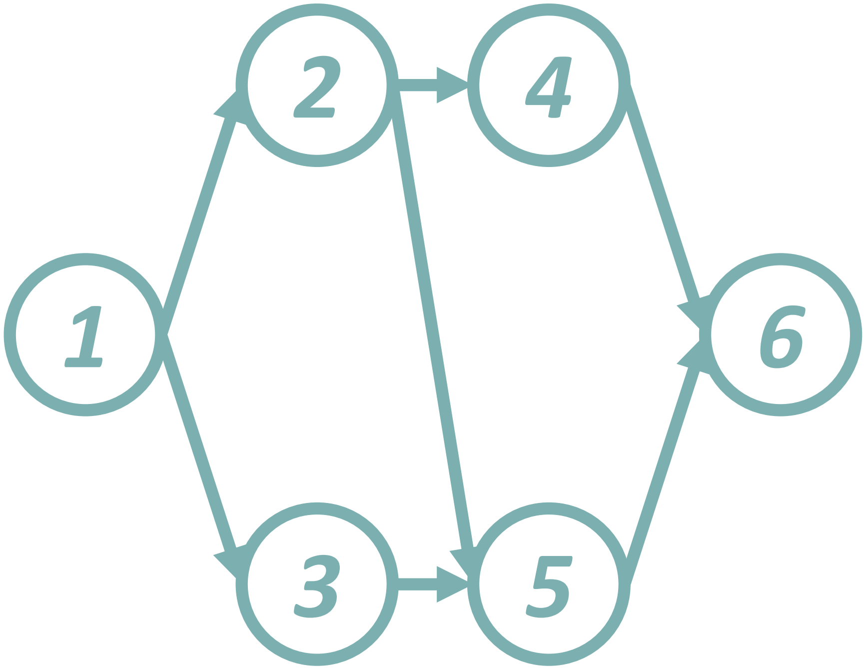
- CTMC of Kulkarni and Adlakha (1986)
- New CTMC
- Comparison of performance for the SRCPSP:
  - CPU times
  - Memory requirements
  - New state-of-the-art results
- Comparison of performance for the SNPV:
  - CPU times
  - Memory requirements
  - New state-of-the-art results
- Conclusion

Kulkarni & Adlakha (1986)

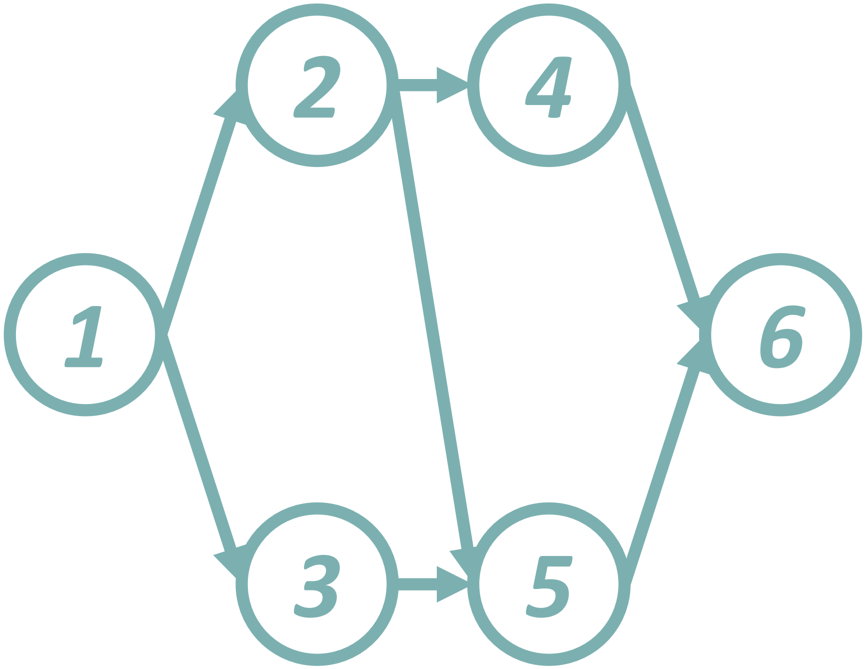
- Markov and Markov-Regenerative PERT Networks, *Operations Research*, 1986
- 208 citations
- First to study Markovian PERT networks
- Use of a CTMC to model a network
- The states of the CTMC are defined by three sets: idle, ongoing, & finished activities

$\Rightarrow$ For a project with n activities there are up to 3^n states!

Example: State space

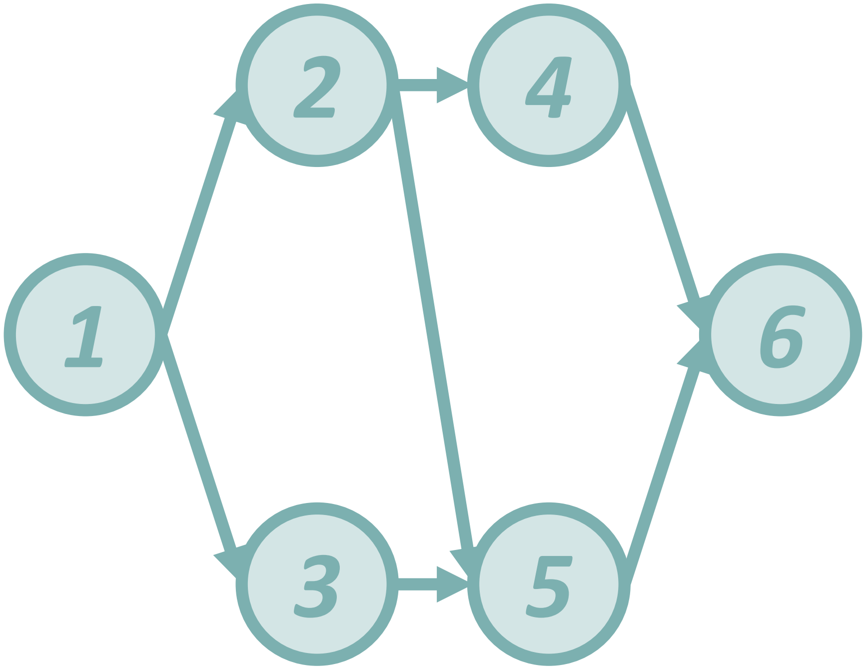


Example: State space



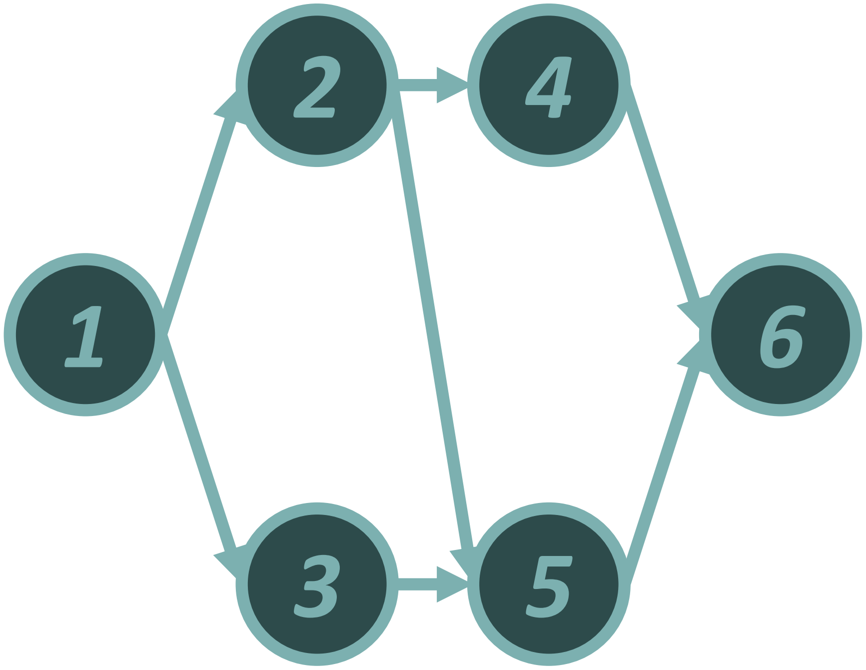
- An activity ***j*** is either:
  - Idle (θ_j=0)

Example: State space



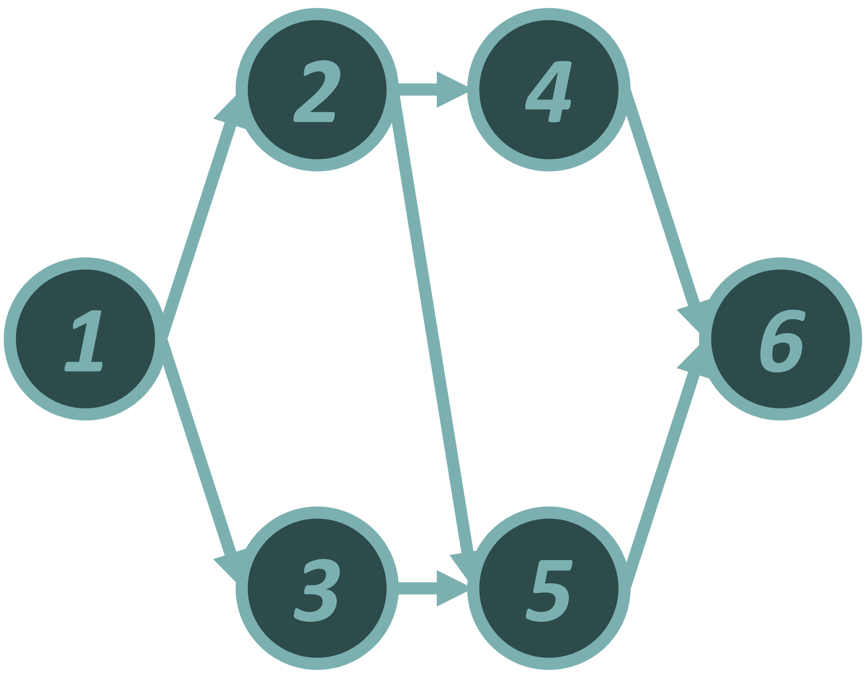
- An activity ***j*** is either:
  - Idle (θ_j=0)
  - Ongoing (θ_j=1)

Example: State space



- An activity ***j*** is either:
  - Idle (θ_j=0)
  - Ongoing (θ_j=1)
  - Finished (θ_j=2)

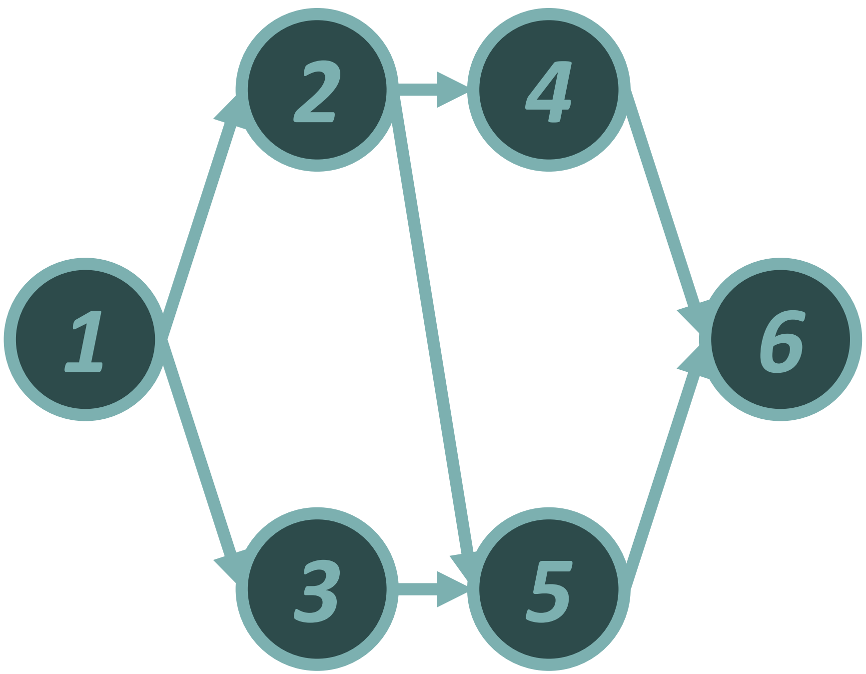
Example: State space



- An activity ***j*** is either:
  - Idle (θ_j=0)
  - Ongoing (θ_j=1)
  - Finished (θ_j=2)
- The state of the system is represented by a vector:

$\boldsymbol{\theta} = \{\theta_1, \theta_2, \ldots \theta_n\}$

Example: State space

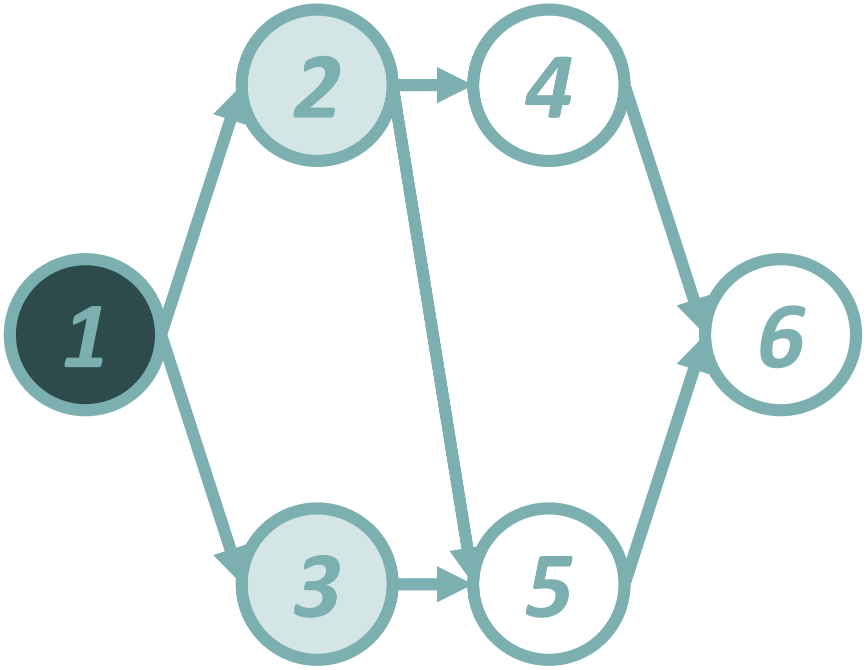


- An activity ***j*** is either:
  - Idle (θ_j=0)
  - Ongoing (θ_j=1)
  - Finished (θ_j=2)
- The state of the system is represented by a vector:

$\boldsymbol{\theta}$ = $\{\theta_1, \theta_2, \ldots \theta_n\}$

- Up to 3^n = 729 states

Example: State space



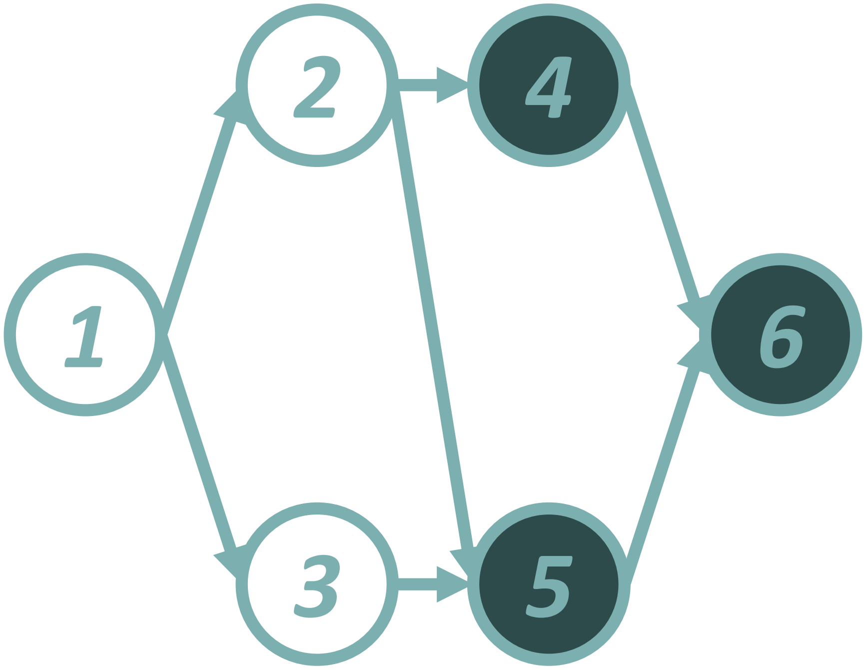
- An activity ***j*** is either:
  - Idle (θ_j=0)
  - Ongoing (θ_j=1)
  - Finished (θ_j=2)
- The state of the system is represented by a vector:

$\boldsymbol{\theta} = \{\theta_1, \theta_2, \ldots \theta_n\}$

- Up to 3^n = 729 states
- Example feasible state:

$\boldsymbol{\theta} = \{2,1,1,0,0,0\}$

Example: State space



- An activity $\boldsymbol{j}$ is either:
  - Idle (θ_j=0)
  - Ongoing (θ_j=1)
  - Finished (θ_j=2)
- The state of the system is represented by a vector:

$\boldsymbol{\theta} = \{\theta_1, \theta_2, \ldots \theta_n\}$

- Up to 3^n = 729 states
- Example feasible state:

$\boldsymbol{\theta} = \{2,1,1,0,0,0\}$

- Example Infeasible state:

$\boldsymbol{\theta} = \{0,0,0,2,2,2\}$

Agenda

- CTMC of Kulkarni and Adlakha (1986)
- New CTMC
- Comparison of performance for the SRCPSP:
  - CPU times
  - Memory requirements
  - New state-of-the-art results
- Comparison of performance for the SNPV:
  - CPU times
  - Memory requirements
  - New state-of-the-art results
- Conclusion

New CTMC

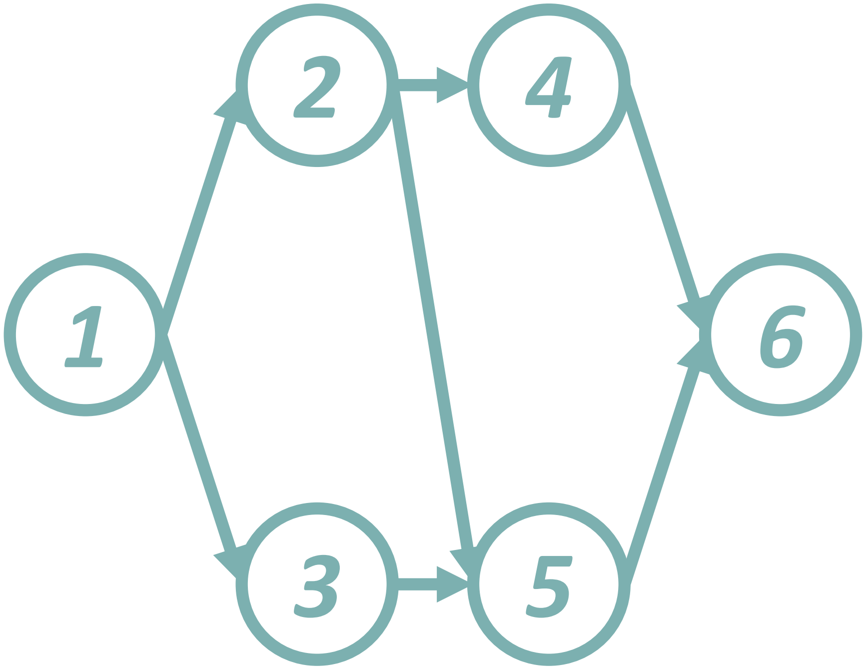
- We are the first to introduce a new CTMC since the CTMC of Kulkarni & Adlakha that was published in 1986
- In this new CTMC, states are defined by the set of finished activities

$\Rightarrow$ up to 2^n states (instead of 3^n states)

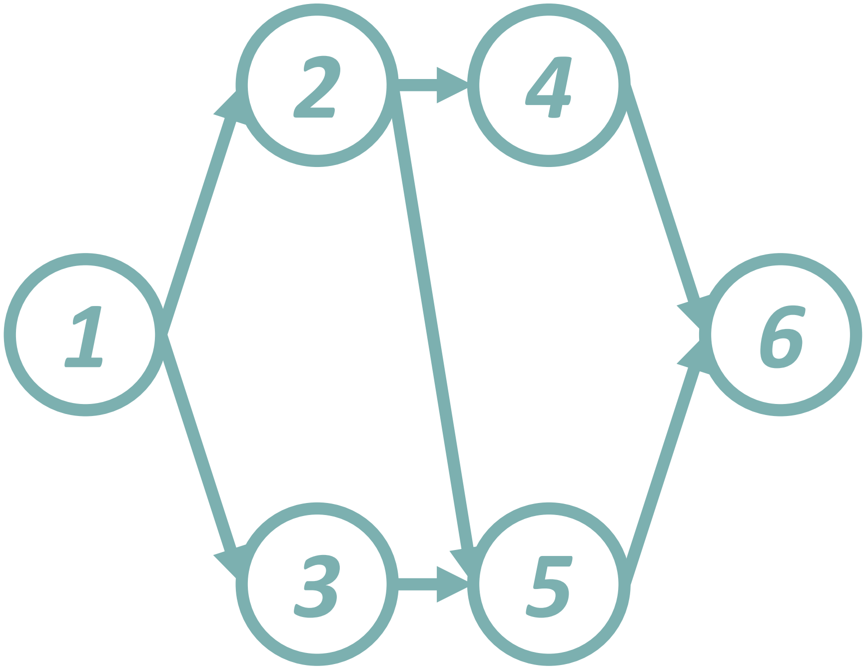
$\Rightarrow$ Huge reduction in memory requirements (= THE bottleneck for CTMC of Kulkarni & Adlakha)

- A potential "drawback" is that the new CTMC allows activities to be preempted

Example: State space

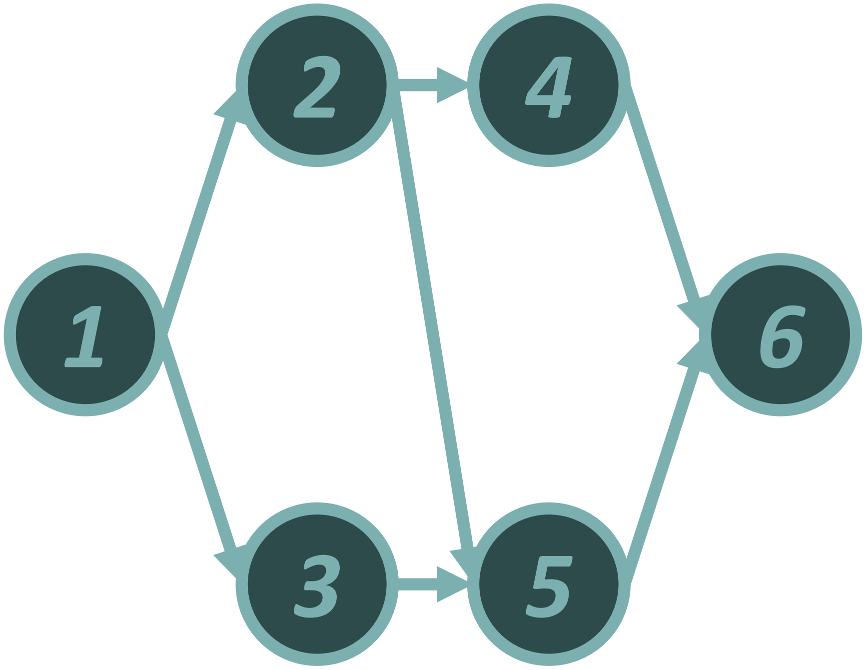


Example: State space



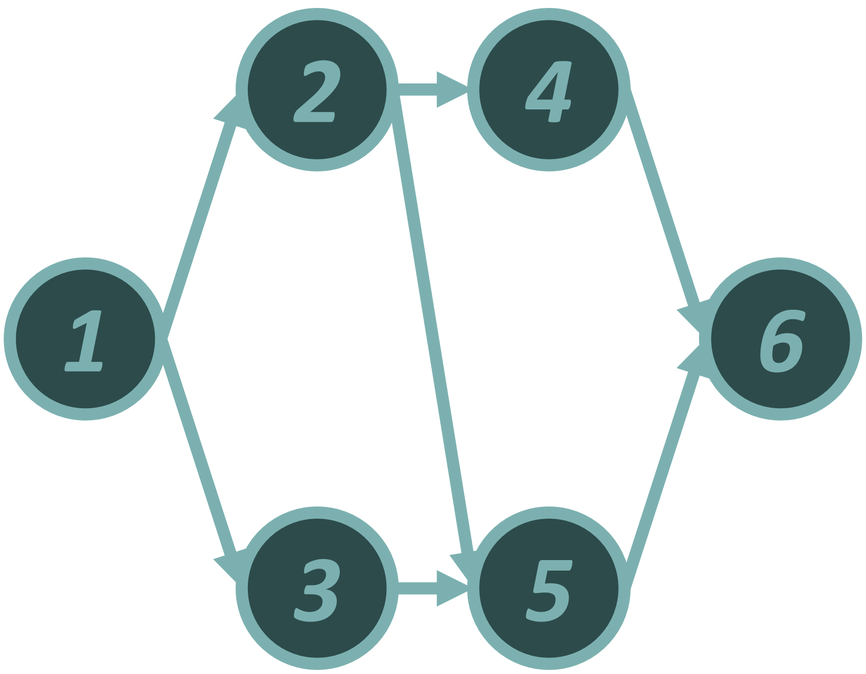
- An activity ***j*** is either:
  - Idle (θ_j=0)

Example: State space



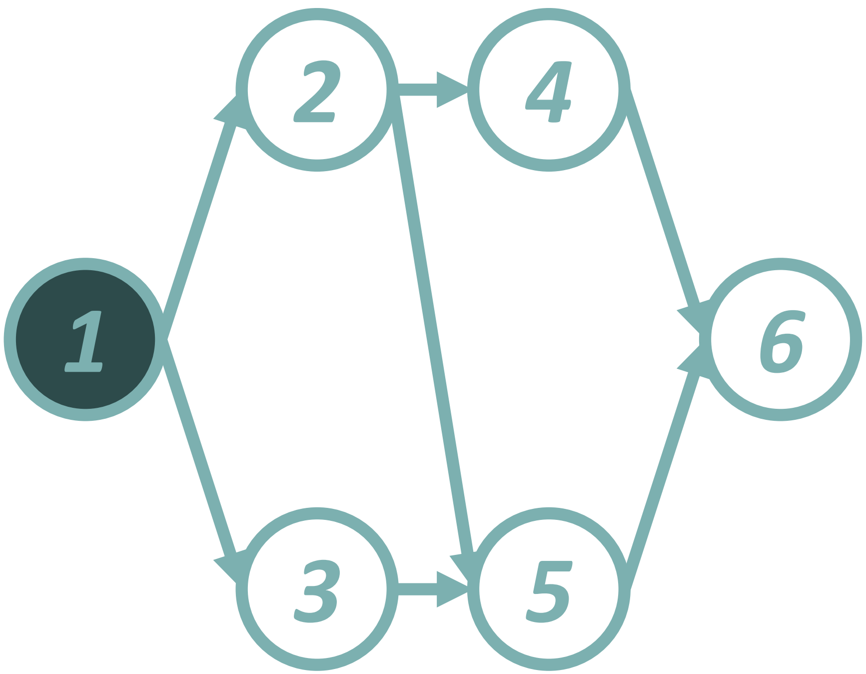
- An activity ***j*** is either:
  - Idle (θ_j=0)
  - Finished (θ_j=1)

Example: State space



- An activity ***j*** is either:
  - Idle (θ_j=0)
  - Finished (θ_j=1)
- Up to 2^n = 64 states

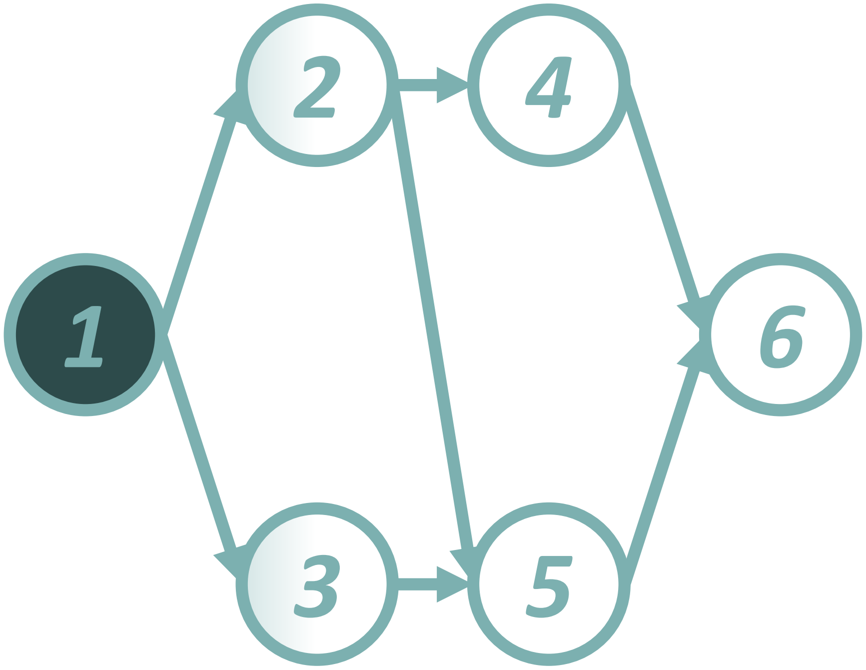
Example: State space



- An activity ***j*** is either:
  - Idle ($\theta_j=0$)
  - Finished ($\theta_j=1$)
- Up to 2^n = 64 states
- Example feasible state:

$\boldsymbol{\theta}$ = $\{1,0,0,0,0,0\}$

Example: State space

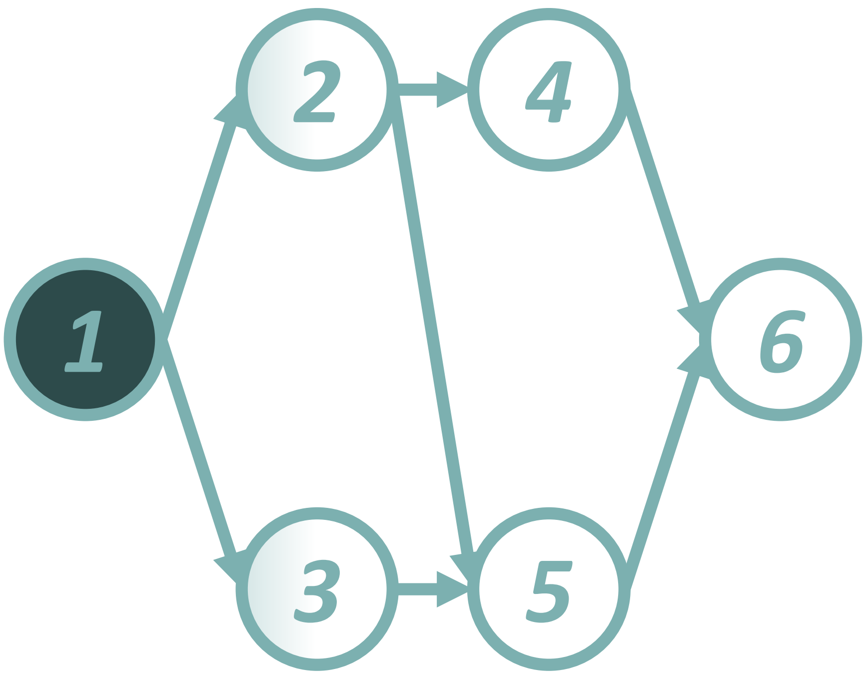


- An activity j is either:
  - Idle (θ_j=0)
  - Finished (θ_j=1)
- Up to 2^n = 64 states
- Example feasible state:

θ = {1,0,0,0,0,0}

- What activities are ongoing? 2? 3? 2 and 3?

Example: State space

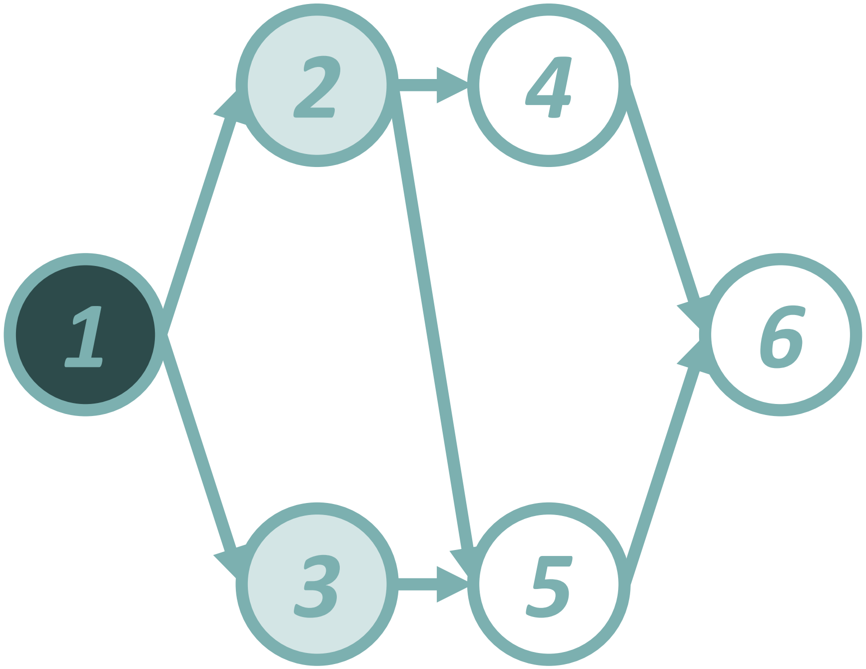


- An activity j is either:
  - Idle (θ_j=0)
  - Finished (θ_j=1)
- Up to 2^n = 64 states
- Example feasible state:

$\boldsymbol{\theta}$ = {1,0,0,0,0,0}

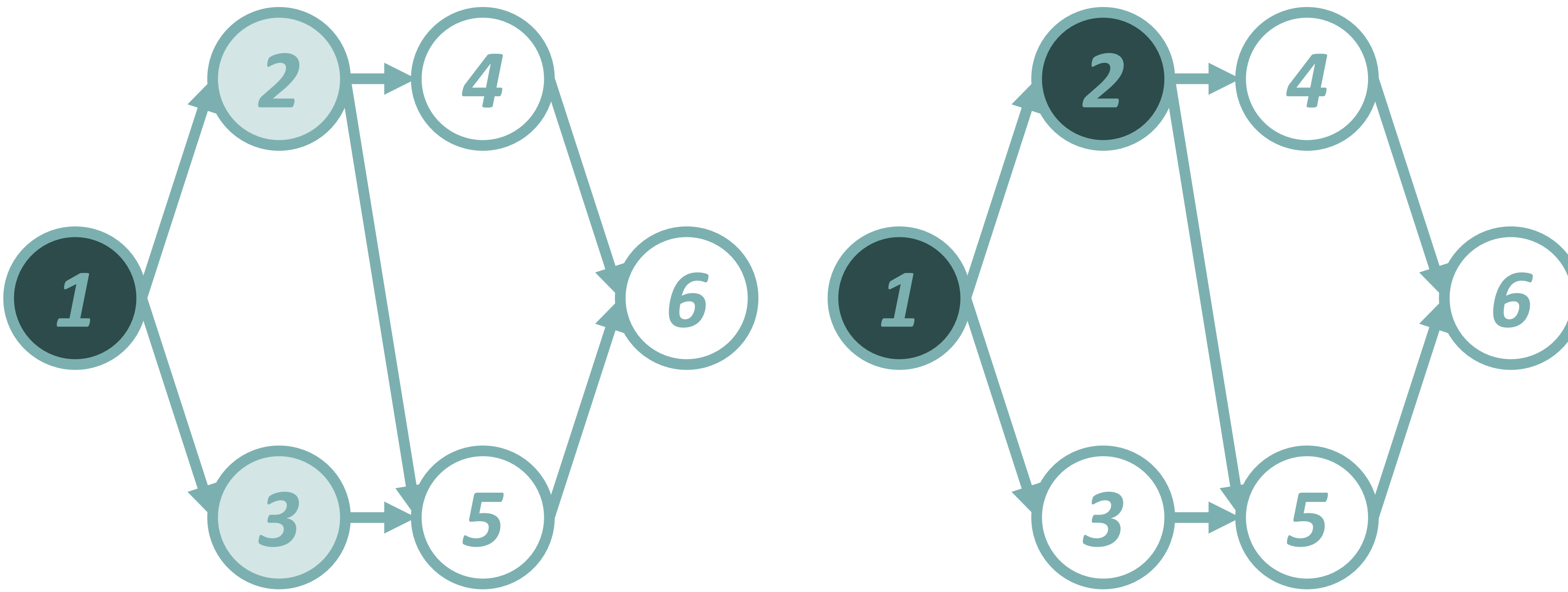
- What activities are ongoing? 2? 3? 2 and 3?
- Preemption is possible

Example: State space



In this state, it is optimal if activities 2 & 3 are ongoing

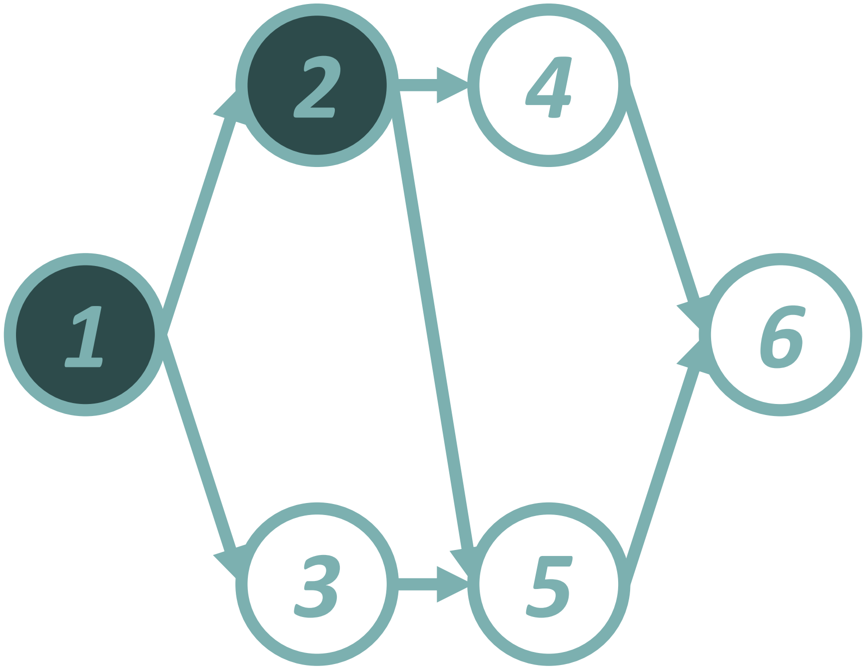
Example: State space



In this state, it is optimal if activities 2 & 3 are ongoing

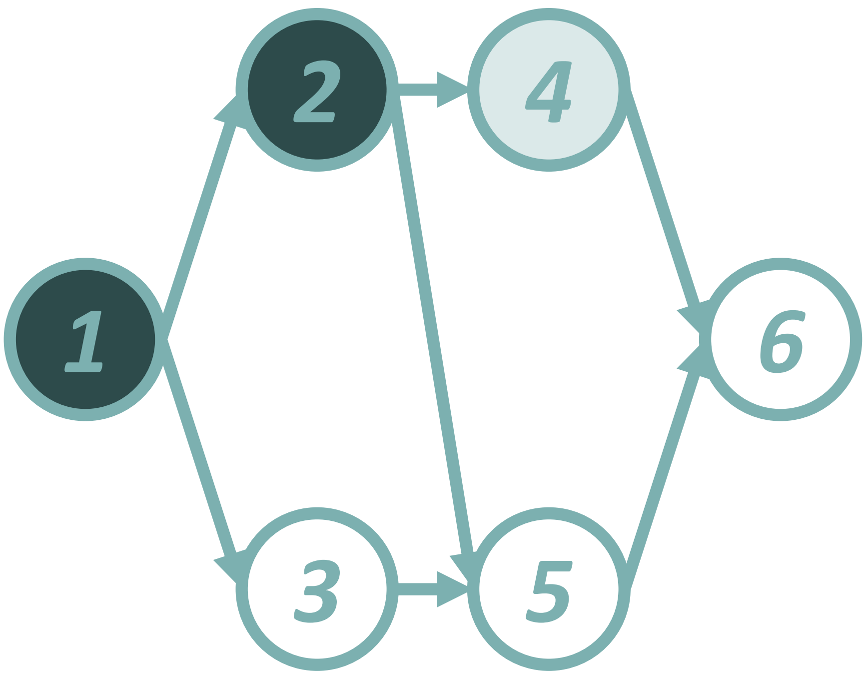
Activity 2 finishes ➔ we end up in state $\theta = \{1,1,0,0,0,0\}$

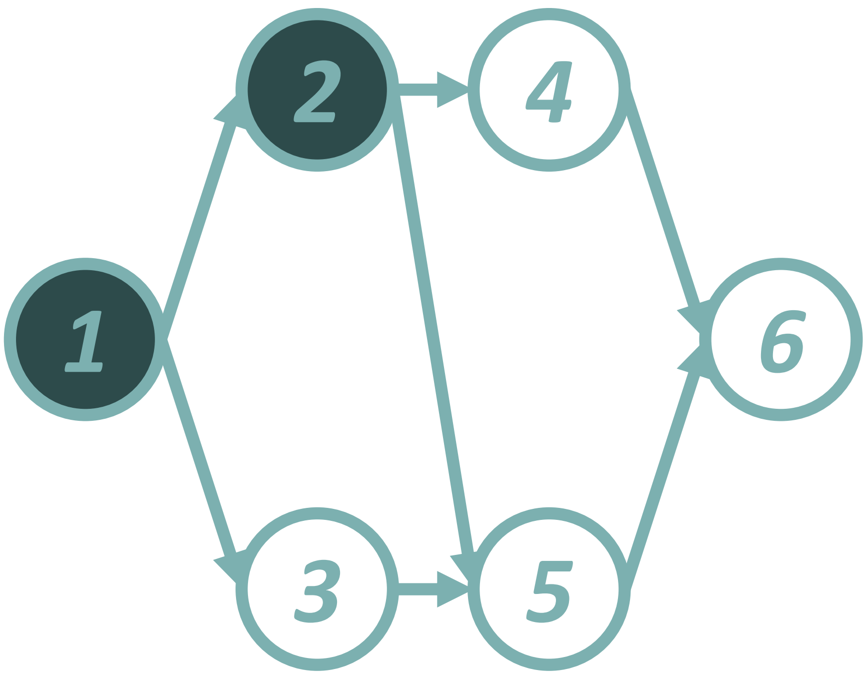
Example: State space



Activity 2 finishes ➔ we end up in state θ = *{1,1,0,0,0,0}*

Example: State space





Here, it is optimal if activity 4 is ongoing ➔ activity 3 is preempted!

Activity 2 finishes ➔ we end up in state θ = {1,1,0,0,0,0}

Agenda

- CTMC of Kulkarni and Adlakha (1986)
- New CTMC
- Comparison of performance for the SRCPSP:
  - CPU times
  - Memory requirements
  - New state-of-the-art results
- Comparison of performance for the SNPV:
  - CPU times
  - Memory requirements
  - New state-of-the-art results
- Conclusion

Creemers (2015)

- Minimizing the expected makespan of a project with stochastic activity durations under resource constraints, *Journal of Scheduling*, 2015
- Current state-of-the-art for solving the **SRCPSP**
- Uses CTMC of Kulkarni & Adlakha
- Computational performance tested on well-known PSPLIB data sets (J30, J60, J90, & J120)
- Bottleneck = memory requirements

SRCPSP

2015 (JOS) Instances Solved

OLD CTMC	
Instances solved (out of 480)	
J30	480
J60	303
J90	NA
J120	NA

SRCPSP

2015 (JOS) CPU Times

OLD CTMC	
Instances solved (out of 480)	
J30	480
J60	303
J90	NA
J120	NA

OLD CTMC	
Average CPU time (s)	
J30	0.48
J60	1591
J90	NA
J120	NA

SRCPSP

2015 (JOS) VS new CTMC

NEW CTMC	
Avg CPU time (s) for same inst.	
J30	0.02
J60	81.6
J90	NA
J120	NA

OLD CTMC	
Average CPU time (s)	
J30	0.48
J60	1591
J90	NA
J120	NA

SRCPSP

2015 (JOS) VS new CTMC

NEW CTMC	
Avg CPU time (s) for same inst.	
J30	0.02
J60	81.6
J90	NA
J120	NA

OLD CTMC	
Average CPU time (s)	
J30	0.48
J60	1591
J90	NA
J120	NA

On average, we improve computation times by a factor of 19!

SRCPSP

2015 (JOS) Instances Solved

OLD CTMC	
Instances solved (out of 480)	
J30	480
J60	303
J90	NA
J120	NA

SRCPSP

2015 (JOS) Memory Requirements

OLD CTMC	
Instances solved (out of 480)	
J30	480
J60	303
J90	NA
J120	NA

OLD CTMC	
Average max # states (x1000)	
J30	176
J60	374499
J90	NA
J120	NA

SRCPSP

2015 (JOS) VS new CTMC

NEW CTMC	
Avg max # states (x1K) for = inst.	
J30	1.99
J60	508
J90	NA
J120	NA

OLD CTMC	
Average max # states (x1000)	
J30	176
J60	374499
J90	NA
J120	NA

SRCPSP

2015 (JOS) VS new CTMC

NEW CTMC	
Avg max # states (x1K) for = inst.	
J30	1.99
J60	508
J90	NA
J120	NA

OLD CTMC	
Average max # states (x1000)	
J30	176
J60	374499
J90	NA
J120	NA

On average, we reduce memory requirements by a factor of 733!

SRCPSP

New CTMC Instances Solved

NEW CTMC	
Instances solved (out of 480)	
J30	480
J60	480
J90	196
J120	10

SRCPSP

New CTMC Instances Solved

NEW CTMC	
Instances solved (out of 480)	
J30	480
J60	480
J90	196
J120	10

We are the first to solve instances of the J90 and J120 data sets to optimality!

Agenda

- CTMC of Kulkarni and Adlakha (1986)
- New CTMC
- Comparison of performance for the SRCPSP:
  - CPU times
  - Memory requirements
  - New state-of-the-art results
- Comparison of performance for the SNPV:
  - CPU times
  - Memory requirements
  - New state-of-the-art results
- Conclusion

Creemers, Leus, & Lambrecht (2010)

- Scheduling Markovian PERT networks to maximize the net present value, *Operations Research Letters*, 2010
- Current state-of-the-art for solving the **SNPV**
- Uses CTMC of Kulkarni & Adlakha
- Computational performance tested on dataset with different *n* and Order Strength (OS)
- Bottleneck = memory requirements

SNPV

2010 (ORL) Instances Solved

OLD CTMC			
Instances solved (out of 30)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	30	30	30
n = 20	30	30	30
n = 30	30	30	30
n = 40	30	30	29
n = 50	30	30	4
n = 60	30	30	0
n = 70	30	22	0

SNPV

2010 (ORL) CPU Times

OLD CTMC			
Instances solved (out of 30)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	30	30	30
n = 20	30	30	30
n = 30	30	30	30
n = 40	30	30	29
n = 50	30	30	4
n = 60	30	30	0
n = 70	30	22	0

OLD CTMC			
Average CPU time (s)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0	0	0
n = 20	0	0	0
n = 30	0	0	27
n = 40	0	7	2338
n = 50	0	100	52268
n = 60	1	2210	NA
n = 70	3	17496	NA

SNPV

2010 (ORL) VS new CTMC

NEW CTMC			
Average CPU time (s) for same instances			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0	0	0
n = 20	0	0	0
n = 30	0	0	0
n = 40	0	0	7
n = 50	0	1	82
n = 60	0	6	NA
n = 70	0	34	NA

OLD CTMC			
Average CPU time (s)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0	0	0
n = 20	0	0	0
n = 30	0	0	27
n = 40	0	7	2338
n = 50	0	100	52268
n = 60	1	2210	NA
n = 70	3	17496	NA

SNPV

2010 (ORL) VS new CTMC

NEW CTMC			
Average CPU time (s) for same instances			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0	0	0
n = 20	0	0	0
n = 30	0	0	0
n = 40	0	0	7
n = 50	0	1	82
n = 60	0	6	NA
n = 70	0	34	NA

OLD CTMC			
Average CPU time (s)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0	0	0
n = 20	0	0	0
n = 30	0	0	27
n = 40	0	7	2338
n = 50	0	100	52268
n = 60	1	2210	NA
n = 70	3	17496	NA

On average, we improve computation times by a factor of 492!

SNPV

2010 (ORL) Memory Requirements

OLD CTMC			
Instances solved (out of 30)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	30	30	30
n = 20	30	30	30
n = 30	30	30	30
n = 40	30	30	29
n = 50	30	30	4
n = 60	30	30	0
n = 70	30	22	0

SNPV

2010 (ORL) Memory Requirements

OLD CTMC			
Instances solved (out of 30)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	30	30	30
n = 20	30	30	30
n = 30	30	30	30
n = 40	30	30	29
n = 50	30	30	4
n = 60	30	30	0
n = 70	30	22	0

OLD CTMC			
Average max # states (x1000)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0	0	1
n = 20	0	4	55
n = 30	2	49	1560
n = 40	8	534	47073
n = 50	27	4346	526020
n = 60	92	42279	NA
n = 70	287	216028	NA

SNPV

2010 (ORL) VS new CTMC

NEW CTMC			
Avg max # states (x1000) for same inst.			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0	0	0
n = 20	0	0	2
n = 30	0	2	17
n = 40	1	9	172
n = 50	2	40	1055
n = 60	4	175	NA
n = 70	8	593	NA

OLD CTMC			
Average max # states (x1000)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0	0	1
n = 20	0	4	55
n = 30	2	49	1560
n = 40	8	534	47073
n = 50	27	4346	526020
n = 60	92	42279	NA
n = 70	287	216028	NA

SNPV

2010 (ORL) VS new CTMC

NEW CTMC			
Avg max # states (x1000) for same inst.			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0	0	0
n = 20	0	0	2
n = 30	0	2	17
n = 40	1	9	172
n = 50	2	40	1055
n = 60	4	175	NA
n = 70	8	593	NA

OLD CTMC			
Average max # states (x1000)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0	0	1
n = 20	0	4	55
n = 30	2	49	1560
n = 40	8	534	47073
n = 50	27	4346	526020
n = 60	92	42279	NA
n = 70	287	216028	NA

On average, we reduce memory requirements by a factor of 403!

SNPV

New CTMC Instances Solved

NEW CTMC			
Instances solved (out of 30)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	30	30	30
n = 20	30	30	30
n = 30	30	30	30
n = 40	30	30	30
n = 50	30	30	30
n = 60	30	30	30
n = 70	30	30	30

SNPV

New CTMC CPU Times

NEW CTMC			
Instances solved (out of 30)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	30	30	30
n = 20	30	30	30
n = 30	30	30	30
n = 40	30	30	30
n = 50	30	30	30
n = 60	30	30	30
n = 70	30	30	30

NEW CTMC			
Average CPU time (s)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0	0	0
n = 20	0	0	0
n = 30	0	0	0
n = 40	0	0	22
n = 50	0	1	476
n = 60	0	11	16869
n = 70	0	99	263012

SNPV

New CTMC CPU Times

NEW CTMC			
Instances solved (out of 30)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	30	30	30
n = 20	30	30	30
n = 30	30	30	30
n = 40	30	30	30
n = 50	30	30	30
n = 60	30	30	30
n = 70	30	30	30

NEW CTMC			
Average CPU time (s)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0	0	0
n = 20	0	0	0
n = 30	0	0	0
n = 40	0	0	22
n = 50	0	1	476
n = 60	0	11	16869
n = 70	0	99	263012

CPU times have become the new bottleneck

SNPV

To preempt or not to preempt?

- If an activity has a zero cost, it is optimal to start that activity as early as possible
- If at time *t* activity *i* is preempted, the remainder of activity *i* joins the set of eligible activities
- The remainder of activity *i* has a zero cost (the cost has already been incurred at the start of activity *i*)

$\Rightarrow$ It is optimal to start the remainder of activity *i* at time *t*

$\Rightarrow$ It is optimal not to preempt activity *i*

Agenda

- CTMC of Kulkarni and Adlakha (1986)
- New CTMC
- Comparison of performance for the SRCPSP:
  - CPU times
  - Memory requirements
  - New state-of-the-art results
- Comparison of performance for the SNPV:
  - CPU times
  - Memory requirements
  - New state-of-the-art results
- Conclusion

Conclusion

- New CTMC that only keeps track of finished activities
- Significantly reduces memory requirements when compared with CTMC of Kulkarni & Adlakha
- New state-of-the-art for solving the SRCPSP and the SNPV
- Bottleneck shifted from memory requirements to CPU times
- Only “drawback” is that the new CTMC allows activities to be preempted
- We prove that there is no preemption when solving the SNPV