# Scheduling Markovian PERT networks with maximum-NPV objective

Stefan Creemers, Marc Lambrecht, Roel Leus

Department of Decision Sciences and Information Management

Katholieke Universiteit Leuven

April 28, 2008

Markov and Markov-regenerative PERT networks
Kulkarni V.G. and Adlaka V.G.
Operations Research (1986) Vol. 34(5) pp.769-781

Markov and Markov-regenerative PERT networks
Kulkarni V.G. and Adlaka V.G.
Operations Research (1986) Vol. 34(5) pp.769-781

- PERT networks with independent exponentially distributed activity durations
- Project execution is a Continuous Time Markov Chain with a single absorbing state (i.e. project completion)
- Early-start policy is optimal

NPV is a nonregular measure of performance, starting activities as soon as possible is not necessarily optimal

Extensive body of literature exists on the deterministic case:

- A.H. Russell (1970)
- R.C. Grinold (1972)
- S. Elmaghraby and W. Herroelen (1990)
- R.H. Möhring, A.S. Schulz, F. Stork and M. Uetz (2001)
- C. Schwindt and J. Zimmermann (2001)

Activity Delay in Stochastic Project Networks
Buss A.H. and Rosenblatt M.J.
Operations Research (1997) Vol. 45(1) pp. 126-139

Activity Delay in Stochastic Project Networks
Buss A.H. and Rosenblatt M.J.
Operations Research (1997) Vol. 45(1) pp. 126-139

- Algorithms to determine delays at the onset of the project (i.e. static decisions)
- Early-start policy after delay
- Performance limited to 25-activity networks

Scheduling projects with stochastic activity duration to maximize EPV
Tilson V., Sobel M.J. and Szmerekovsky J.G.
Submitted Working Paper (2006)

Scheduling projects with stochastic activity duration to maximize EPV
Tilson V., Sobel M.J. and Szmerekovsky J.G.
Submitted Working Paper (2006)

- Optimization over the set of policies that start activities at the end of other activities (dynamic)
- Process is a Continuous Time Markov Decision Chain
- Performance limited to 25-activity networks

Our contribution:

Significant improvement of performance compared to existing models:

- CPU-time reduction up to factor 15
- Memory requirement reduction up to factor 360 (largest statespace analyzed: 867,589,281 states)

Setting:

- Stochastic activity durations (exponentially distributed)
- Expected NPV-objective: incurred cash flow $c_i$ at the start of activity $i$
- Optimization over all policies that start activities at the end of other activities
- No resources

Model outline:

- Definition of the statespace
- Dynamic program to obtain optimal NPV

# Preliminary Concepts

Status of activity $i$ at time $t$:

- not started $\Omega_i(t) = 0$
- in progress $\Omega_i(t) = 1$
- finished $\Omega_i(t) = 2$

$\Omega(t) = (\Omega_0(t), \Omega_1(t), \ldots, \Omega_n(t))$ defines the state of the system

# Preliminary Concepts

Status of activity $i$ at time $t$:

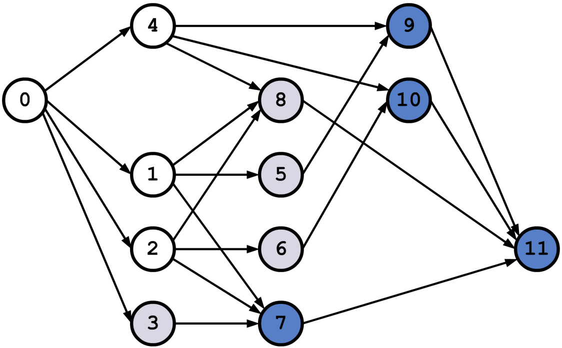
- not started $\Omega_i(t) = 0$
- in progress $\Omega_i(t) = 1$
- finished $\Omega_i(t) = 2$

$\Omega(t) = (\Omega_0(t), \Omega_1(t), \ldots, \Omega_n(t))$ defines the state of the system

Size of statespace $Q$ has upper bound $|Q| = 3^n$

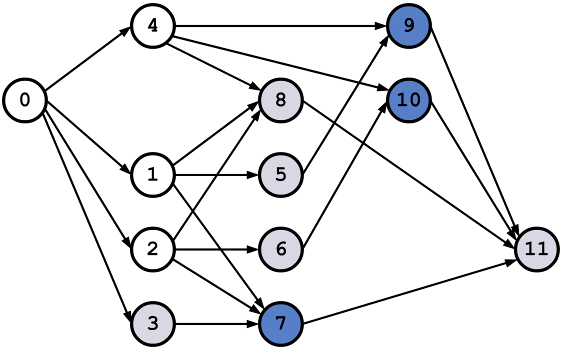
Most of these states do not satisfy precedence constraints
$\Rightarrow$ Strict and clear definition of the statespace is essential

# Example of a Feasible State




Feasible state $\Omega = (2, 2, 2, 1, 2, 1, 1, 0, 1, 0, 0, 0)$

# Example of an Infeasible State




Infeasible state $\Omega = (2, 2, 2, 1, 2, 1, 1, 0, 1, 0, 0, 1)$

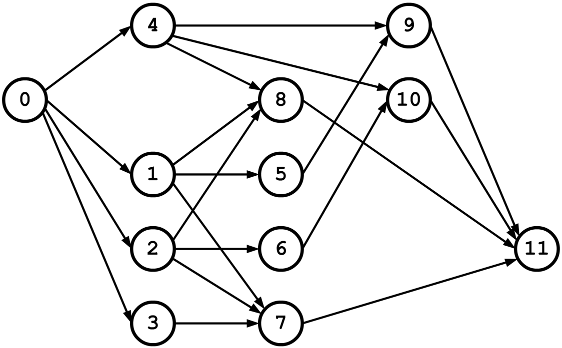
# The UDC-concept

A UDC is an inclusion-maximal set of activities that can be executed in parallel at a given moment in time

Traditionally used in AoA representation, we apply the concept in AoN representation

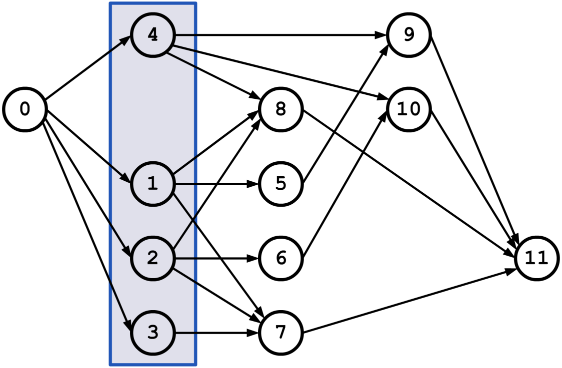
# The UDC-concept

A UDC is a set of all activities that can be executed in parallel at a given moment in time



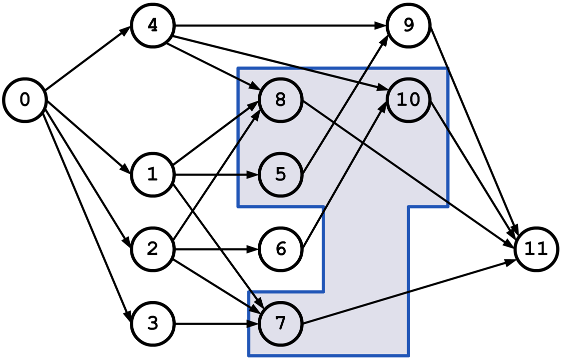
# The UDC-concept

A UDC is a set of all activities that can be executed in parallel at a given moment in time

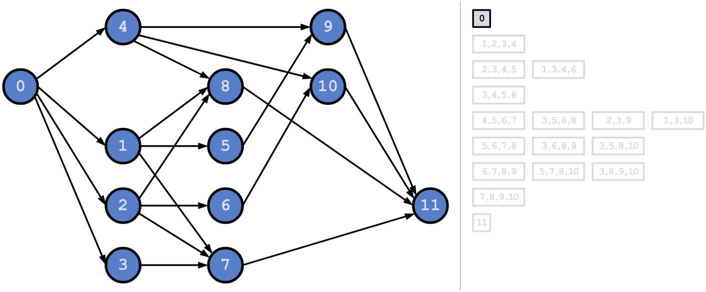


# The UDC-concept

A UDC is a set of all activities that can be executed in parallel at a given moment in time

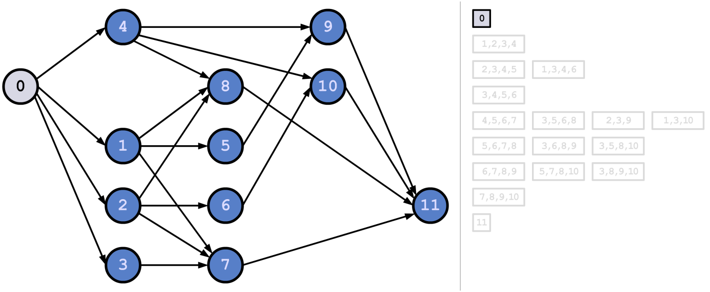


# The UDC-network



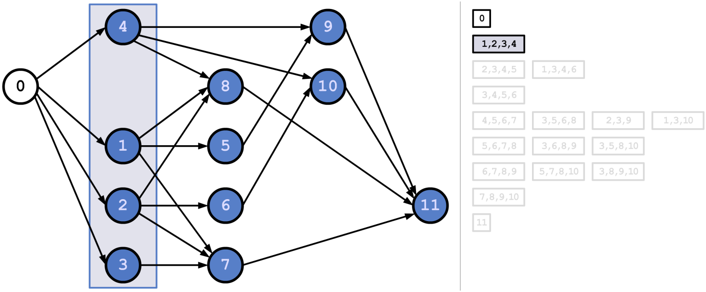

State $\Omega = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$

# The UDC-network



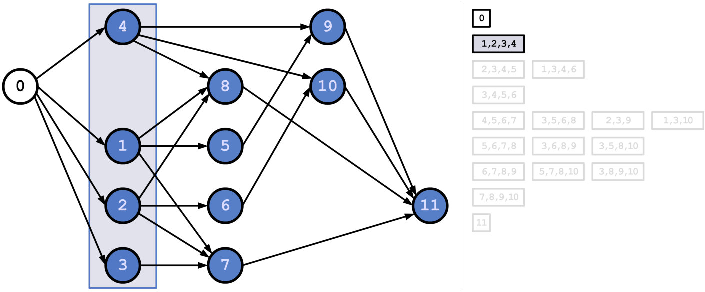

State $\Omega = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$

# The UDC-network




State $\Omega = (2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$

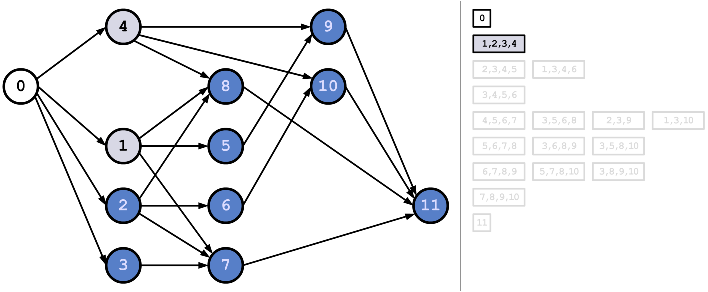
# The UDC-network




State $\Omega = (2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$

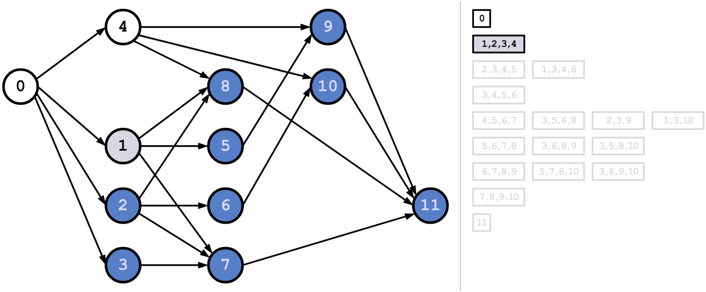
**Lemma 1.** Each feasible state is assigned to a single UDC

# The UDC-network



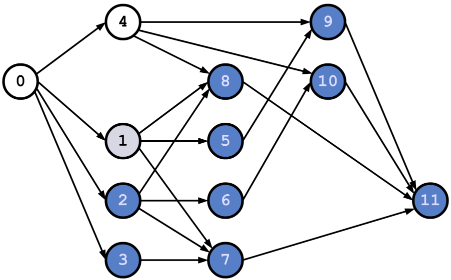

State $\Omega = (2, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0)$

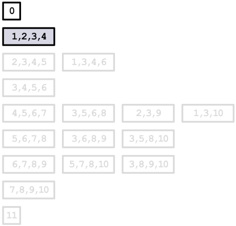
# The UDC-network




State $\Omega = (2, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0)$

# The UDC-network

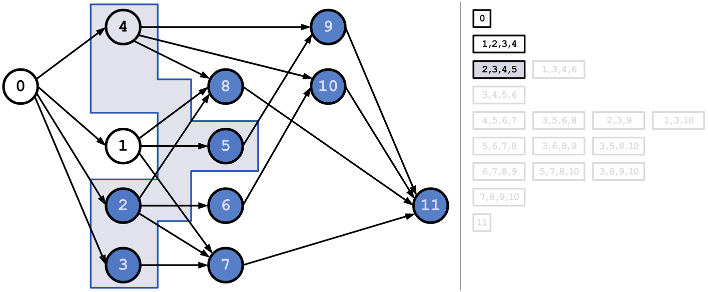



State $\Omega = (2, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0)$

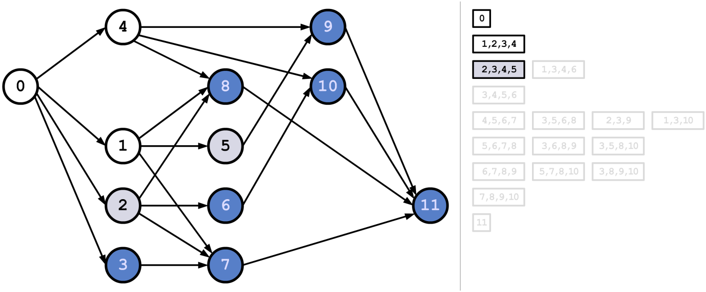
**Lemma 2.** If at least one new activity becomes eligible then the system moves to a different UDC

# The UDC-network



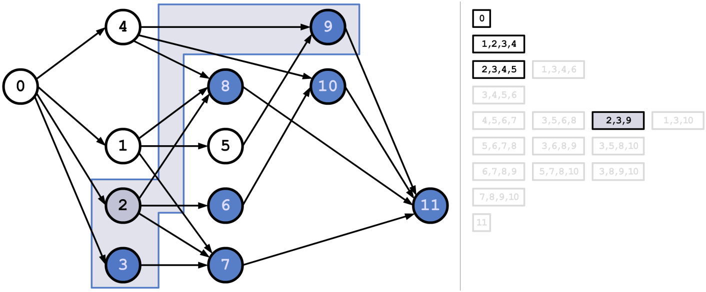

State $\Omega = (2, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0)$

# The UDC-network



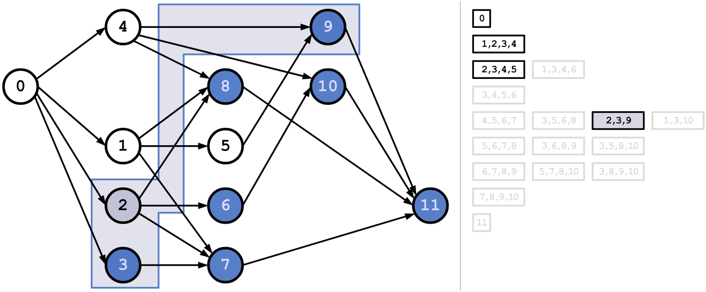

State $\Omega = (2, 2, 1, 0, 2, 1, 0, 0, 0, 0, 0, 0)$

# The UDC-network




State $\Omega = (2, 2, 1, 0, 2, 2, 0, 0, 0, 0, 0, 0)$

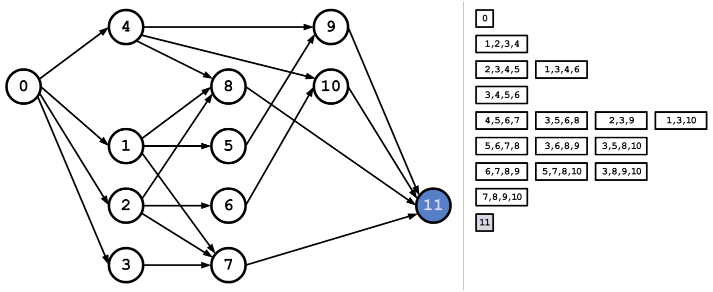
# The UDC-network




State $\Omega = (2, 2, 1, 0, 2, 2, 0, 0, 0, 0, 0, 0)$

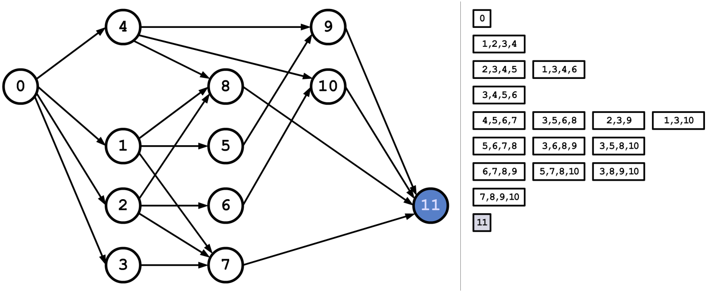
**Lemma 3.** Inter-UDC-transitions can only lead from lower- to higher-ranked UDCs

# The UDC-network




State $\Omega = (2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0)$

# The UDC-network




State $\Omega = (2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0)$

**Observation 1.** Note that the assignment of states to UDCs establishes a partition of Q

**Algorithm** Global algorithmic structure

Generate the UDC-network
Let NPV at state $\Omega = (2, 2, \ldots, 2, 0)$ equal $c_n$
**For all** UDCs in decreasing rank
    Allocate storage for all states in the UDC
    **For all** states
        Determine optimal decision and compute NPV (SDP-recursion)
    **End For**
    **For all** UDCs that are linked to the current UDC
        Reduce the number of incoming links
        **If** there are no more incoming links
            Free storage occupied by the UDC
        **End If**
    **End For**
**End For**

**Lemma 4.** For an arbitrary UDC, in any state, the backward recursion only needs value-function lookups within higher ranked UDCs or within the same UDC for states which have already been evaluated.

| | Tilson et al. | Creemers et al. |
|---|---|---|
| Configuration | Intel Pentium IV | AMD Athlon 64 |
| Clock Speed | 2.8GHz | 1.8GHz |
| PCMark* (CPU) | 3,646 | 2,602 |
| RAM | 512MB | 2,048MB |
| CPU time | 210 sec | 14 sec |
| Max statespace | 600,000 | 268,435,456 |
| | | (867,589,281) |

CPU time reduction: factor 15 (uncorrected)
Memory reduction: factor 360 (corrected)

* CPU benchmarking tool: http://www.futuremark.com/

| $N$ | $N_s$ | | | Average statespace size | | |
|---|---|---|---|---|---|---|
| | $OS = 0.8$ | $OS = 0.6$ | $OS = 0.4$ | $OS = 0.8$ | $OS = 0.6$ | $OS = 0.4$ |
| 10 | 30 | 30 | 30 | 71 | 206 | 695 |
| 20 | 30 | 30 | 30 | 484 | 4, 006 | 55, 016 |
| 30 | 30 | 30 | 30 | 1, 995 | 49, 388 | 1, 560, 364 |
| 40 | 30 | 30 | 29 | 7, 860 | 534, 014 | 47, 072, 515 |
| 50 | 30 | 30 | 4 | 26, 667 | 4, 346, 215 | 526, 020, 237 |
| 60 | 30 | 30 | 0 | 92, 003 | 216, 027, 815 | |
| 70 | 30 | 22 | 0 | 286, 831 | 216, 027, 815 | |
| 80 | 30 | 5 | 0 | 829, 741 | 758, 644, 207 | |
| 90 | 30 | 0 | 0 | 2, 596, 419 | | |
| 100 | 30 | 0 | 0 | 6, 868, 100 | | |
| 110 | 30 | 0 | 0 | 24, 235, 588 | | |
| 120 | | | | | | |

| $N$ | Avg CPU Time NPV | | | Max CPU Time NPV | | |
|---|---|---|---|---|---|---|
| | $OS = 0.8$ | $OS = 0.6$ | $OS = 0.4$ | $OS = 0.8$ | $OS = 0.6$ | $OS = 0.4$ |
| 10 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.03 |
| 20 | 0.00 | 0.03 | 0.90 | 0.00 | 0.08 | 3.77 |
| 30 | 0.01 | 0.64 | 52.98 | 0.02 | 1.78 | 326.16 |
| 40 | 0.06 | 13.29 | 4, 273 | 0.11 | 51.72 | 19, 916 |
| 50 | 0.27 | 171.56 | 99, 216 | 0.66 | 849.42 | 132, 984 |
| 60 | 1.28 | 0.00 | | 5.30 | 0.00 | |
| 70 | 5.37 | 33, 203 | | 15.52 | 114, 424 | |
| 80 | 19.13 | 124, 831 | | 73.09 | 145, 922 | |
| 90 | 86.86 | | | 538 | | |
| 100 | 301 | | | 1, 626 | | |
| 110 | 1, 774 | | | 19, 571 | | |
| 120 | | | | | | |

| $N$ | Average statespace size | | | Maximum statespace size | | |
|---|---|---|---|---|---|---|
| | $OS = 0.8$ | $OS = 0.6$ | $OS = 0.4$ | $OS = 0.8$ | $OS = 0.6$ | $OS = 0.4$ |
| 10 | 71 | 206 | 695 | 105 | 333 | 2,361 |
| 20 | 484 | 4,006 | 55,016 | 953 | 7,673 | 153,441 |
| 30 | 1,995 | 49,388 | 1,560,364 | 3,233 | 84,837 | 5,966,721 |
| 40 | 7,860 | 534,014 | 47,072,515 | 11,945 | 1,543,113 | 146,560,473 |
| 50 | 26,667 | 4,346,215 | 526,020,237 | 53,481 | 13,893,741 | 737,047,953 |
| 60 | 92,003 | 42,278,506 | | 236,889 | 165,102,585 | |
| 70 | 286,831 | 216,027,815 | | 605,649 | 426,644,253 | |
| 80 | 829,741 | 758,644,207 | | 2,278,353 | 867,589,281 | |
| 90 | 2,596,419 | | | 9,322,153 | | |
| 100 | 6,868,100 | | | 22,963,321 | | |
| 110 | 24,235,588 | | | 117,261,489 | | |
| 120 | | | | | | |

| $N$ | Maximum statespace use | | | Average statespace use | | |
|---|---|---|---|---|---|---|
| | $OS = 0.8$ | $OS = 0.6$ | $OS = 0.4$ | $OS = 0.8$ | $OS = 0.6$ | $OS = 0.4$ |
| 10 | 0.41 | 0.55 | 0.63 | 0.25 | 0.37 | 0.44 |
| 20 | 0.38 | 0.49 | 0.62 | 0.22 | 0.27 | 0.38 |
| 30 | 0.30 | 0.44 | 0.55 | 0.15 | 0.24 | 0.30 |
| 40 | 0.31 | 0.46 | 0.52 | 0.15 | 0.28 | 0.29 |
| 50 | 0.33 | 0.46 | 0.28 | 0.16 | 0.24 | 0.17 |
| 60 | 0.37 | 0.49 | | 0.16 | 0.33 | |
| 70 | 0.34 | 0.40 | | 0.16 | 0.19 | |
| 80 | 0.30 | 0.25 | | 0.13 | 0.11 | |
| 90 | 0.35 | | | 0.16 | | |
| 100 | 0.39 | | | 0.17 | | |
| 110 | 0.42 | | | 0.19 | | |
| 120 | | | | | | |