A new approach to maximize the expected NPV (eNPV) of a project with activity duration uncertainty

Stefan Creemers

(July 13, 2015)

IÉSEG SCHOOL OF MANAGEMENT

KATHOLIEKE UNIVERSITEIT LEUVEN

Agenda

- Past work
- New approach
- What about the SRCPSP?
- Contribution

Agenda

- Past work
- New approach
- What about the SRCPSP?
- Contribution

Past work: overview

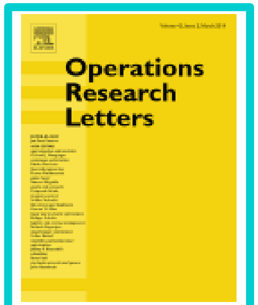
Creemers, Leus, Lambrecht (2010). Scheduling Markovian PERT networks to maximize the net present value, Operations Research Letters.

Past work: overview

Creemers, Leus, Lambrecht (2010). Scheduling Markovian PERT networks to maximize the net present value, Operations Research Letters.

1. Maximum-eNPV objective
2. No resources
3. Exponentially-distributed activity durations
4. Use of a SDP recursion to obtain the optimal policy

Past work: overview



Creemers, Leus, Lambrecht (2010). Scheduling Markovian PERT networks to maximize the net present value, Operations Research Letters.

Creemers (2015) Minimizing the makespan of a project with stochastic activity durations under resource constraints, Journal of Scheduling.

1. Maximum-eNPV objective
2. No resources
3. Exponentially-distributed activity durations
4. Use of a SDP recursion to obtain the optimal policy

Past work: overview

Creemers, Leus, Lambrecht (2010). Scheduling Markovian PERT networks to maximize the net present value, Operations Research Letters.

1. Maximum-eNPV objective
2. No resources
3. Exponentially-distributed activity durations
4. Use of a SDP recursion to obtain the optimal policy

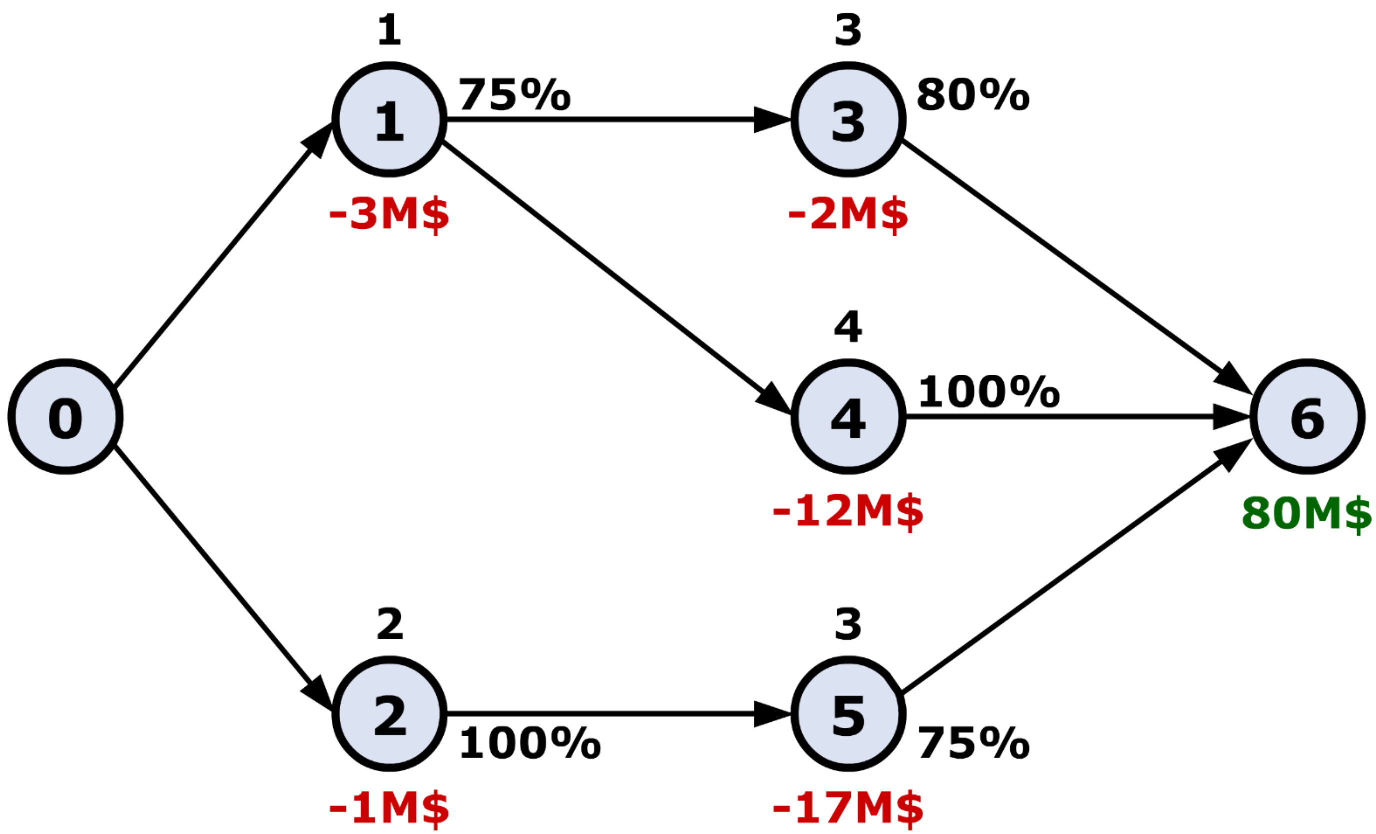
Creemers (2015) Minimizing the makespan of a project with stochastic activity durations under resource constraints, Journal of Scheduling.

1. Minimum-makespan objective
2. Renewable resources
3. General activity durations (PH approximation)
4. Use of an improved/modified SDP recursion

Past work: overview

Fear not!
we have an example to help you understand all this **mumbo jumbo**! In the example, we consider an **eNPV objective** and assume that activity durations are **exponentially distributed** (to keep things simple).

Past work: example



AON network with 5 non-dummy activities

Past work: state & state space

- The state of the project is determined by the status of the activities

Past work: state & state space

- The state of the project is determined by the status of the activities
- The status of an activity j at time t is either:
  - Idle ($\theta_j(t) = 0$)
  - Busy ($\theta_j(t) = 1$)
  - Finished ($\theta_j(t) = 2$)

Past work: state & state space

- The state of the project is determined by the status of the activities
- The status of an activity *j* at time *t* is either:
  - Idle ($\theta_j(t) = 0$)
  - Busy ($\theta_j(t) = 1$)
  - Finished ($\theta_j(t) = 2$)
- $\boldsymbol{\theta}(t) = \{\theta_1(t), \theta_2(t), \ldots \theta_n(t)\}$ defines the state of the system

Past work: state & state space

- The state of the project is determined by the status of the activities
- The status of an activity j at time t is either:
  - Idle ($\theta_j(t) = 0$)
  - Busy ($\theta_j(t) = 1$)
  - Finished ($\theta_j(t) = 2$)
- $\boldsymbol{\theta}(t) = \{\theta_1(t), \theta_2(t), \ldots \theta_n(t)\}$ defines the state of the system
- The size of the state space has upper bound 3^n

Past work: state & state space

- The state of the project is determined by the status of the activities
- The status of an activity j at time t is either:
  - Idle ($\theta_j(t) = 0$)
  - Busy ($\theta_j(t) = 1$)
  - Finished ($\theta_j(t) = 2$)
- $\boldsymbol{\theta}(t) = \{\theta_1(t), \theta_2(t), \ldots \theta_n(t)\}$ defines the state of the system
- The size of the state space has upper bound 3^n
- Most of these states do not meet precedence constraints

Past work: state & state space

- The state of the project is determined by the status of the activities
- The status of an activity *j* at time *t* is either:
  - Idle ($\theta_j(t) = 0$)
  - Busy ($\theta_j(t) = 1$)
  - Finished ($\theta_j(t) = 2$)
- $\boldsymbol{\theta}(t) = \{\theta_1(t), \theta_2(t), \ldots \theta_n(t)\}$ defines the state of the system
- The size of the state space has upper bound 3^n
- Most of these states do not meet precedence constraints

$\Rightarrow$ A clear and strict definition of the state space is essential

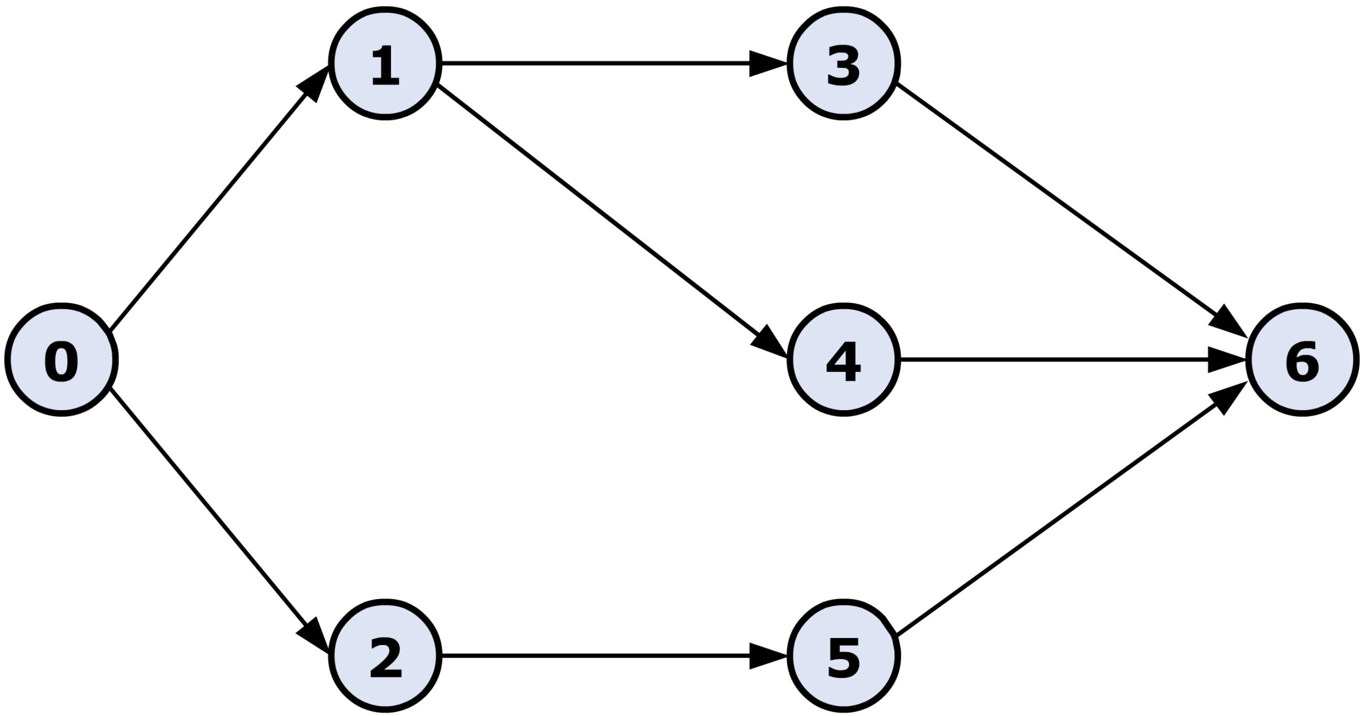
Past work: state & state space

- The state of the project is determined by the status of the activities
- The status of an activity *j* at time *t* is either:
  - Idle ($\theta_j(t) = 0$)
  - Busy ($\theta_j(t) = 1$)
  - Finished ($\theta_j(t) = 2$)
- $\boldsymbol{\theta}(t) = \{\theta_1(t), \theta_2(t), \ldots \theta_n(t)\}$ defines the state of the system
- The size of the state space has upper bound 3^n
- Most of these states do not meet precedence constraints

$\Rightarrow$ A clear and strict definition of the state space is essential

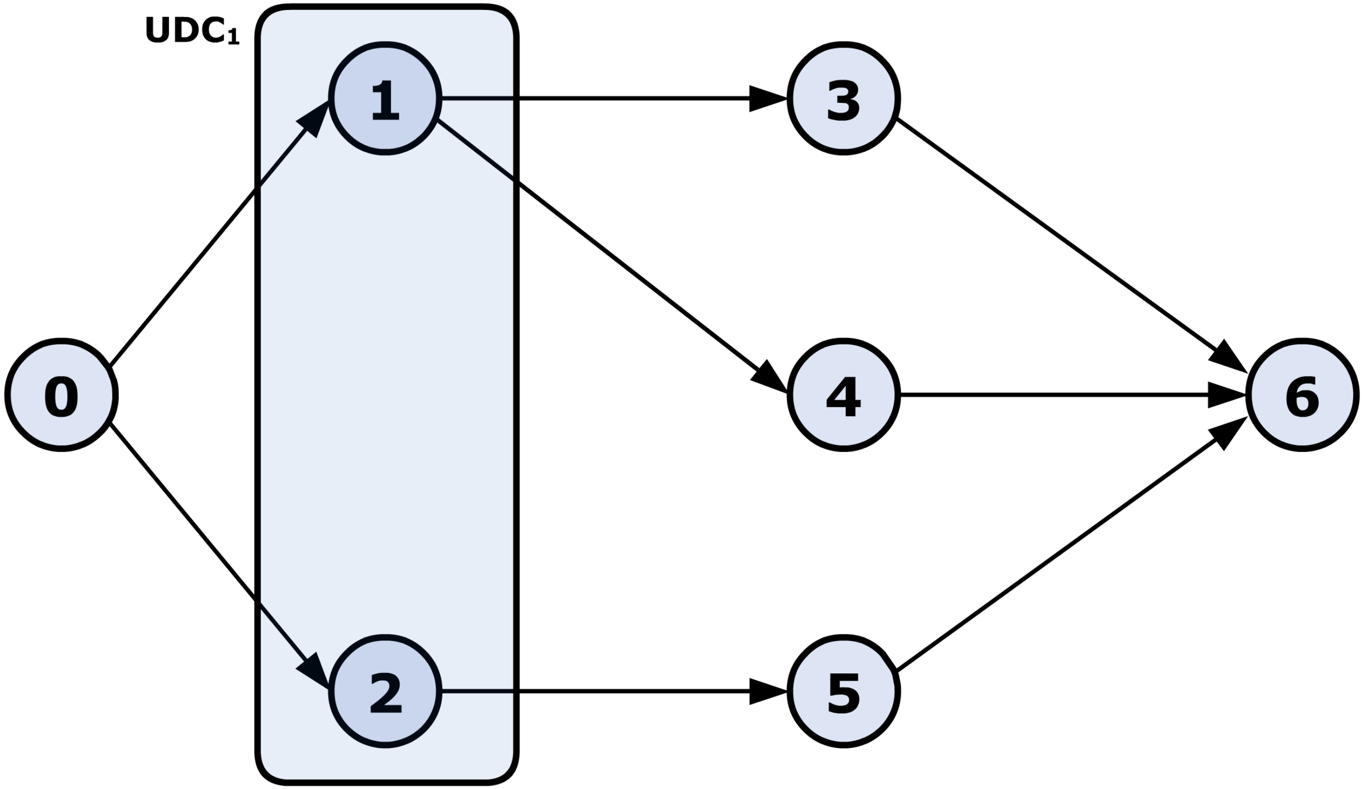
$\Rightarrow$ We use UDCs to structure the state space

Past work: example



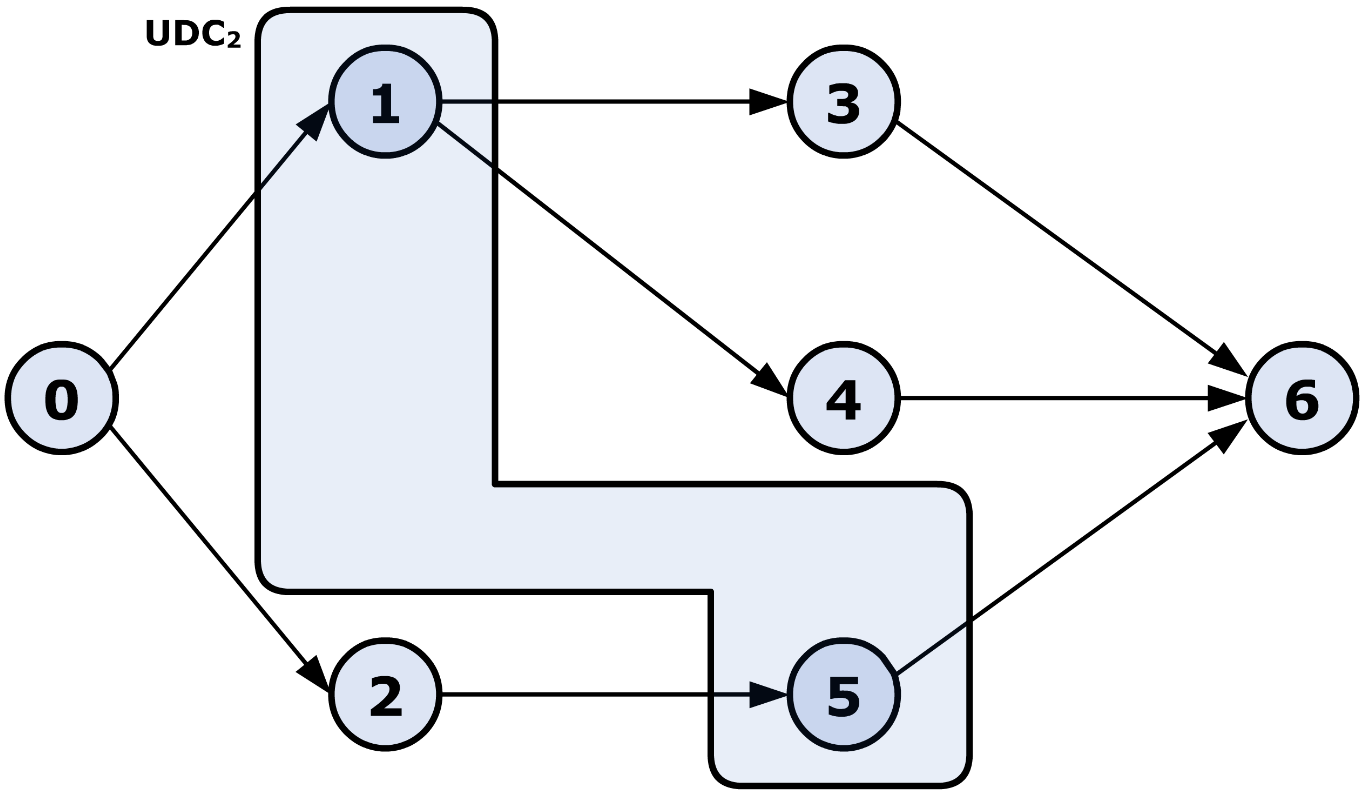
UDC = set of all activities that can be executed in parallel

Past work: example



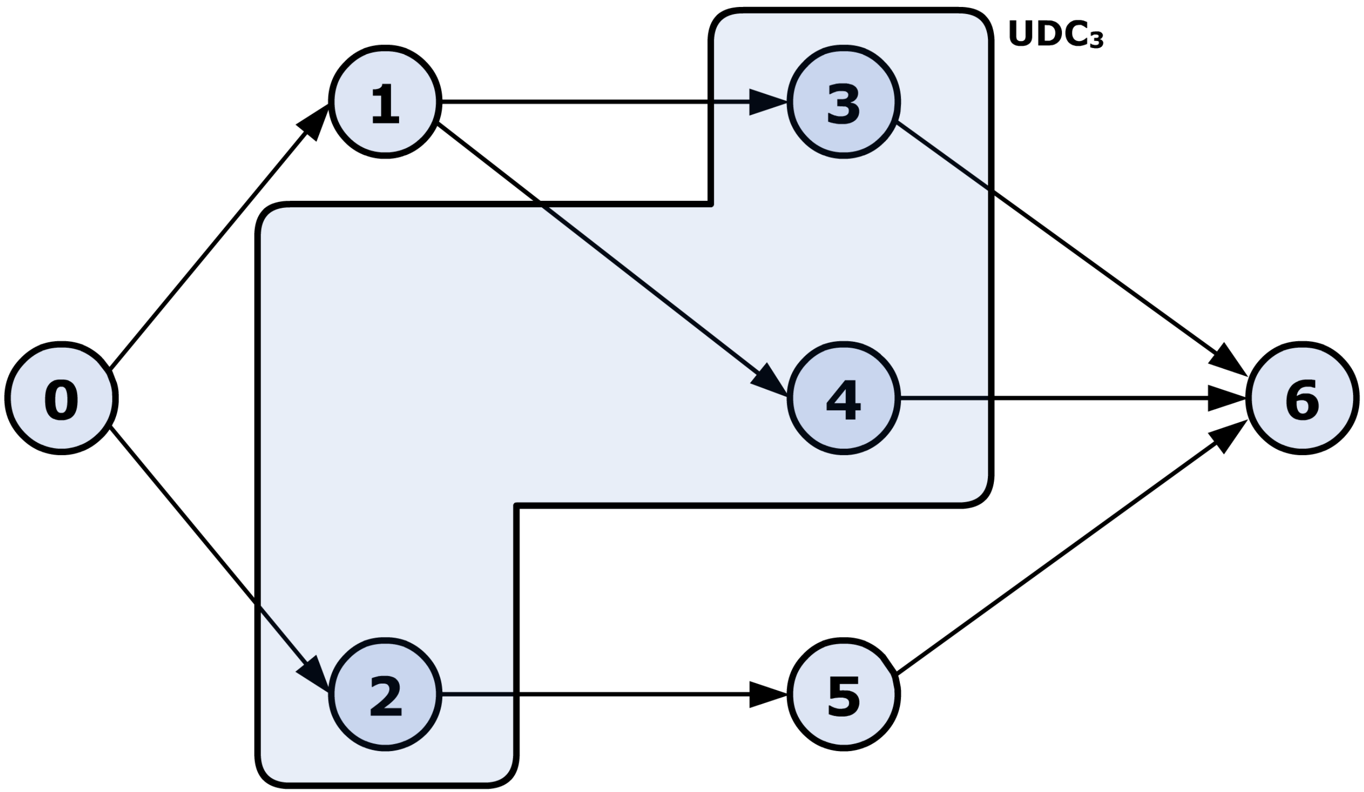
UDC = set of all activities that can be executed in parallel

Past work: example



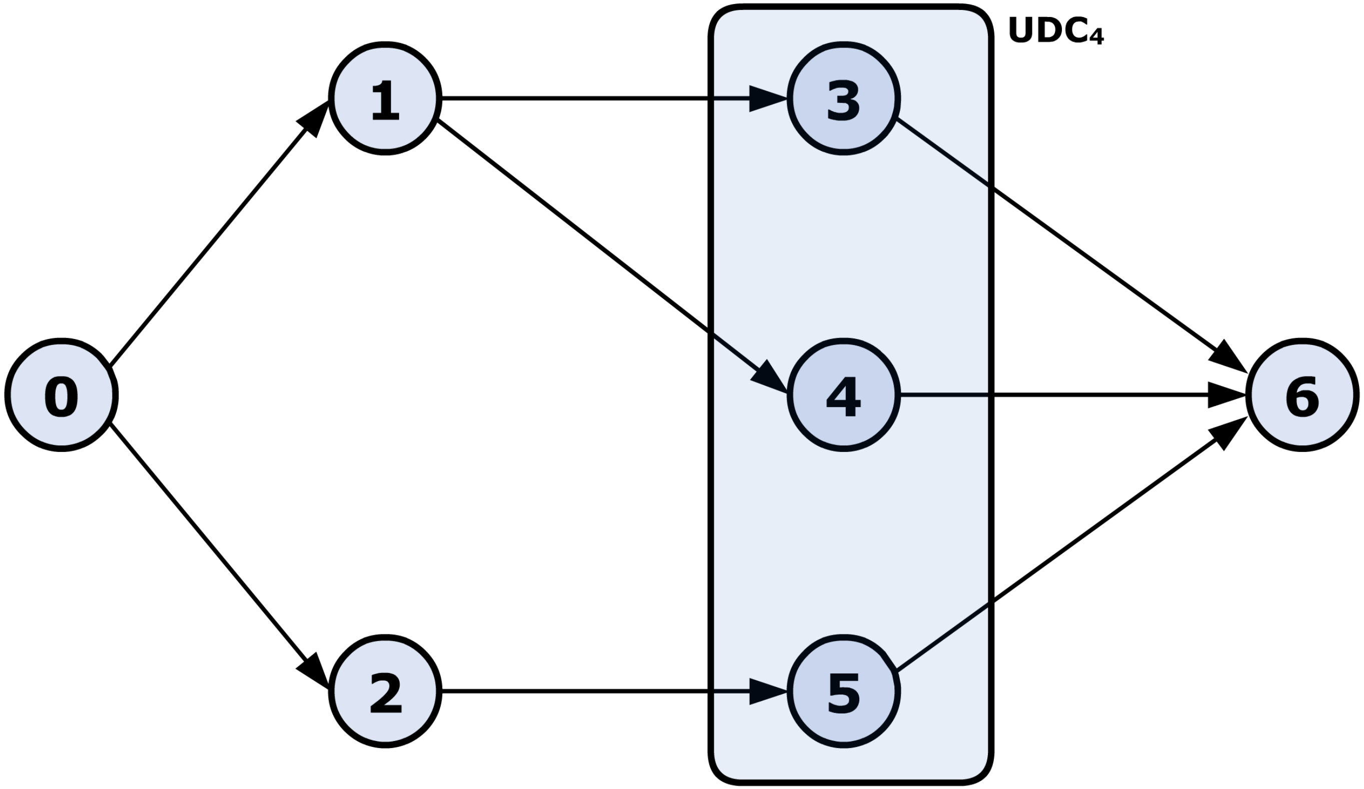
UDC = set of all activities that can be executed in parallel

Past work: example



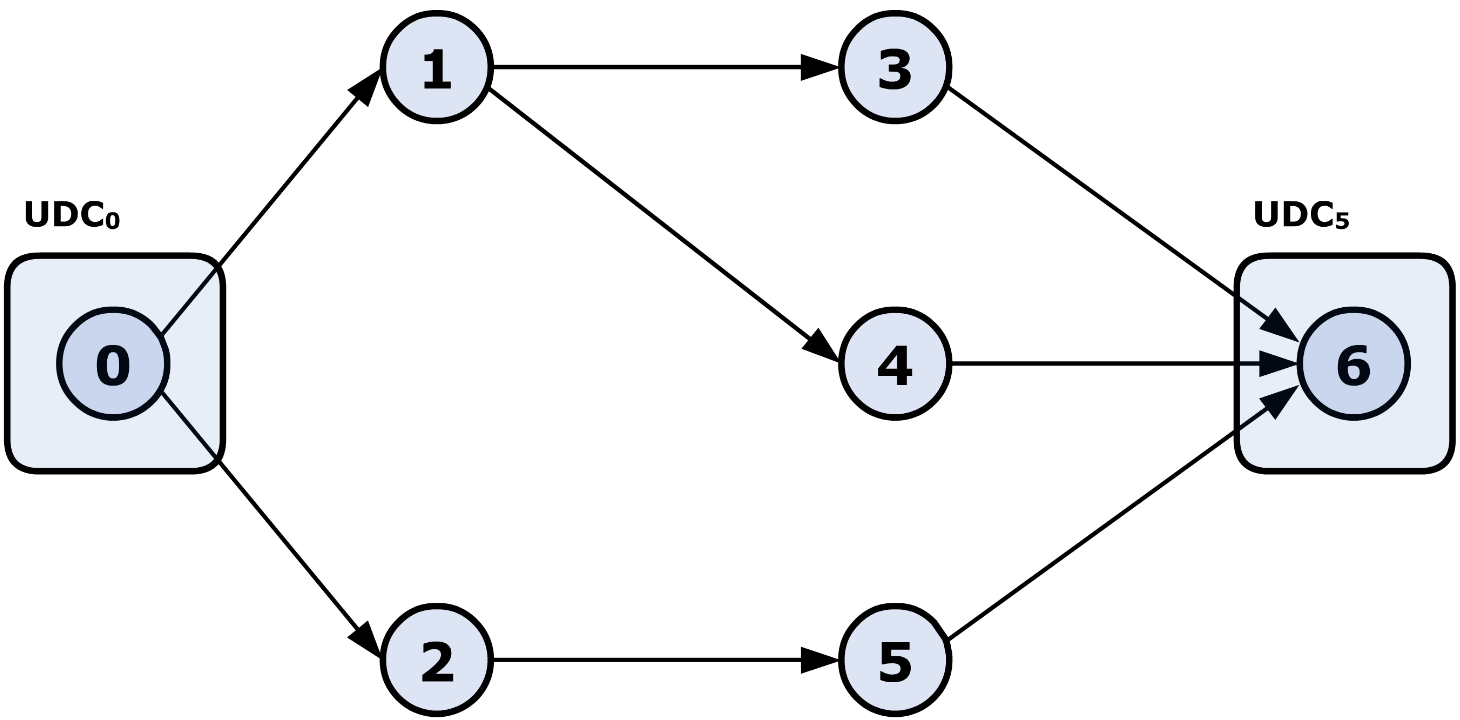
UDC = set of all activities that can be executed in parallel

Past work: example



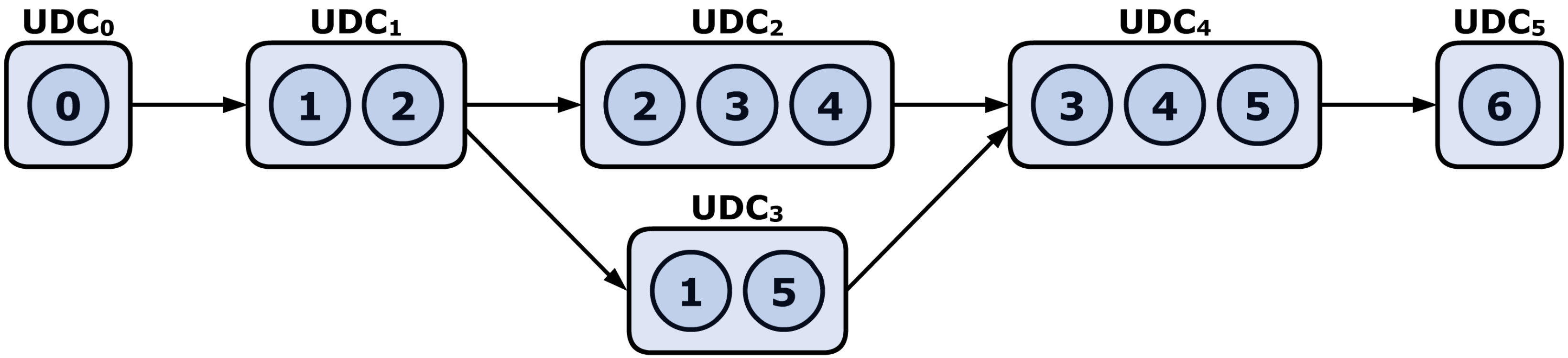
UDC = set of all activities that can be executed in parallel

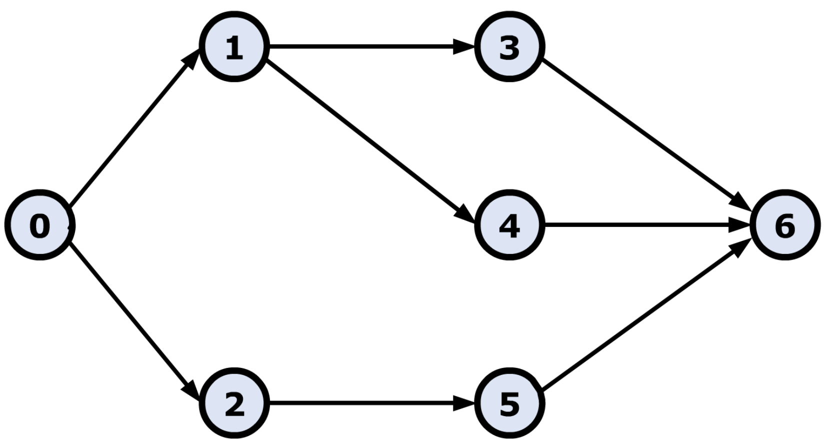
Past work: example



UDC = set of all activities that can be executed in parallel

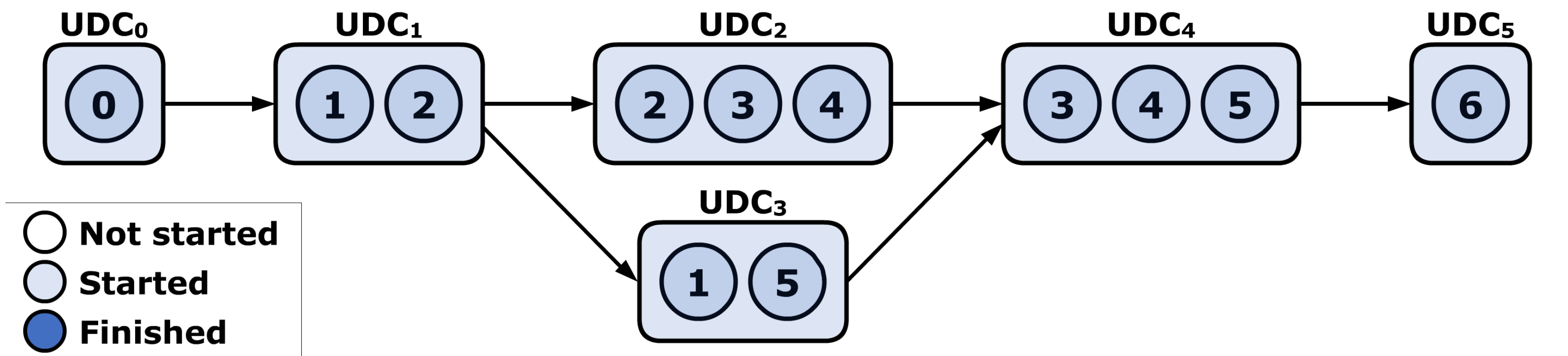
Past work: example

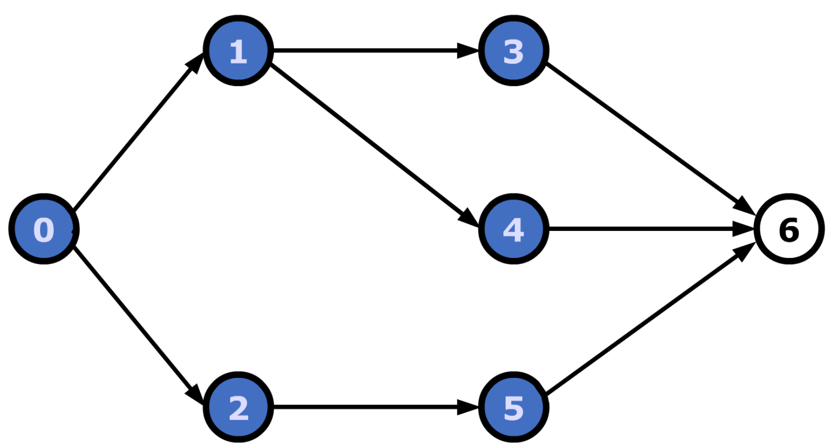


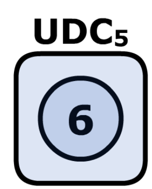


Network of UDCs

Past work: example





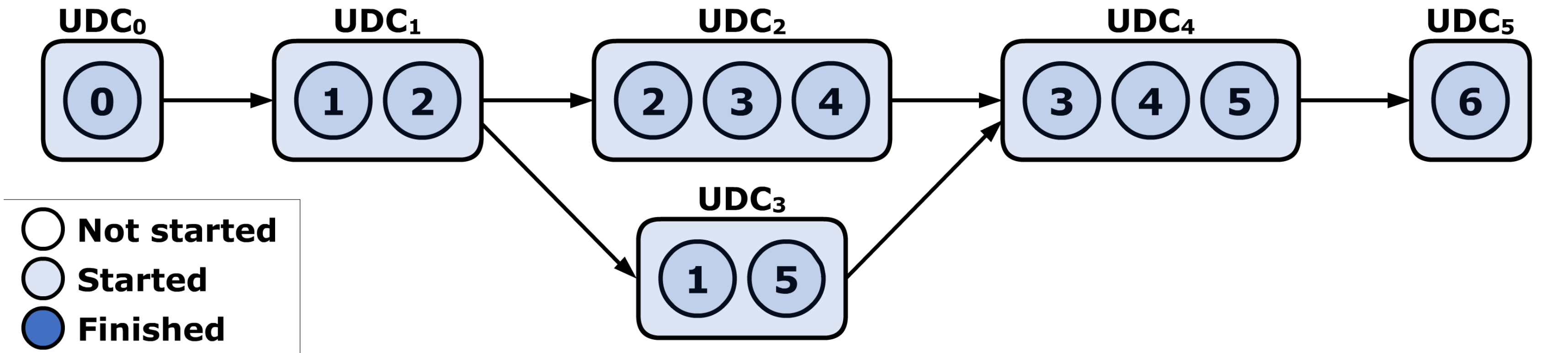


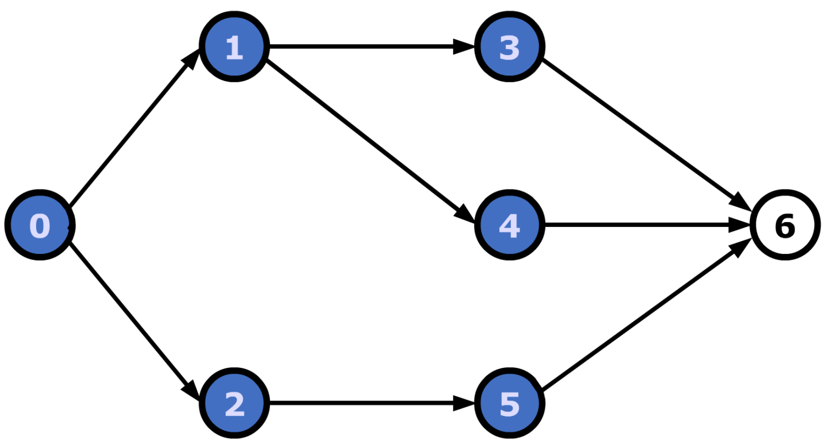
States assigned to UDC:

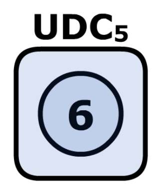
(2,2,2,2,2,2,0)

Illustration of state space & SDP recursion

Past work: example







States assigned to UDC:

(2,2,2,2,2,2,0) -> 80M$

Illustration of state space & SDP recursion

Past work: example

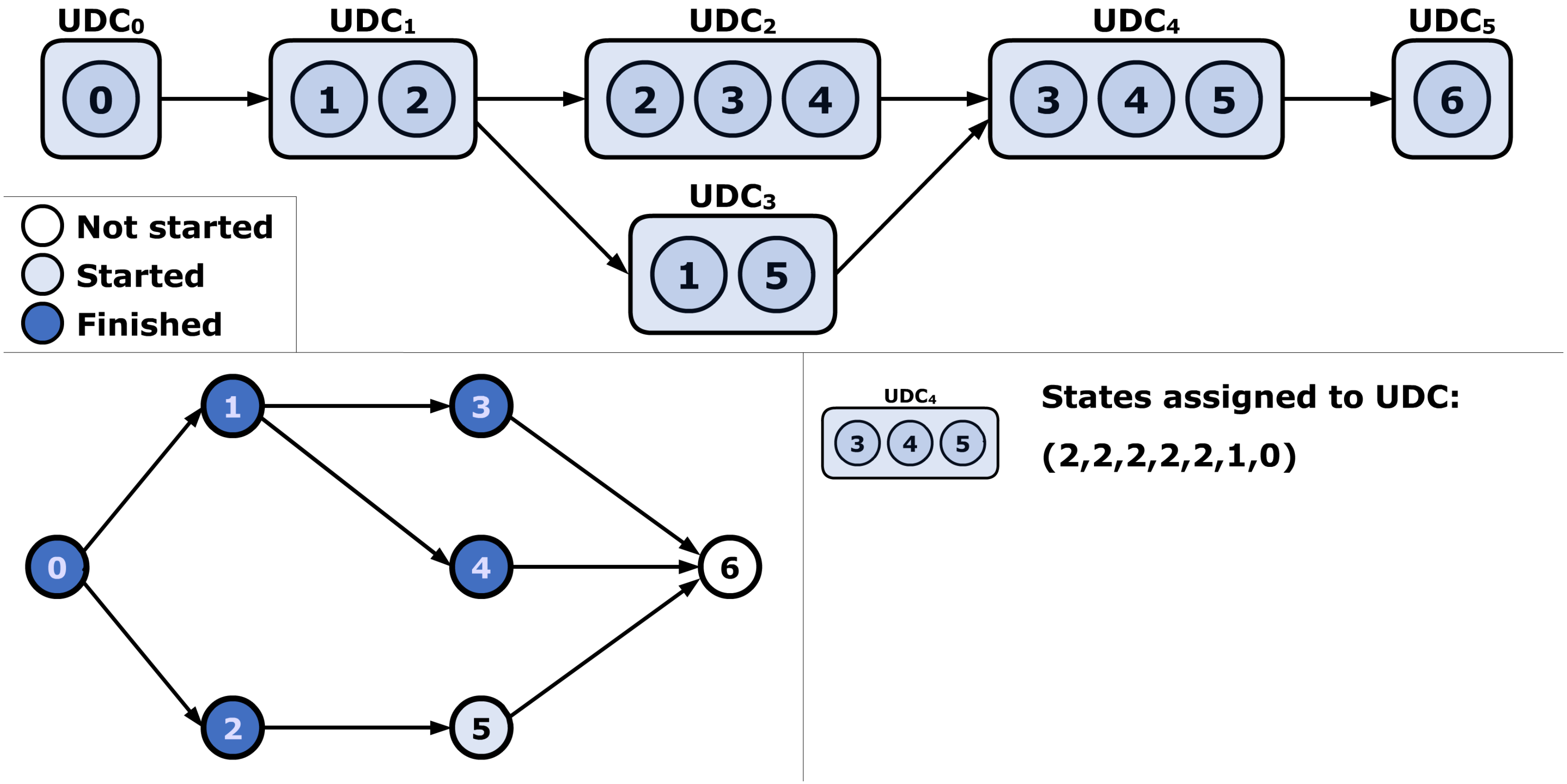


Illustration of state space & SDP recursion

Past work: example

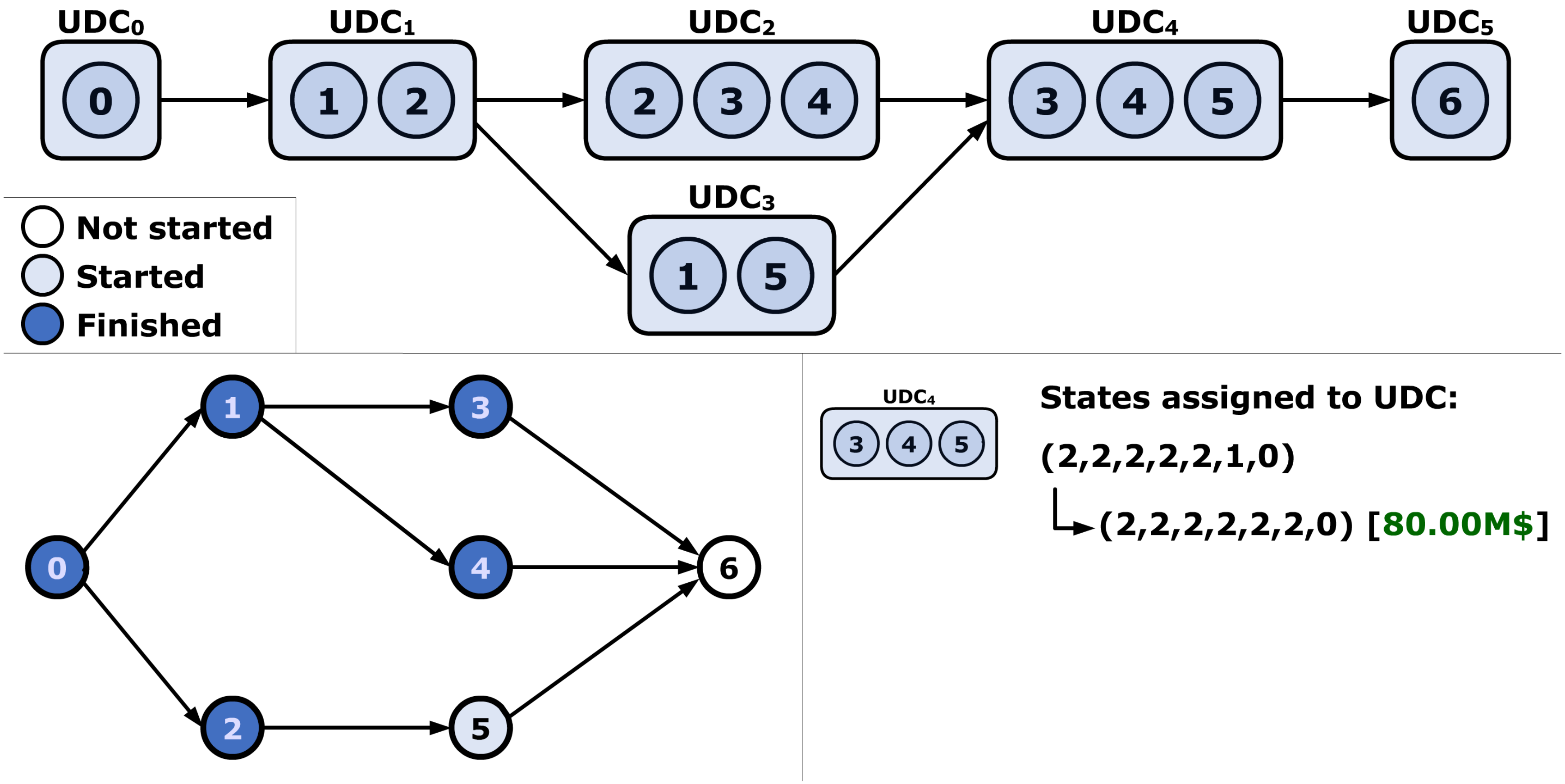
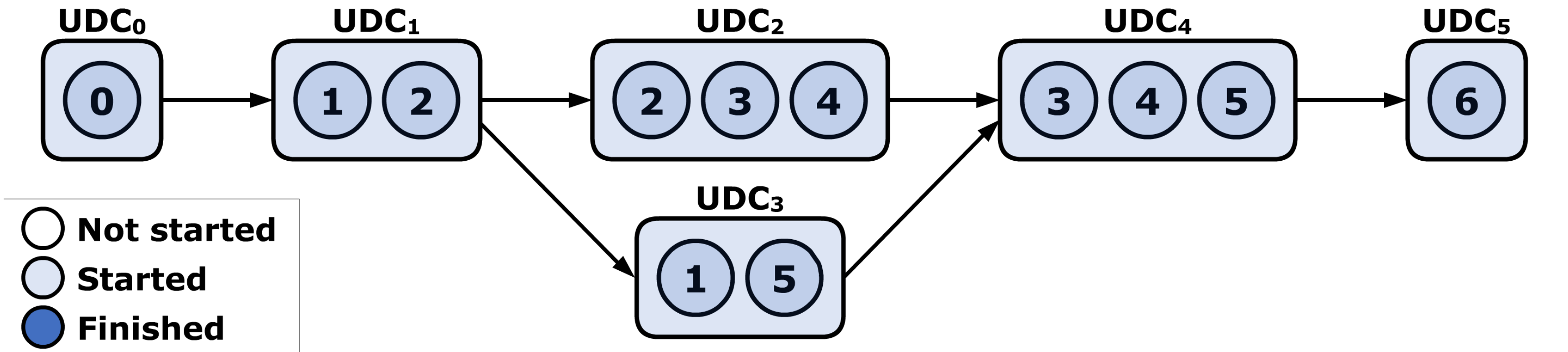
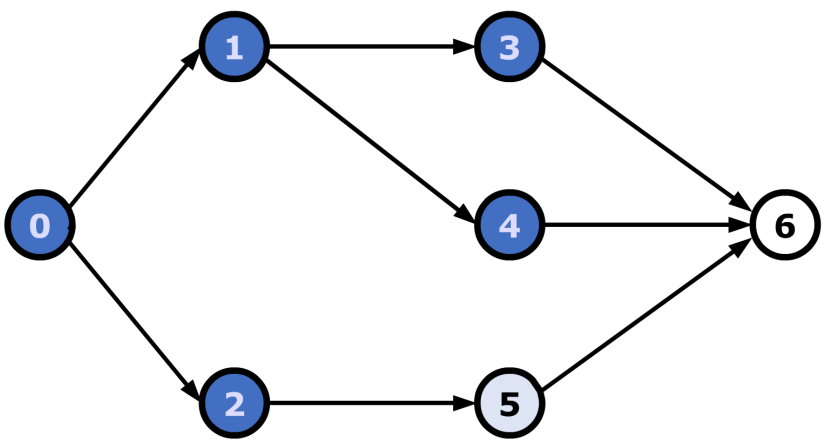
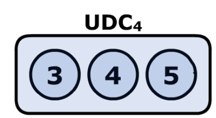


Illustration of state space & SDP recursion

Past work: example







States assigned to UDC:

(2,2,2,2,2,1,0)

→(2,2,2,2,2,2,0) [80.00M$]

Discount factor : $(1/d_i)\times(r+(1/d_i))^{-1}$

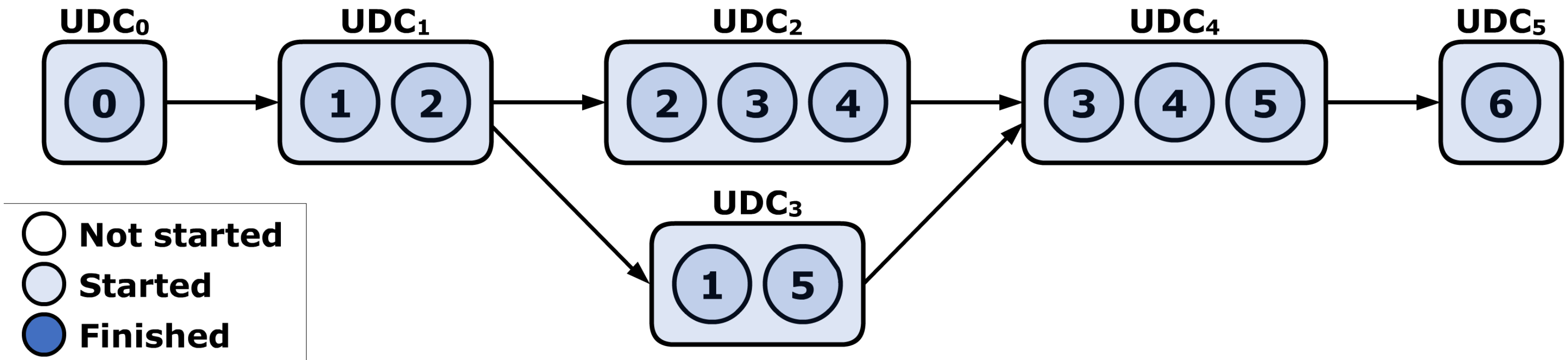
d_5 = 3 => Discount factor = 0.97

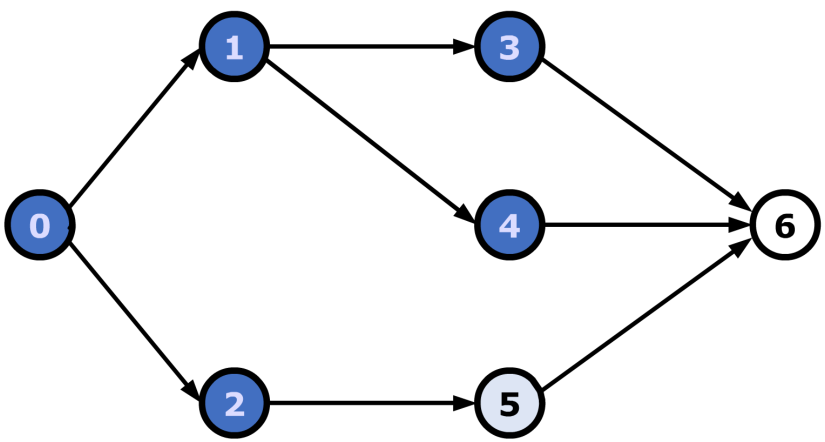
Discounted value at state entry = 77.67M$

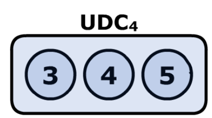
p_5 = 0.75 => NPV at state entry = 58.25M$

Illustration of state space & SDP recursion

Past work: example







States assigned to UDC:

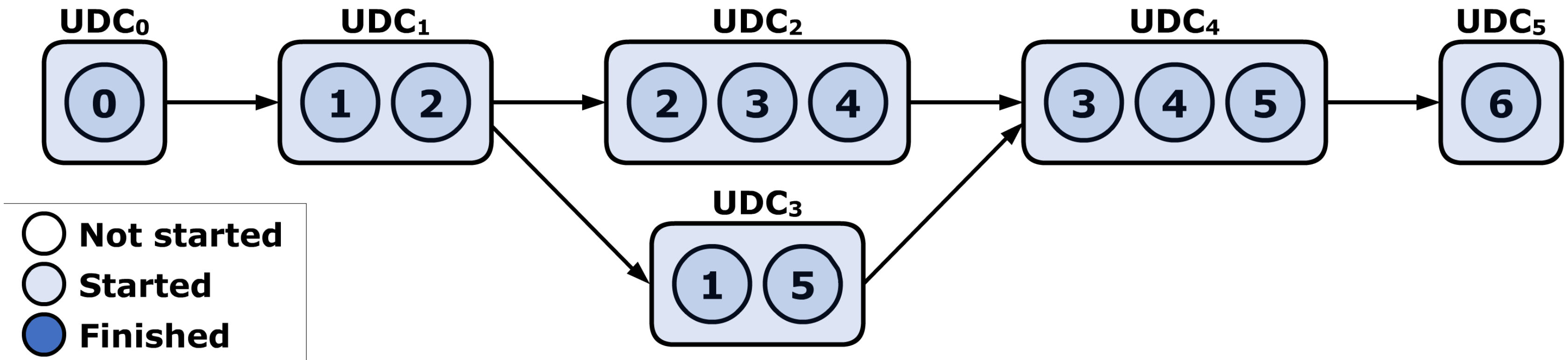
(2,2,2,2,2,1,0) -> 58.25M$

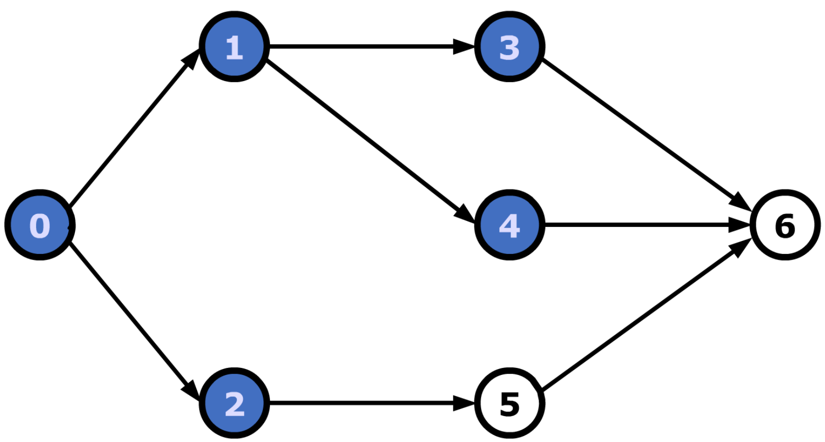
(2,2,2,2,2,2,0) [80.00M$]

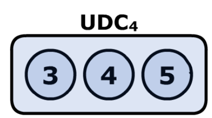
Discount factor : $(1/d_i)\times(r+(1/d_i))^{-1}$
d_5 = 3 => Discount factor = 0.97
Discounted value at state entry = 77.67M$
p_5 = 0.75 => NPV at state entry = 58.25M$

Illustration of state space & SDP recursion

Past work: example





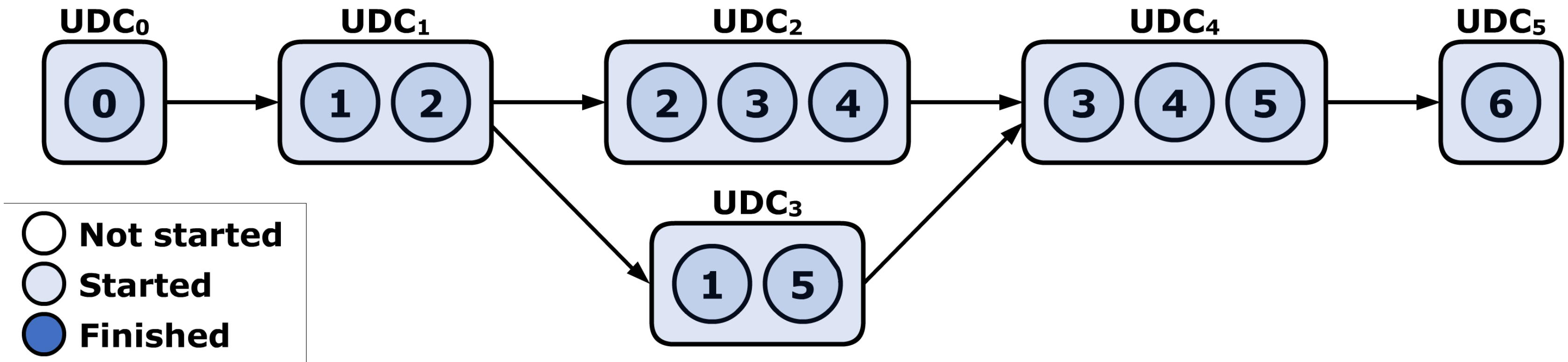


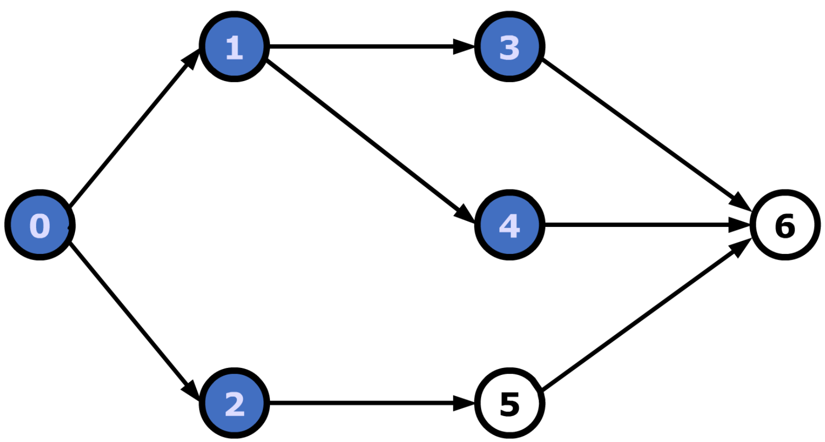
States assigned to UDC:

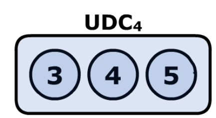
(2,2,2,2,2,1,0) -> 58.25M$
(2,2,2,2,2,0,0)

Illustration of state space & SDP recursion

Past work: example





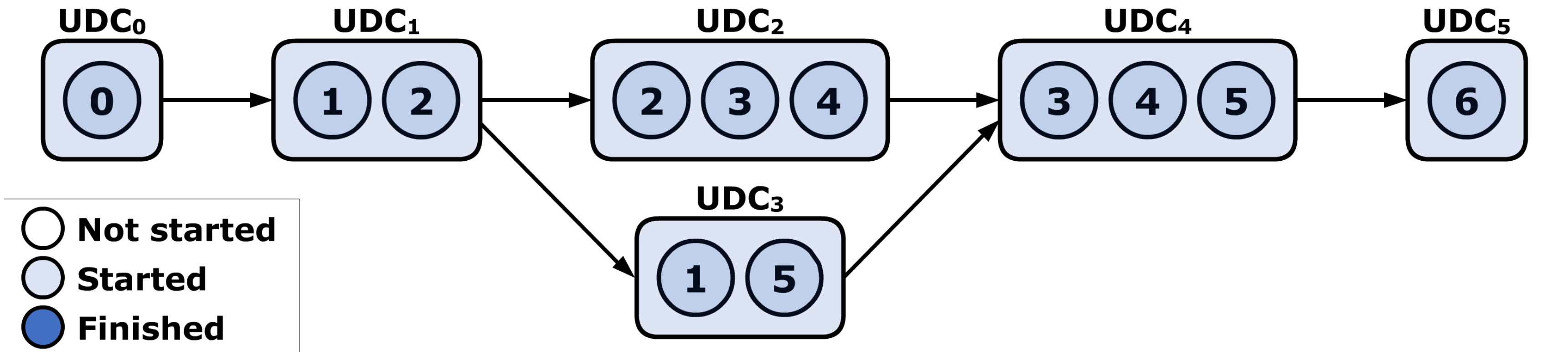


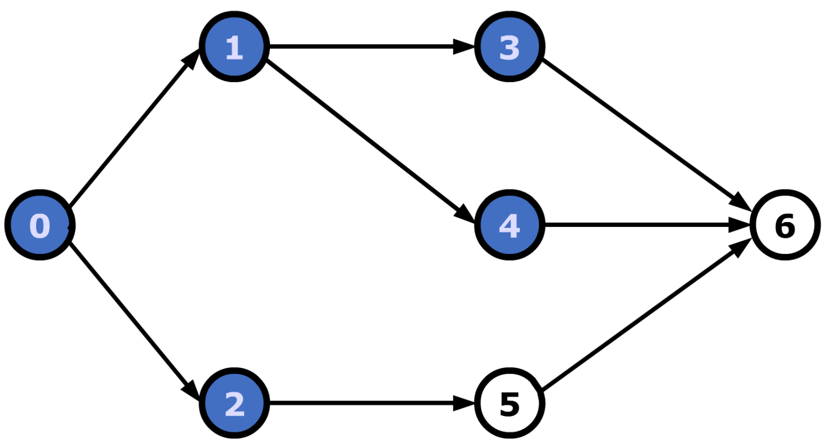
States assigned to UDC:

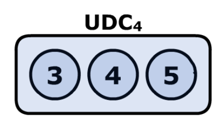
(2,2,2,2,2,1,0) -> 58.25M$
(2,2,2,2,2,0,0)
└► (2,2,2,2,2,1,0) [58.25M$]

Illustration of state space & SDP recursion

Past work: example







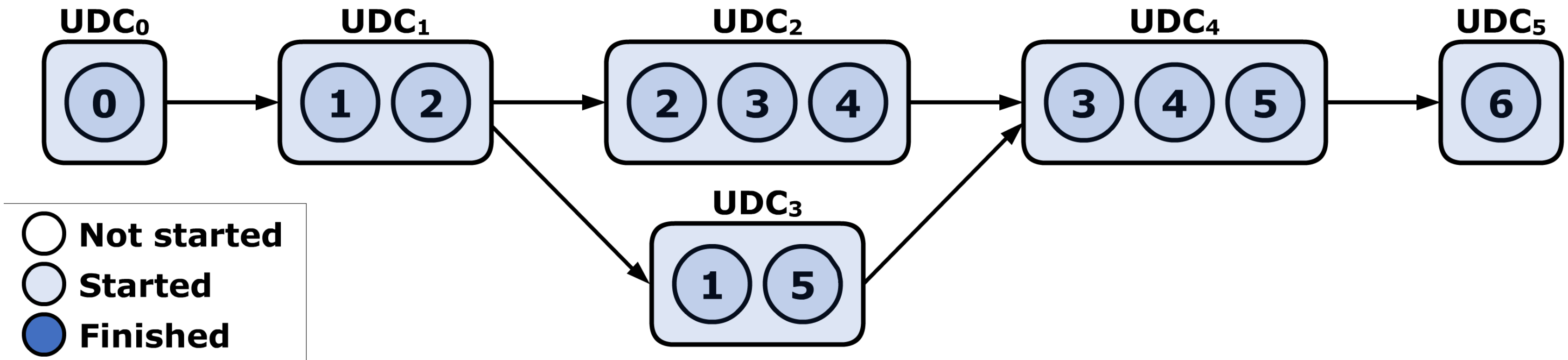
States assigned to UDC:

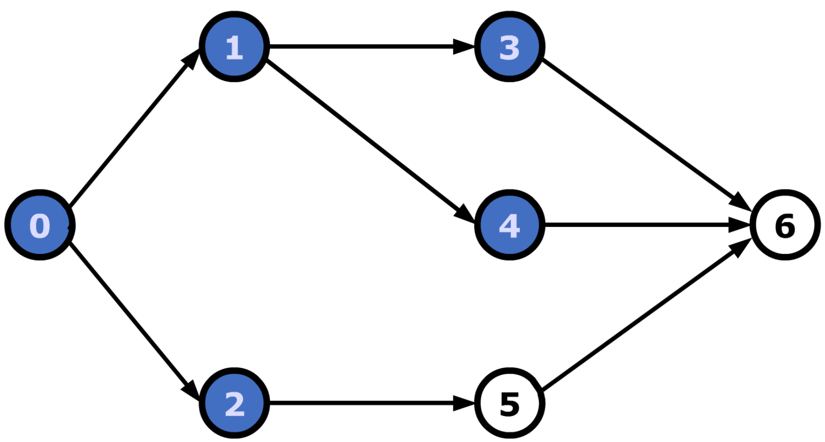
(2,2,2,2,2,1,0) -> 58.25M$
(2,2,2,2,2,0,0)
(2,2,2,2,2,1,0) [58.25M$]

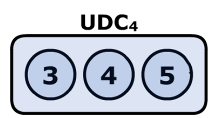
Only decision left is to start activity 5
=> incur cost c_5 = -17M$
=> NPV at state entry = 41.25M$

Illustration of state space & SDP recursion

Past work: example







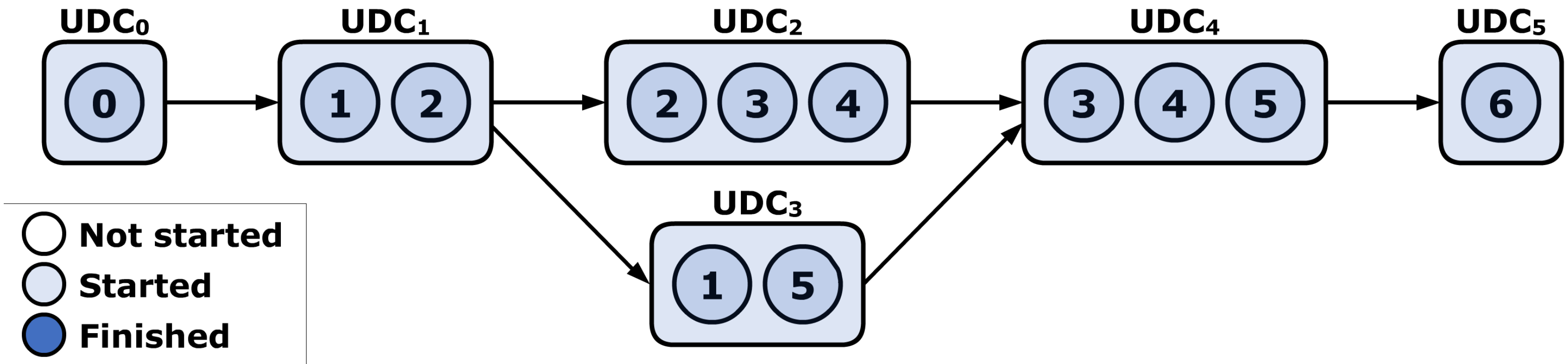
States assigned to UDC:

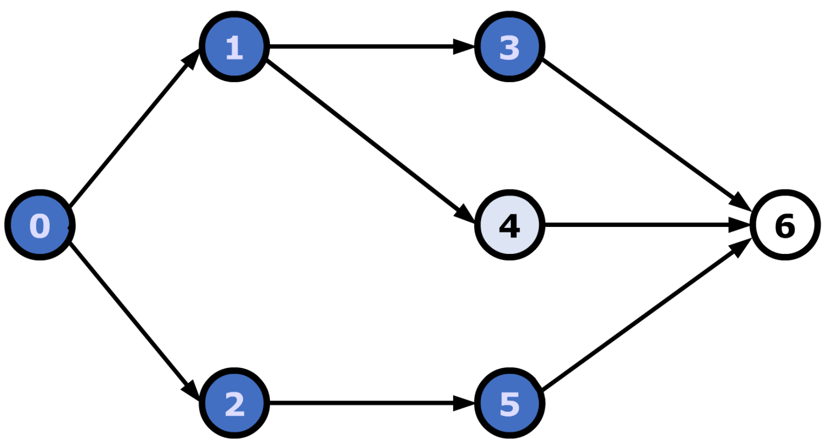
(2,2,2,2,2,1,0) -> 58.25M$
(2,2,2,2,2,0,0) -> 41.25M$
↳ (2,2,2,2,2,1,0) [58.25M$]

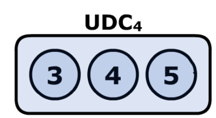
Only decision left is to start activity 5
=> incur cost c_5 = -17M$
=> NPV at state entry = 41.25M$

Illustration of state space & SDP recursion

Past work: example







States assigned to UDC:

(2,2,2,2,2,1,0) -> 58.25M$
(2,2,2,2,2,0,0) -> 41.25M$
(2,2,2,2,1,2,0)

Illustration of state space & SDP recursion

Past work: example

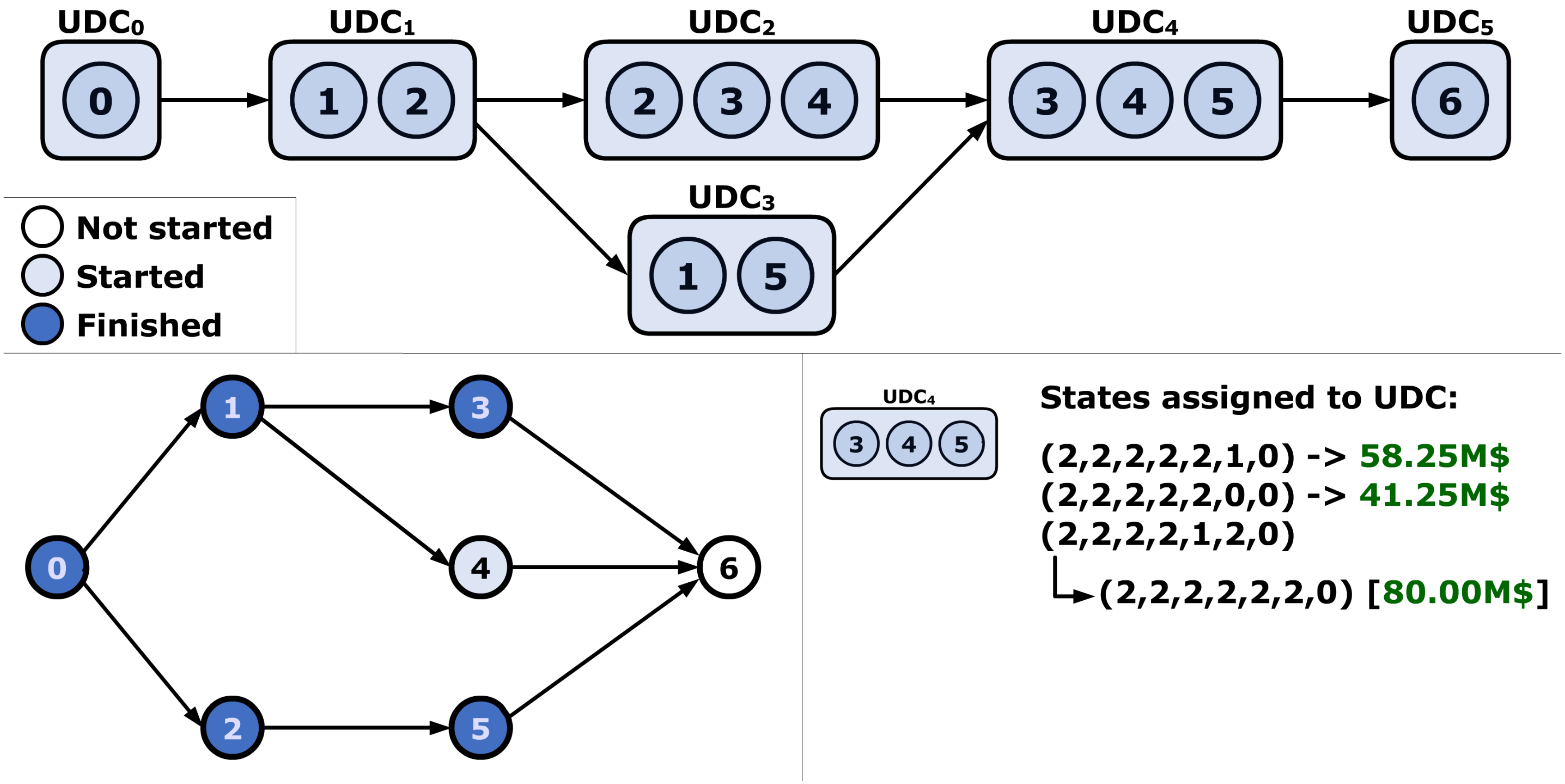


Illustration of state space & SDP recursion

Past work: example

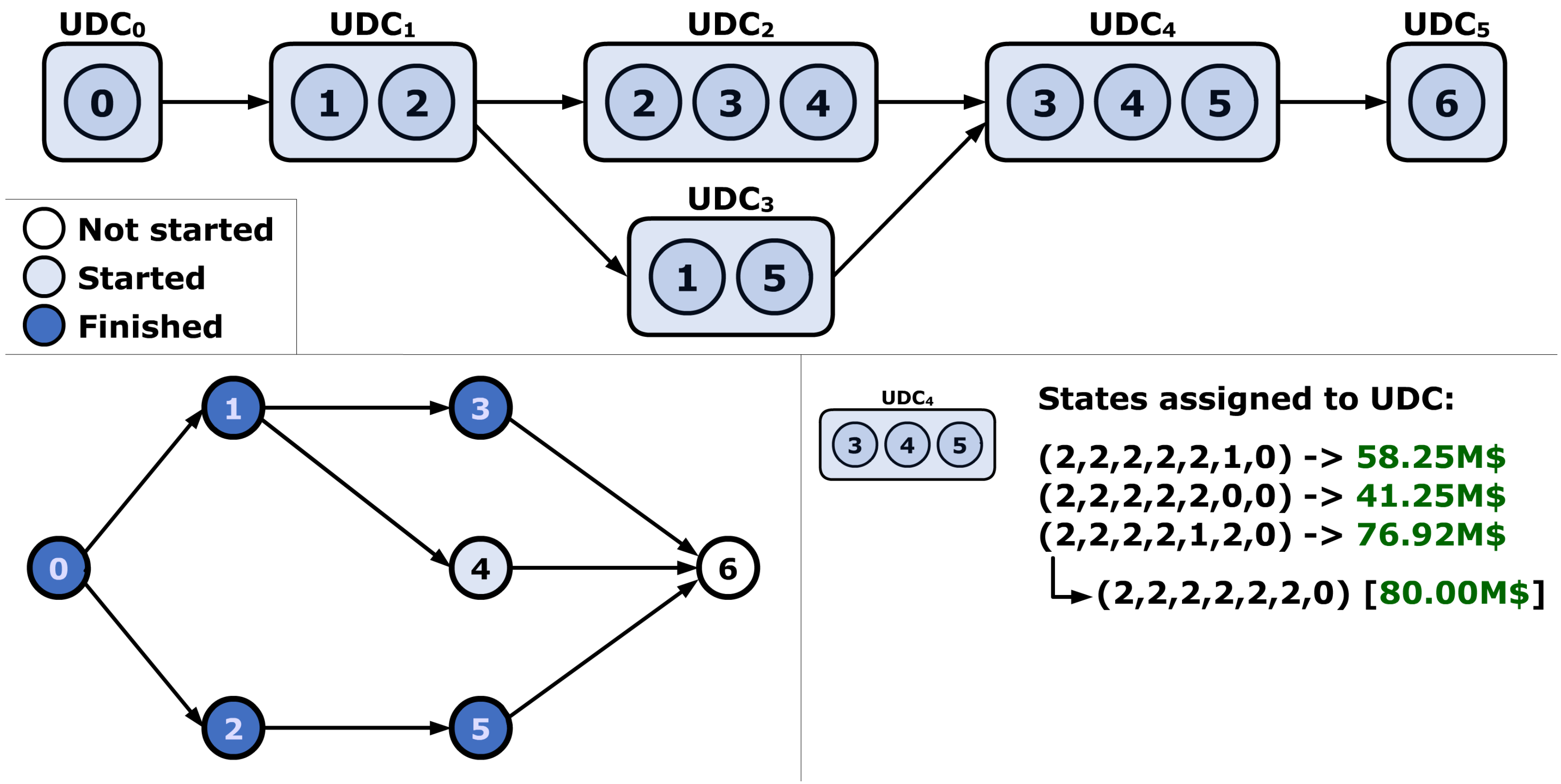


Illustration of state space & SDP recursion

Past work: example

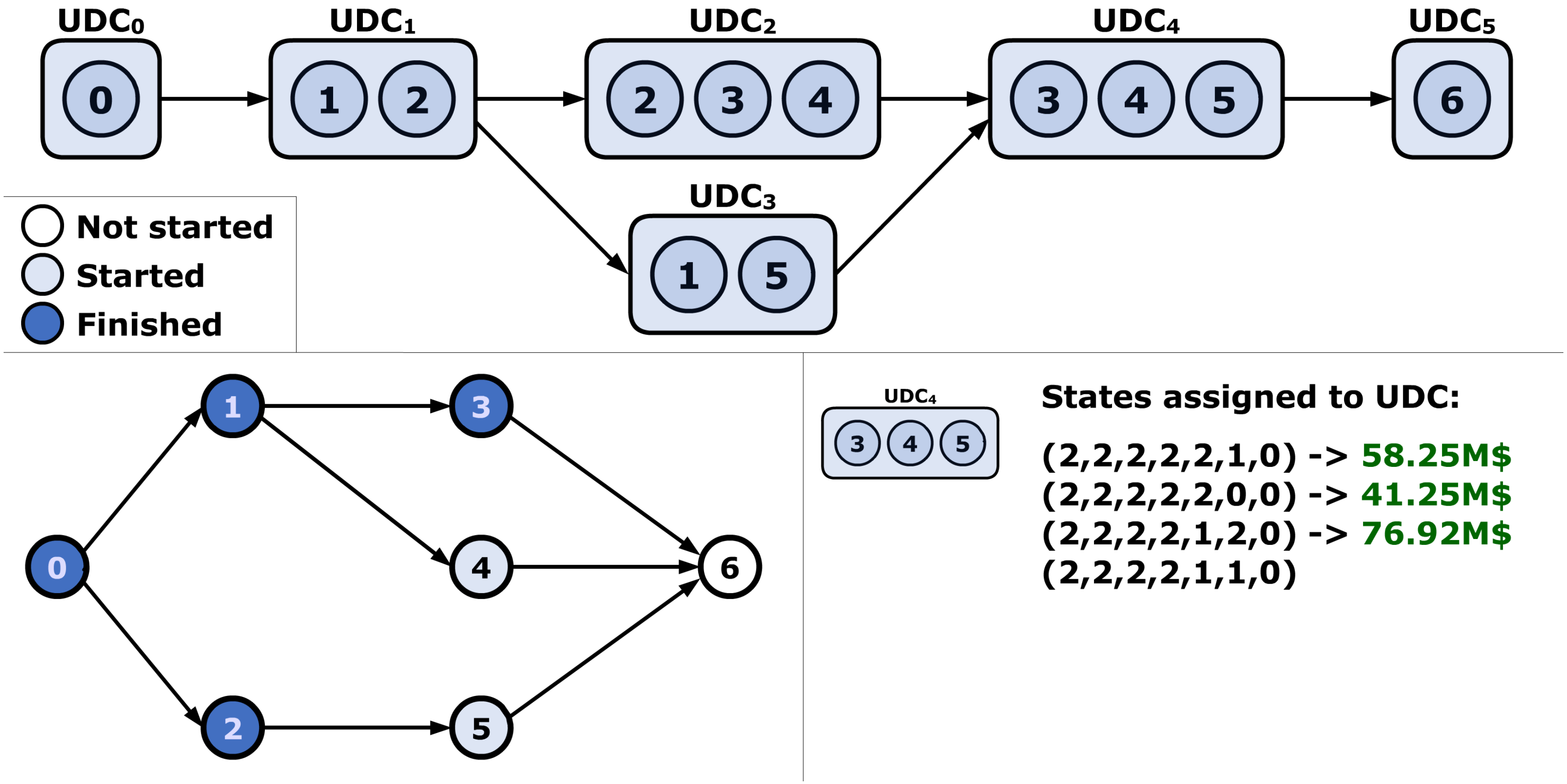
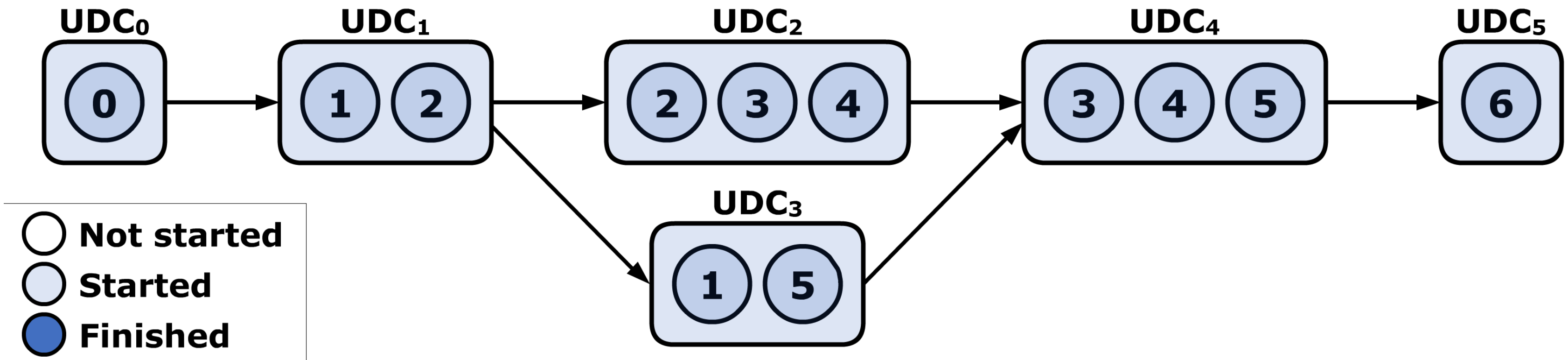
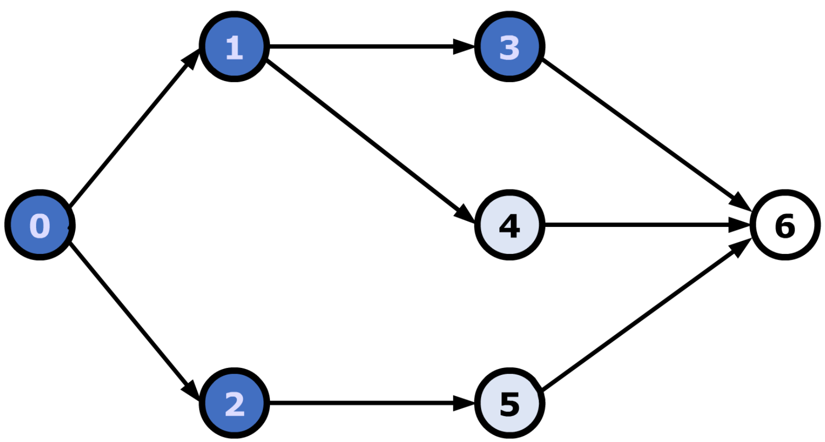
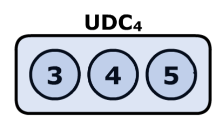


Illustration of state space & SDP recursion

Past work: example





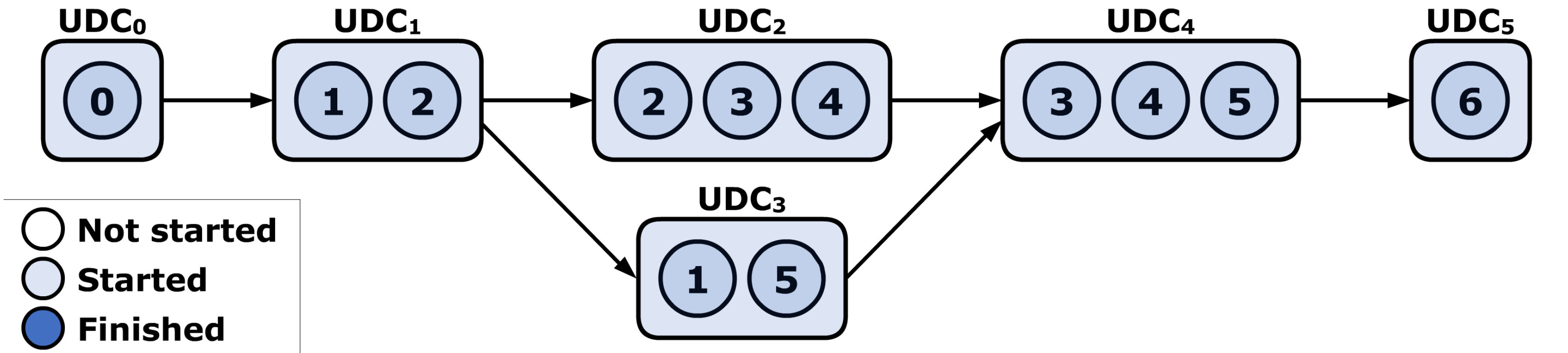


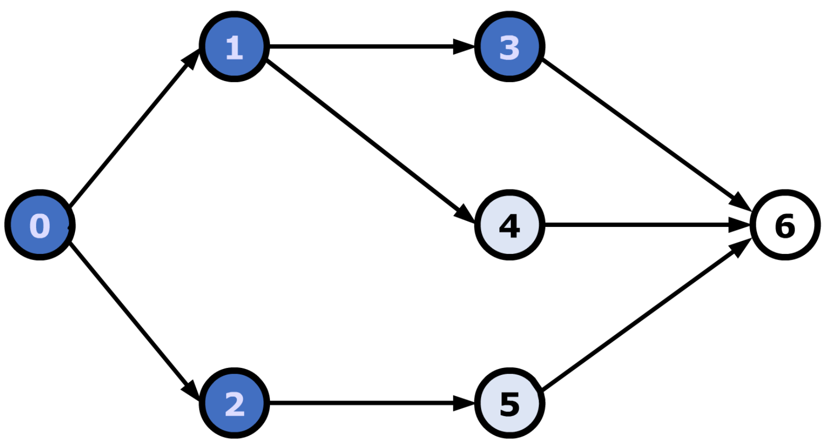
States assigned to UDC:

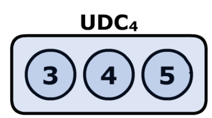
(2,2,2,2,2,1,0) -> 58.25M$
(2,2,2,2,2,0,0) -> 41.25M$
(2,2,2,2,1,2,0) -> 76.92M$
(2,2,2,2,1,1,0)
- (2,2,2,2,2,1,0) [58.25M$]
- (2,2,2,2,1,2,0) [76.92M$]

Illustration of state space & SDP recursion

Past work: example







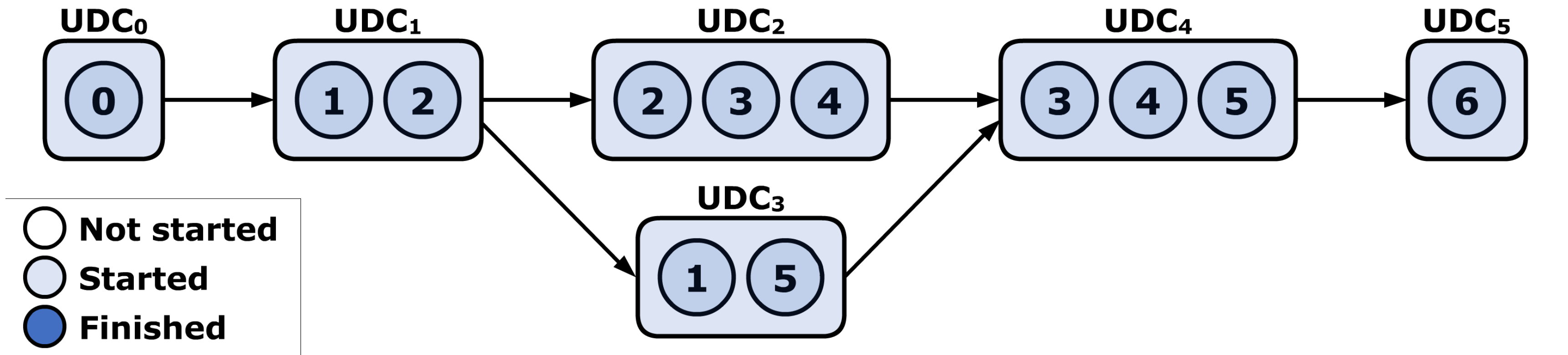
States assigned to UDC:

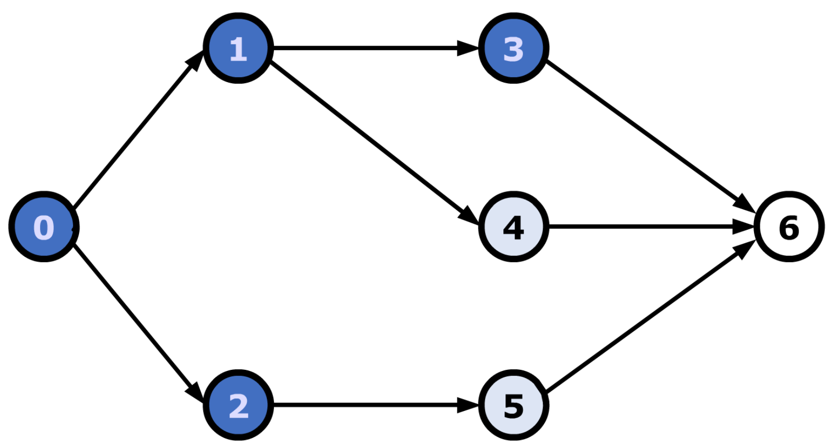
(2,2,2,2,2,1,0) -> 58.25M$
(2,2,2,2,2,0,0) -> 41.25M$
(2,2,2,2,1,2,0) -> 76.92M$
(2,2,2,2,1,1,0)
→(2,2,2,2,2,1,0) [58.25M$]
→(2,2,2,2,1,2,0) [76.92M$]

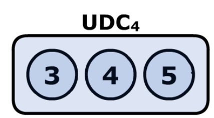
Probability activity finishing first
=> $(1/d_i)\times(\mathrm{SUM}(1/d_i))^{-1}$

Illustration of state space & SDP recursion

Past work: example







States assigned to UDC:

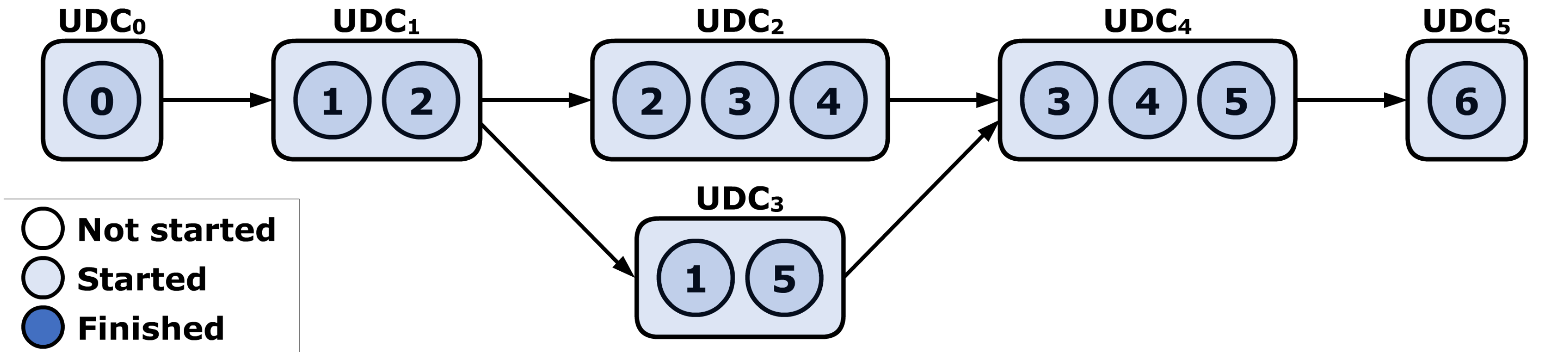
(2,2,2,2,2,1,0) -> 58.25M$
(2,2,2,2,2,0,0) -> 41.25M$
(2,2,2,2,1,2,0) -> 76.92M$
(2,2,2,2,1,1,0)

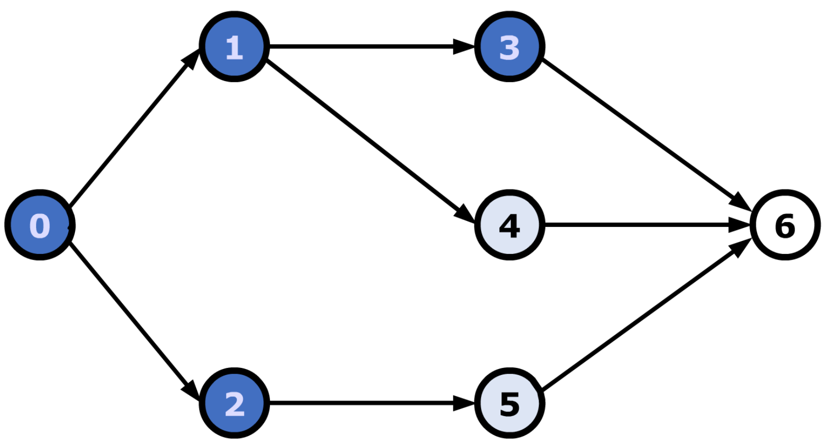
43% → (2,2,2,2,2,1,0) [58.25M$]
57% → (2,2,2,2,1,2,0) [76.92M$]

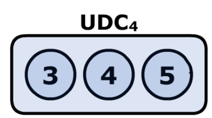
Probability activity finishing first
=> $(1/d_i) \times (\mathrm{SUM}(1/d_i))^{-1}$

Illustration of state space & SDP recursion

Past work: example







States assigned to UDC:

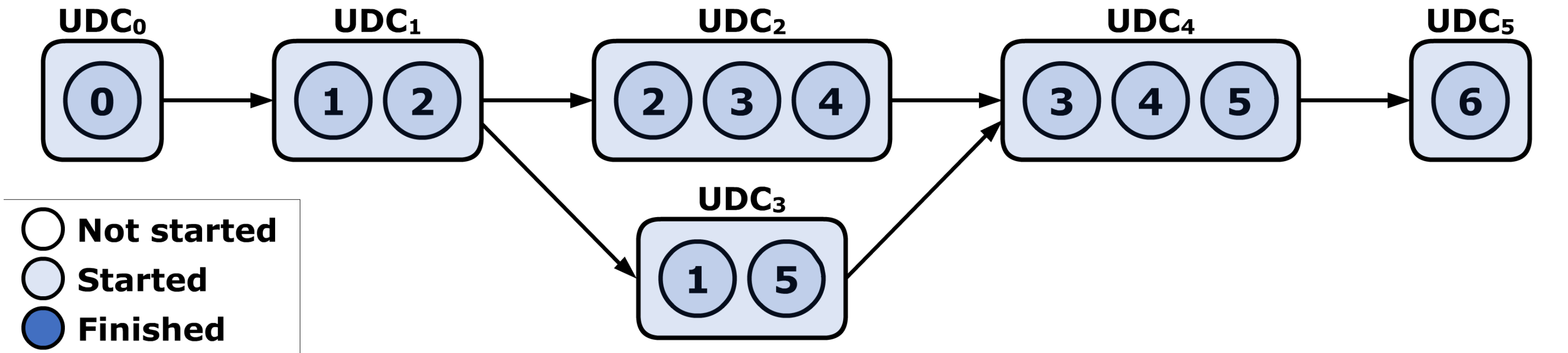
(2,2,2,2,2,1,0) -> 58.25M$
(2,2,2,2,2,0,0) -> 41.25M$
(2,2,2,2,1,2,0) -> 76.92M$
(2,2,2,2,1,1,0) -> 56.96M$

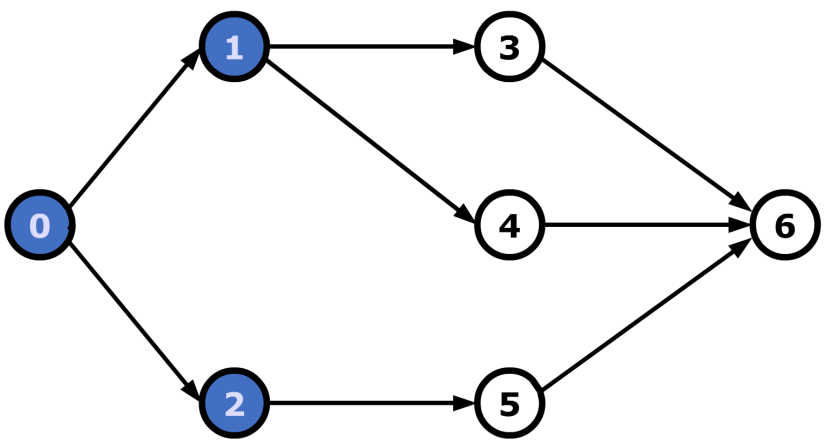
43% → (2,2,2,2,2,1,0) [58.25M$]
57% → (2,2,2,2,1,2,0) [76.92M$]

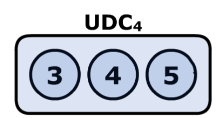
Probability activity finishing first
=> $(1/d_i)\times(\mathrm{SUM}(1/d_i))^{-1}$

Illustration of state space & SDP recursion

Past work: example





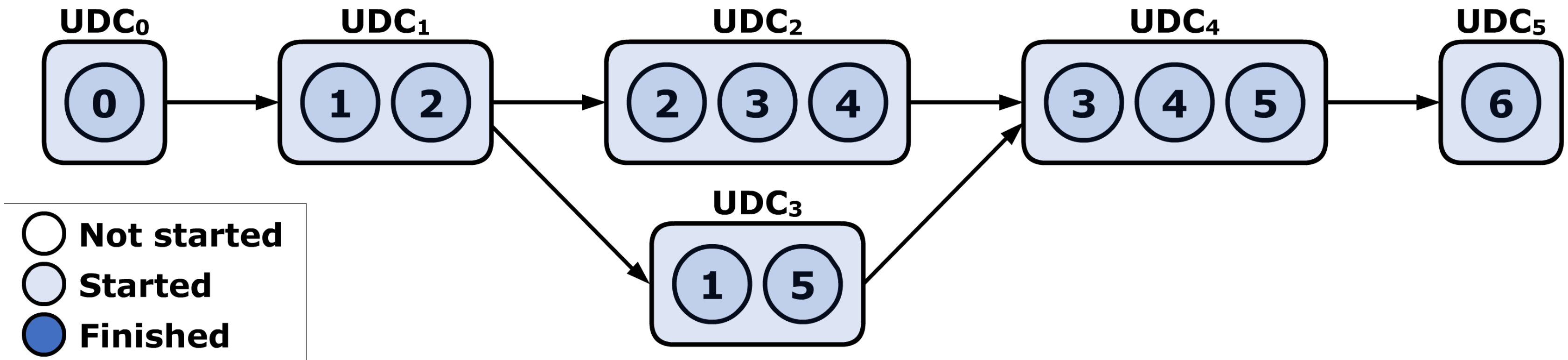


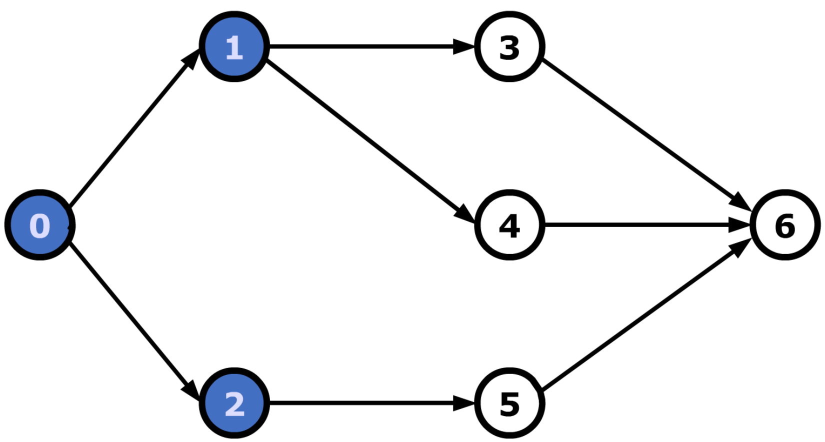
States assigned to UDC:

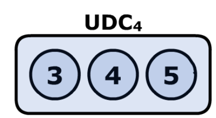
(2,2,2,0,0,0,0)

Illustration of state space & SDP recursion

Past work: example







States assigned to UDC:

(2,2,2,0,0,0,0)

- **(2,2,2,1,0,0,0) [21.51M$]**
- **(2,2,2,0,1,0,0) [17.46M$]**
- **(2,2,2,0,0,1,0) [18.26M$]**
- **(2,2,2,1,1,0,0) [17.46M$]**
- **(2,2,2,1,0,1,0) [18.26M$]**
- **(2,2,2,0,1,1,0) [14.26M$]**
- **(2,2,2,1,1,1,0) [14.17M$]**

Illustration of state space & SDP recursion

Past work: example

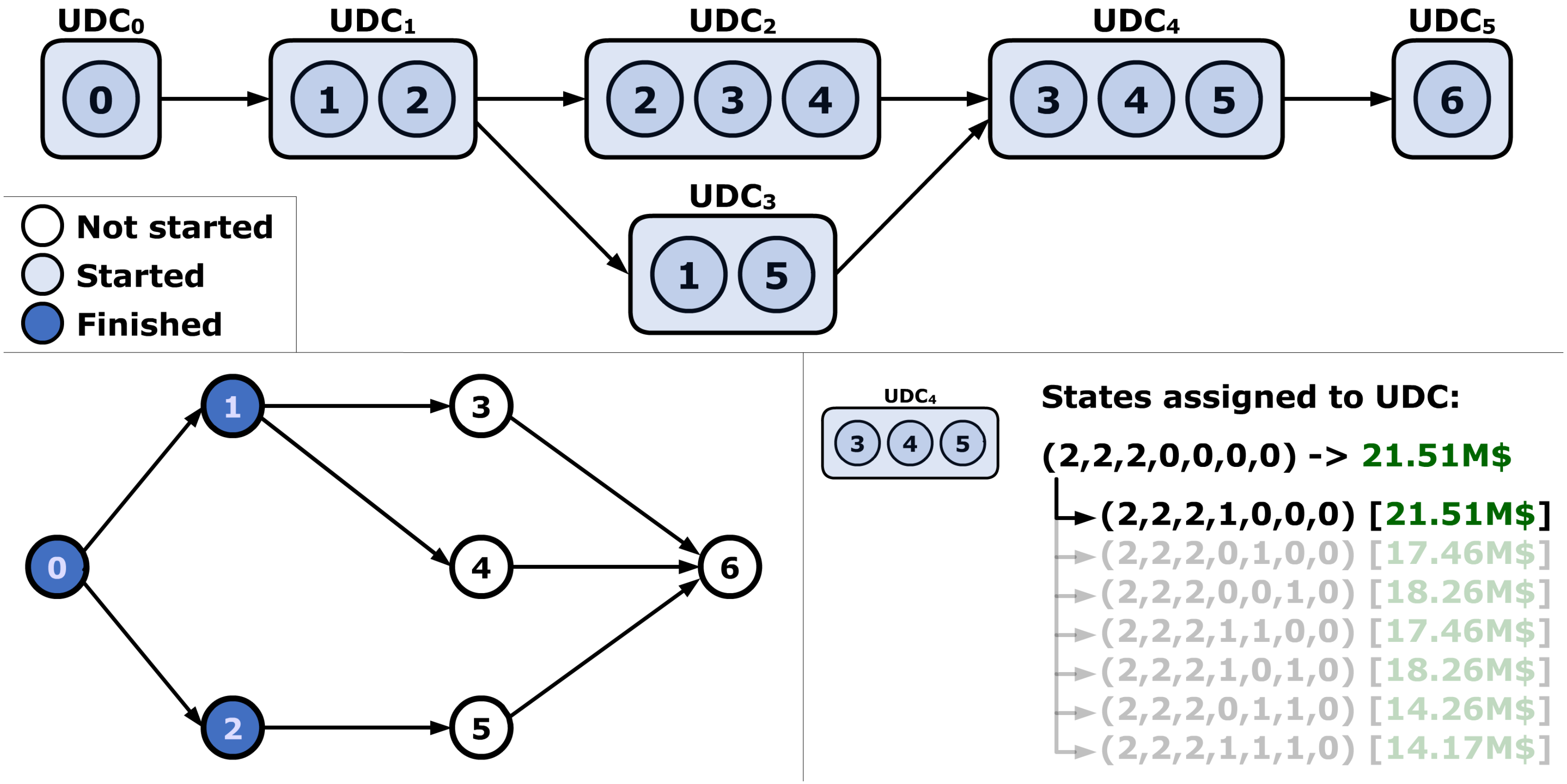


Illustration of state space & SDP recursion

Past work: example

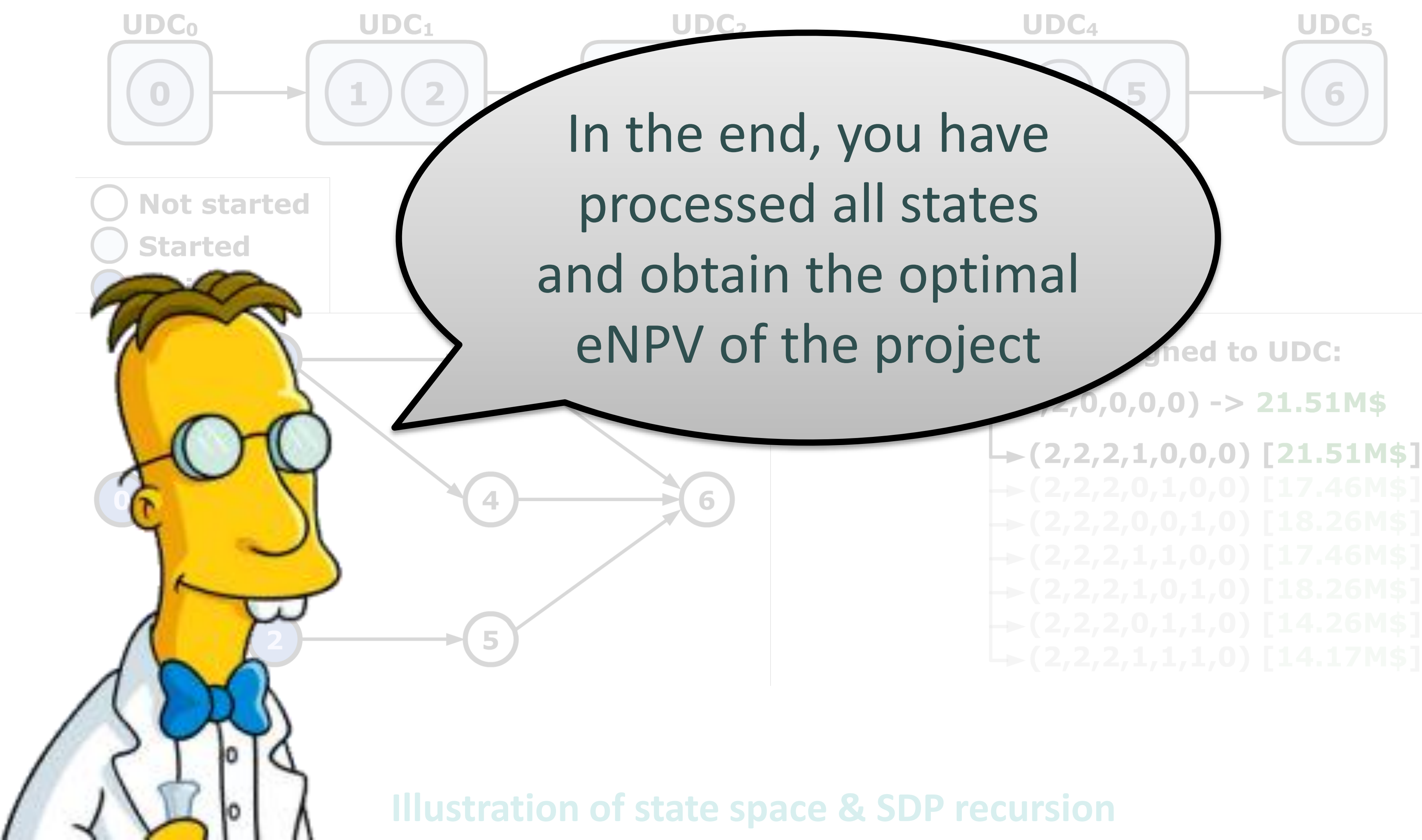


Illustration of state space & SDP recursion

Agenda

- Past work
- New approach
- What about the SRCPSP?
- Contribution

New approach

PAST WORK

1. SDP recursion
2. Optimal solution
3. General activity durations
4. eNPV & SRCPSP
5. UDCs to structure state space
6. Upper bound state space = 3^n

New approach

PAST WORK

1. SDP recursion
2. Optimal solution
3. General activity durations
4. eNPV & SRCPSP
5. UDCs to structure state space
6. Upper bound state space = 3^n

Main bottleneck = memory!

New approach

PAST WORK

1. SDP recursion
2. Optimal solution
3. General activity durations
4. eNPV & SRCPSP
5. UDCs to structure state space
6. Upper bound state space = 3^n

Main bottleneck = memory!

?

NEW APPROACH

1. SDP recursion
2. Optimal solution
3. General activity durations
4. eNPV (SRCPSP = see infra)
5. No UDCs
6. Upper bound state space = 2^n

New approach

How does
he do that?

1.
2. Optimal so
3. Gen activ urations
4. SRC
5. st e state space
6. Upp bou ate space = 3^n

Main b neck = memory!

W
ACH

1. ursion
2. Optimal solution
3. General activity durations
4. eNPV (SRCPSP = see infra)
5. No UDCs
6. Upper bound state space = 2^n

New approach: relax state space definition

- Before, a state was defined by the set of idle, busy, and finished activities
- In the new approach, a state is defined only by the set of finished activities

New approach: relax state space definition

- Before, a state was defined by the set of idle, busy, and finished activities
- In the new approach, a state is defined only by the set of finished activities

$\Rightarrow$We don’t know which activities are ongoing!

New approach: relax state space definition

- Before, a state was defined by the set of idle, busy, and finished activities
- In the new approach, a state is defined only by the set of finished activities

$\Rightarrow$We don’t know which activities are ongoing!

- We, however, do know the set of activities that are eligible to start

New approach: relax state space definition

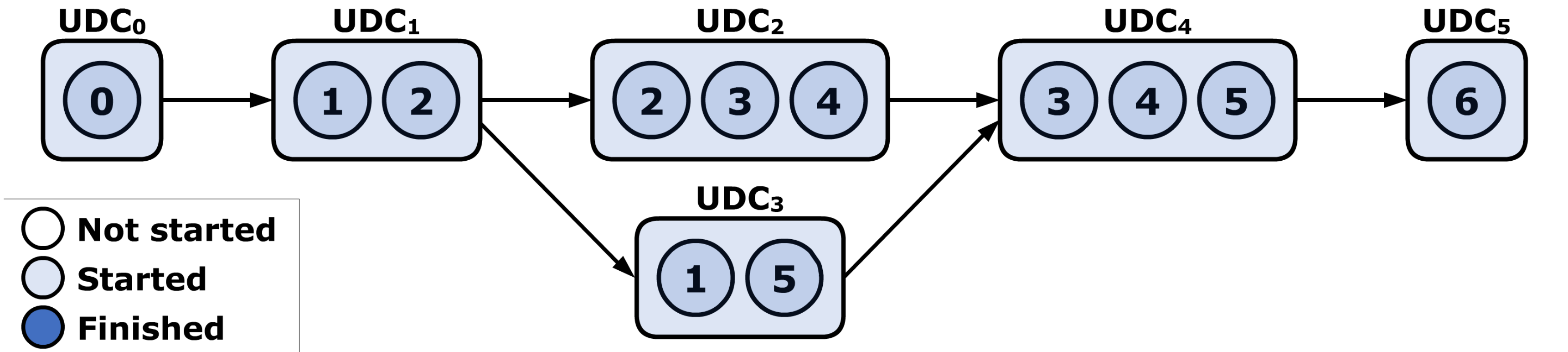
- Before, a state was defined by the set of idle, busy, and finished activities
- In the new approach, a state is defined only by the set of finished activities

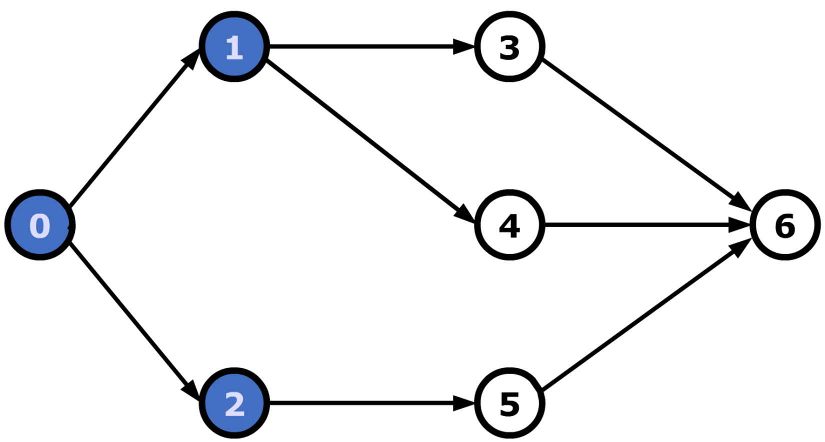
$\Rightarrow$We don’t know which activities are ongoing!

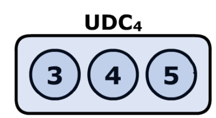
- We, however, do know the set of activities that are eligible to start

$\Rightarrow$We can determine the optimal set of ongoing activities (i.e., the set of ongoing activities that maximizes the eNPV)

New approach: relax state space definition





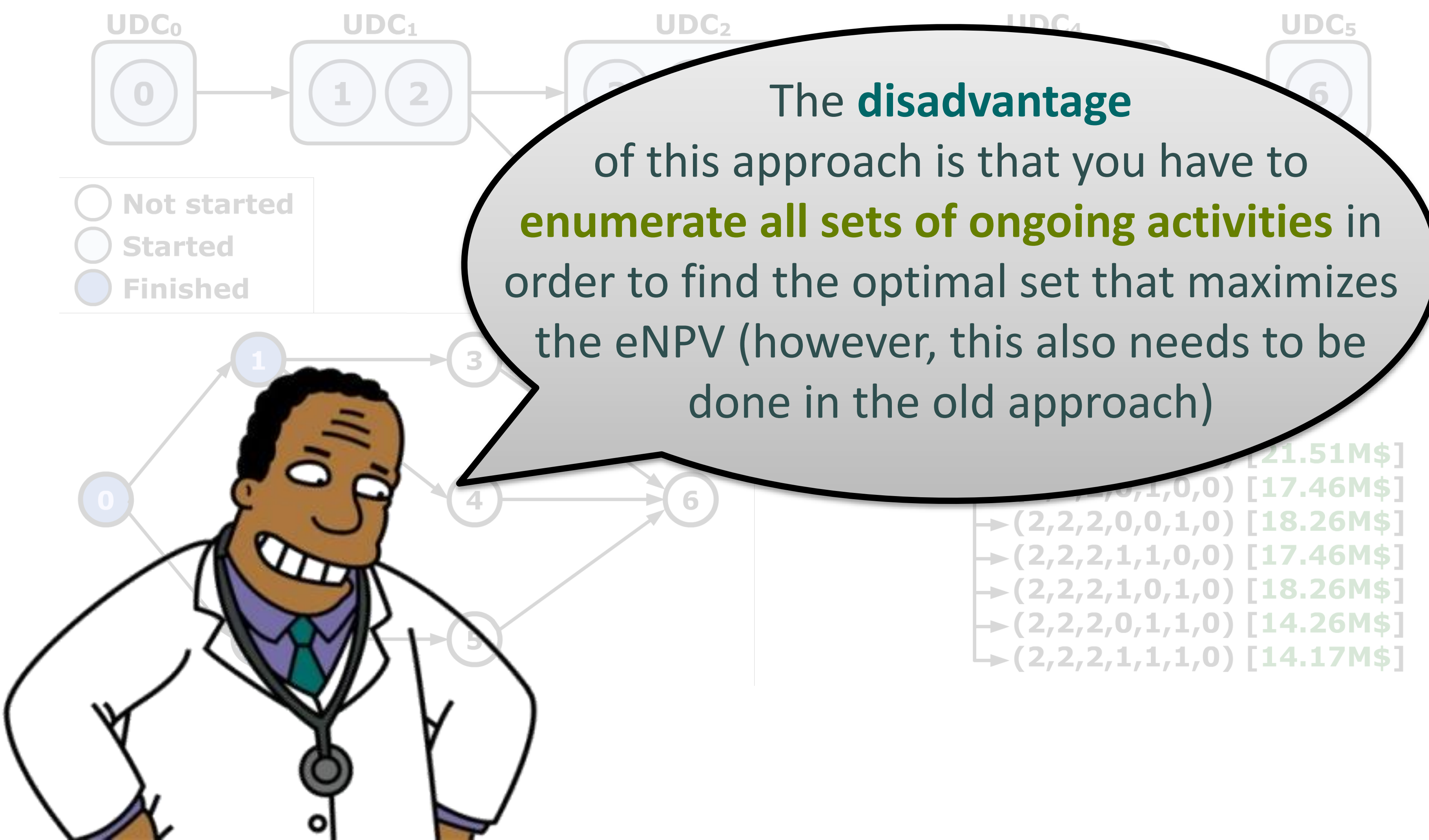


States assigned to UDC:

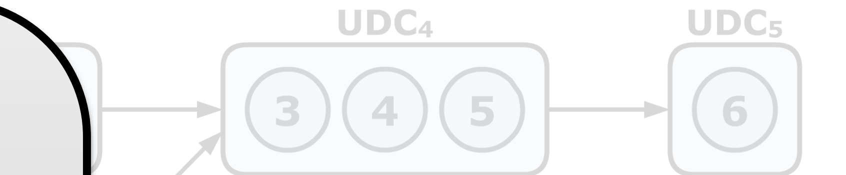
(2,2,2,0,0,0,0)

- **(2,2,2,1,0,0,0) [21.51M$]**
- **(2,2,2,0,1,0,0) [17.46M$]**
- **(2,2,2,0,0,1,0) [18.26M$]**
- **(2,2,2,1,1,0,0) [17.46M$]**
- **(2,2,2,1,0,1,0) [18.26M$]**
- **(2,2,2,0,1,1,0) [14.26M$]**
- **(2,2,2,1,1,1,0) [14.17M$]**

New approach: relax state space definition

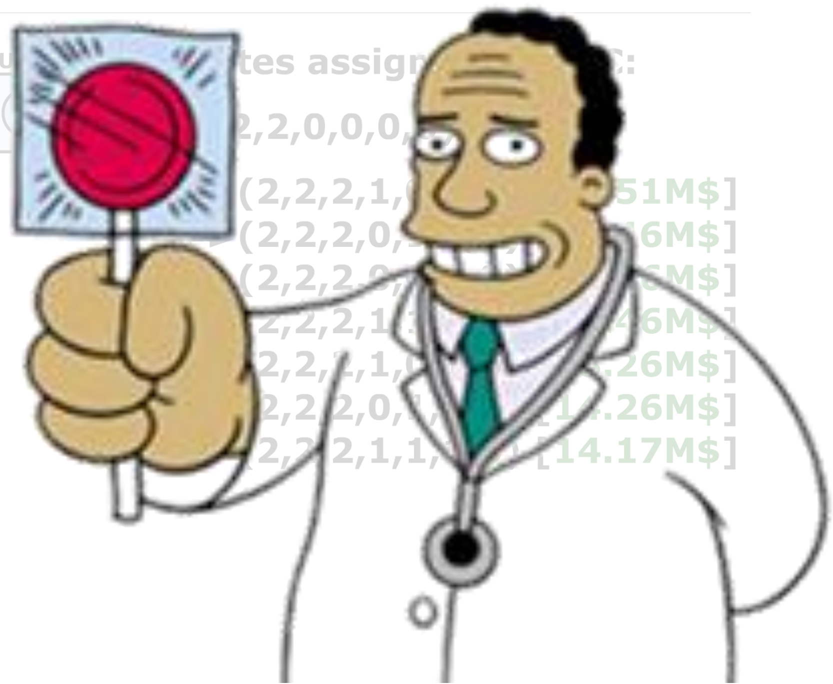


New approach: relax state space definition



The **advantages** of this approach are clear:

1. You only need up to 2^n states instead of 3^n states => **huge reduction in memory requirements**.
2. It is **easy to implement heuristics** that are able to quickly identify a “good” set of ongoing activities.



New approach: no longer uses UDCs

- Before, UDCs where used to structure the state space

$\Rightarrow$ We needed to determine the UDC network (which in itself is a NP-hard task)

New approach: no longer uses UDCs

- Before, UDCs where used to structure the state space

$\Rightarrow$ We needed to determine the UDC network (which in itself is a NP-hard task)

- In the new approach, we use arrays of states that have the same number of finished activities

New approach: no longer uses UDCs

- Before, UDCs where used to structure the state space

$\Rightarrow$ We needed to determine the UDC network (which in itself is a NP-hard task)

- In the new approach, we use arrays of states that have the same number of finished activities
- Let $\boldsymbol{X}_f$ denote the array of states for which f activities are finished
- States in $\boldsymbol{X}_f$ link only to states in $\boldsymbol{X}_{(f+1)}$

New approach: no longer uses UDCs

- Before, UDCs where used to structure the state space

$\Rightarrow$ We needed to determine the UDC network (which in itself is a NP-hard task)

- In the new approach, we use arrays of states that have the same number of finished activities
- Let $\boldsymbol{X}_f$ denote the array of states for which f activities are finished
- States in $\boldsymbol{X}_f$ link only to states in $\boldsymbol{X}_{(f+1)}$

$\Rightarrow$ Once we have determined the objective value in all states of $\boldsymbol{X}_f$, we no longer need the states in $\boldsymbol{X}_{(f+1)}$ & the memory occupied by these states can be freed

New approach: no longer uses UDCs

- Before, UDCs where used to structure the state space

⇒ We needed to determine the UDC network (which in itself is a NP-hard task)

- In the new approach, we use arrays of states that have the same number of finished activities
- Let $\boldsymbol{X}_f$ denote the array of states for which f activities are finished
- States in $\boldsymbol{X}_f$ link only to states in $\boldsymbol{X}_{(f+1)}$

⇒ Once we have determined the objective value in all states of $\boldsymbol{X}_f$, we no longer need the states in $\boldsymbol{X}_{(f+1)}$ & the memory occupied by these states can be freed

⇒ We keep at most two arrays of states in memory which again results in a huge reduction of memory requirements

New approach: no longer uses UDCs

- Before, UDCs where used

⇒ We needed to determi
is a NP-hard task)

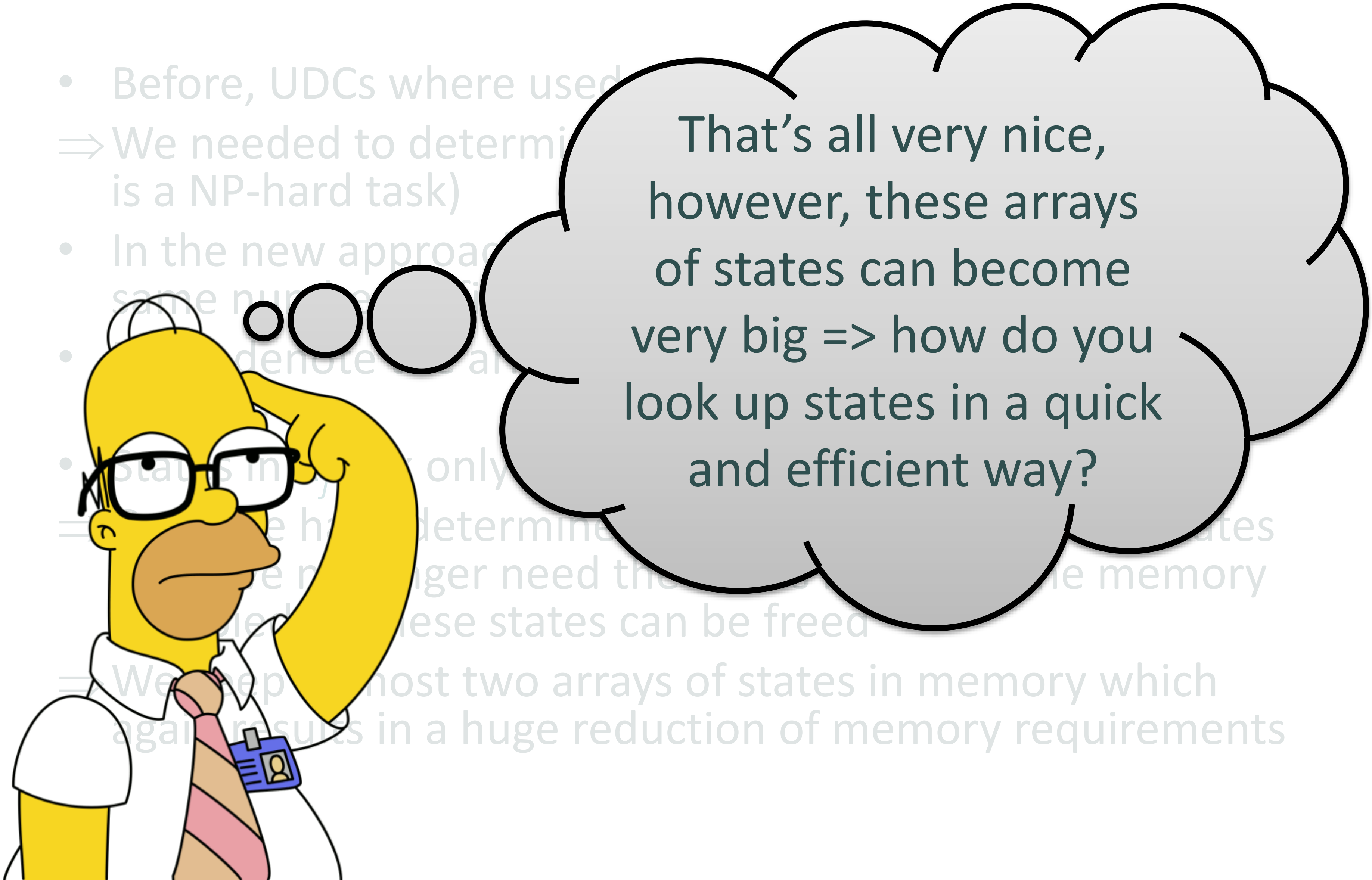
- In the new approac
same nu

- denote ar

- Sta s in only

⇒ e h determine tes
e n ger need th e memory
ie ese states can be freed

⇒ We ep most two arrays of states in memory which
aga sults in a huge reduction of memory requirements



New approach: no longer uses UDCs

- Before, UDCs w… structure the state space

⇒ We need… etwork (which in itself is a N…

- In th… ates that have the sam…
- Let X_f … ch f activities a… finishe…
- States in X_f … $X_{(f+1)}$

⇒ Once we have determined the objective value in a… of X_f, we no longer need the states in $X_{(f+1)}$ & the m…y occupied by these states can be freed

⇒ We keep at most two arrays of states in memory, which again results in a huge reduction of memory requirements

That is **trivial**! The array is constructed in such a way that states are ordered => we can use **binary search** to look up states in a quick & efficient way.

New approach: no longer uses UDCs

- Before, UDCs where
- ⇒ We needed to deter… is a NP-hard task)
- In the new approac… same number of fin…
- Let X_f denote the
- S… X_f link only…
- ⇒ O… have deter…
  … no longer need the states in $X_{(f+1)}$ & the memory …ese states can be freed
  … two arrays of states in memory which … huge reduction of memory requirements

To summarize, the **advantages** of no longer using UDCs are:

1. Huge **reduction in required memory**
2. **Improved computational efficiency** because we no longer need to determine the UDC network
3. Easy to use **parallel computing** to further improve computational efficiency

New approach: results

- Computational experiment to compare the old and the new approach with respect to:
  - The number of instances solved
  - The computation speed (CPU times)
  - The average maximum number of states stored in memory
- We use a dataset with 30 projects for each:
  - Number of activities (*n* between 10 & 70)
  - Order Strength (OS equal to 0.8, 0.6, and 0.4)

New approach: number of instances solved

OLD			
Number 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	16
n = 60	30	30	0
n = 70	30	29	0

New approach: number of instances solved

OLD			
Number 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	16
n = 60	30	30	0
n = 70	30	29	0

NEW			
Number 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 approach: average CPU time (sec)

OLD			
Average CPU time (sec)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0.00	0.00	0.00
n = 20	0.00	0.00	0.00
n = 30	0.00	0.00	0.00
n = 40	0.00	0.00	41.1
n = 50	0.00	3.02	899
n = 60	0.00	39.4	NA
n = 70	0.00	365	NA

New approach: average CPU time (sec)

OLD			
Average CPU time (sec)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0.00	0.00	0.00
n = 20	0.00	0.00	0.00
n = 30	0.00	0.00	0.00
n = 40	0.00	0.00	41.1
n = 50	0.00	3.02	899
n = 60	0.00	39.4	NA
n = 70	0.00	365	NA

NEW			
Average CPU time (sec)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0.00	0.00	0.00
n = 20	0.00	0.00	0.00
n = 30	0.00	0.00	0.00
n = 40	0.00	0.00	12.3
n = 50	0.00	0.00	270
n = 60	0.00	6.57	8960
n = 70	0.00	61.2	195691

New approach:
average maximum number of states

OLD			
Average maximum # states (x1000)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0.00	0.00	0.00
n = 20	0.00	2.39	38.6
n = 30	0.00	24.8	934
n = 40	2.9	273	25413
n = 50	9.97	2155	315807
n = 60	37.9	21140	NA
n = 70	112	149925	NA

New approach: average maximum number of states

OLD			
Average maximum # states (x1000)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0.00	0.00	0.00
n = 20	0.00	2.39	38.6
n = 30	0.00	24.8	934
n = 40	2.9	273	25413
n = 50	9.97	2155	315807
n = 60	37.9	21140	NA
n = 70	112	149925	NA

NEW			
Average maximum # states (x1000)			
	OS = 0.8	OS = 0.6	OS = 0.4
n = 10	0.00	0.00	0.00
n = 20	0.00	0.00	0.00
n = 30	0.00	0.00	2.87
n = 40	0.00	1.28	30.4
n = 50	0.00	4.87	210
n = 60	0.00	20.2	1693
n = 70	0.00	79.1	11006

Agenda

- Past work
- New approach
- What about the SRCPSP?
- Contribution

What about the SRCPSP?

- We no longer keep track of the ongoing activities

$\Rightarrow$In every state we determine the optimal set of ongoing activities

What about the SRCPSP?

- We no longer keep track of the ongoing activities

$\Rightarrow$In every state we determine the optimal set of ongoing activities

$\Rightarrow$It is possible to interrupt the execution of an activity/that an activity is started multiple times

What about the SRCPSP?

- We no longer keep track of the ongoing activities

$\Rightarrow$ In every state we determine the optimal set of ongoing activities

$\Rightarrow$ It is possible to interrupt the execution of an activity/that an activity is started multiple times

$\Rightarrow$ We cannot solve the traditional SRCPSP

What about the SRCPSP?

- We no longer keep track of the ongoing activities

$\Rightarrow$In every state we determine the optimal set of ongoing activities

$\Rightarrow$It is possible to interrupt the execution of an activity/that an activity is started multiple times

$\Rightarrow$We cannot solve the traditional SRCPSP

- We can, however, solve the SRCPSP where the execution of activities is allowed to be interrupted (in addition, the results may serve as a proxy/lower bound for the traditional SRCPSP)

What about the SRCPSP?

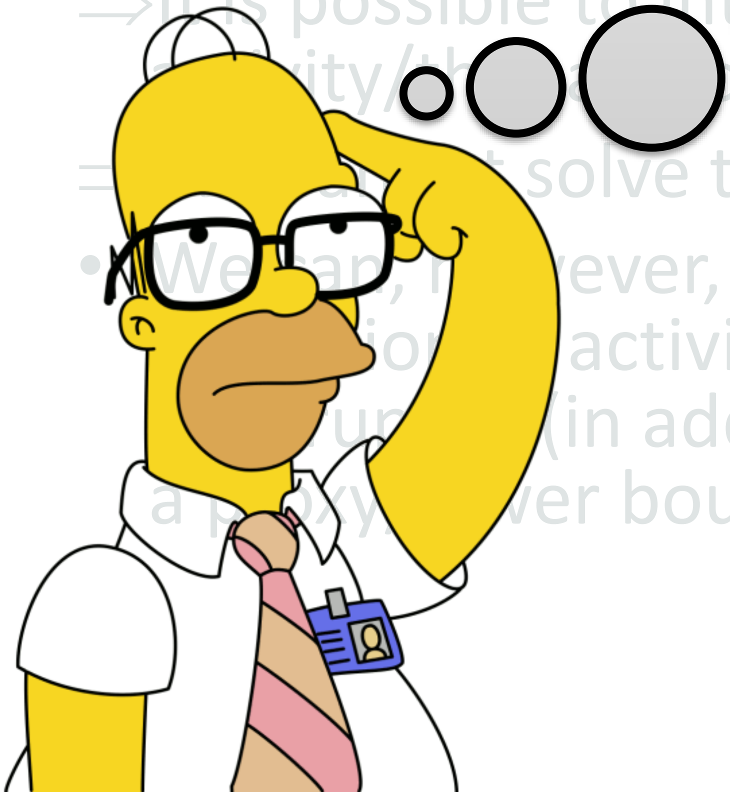
- We no longer keep trac… …es

⇒In every state we det…
ongoing activities

⇒It is possible to int…
…ity/th…

⇒… solve th…

- We can, however, s… …e
…ion… activities is…
…up… (in addition, the res… …ay serve as
a proxy … …wer bound for the traditional SRCPSP)



Why are activities not interrupted when the objective is to maximize eNPV?

What about the SRCPSP?

- We no longer … k of the ongoing activities

⇒In eve … e optimal set of ong …

⇒It i … ution of an acti … multiple times

⇒We ca … RCPSP

- We can, howev … the SRCPSP where execution of activities is allowed to be interrupted (in addition, the results may serve as a proxy/lower bound for the traditional SRCPSP)

In theory this is possible, however, interrupting an activity would result in incurring its cost twice.

SRCPSP: results

- Computational experiment to compare the old and the new approach with respect to:
  - The computation speed (CPU times)
  - The average maximum number of states stored in memory
  - The gap in between the solutions of the old approach (without activity splitting) & those of the new approach (with activity splitting)
- We use the J30 & J60 PSPLIP datasets

SRCPSP results: computational performance

	J30	
	Old	New
Instances in set	480	480
Instances solved	480	480
Average CPU time (sec)	0.48	0.02
Average max # states (x1000)	176	1.99

SRCPSP results: computational performance

	J30		J60	
	Old	New	Old	New
Instances in set	480	480	480	480
Instances solved	480	480	303	303 (480)
Average CPU time (sec)	0.48	0.02	1591	81.6
Average max # states (x1000)	176	1.99	374499	508

SRCPSP results: gap with traditional SRCPSP

	J30	J60
Instances in set	480	480
Instances solved	480	303
Minimum gap	0.00 %	0.00 %
Average gap	1.55 %	1.92 %
Maximum gap	6.65 %	7.91 %

Agenda

- Past work
- New approach
- What about the SRCPSP?
- Contribution

Contributions

We improve the models of Creemers et al. (2010) and Creemers (2015) and obtain an increase in computational efficiency with factor 6.85 and a reduction of memory requirements with factor 335!

Contributions

We improve the models of Creemers et al. (2010) and Creemers (2015) and obtain an increase in computational efficiency with factor 6.85 and a reduction of memory requirements with factor 335!

We can use our model to find the optimal expected NPV for projects with up to 120 activities that have general activity durations!

Contributions

We improve the models of Creemers et al. (2010) and Creemers (2015) and obtain an increase in computational efficiency with factor 6.85 and a reduction of memory requirements with factor 335!

We can use our model to find the optimal expected NPV for projects with up to 120 activities that have general activity durations!

Our model can also be used to study the SRCPSP where the execution of activities is allowed to be interrupted (i.e., we can assess the value of splitting activities).

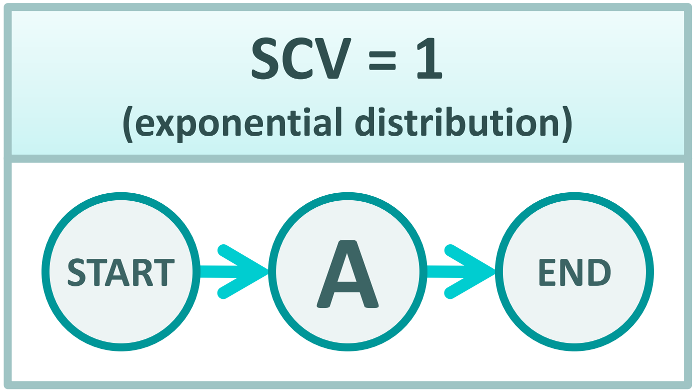
?

PH distributions

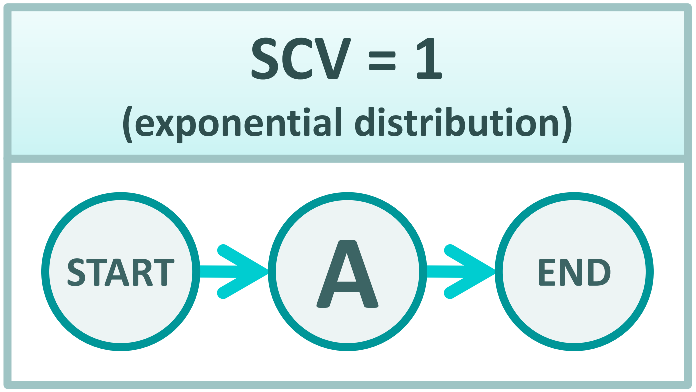
- Introduced by Neuts in 1981
- A Phase Type (PH) distribution is a mixture of exponential distributions
- The exponential, Erlang, Coxian, and hyper-exponential distribution are all examples of a PH distribution
- We use simple PH distributions to match the first two moments of the distribution of the activity duration (more advanced PH distributions, however, can also be used)

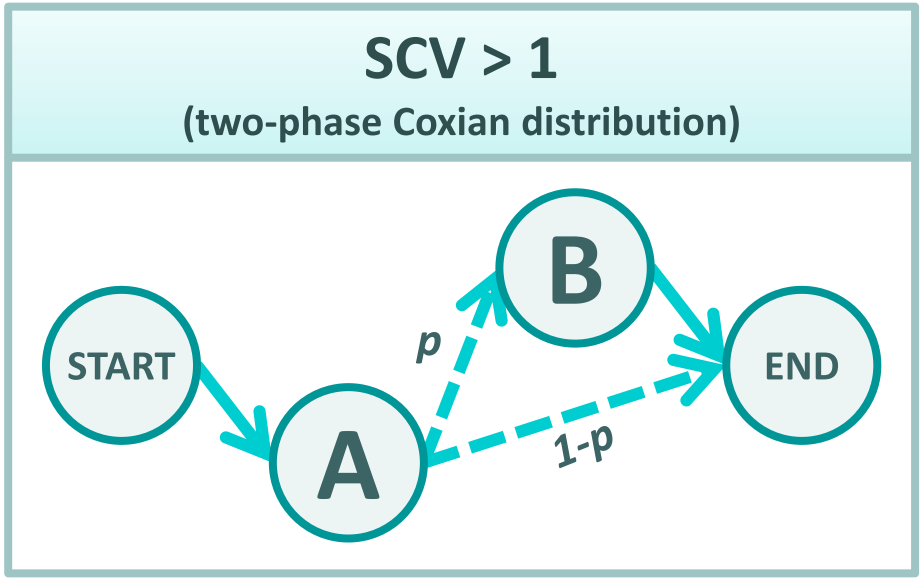
PH distributions: Example of a single activity

PH distributions: Example of a single activity

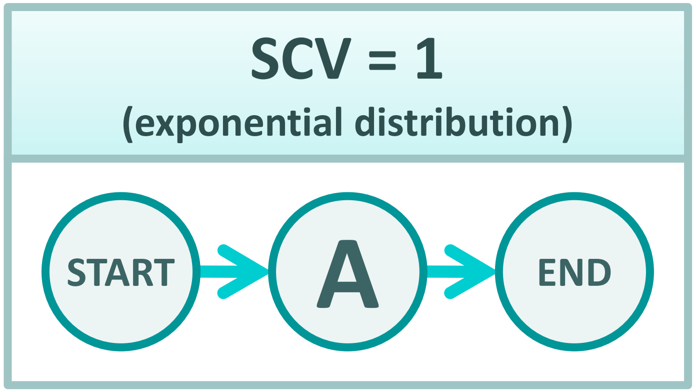


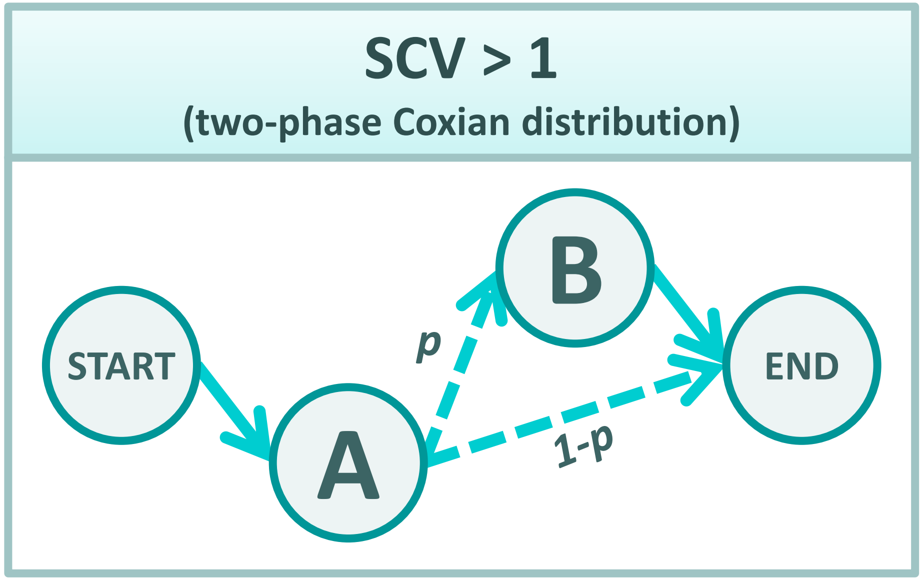
PH distributions: Example of a single activity

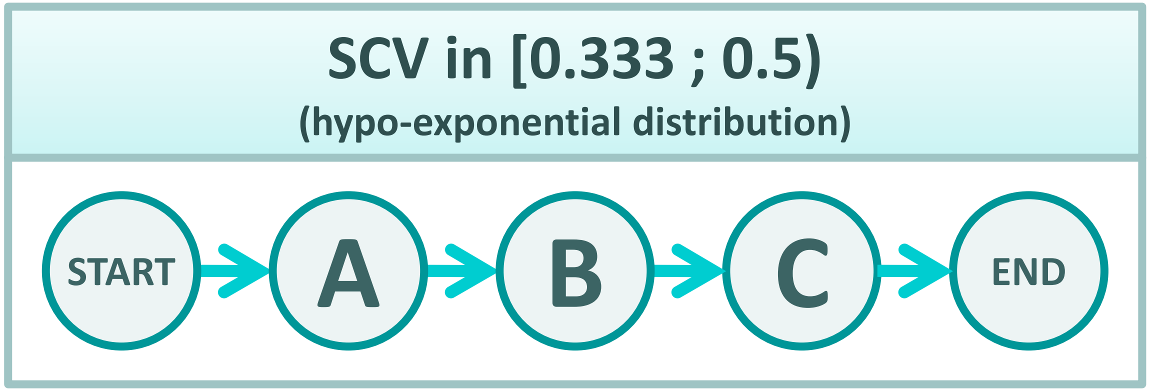




PH distributions: Example of a single activity

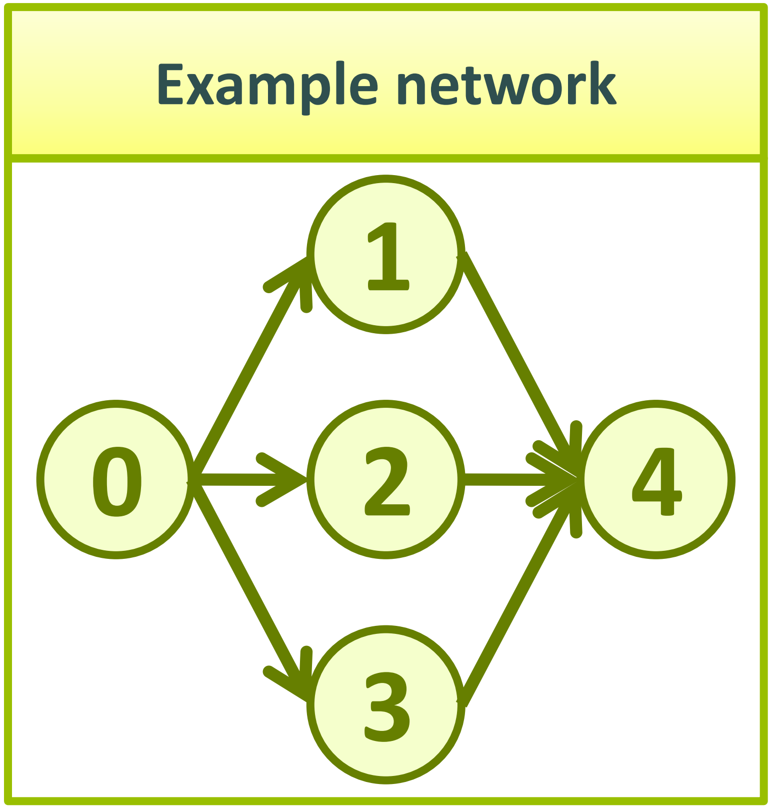






PH distributions: Example of a project network

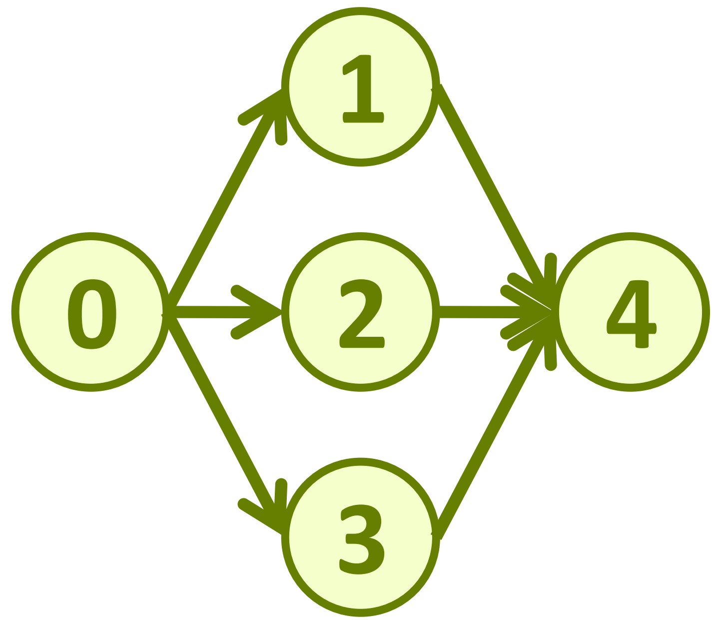
PH distributions: Example of a project network



PH distributions: Example of a project network

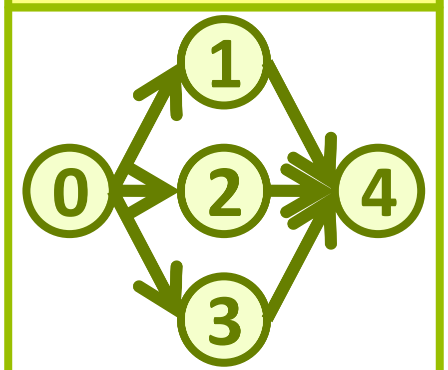
Activity	SCV
0	Dummy start
1	SCV in [0.33;0.5)
2	SCV = 1
3	SCV > 1
4	Dummy finish

Example network



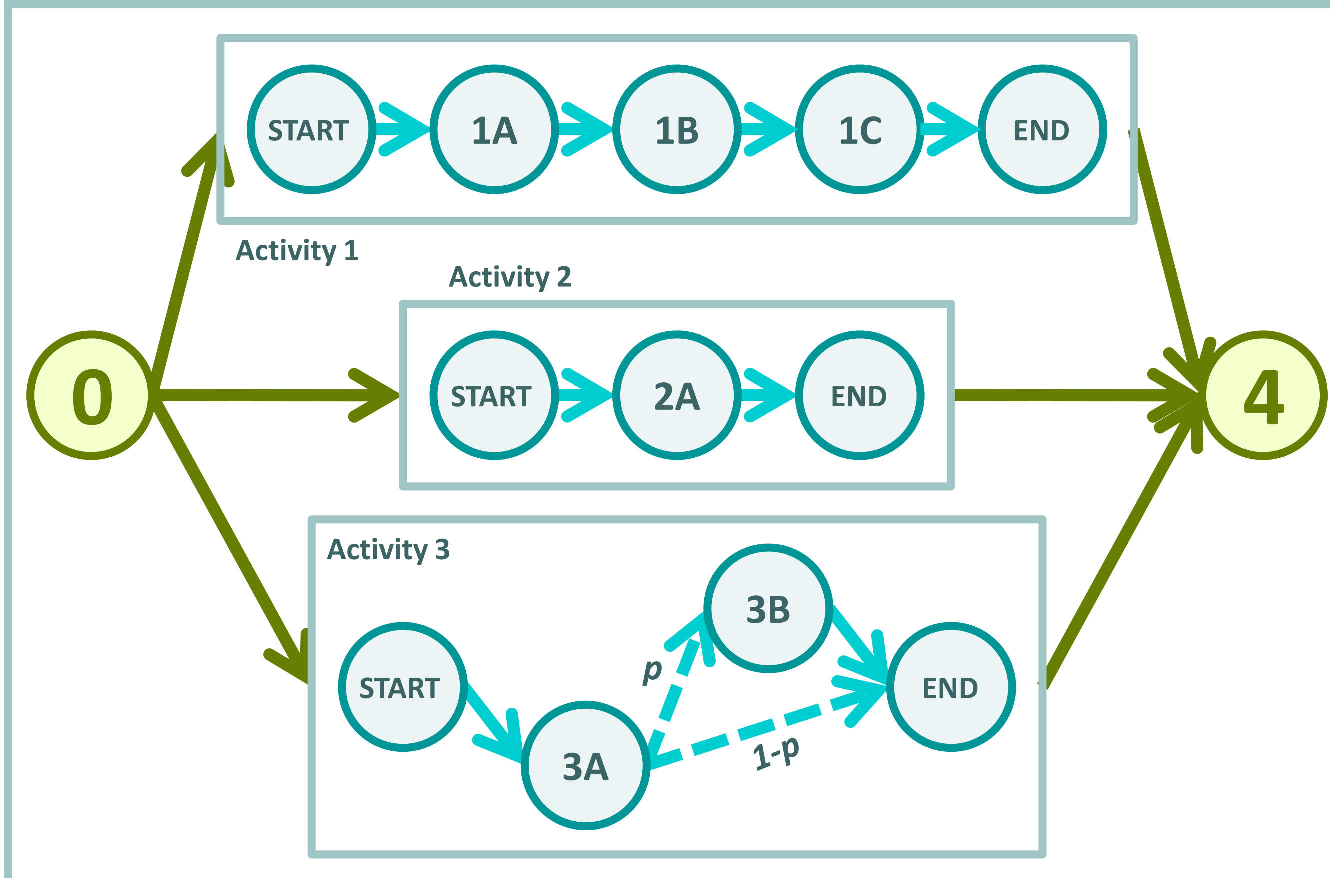
PH distributions: Example of a project network

Example network

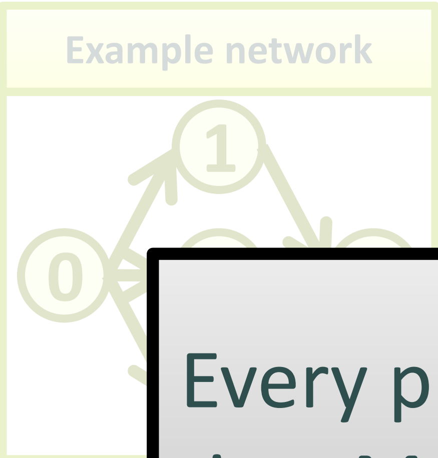


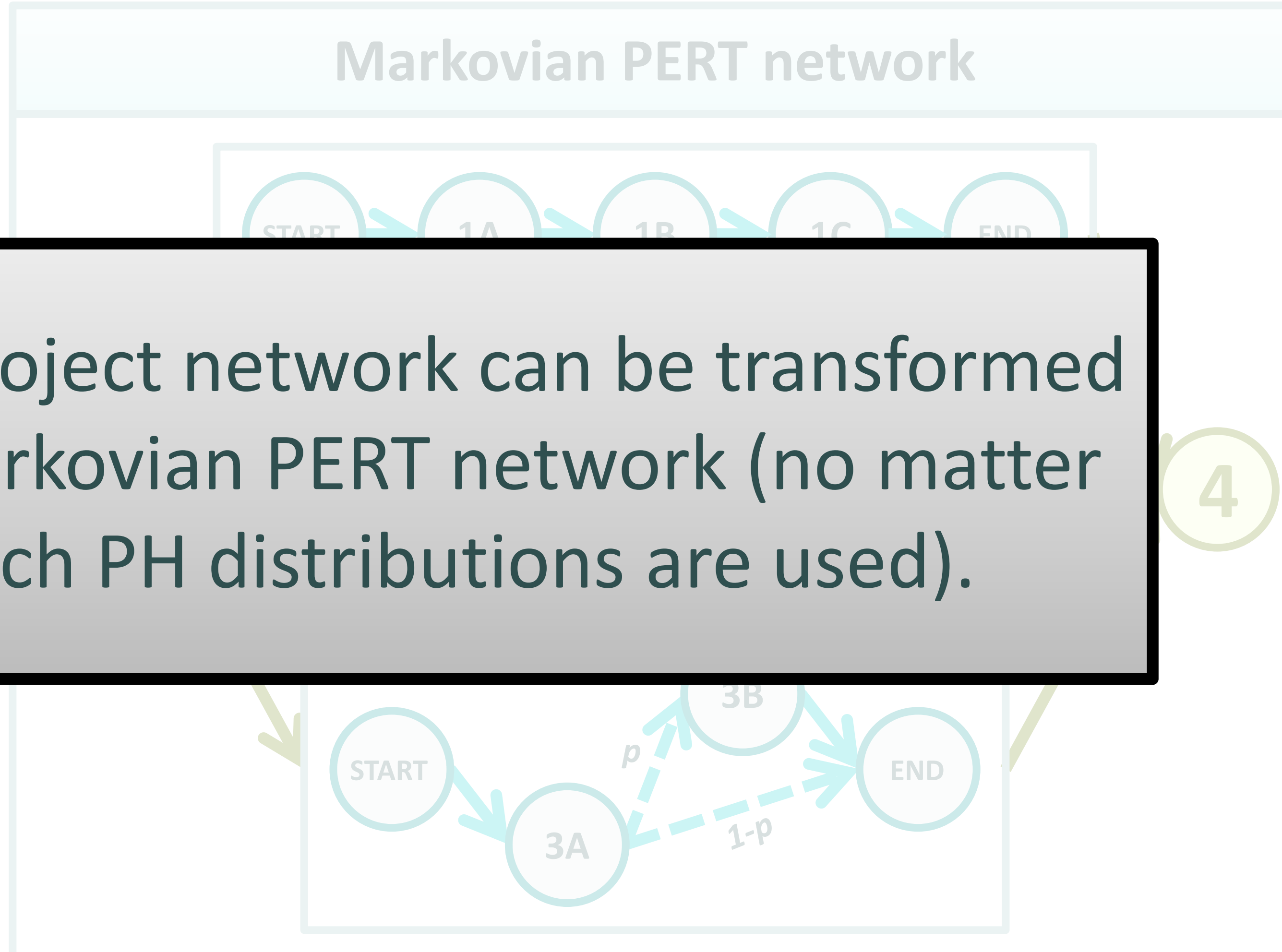
Activity	SCV
0	Dummy start
1	SCV in [0.33;0.5)
2	SCV = 1
3	SCV > 1
4	Dummy finish

Markovian PERT network



PH distributions: Example of a project network



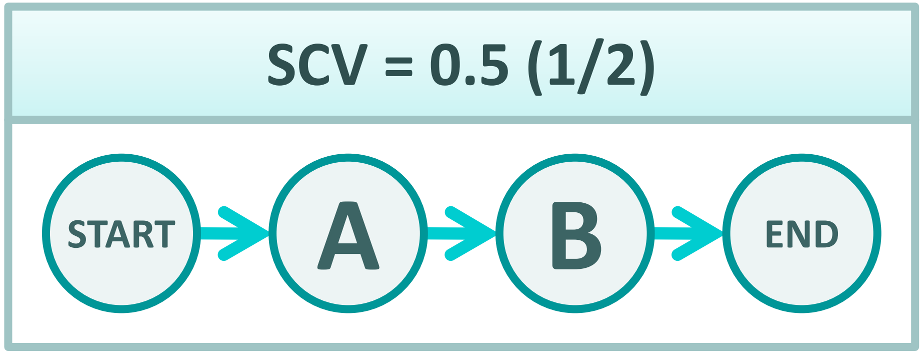


Activity	
0	
1	SCV in [0.33;0.5)
2	SCV = 1
3	SCV > 1
4	Dummy finish

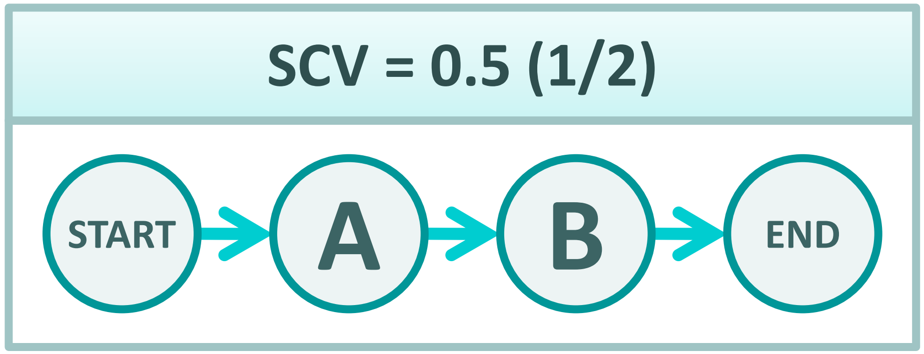
Every project network can be transformed in a Markovian PERT network (no matter which PH distributions are used).

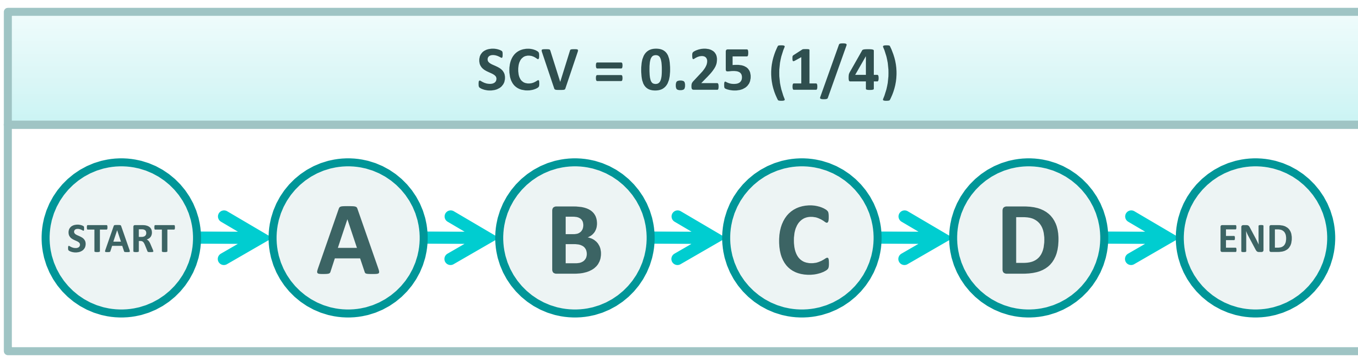
PH distributions: What about low variability?

PH distributions: What about low variability?

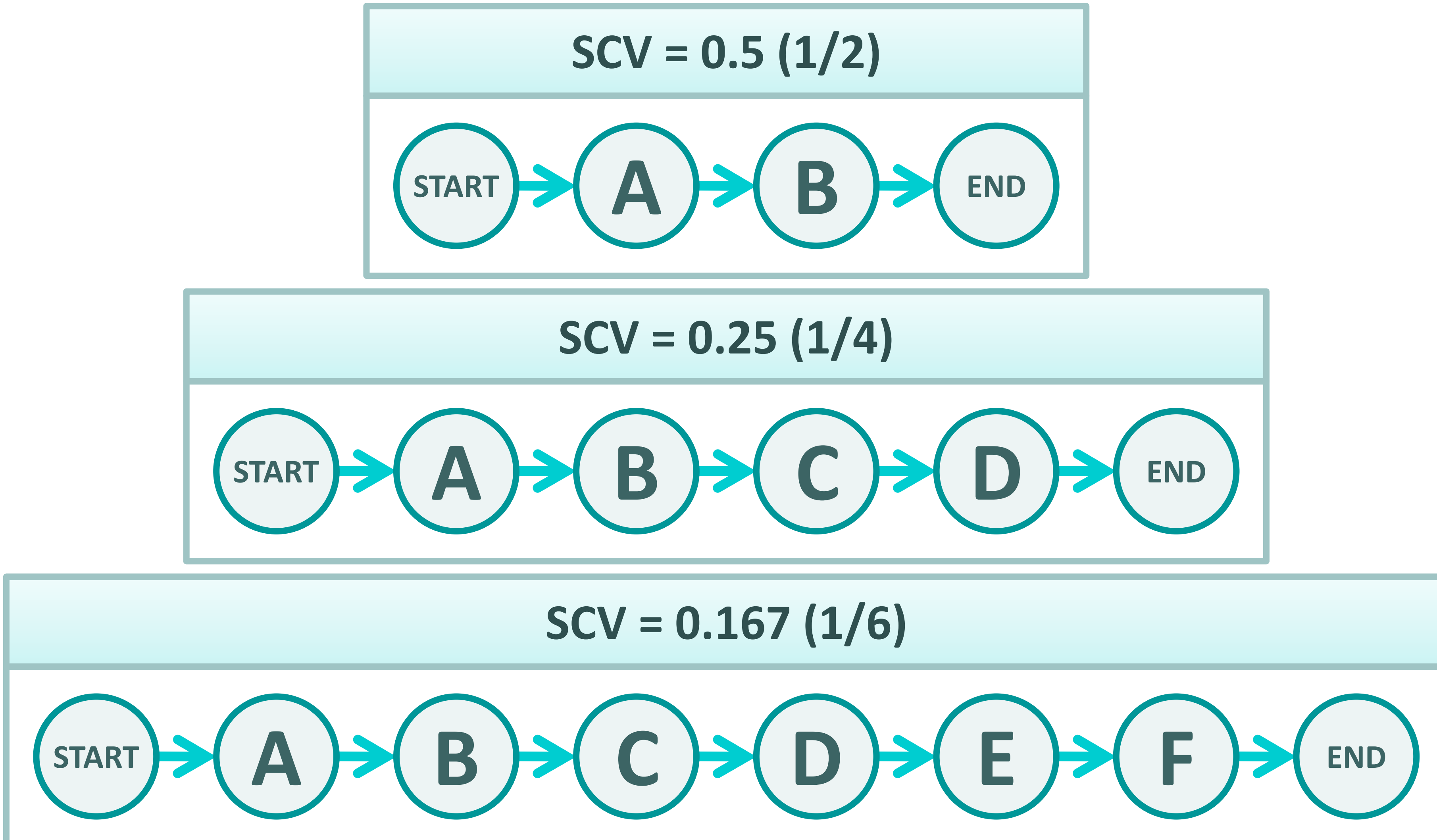


PH distributions: What about low variability?





PH distributions: What about low variability?



PH distributions: What about low variability?

Low variability duration variability inflates the size of the Markovian PERT network.

=>

Our model works best when duration variability is moderate to high.

Past work: example

Illustration of optimal policy

Past work: example

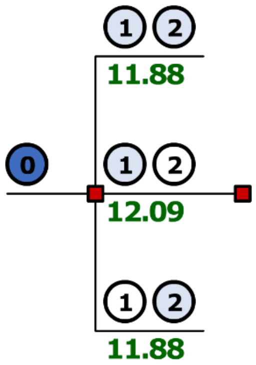


Illustration of optimal policy

Past work: example

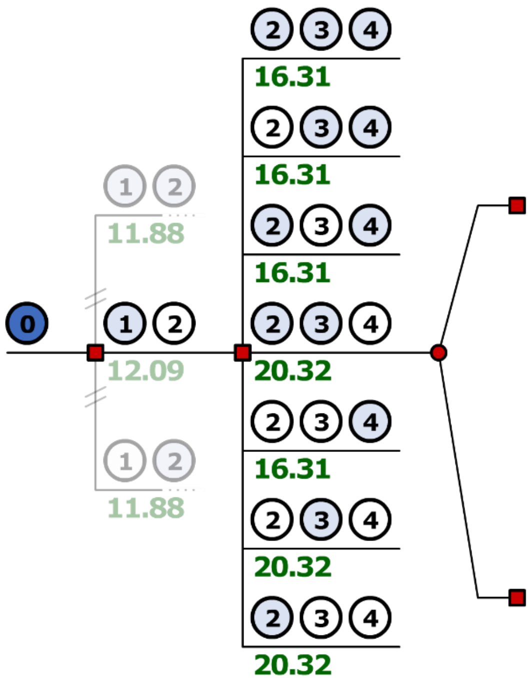


Illustration of optimal policy

Past work: example

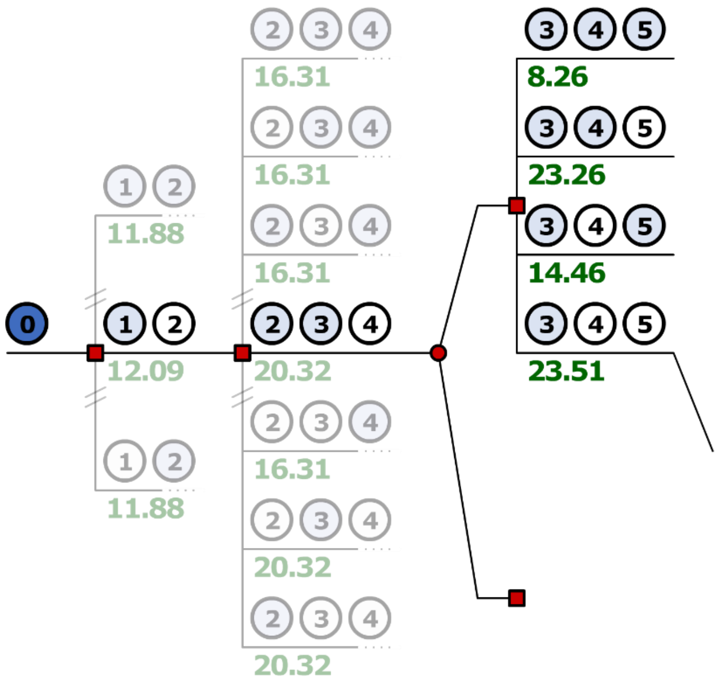


Illustration of optimal policy

Past work: example

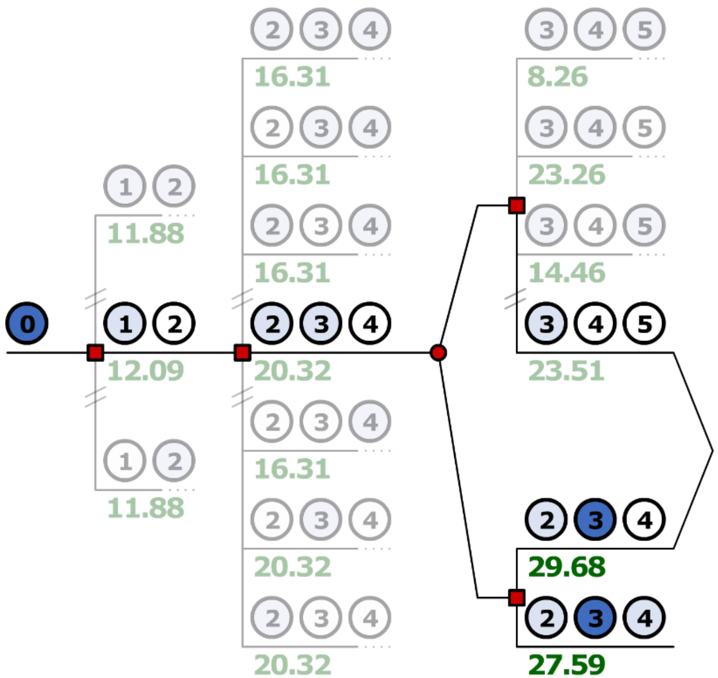


Illustration of optimal policy

Past work: example

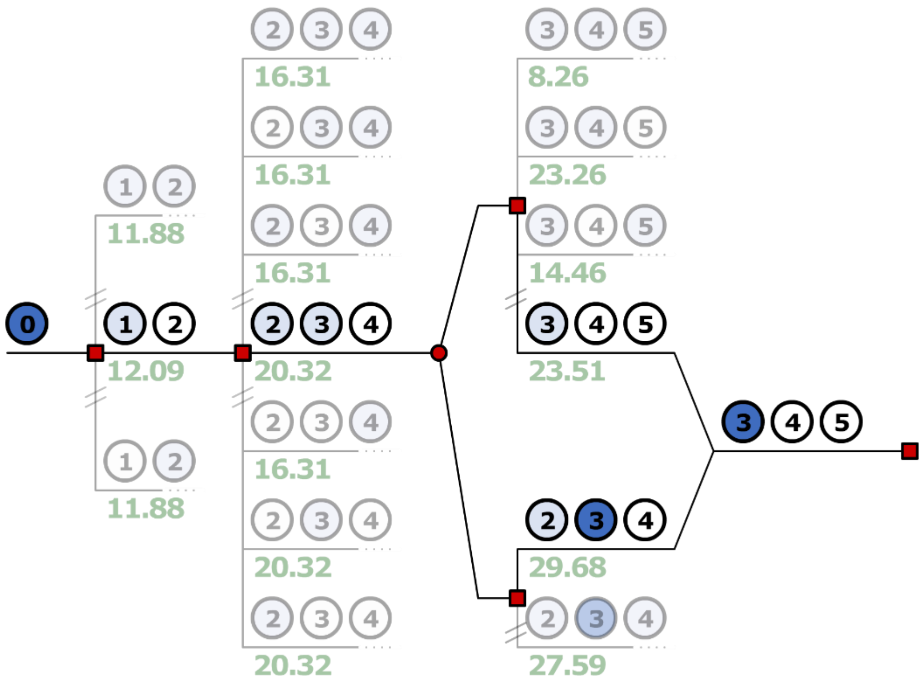


Illustration of optimal policy

Past work: example

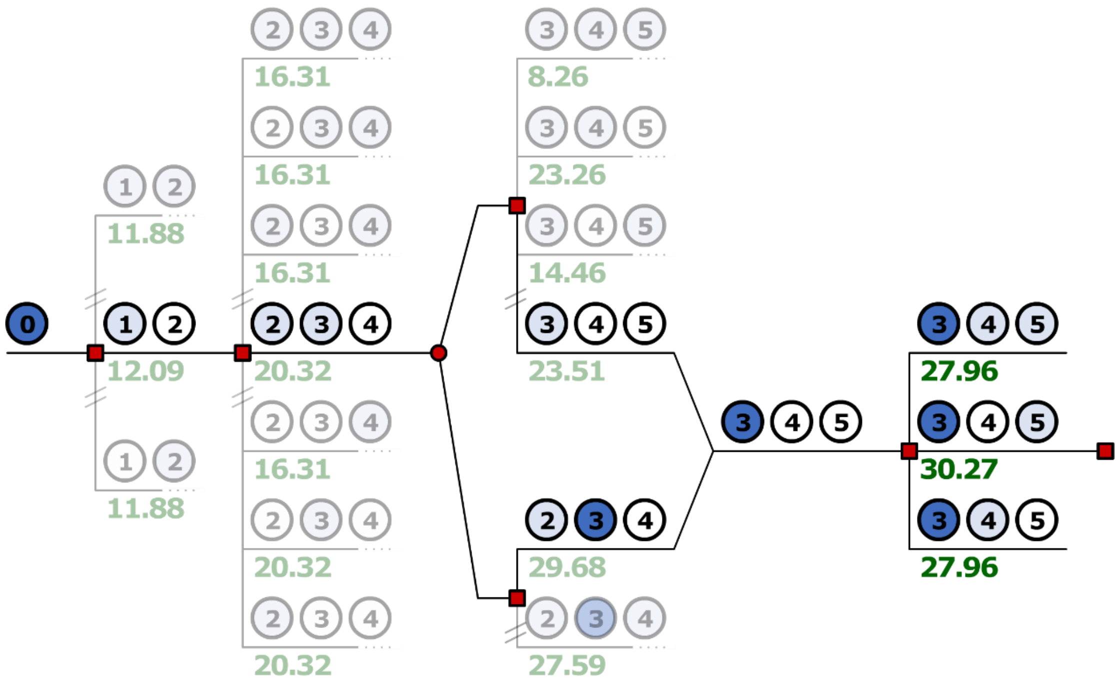


Illustration of optimal policy

Past work: example

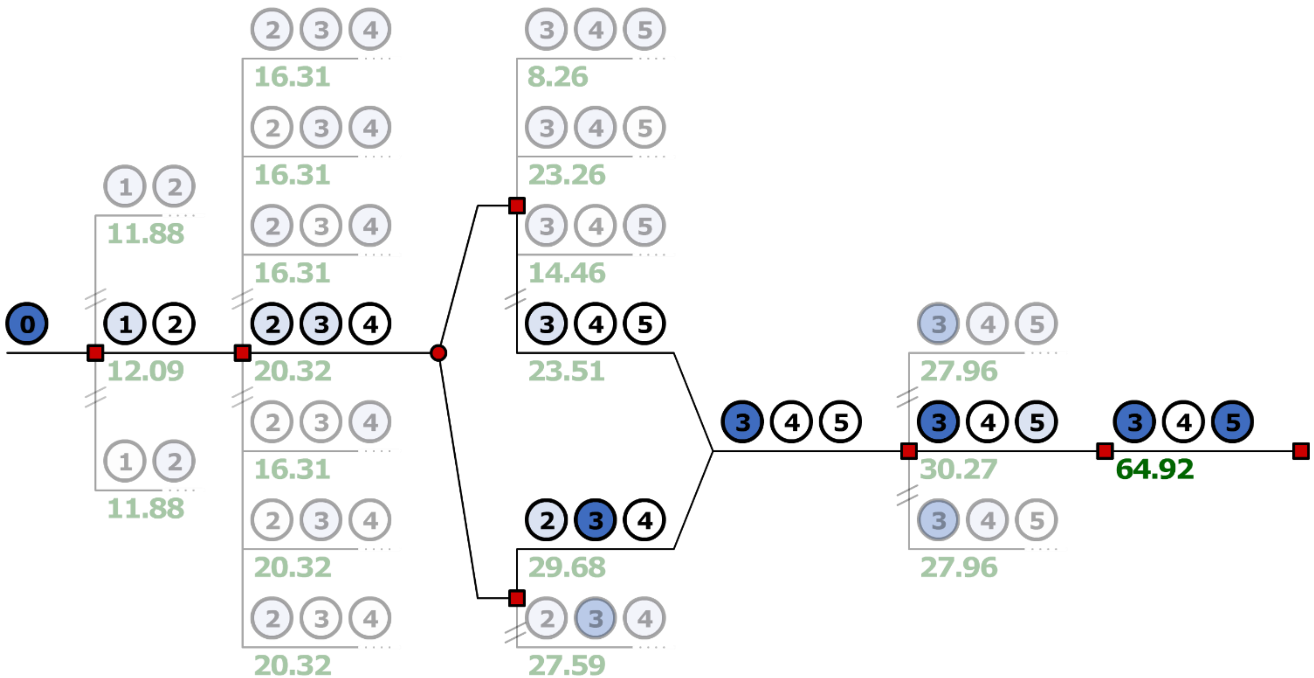


Illustration of optimal policy

Past work: example

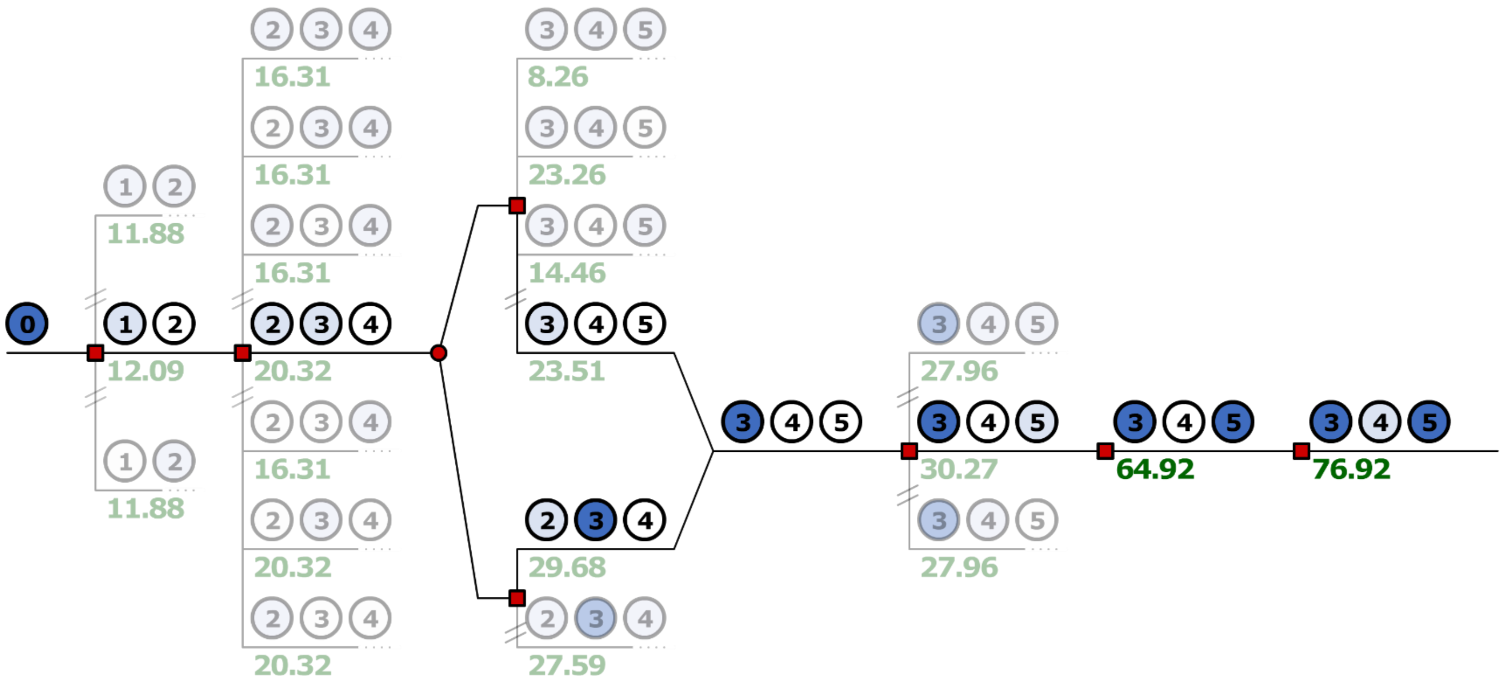


Illustration of optimal policy

Past work: example

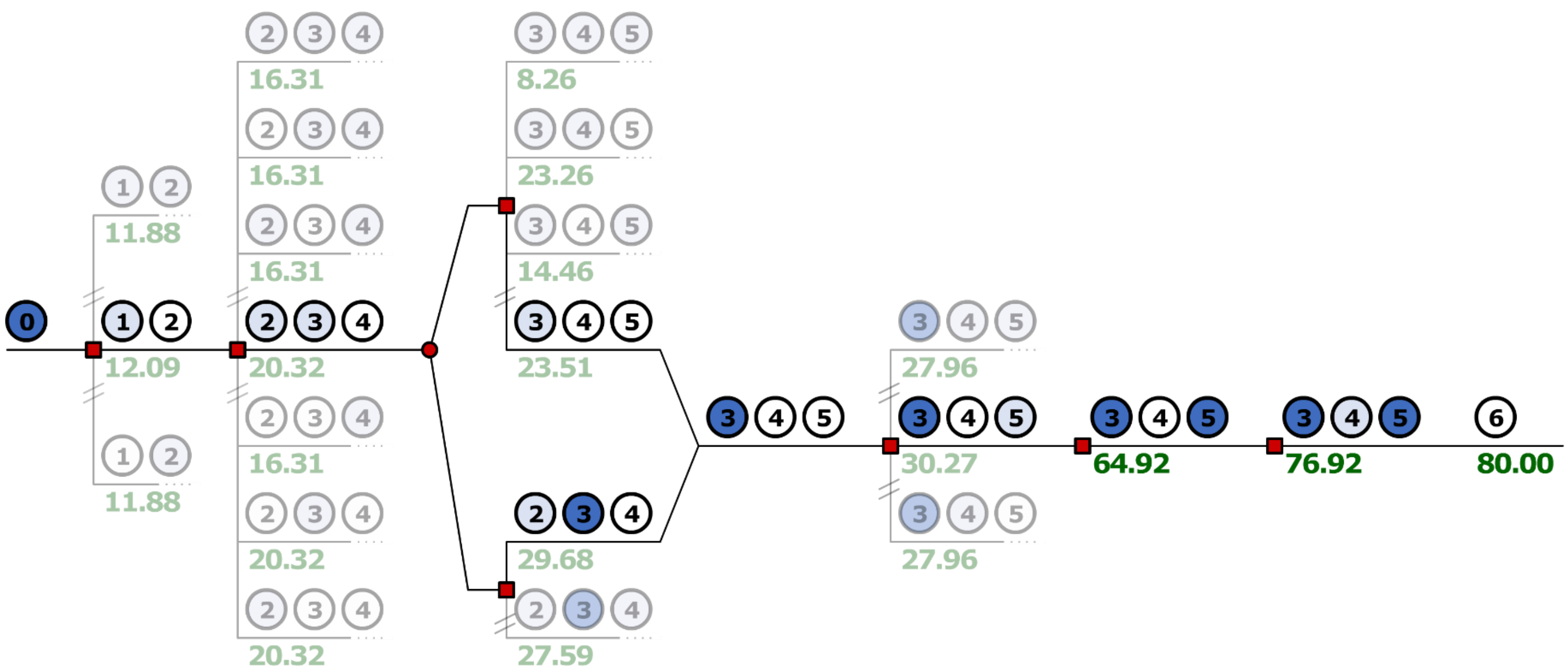


Illustration of optimal policy