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

Stefan Creemers
(July 13, 2015)
Agenda

• Past work
• New approach
• What about the SRCPSP?
• Contribution
Agenda

• Past work
• New approach
• What about the SRCPSP?
• Contribution
Past work: overview

Past work: overview

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

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

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


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

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

---

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

  The status of an activity $j$ at time $t$ is either:

  - Idle ($q_j(t) = 0$)
  - Busy ($q_j(t) = 1$)
  - Finished ($q_j(t) = 2$)

- $q(t) = \{q_1(t), q_2(t), \ldots, q_n(t)\}$ 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

  ➞ A clear and strict definition of the state space is essential

  ➞ We use UDCs to structure the state space
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 ($\theta_j(t) = 2$)
• $\theta(t) = \{\theta_1(t), \theta_2(t), \ldots, \theta_n(t)\}$ 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 ($\theta_j(t) = 2$)

• $\Theta(t) = \{\theta_1(t), \theta_2(t), \ldots \theta_n(t)\}$ 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 \( (\theta_j(t) = 2) \)
• \( \theta(t) = \{\theta_1(t), \theta_2(t), \ldots, \theta_n(t)\} \) 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 (\( \theta_j(t) = 2 \))

• \( \theta(t) = \{ \theta_1(t), \theta_2(t), \ldots, \theta_n(t) \} \) 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

⇒ 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 ($\theta_j(t) = 2$)
• $\Theta(t) = \{\theta_1(t), \theta_2(t), \ldots, \theta_n(t)\}$ 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.
  ⇒ A clear and strict definition of the state space is essential.
  ⇒ 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

Illustration of state space & SDP recursion
Past work: example

Illustration of state space & SDP recursion

States assigned to UDC: (2,2,2,2,2,0) -> 80M$
Past work: example

Illustration of state space & SDP recursion
Past work: example

Illustration of state space & SDP recursion

States assigned to UDC: 
\[(2,2,2,2,2,1,0) \rightarrow (2,2,2,2,2,2,0) \text{ [80.00M$]}\]
Past work: example

Illustration of state space & SDP recursion

States assigned to UDC:
(2,2,2,2,2,1,0)

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

Illustration of state space & SDP recursion

States assigned to UDC:

\((2,2,2,2,2,1,0) \rightarrow 58.25M\$\)
\((2,2,2,2,2,2,0) [80.00M\$]\)

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

Illustration of state space & SDP recursion

States assigned to UDC:
(2,2,2,2,2,1,0) -> 58.25M$
(2,2,2,2,2,0,0)
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

States assigned to UDC:

\[(2,2,2,2,2,1,0) \rightarrow 58.25 \text{M}\$
\[(2,2,2,2,2,0,0) \rightarrow 41.25 \text{M}\$
\[(2,2,2,2,2,1,0) \rightarrow 58.25 \text{M}\$

Only decision left is to start activity 5
\[\Rightarrow \text{incur cost } c_5 = -17 \text{M}\$
\[\Rightarrow \text{NPV at state entry} = 41.25 \text{M}\$

Past work: example

Illustration of state space & SDP recursion
Past work: example

Illustration of state space & SDP recursion

States assigned to UDC:
1. \((2,2,2,2,2,1,0) \rightarrow 58.25\text{M}$
2. \((2,2,2,2,2,0,0) \rightarrow 41.25\text{M}$
3. \((2,2,2,2,1,2,0) \rightarrow 80.00\text{M}$
Past work: example

Illustration of state space & SDP recursion
Past work: example

Illustration of state space & SDP recursion

States assigned to UDC:
- \((2,2,2,2,2,1,0)\) \(\rightarrow\) 58.25M$
- \((2,2,2,2,2,0,0)\) \(\rightarrow\) 41.25M$
- \((2,2,2,2,1,2,0)\) \(\rightarrow\) 76.92M$
- \((2,2,2,2,1,1,0)\)
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

Illustration of state space & SDP recursion
Past work: example

Illustration of state space & SDP recursion
Past work: example

Illustration of state space & SDP recursion

States assigned to UDC:

- $(2,2,2,0,0,0,0) \ [21.51M\$]
- $(2,2,2,1,0,0,0) \ [17.46M\$]
- $(2,2,2,0,1,0,0) \ [18.26M\$]
- $(2,2,2,1,1,0,0) \ [17.46M\$]
- $(2,2,2,0,1,1,0) \ [18.26M\$]
- $(2,2,2,1,1,1,0) \ [14.26M\$]
- $(2,2,2,1,1,1,0) \ [14.17M\$]
Past work: example

Illustration of state space & SDP recursion

States assigned to UDC:
(2,2,2,0,0,0,0) -> 21.51M$
(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$]
Past work: example

In the end, you have processed all states and obtain the optimal eNPV of the project
Agenda

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

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

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
5. UDCs to structure state space
6. Upper bound state space = $3^n$

Main bottleneck = memory!

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

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

1. SDP recursion
2. Optimal solution
3. General activity durations
4. eNPV (SRCPSP = see infra)
5. Upper bound state space = $3^n$
6. Main bottleneck = memory!

How does he do that?
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
⇒ 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

⇒ 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
  ⇒ We don’t know which activities are ongoing!
• We, however, do know the set of activities that are eligible to start
  ⇒ 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) [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

The disadvantage of this approach is that you have to enumerate all sets of ongoing activities in order to find the optimal set that maximizes the eNPV (however, this also needs to be done in the old approach).
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 were used to structure the state space
  ⇒ We needed to determine the UDC network (which in itself is a NP-hard task)
New approach: no longer uses UDCs

• Before, UDCs were 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
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 $X_f$ denote the array of states for which $f$ activities are finished
• States in $X_f$ link only to states in $X_{(f+1)}$
New approach: no longer uses UDCs

• Before, UDCs were used to structure the state space
  \[\Rightarrow \text{We needed to determine the UDC network (which in itself is a NP-hard task)}\]
• In the new approach, we use arrays of states that have the same number of finished activities
• Let \(X_f\) denote the array of states for which \(f\) activities are finished
• States in \(X_f\) link only to states in \(X_{(f+1)}\)
  \[\Rightarrow \text{Once we have determined the objective value in all states of } X_f, \text{ we no longer need the states in } X_{(f+1)} & \text{ the memory occupied by these states can be freed}\]
New approach: no longer uses UDCs

• Before, UDCs were 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 $X_f$ denote the array of states for which $f$ activities are finished
• States in $X_f$ link only to states in $X_{(f+1)}$
  ⇒ Once we have determined the objective value in all states of $X_f$, we no longer need the states in $X_{(f+1)}$ & 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 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 $X_f$ denote the array of states for which $f$ activities are finished
• States in $X_f$ link only to states in $X_{f+1}$
  ⇒ Once we have determined the objective value in all states of $X_f$, we no longer need the states in $X_{f+1}$, and 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

That’s all very nice, however, these arrays of states can become very big ⇒ how do you look up states in a quick and efficient way?
New approach: no longer uses UDCs

- Before, UDCs were 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 $X_f$ denote the array of states for which $f$ activities are finished.
- States in $X_f$ link only to states in $X_{(f+1)}$.
- Once we have determined the objective value in all states of $X_f$, we no longer need the states in $X_{(f+1)}$, and 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.

That is **trivial**! The array is constructed in such a way that states are ordered, so we can use binary search to look up states in a quick & efficient way.
New approach: no longer uses UDCs

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

<table>
<thead>
<tr>
<th>OLD</th>
<th>Number solved (out of 30)</th>
<th>OS = 0.8</th>
<th>OS = 0.6</th>
<th>OS = 0.4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>n = 10</td>
<td>n = 20</td>
<td>n = 30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n = 40</td>
<td>n = 50</td>
<td>n = 60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>
New approach: number of instances solved

<table>
<thead>
<tr>
<th></th>
<th>Old Method</th>
<th>New Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OS = 0.8</td>
<td>OS = 0.6</td>
</tr>
<tr>
<td>n = 10</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>n = 20</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>n = 30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>n = 40</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>n = 50</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>n = 60</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>n = 70</td>
<td>30</td>
<td>29</td>
</tr>
</tbody>
</table>
New approach: average CPU time (sec)

<table>
<thead>
<tr>
<th></th>
<th>Average CPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OS = 0.8</td>
</tr>
<tr>
<td>n = 10</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 20</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 30</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 40</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 50</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 60</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 70</td>
<td>0.00</td>
</tr>
</tbody>
</table>
New approach:
average CPU time (sec)

<table>
<thead>
<tr>
<th></th>
<th>OS = 0.8</th>
<th>OS = 0.6</th>
<th>OS = 0.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 10</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 20</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 30</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 40</td>
<td>0.00</td>
<td>0.00</td>
<td>41.1</td>
</tr>
<tr>
<td>n = 50</td>
<td>0.00</td>
<td>3.02</td>
<td>899</td>
</tr>
<tr>
<td>n = 60</td>
<td>0.00</td>
<td>39.4</td>
<td>NA</td>
</tr>
<tr>
<td>n = 70</td>
<td>0.00</td>
<td>365</td>
<td>NA</td>
</tr>
</tbody>
</table>

Average CPU time (sec)

<table>
<thead>
<tr>
<th></th>
<th>OS = 0.8</th>
<th>OS = 0.6</th>
<th>OS = 0.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 10</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 20</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 30</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 40</td>
<td>0.00</td>
<td>0.00</td>
<td>12.3</td>
</tr>
<tr>
<td>n = 50</td>
<td>0.00</td>
<td>0.00</td>
<td>270</td>
</tr>
<tr>
<td>n = 60</td>
<td>0.00</td>
<td>6.57</td>
<td>8960</td>
</tr>
<tr>
<td>n = 70</td>
<td>0.00</td>
<td>61.2</td>
<td>195691</td>
</tr>
</tbody>
</table>
New approach:
average maximum number of states

<table>
<thead>
<tr>
<th>n</th>
<th>OS = 0.8</th>
<th>OS = 0.6</th>
<th>OS = 0.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>20</td>
<td>0.00</td>
<td>2.39</td>
<td>38.6</td>
</tr>
<tr>
<td>30</td>
<td>0.00</td>
<td>24.8</td>
<td>934</td>
</tr>
<tr>
<td>40</td>
<td>2.9</td>
<td>273</td>
<td>25413</td>
</tr>
<tr>
<td>50</td>
<td>9.97</td>
<td>2155</td>
<td>315807</td>
</tr>
<tr>
<td>60</td>
<td>37.9</td>
<td>21140</td>
<td>NA</td>
</tr>
<tr>
<td>70</td>
<td>112</td>
<td>149925</td>
<td>NA</td>
</tr>
</tbody>
</table>
New approach: average maximum number of states

<table>
<thead>
<tr>
<th></th>
<th>OLD</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>OS = 0.8</td>
<td>OS = 0.6</td>
</tr>
<tr>
<td>n = 10</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 20</td>
<td>0.00</td>
<td>2.39</td>
<td>38.6</td>
</tr>
<tr>
<td>n = 30</td>
<td>0.00</td>
<td>24.8</td>
<td>934</td>
</tr>
<tr>
<td>n = 40</td>
<td>2.9</td>
<td>273</td>
<td>25413</td>
</tr>
<tr>
<td>n = 50</td>
<td>9.97</td>
<td>2155</td>
<td>315807</td>
</tr>
<tr>
<td>n = 60</td>
<td>37.9</td>
<td>21140</td>
<td>NA</td>
</tr>
<tr>
<td>n = 70</td>
<td>112</td>
<td>149925</td>
<td>NA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>OLD</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>OS = 0.8</td>
<td>OS = 0.6</td>
</tr>
<tr>
<td>n = 10</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 20</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 30</td>
<td>0.00</td>
<td>0.00</td>
<td>2.87</td>
</tr>
<tr>
<td>n = 40</td>
<td>0.00</td>
<td>1.28</td>
<td>30.4</td>
</tr>
<tr>
<td>n = 50</td>
<td>0.00</td>
<td>0.00</td>
<td>4.87</td>
</tr>
<tr>
<td>n = 60</td>
<td>0.00</td>
<td>0.00</td>
<td>20.2</td>
</tr>
<tr>
<td>n = 70</td>
<td>0.00</td>
<td>0.00</td>
<td>79.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>OLD</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>OS = 0.8</td>
<td>OS = 0.6</td>
</tr>
<tr>
<td>n = 10</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 20</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 30</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 40</td>
<td>0.00</td>
<td>0.00</td>
<td>2.87</td>
</tr>
<tr>
<td>n = 50</td>
<td>0.00</td>
<td>0.00</td>
<td>4.87</td>
</tr>
<tr>
<td>n = 60</td>
<td>0.00</td>
<td>0.00</td>
<td>20.2</td>
</tr>
<tr>
<td>n = 70</td>
<td>0.00</td>
<td>0.00</td>
<td>79.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>NEW</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>OS = 0.8</td>
<td>OS = 0.6</td>
</tr>
<tr>
<td>n = 10</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 20</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 30</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>n = 40</td>
<td>0.00</td>
<td>1.28</td>
<td>30.4</td>
</tr>
<tr>
<td>n = 50</td>
<td>0.00</td>
<td>1.28</td>
<td>30.4</td>
</tr>
<tr>
<td>n = 60</td>
<td>0.00</td>
<td>1.28</td>
<td>30.4</td>
</tr>
<tr>
<td>n = 70</td>
<td>0.00</td>
<td>79.1</td>
<td>11006</td>
</tr>
</tbody>
</table>
Agenda

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

• We no longer keep track of the ongoing activities
  → In every state we determine the optimal set of ongoing activities
What about the SRCPSP?

• We no longer keep track of the ongoing activities
  ⇒ In every state we determine the optimal set of ongoing activities
  ⇒ 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
  ⇒ In every state we determine the optimal set of ongoing activities
  ⇒ It is possible to interrupt the execution of an activity/that an activity is started multiple times
  ⇒ We cannot solve the traditional SRCPSP
What about the SRCPSP?

• We no longer keep track of the ongoing activities
  ⇒ In every state we determine the optimal set of ongoing activities
  ⇒ It is possible to interrupt the execution of an activity/that an activity is started multiple times
  ⇒ 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 track of the ongoing activities
  ⇒ In every state we determine the optimal set of ongoing activities
  ⇒ It is possible to interrupt the execution of an activity/that an activity is started multiple times
  ⇒ 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)

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

- We no longer keep track of the ongoing activities.
  - In every state, we determine the optimal set of ongoing activities.
  - It is possible to interrupt the execution of an activity or start an activity multiple times.
  - 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).

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

<table>
<thead>
<tr>
<th></th>
<th>J30</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Old</td>
</tr>
<tr>
<td>Instances in set</td>
<td>480</td>
</tr>
<tr>
<td>Instances solved</td>
<td>480</td>
</tr>
<tr>
<td>Average CPU time (sec)</td>
<td>0.48</td>
</tr>
<tr>
<td>Average max # states (x1000)</td>
<td>176</td>
</tr>
</tbody>
</table>
## SRCPSP results: computational performance

<table>
<thead>
<tr>
<th></th>
<th>J30</th>
<th>J60</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Old</td>
<td>New</td>
</tr>
<tr>
<td>Instances in set</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>Instances solved</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>Average CPU time (sec)</td>
<td>0.48</td>
<td>0.02</td>
</tr>
<tr>
<td>Average max # states (x1000)</td>
<td>176</td>
<td>1.99</td>
</tr>
</tbody>
</table>
SRCPSP results: gap with traditional SRCPSP

<table>
<thead>
<tr>
<th></th>
<th>J30</th>
<th>J60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instances in set</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>Instances solved</td>
<td>480</td>
<td>303</td>
</tr>
<tr>
<td>Minimum gap</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Average gap</td>
<td>1.55%</td>
<td>1.92%</td>
</tr>
<tr>
<td>Maximum gap</td>
<td>6.65%</td>
<td>7.91%</td>
</tr>
</tbody>
</table>
Agenda

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

SCV = 1  
(exponential distribution)
PH distributions: Example of a single activity

SCV = 1
(exponential distribution)

SCV > 1
(two-phase Coxian distribution)
PH distributions: Example of a single activity

- **SCV = 1** (exponential distribution)
  - START → A → END

- **SCV > 1** (two-phase Coxian distribution)
  - START → A → B → END
  - Probability: p → A, (1-p) → B

- **SCV in [0.333 ; 0.5)** (hypo-exponential distribution)
  - START → A → B → C → END
PH distributions: Example of a project network
PH distributions: Example of a project network
### PH distributions: Example of a project network

<table>
<thead>
<tr>
<th>Activity</th>
<th>SCV</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Dummy start</td>
</tr>
<tr>
<td>1</td>
<td>SCV in [0.33;0.5)</td>
</tr>
<tr>
<td>2</td>
<td>SCV = 1</td>
</tr>
<tr>
<td>3</td>
<td>SCV &gt; 1</td>
</tr>
<tr>
<td>4</td>
<td>Dummy finish</td>
</tr>
</tbody>
</table>

**Example network**

![Diagram of the example network](image)
PH distributions: Example of a project network

<table>
<thead>
<tr>
<th>Activity</th>
<th>SCV</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Dummy start</td>
</tr>
<tr>
<td>1</td>
<td>SCV in [0.33;0.5)</td>
</tr>
<tr>
<td>2</td>
<td>SCV = 1</td>
</tr>
<tr>
<td>3</td>
<td>SCV &gt; 1</td>
</tr>
<tr>
<td>4</td>
<td>Dummy finish</td>
</tr>
</tbody>
</table>
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?

SCV = 0.5 (1/2)

START → A → B → END
PH distributions: What about low variability?

SCV = 0.5 (1/2)

START → A → B → END

SCV = 0.25 (1/4)

START → A → B → C → D → END
PH distributions: What about low variability?

SCV = 0.5 (1/2)

START → A → B → END

SCV = 0.25 (1/4)

START → A → B → C → D → END

SCV = 0.167 (1/6)

START → A → B → C → D → E → F → END
PH distributions: What about low variability?

SCV = 0.5 (1/2)

SCV = 0.25 (1/4)

SCV = 0.167 (1/6)

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
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