On-the-Fly Exact Computation of Bisimilarity Distances

Tutorial for the prototype implementation

Authors: Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare

This is a tutorial for the prototype implementation of our on-the-fly algorithm for computing the bisimilarity pseudometric of Desharnais et al.
This algorithm has been presented in a paper submitted to TACAS 2013 by the same authors of this tutorial, the *Mathematica* notebook of this tutorial is available here.

Tutorial

■ Encoding and manipulating Markov Chains

A Markov chain $\mathcal{M}$ is encoded as a term of the form MC[π, ℓ] where π and ℓ are respectively the transition funtion and the label function of $\mathcal{M}$.
The set of states is implicitly respresented as a list of integer numbers {1,..., n} indicating the indices of π and ℓ.
The label function ℓ is encoded as a vector $\{l_1, \ldots, l_n\}$ (of integers) indicating that the i-th state in $\mathcal{M}$ is labelled l_i.
The transition function π is encoded as a n × n matrix of the form {{π(1,1),..., π(1,n)},...,{π(n,1),..., π(n,n)}}.

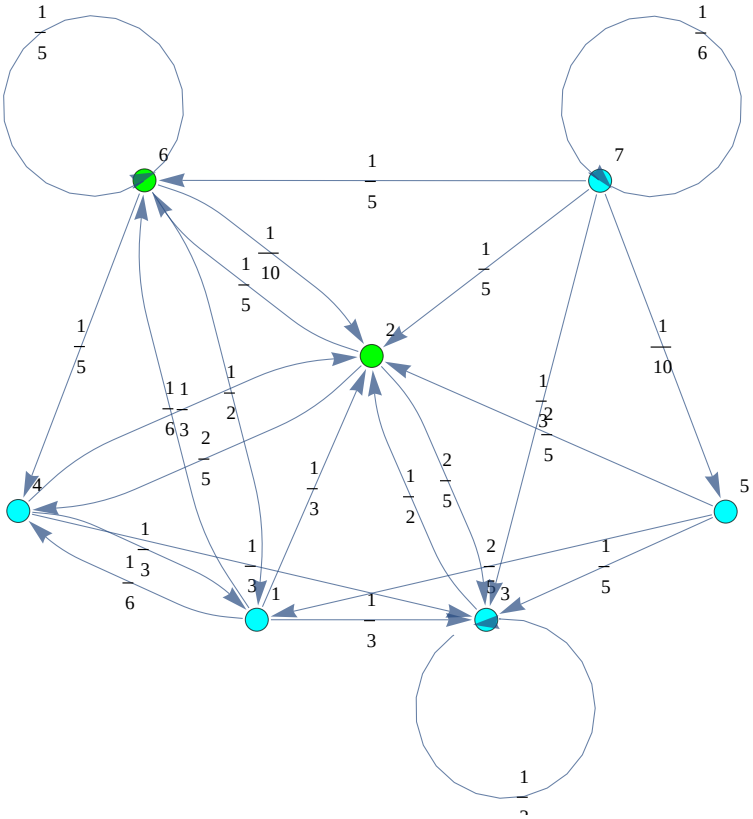
For instance we can encode the Markov chain $\mathcal{M}$ = `MC[tm, labels]` by defining the variables `tm` and `labels` as follows:

In[251]:=
```
labels = {1, 2, 1, 1, 1, 2, 1}; tm =
```
$$\begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} & 0 & \frac{1}{6} & 0 \\ 0 & 0 & \frac{2}{5} & \frac{2}{5} & 0 & \frac{1}{5} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 \\ \frac{2}{5} & \frac{2}{5} & \frac{1}{5} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{10} & 0 & \frac{1}{5} & 0 & \frac{1}{5} & 0 \\ 0 & \frac{1}{5} & \frac{1}{3} & 0 & \frac{1}{10} & \frac{1}{5} & \frac{1}{6} \end{pmatrix}$$
```
;
```

The Markov chain $\mathcal{M}$ can be displayed by means of its undelying graph. Labels are displayed using different colors.

In[252]:=
```
PlotMC[MC[tm, labels]]
```

Out[252]=



Another way to encode a transition function is by means of a list of terms of the form `Edge[i,π(i,j),j]`, for $1 \le$ `i, j` $\le$ n.

In[253]:=
```
EdgesList = {Edge[1, 1 / 3, 2], Edge[1, 1 / 3, 3], Edge[1, 1 / 6, 4], Edge[1, 1 / 6, 6],
    Edge[2, 2 / 5, 4], Edge[2, 2 / 5, 3],
    Edge[2, 1 / 5, 6], Edge[3, 1 / 2, 2], Edge[3, 1 / 2, 3],
    Edge[4, 1 / 3, 1], Edge[4, 1 / 3, 3], Edge[4, 1 / 3, 2],
    Edge[5, 1 / 5, 3], Edge[5, 2 / 5, 1], Edge[5, 2 / 5, 2],
    Edge[6, 1 / 2, 1], Edge[6, 1 / 10, 2], Edge[6, 1 / 5, 4], Edge[6, 1 / 5, 6],
    Edge[7, 1 / 3, 3], Edge[7, 1 / 5, 2],
    Edge[7, 1 / 10, 5], Edge[7, 1 / 5, 6], Edge[7, 1 / 6, 7]};
```

`EdgesList` is then transformed in a transtition matrix using the function `TransitionMatrix[n,EdgeList]`

In[254]:=
```
TransitionMatrix[7, EdgesList] // MatrixForm
```

Out[254]//MatrixForm=

$$\begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} & 0 & \frac{1}{6} & 0 \\ 0 & 0 & \frac{2}{5} & \frac{2}{5} & 0 & \frac{1}{5} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 \\ \frac{2}{5} & \frac{2}{5} & \frac{1}{5} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{10} & 0 & \frac{1}{5} & 0 & \frac{1}{5} & 0 \\ 0 & \frac{1}{5} & \frac{1}{3} & 0 & \frac{1}{10} & \frac{1}{5} & \frac{1}{6} \end{pmatrix}$$

One can check if the encoding corresponds to a well-formed MC by calling `MarkovChainQ`

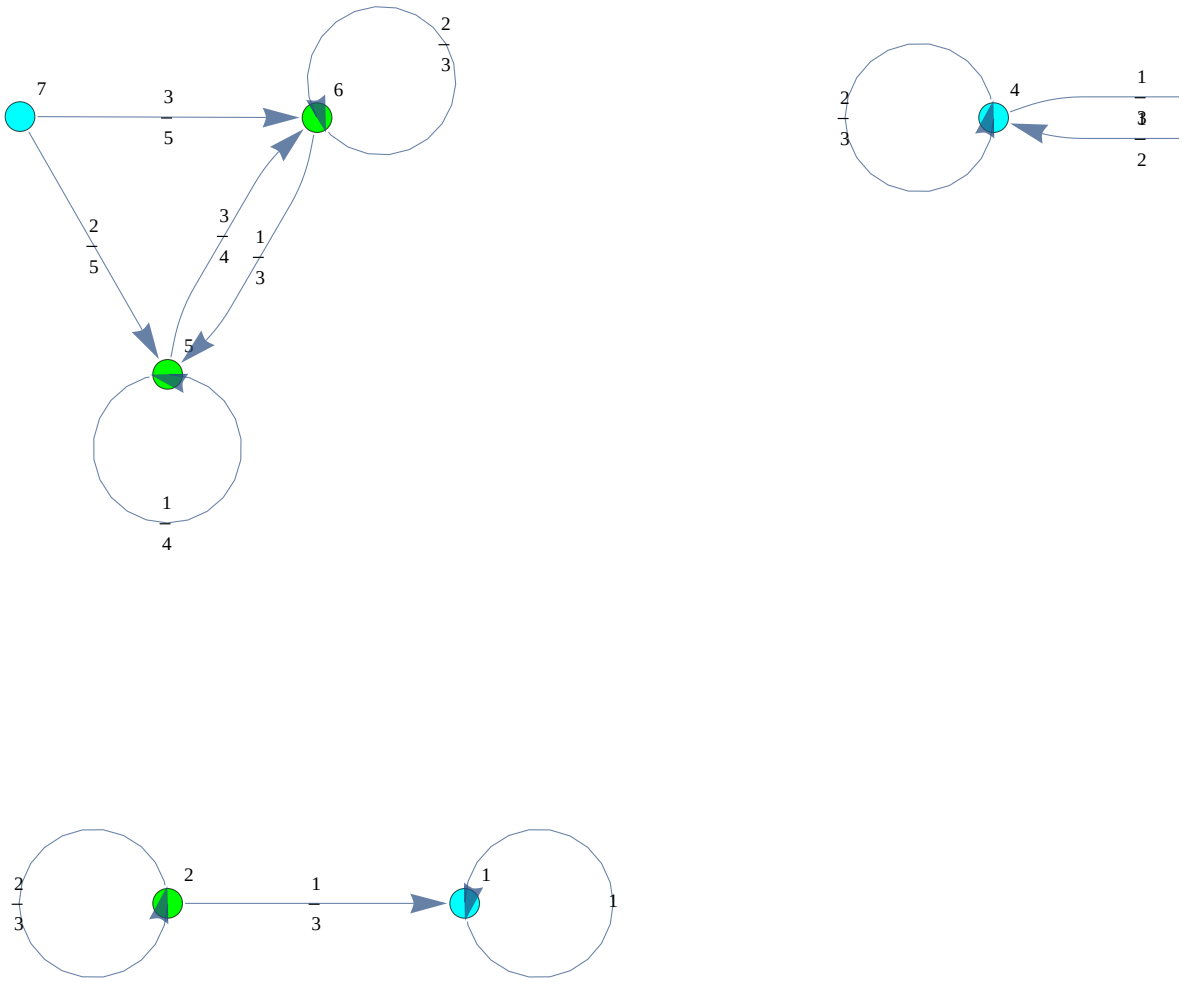
In[255]:=
```
MarkovChainQ[MC[tm, labels]]
```

Out[255]=
```
True
```

Given a sequence of Markov chains $\mathcal{M}_1$, ..., $\mathcal{M}_k$ they can be joined together using the function `JoinMC`

In[256]:=
```
M1 = MC[TransitionMatrix[2,
     {Edge[1, 1, 1], Edge[2, 1 / 3, 1], Edge[2, 2 / 3, 2]}], {1, 2}];
M2 = MC[TransitionMatrix[2, {Edge[1, 1 / 2, 1], Edge[1, 1 / 2, 2],
      Edge[2, 1 / 3, 1], Edge[2, 2 / 3, 2]}], {2, 1}];
M3 = MC[TransitionMatrix[3, {Edge[1, 1 / 4, 1], Edge[1, 3 / 4, 2],
      Edge[2, 1 / 3, 1], Edge[2, 2 / 3, 2],
      Edge[3, 3 / 5, 2], Edge[3, 2 / 5, 1]}], {2, 2, 1}];
JoinMC[M1, M2, M3] // PlotMC
```

Out[259]=



■ Run the on-the-fly algorithm

Given a MC $\mathcal{M}$ = MC[π,ℓ], a discount factor $\lambda \in (0,1]$, and a list of pairs of states Q, the bisimilarity distance of the pairs in Q can be computed as

In[260]:=
```
M = MC[tm, labels]; λ = 1; Q = {{1, 4}, {3, 5}, {1, 2}};
ComputeDistances[M, λ, Q]
```

Out[261]=
$$\left\{\{1, 4\} \to \frac{1}{5}, \{3, 5\} \to \frac{472}{4075}, \{1, 2\} \to 1\right\}$$

If one is interested is tracing the computation he can set the option `Verbose` as `True`

In[262]:= **ComputeDistances[M, λ, {{1, 4}}, Verbose → True]**

Pairs to compute: {{1, 4}}

Picked pair: {{1, 4}}

Reachable states from {{1, 4}} in the current coupling

{1, 4} →

	1	2	3
2	$\frac{1}{3}$	**0**	0
3	0	$\frac{1}{3}$	**0**
4	0	0	$\frac{1}{6}$
6	0	0	$\frac{1}{6}$

{3, 4} →

	1	2	3
2	$\frac{1}{3}$	$\frac{1}{6}$	0
3	0	$\frac{1}{6}$	$\frac{1}{3}$

Variables detected to be zero:{}

Reduced Linear System: $\begin{pmatrix} 1 & -\frac{1}{6} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} d(1,4) \\ d(3,4) \end{pmatrix} == \begin{pmatrix} \frac{5}{6} \\ \frac{1}{2} \end{pmatrix}$; Solution: $\begin{pmatrix} d(1,4) \\ d(3,4) \end{pmatrix} = \begin{pmatrix} \frac{11}{12} \\ \frac{1}{2} \end{pmatrix}$

Non optimal schedule for: {1, 4} reduced cost: −2 for introducing {4, 1} as basic

Reachable states from {{1, 4}} in the current coupling

{1, 4} →

	1	2	3
2	**0**	$\frac{1}{3}$	0
3	0	0	$\frac{1}{3}$
4	$\frac{1}{6}$	0	**0**
6	$\frac{1}{6}$	0	0

Variables detected to be zero:{}

Reduced Linear System: $\left(\frac{5}{6}\right)(d(1,4)) == \left(\frac{1}{6}\right)$; Solution: $(d(1,4)) = \left(\frac{1}{5}\right)$

Non optimal schedule for: {1, 4} reduced cost: $-\frac{3}{10}$ for introducing {1, 3} as basic

Reachable states from {{1, 4}} in the current coupling

{1, 4} →

	1	2	3
2	0	$\frac{1}{3}$	**0**
3	0	0	$\frac{1}{3}$
4	$\frac{1}{6}$	0	0
6	$\frac{1}{6}$	0	**0**

Variables detected to be zero:{}

Reduced Linear System: $\left(\frac{5}{6}\right)(d(1,4)) == \left(\frac{1}{6}\right)$; Solution: $(d(1,4)) = \left(\frac{1}{5}\right)$

Total solved TPs: 2

```
Execution Time: 0.034067 sec
```

Out[262]= $\left\{\{1, 4\} \to \frac{1}{5}\right\}$

If one is iterestes in computing the distance between all pair of states it suffices to ask for `All` pairs

In[263]:= **`ComputeDistances[M, λ, All]`**

Out[263]=
$$\left\{\{1, 1\} \to 0, \{1, 2\} \to 1, \{1, 3\} \to \frac{43}{815}, \{1, 4\} \to \frac{1}{5}, \{1, 5\} \to \frac{1831}{12\,225},\right.$$
$$\{1, 6\} \to 1, \{1, 7\} \to \frac{8171}{61\,125}, \{2, 1\} \to 1, \{2, 2\} \to 0, \{2, 3\} \to 1, \{2, 4\} \to 1,$$
$$\{2, 5\} \to 1, \{2, 6\} \to \frac{23}{163}, \{2, 7\} \to 1, \{3, 1\} \to \frac{43}{815}, \{3, 2\} \to 1, \{3, 3\} \to 0,$$
$$\{3, 4\} \to \frac{143}{815}, \{3, 5\} \to \frac{472}{4075}, \{3, 6\} \to 1, \{3, 7\} \to \frac{627}{4075}, \{4, 1\} \to \frac{1}{5}, \{4, 2\} \to 1,$$
$$\{4, 3\} \to \frac{143}{815}, \{4, 4\} \to 0, \{4, 5\} \to \frac{286}{4075}, \{4, 6\} \to 1, \{4, 7\} \to \frac{22\,507}{183\,375},$$
$$\{5, 1\} \to \frac{1831}{12\,225}, \{5, 2\} \to 1, \{5, 3\} \to \frac{472}{4075}, \{5, 4\} \to \frac{286}{4075}, \{5, 5\} \to 0,$$
$$\{5, 6\} \to 1, \{5, 7\} \to \frac{13\,297}{183\,375}, \{6, 1\} \to 1, \{6, 2\} \to \frac{23}{163}, \{6, 3\} \to 1,$$
$$\{6, 4\} \to 1, \{6, 5\} \to 1, \{6, 6\} \to 0, \{6, 7\} \to 1, \{7, 1\} \to \frac{8171}{61\,125}, \{7, 2\} \to 1,$$
$$\left.\{7, 3\} \to \frac{627}{4075}, \{7, 4\} \to \frac{22\,507}{183\,375}, \{7, 5\} \to \frac{13\,297}{183\,375}, \{7, 6\} \to 1, \{7, 7\} \to 0\right\}$$

The output is returned as a set of ArrayRules, so that one can easily transform it into a matrix using the function SparseArray

In[264]:= **`SparseArray[ComputeDistances[M, λ, All], {7, 7}] // MatrixForm`**

Out[264]//MatrixForm=

$$\begin{pmatrix}
0 & 1 & \frac{43}{815} & \frac{1}{5} & \frac{1831}{12\,225} & 1 & \frac{8171}{61\,125} \\
1 & 0 & 1 & 1 & 1 & \frac{23}{163} & 1 \\
\frac{43}{815} & 1 & 0 & \frac{143}{815} & \frac{472}{4075} & 1 & \frac{627}{4075} \\
\frac{1}{5} & 1 & \frac{143}{815} & 0 & \frac{286}{4075} & 1 & \frac{22\,507}{183\,375} \\
\frac{1831}{12\,225} & 1 & \frac{472}{4075} & \frac{286}{4075} & 0 & 1 & \frac{10\,351}{122\,250} \\
1 & \frac{23}{163} & 1 & 1 & 1 & 0 & 1 \\
\frac{8171}{61\,125} & 1 & \frac{627}{4075} & \frac{22\,507}{183\,375} & \frac{10\,351}{122\,250} & 1 & 0
\end{pmatrix}$$

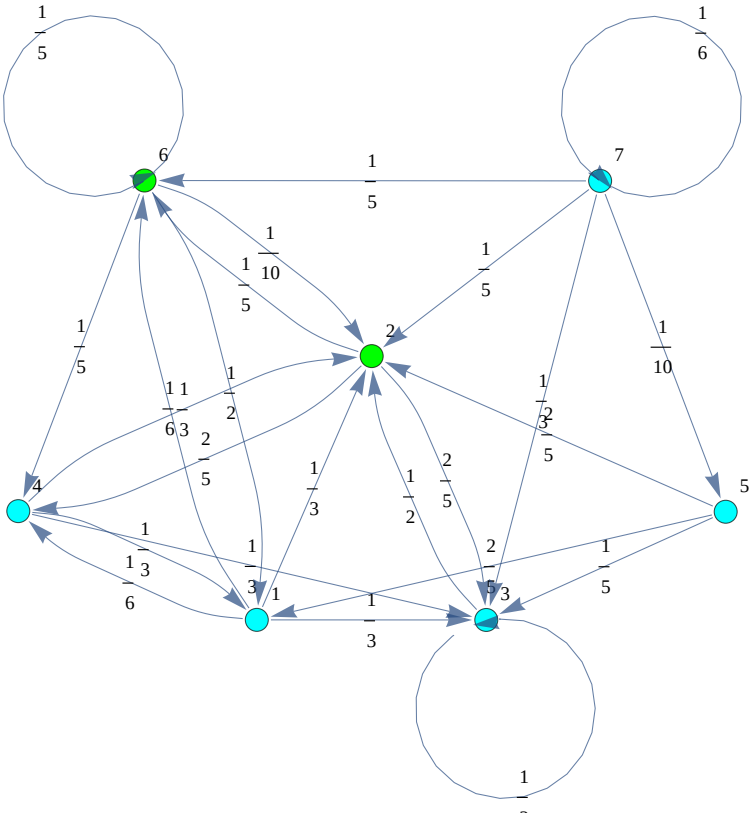
Examples

■ Example taken from the paper submitted at TACAS 2013

Here we use the MC presented in the paper submitted at TACAS 2013 in order to show most of the features of our on-the-fly algorithm.

In[265]:=
```
EdgesPaper =
  {Edge[1, 1 / 3, 2], Edge[1, 1 / 3, 3], Edge[1, 1 / 6, 4], Edge[1, 1 / 6, 6],
   Edge[2, 2 / 5, 4], Edge[2, 2 / 5, 3], Edge[2, 1 / 5, 6],
   Edge[3, 1 / 2, 2], Edge[3, 1 / 2, 3],
   Edge[4, 1 / 3, 1], Edge[4, 1 / 3, 3], Edge[4, 1 / 3, 2],
   Edge[5, 1 / 5, 3], Edge[5, 2 / 5, 1], Edge[5, 2 / 5, 2],
   Edge[6, 1 / 2, 1], Edge[6, 1 / 10, 2], Edge[6, 1 / 5, 4], Edge[6, 1 / 5, 6],
   Edge[7, 1 / 3, 3], Edge[7, 1 / 5, 2],
   Edge[7, 1 / 10, 5], Edge[7, 1 / 5, 6], Edge[7, 1 / 6, 7]};
labelsPaper = {1, 2, 1, 1, 1, 2, 1};
mcPaper = MC[TransitionMatrix[Length@labelsPaper, EdgesPaper], labelsPaper];
PlotMC[mcPaper]
```

Out[268]=



Let us compute the distance between the state 1 and 4, namely d(1,4)

In[269]:=
```
ComputeDistances[mcPaper, 1, {{1, 4}}, Verbose → True]
```

```
Pairs to compute: {{1, 4}}

Picked pair: {{1, 4}}

Reachable states from {{1, 4}} in the current coupling
```

{1, 4} →	1	2	3
2	1/3	0	0
3	0	1/3	0
4	0	0	1/6
6	0	0	1/6

$\{3, 4\} \to$

	1	2	3
2	$\frac{1}{3}$	$\frac{1}{6}$	0
3	0	$\frac{1}{6}$	$\frac{1}{3}$

Variables detected to be zero: {}

Reduced Linear System: $\begin{pmatrix} 1 & -\frac{1}{6} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} d(1,4) \\ d(3,4) \end{pmatrix} == \begin{pmatrix} \frac{5}{6} \\ \frac{1}{2} \end{pmatrix}$; Solution: $\begin{pmatrix} d(1,4) \\ d(3,4) \end{pmatrix} = \begin{pmatrix} \frac{11}{12} \\ \frac{1}{2} \end{pmatrix}$

Non optimal schedule for: {1, 4} reduced cost: −2 for introducing {4, 1} as basic

Reachable states from {{1, 4}} in the current coupling

$\{1, 4\} \to$

	1	2	3
2	**0**	$\frac{1}{3}$	0
3	0	0	$\frac{1}{3}$
4	$\frac{1}{6}$	0	**0**
6	$\frac{1}{6}$	0	0

Variables detected to be zero: {}

Reduced Linear System: $\left(\frac{5}{6}\right)(d(1,4)) == \left(\frac{1}{6}\right)$; Solution: $(d(1,4)) = \left(\frac{1}{5}\right)$

Non optimal schedule for: {1, 4} reduced cost: $-\frac{3}{10}$ for introducing {1, 3} as basic

Reachable states from {{1, 4}} in the current coupling

$\{1, 4\} \to$

	1	2	3
2	0	$\frac{1}{3}$	**0**
3	0	0	$\frac{1}{3}$
4	$\frac{1}{6}$	0	0
6	$\frac{1}{6}$	0	**0**

Variables detected to be zero: {}

Reduced Linear System: $\left(\frac{5}{6}\right)(d(1,4)) == \left(\frac{1}{6}\right)$; Solution: $(d(1,4)) = \left(\frac{1}{5}\right)$

Total solved TPs: 2

Execution Time: 0.037961 sec

Out[269]= $\left\{\{1, 4\} \to \frac{1}{5}\right\}$

Looking the execution trace we see that during the computation of d(1,4) an over-approximation for the distance d(3,4) is computed.
This happens because the we encountered a coupling that demands d(3,4) for computing an over-approximation of d(1,4). However, the first improvement of the current coupling makes the computation of d(3,4) no more demanded for d(1,4), so that no further improvement on (3,4) is made.

On the other hand, as shown in the execution trace that follows, the exact computation of d(3,4) demands an exact computation of d(1,3), d(1,4) and d(2,6)

In[270]:= **ComputeDistances[mcPaper, 1, {{3, 4}}, Verbose → True]**

Pairs to compute: {{3, 4}}

Picked pair: {{3, 4}}

Reachable states from {{3, 4}} in the current coupling

{3, 4} →

	1	2	3
2	$\frac{1}{3}$	$\frac{1}{6}$	0
3	0	$\frac{1}{6}$	$\frac{1}{3}$

Variables detected to be zero: {}

Reduced Linear System: $(1)(d(3,4)) == \left(\frac{1}{2}\right)$; Solution: $(d(3,4)) = \left(\frac{1}{2}\right)$

Non optimal schedule for: {3, 4} reduced cost: −1 for introducing {2, 1} as basic

Reachable states from {{3, 4}} in the current coupling

{1, 3} →

	2	3
2	$\frac{1}{3}$	0
3	$\frac{1}{6}$	$\frac{1}{6}$
4	0	$\frac{1}{6}$
6	0	$\frac{1}{6}$

{3, 4} →

	1	2	3
2	$\frac{1}{6}$	$\frac{1}{3}$	0
3	$\frac{1}{6}$	0	$\frac{1}{3}$

Variables detected to be zero: {}

Reduced Linear System: $\begin{pmatrix} 1 & -\frac{1}{6} \\ -\frac{1}{6} & 1 \end{pmatrix} \begin{pmatrix} d(1,3) \\ d(3,4) \end{pmatrix} == \begin{pmatrix} \frac{1}{3} \\ \frac{1}{6} \end{pmatrix}$; Solution: $\begin{pmatrix} d(1,3) \\ d(3,4) \end{pmatrix} = \begin{pmatrix} \frac{13}{35} \\ \frac{8}{35} \end{pmatrix}$

Non optimal schedule for: {1, 3} reduced cost: −1 for introducing {4, 1} as basic

Reachable states from {{3, 4}} in the current coupling

{1, 3} →

	2	3
2	$\frac{1}{3}$	0
3	0	$\frac{1}{3}$
4	0	$\frac{1}{6}$
6	$\frac{1}{6}$	**0**

{1, 4} →

	1	2	3
2	$\frac{1}{3}$	**0**	0
3	0	$\frac{1}{3}$	**0**
4	0	0	$\frac{1}{6}$
6	0	0	$\frac{1}{6}$

{2, 6} →

	1	2	4	6
3	$\frac{2}{5}$	0	0	0
4	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{5}$	**0**
6	0	0	0	$\frac{1}{5}$

$\{3, 4\} \to$

	1	2	3
2	$\frac{1}{6}$	$\frac{1}{3}$	0
3	$\frac{1}{6}$	0	$\frac{1}{3}$

Variables detected to be zero: $\{\}$

Reduced Linear System: $\begin{pmatrix} 1 & 0 & -\frac{1}{6} & -\frac{1}{6} \\ 0 & 1 & 0 & -\frac{1}{6} \\ -\frac{2}{5} & -\frac{1}{10} & 1 & 0 \\ -\frac{1}{6} & 0 & 0 & 1 \end{pmatrix}$

$\begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} == \begin{pmatrix} 0 \\ \frac{5}{6} \\ \frac{1}{10} \\ \frac{1}{6} \end{pmatrix}$; Solution: $\begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} = \begin{pmatrix} \frac{127}{1955} \\ \frac{1687}{1955} \\ \frac{83}{391} \\ \frac{347}{1955} \end{pmatrix}$

Non optimal schedule for: $\{1, 4\}$ reduced cost: -2 for introducing $\{4, 1\}$ as basic

Reachable states from $\{\{3, 4\}\}$ in the current coupling

$\{1, 3\} \to$

	2	3
2	$\frac{1}{3}$	0
3	0	$\frac{1}{3}$
4	0	$\frac{1}{6}$
6	$\frac{1}{6}$	**0**

$\{1, 4\} \to$

	1	2	3
2	**0**	$\frac{1}{3}$	0
3	$\frac{1}{6}$	0	$\frac{1}{6}$
4	0	0	$\frac{1}{6}$
6	$\frac{1}{6}$	0	0

$\{2, 6\} \to$

	1	2	4	6
3	$\frac{2}{5}$	0	0	0
4	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{5}$	**0**
6	0	0	0	$\frac{1}{5}$

$\{3, 4\} \to$

	1	2	3
2	$\frac{1}{6}$	$\frac{1}{3}$	0
3	$\frac{1}{6}$	0	$\frac{1}{3}$

Variables detected to be zero: $\{\}$

Reduced Linear System: $\begin{pmatrix} 1 & 0 & -\frac{1}{6} & -\frac{1}{6} \\ -\frac{1}{6} & 1 & 0 & -\frac{1}{6} \\ -\frac{2}{5} & -\frac{1}{10} & 1 & 0 \\ -\frac{1}{6} & 0 & 0 & 1 \end{pmatrix}$

$\begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} == \begin{pmatrix} 0 \\ \frac{1}{6} \\ \frac{1}{10} \\ \frac{1}{6} \end{pmatrix}$; Solution: $\begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} = \begin{pmatrix} \frac{103}{1949} \\ \frac{399}{1949} \\ \frac{276}{1949} \\ \frac{342}{1949} \end{pmatrix}$

Non optimal schedule for: {1, 4}

reduced cost: $-\frac{46}{1949}$ for introducing {3, 1} as basic

Reachable states from {{3, 4}} in the current coupling

{1, 3} →

	2	3
2	$\frac{1}{3}$	0
3	0	$\frac{1}{3}$
4	0	$\frac{1}{6}$
6	$\frac{1}{6}$	**0**

{1, 4} →

	1	2	3
2	**0**	$\frac{1}{3}$	0
3	0	0	$\frac{1}{3}$
4	$\frac{1}{6}$	0	**0**
6	$\frac{1}{6}$	0	0

{2, 6} →

	1	2	4	6
3	$\frac{2}{5}$	0	0	0
4	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{5}$	**0**
6	0	0	0	$\frac{1}{5}$

{3, 4} →

	1	2	3
2	$\frac{1}{6}$	$\frac{1}{3}$	0
3	$\frac{1}{6}$	0	$\frac{1}{3}$

Variables detected to be zero:{}

Reduced Linear System: $\begin{pmatrix} 1 & 0 & -\frac{1}{6} & -\frac{1}{6} \\ 0 & \frac{5}{6} & 0 & 0 \\ -\frac{2}{5} & -\frac{1}{10} & 1 & 0 \\ -\frac{1}{6} & 0 & 0 & 1 \end{pmatrix}$

$\begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} == \begin{pmatrix} 0 \\ \frac{1}{6} \\ \frac{1}{10} \\ \frac{1}{6} \end{pmatrix}$; Solution: $\begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} = \begin{pmatrix} \frac{43}{815} \\ \frac{1}{5} \\ \frac{23}{163} \\ \frac{143}{815} \end{pmatrix}$

Total solved TPs: 4

Execution Time: 0.090483 sec

Out[270]= $\left\{\{3, 4\} \to \frac{143}{815}\right\}$

We have seen that the exact computation of d(3,4) demands for the exact compuatation of d(1,3), d(1,4), d(2,6).
One would claim that if we ask for the exact computation of d(3,4), d(1,4) and d(2,6) the compuatation will do the same steps made for d(3,4).
In some sense this is true, since we will need to compute the same set of distances, however we have seen that the exact computation of d(1,4) does not need an exact compuation of the remaining pairs.
Thus, considering the pair (1,4) before (3,4) will allow us to reduce the size of the linear equation systems computed for d(3,4).

In[271]:= **ComputeDistances[mcPaper, 1, {{1, 4}, {2, 6}, {3, 4}}, Verbose → True]**

Pairs to compute: {{1, 4}, {2, 6}, {3, 4}}

Picked pair: {{2, 6}}

Reachable states from {{2, 6}} in the current coupling

{1, 3} →

	2	3
2	$\frac{1}{3}$	0
3	$\frac{1}{6}$	$\frac{1}{6}$
4	0	$\frac{1}{6}$
6	0	$\frac{1}{6}$

{1, 4} →

	1	2	3
2	$\frac{1}{3}$	**0**	0
3	0	$\frac{1}{3}$	**0**
4	0	0	$\frac{1}{6}$
6	0	0	$\frac{1}{6}$

{2, 6} →

	1	2	4	6
3	$\frac{2}{5}$	0	0	0
4	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{5}$	**0**
6	0	0	0	$\frac{1}{5}$

$\{3, 4\} \to$

	1	2	3
2	$\frac{1}{3}$	$\frac{1}{6}$	0
3	0	$\frac{1}{6}$	$\frac{1}{3}$

Variables detected to be zero: $\{\}$

Reduced Linear System: $\begin{pmatrix} 1 & 0 & 0 & -\frac{1}{6} \\ 0 & 1 & 0 & -\frac{1}{6} \\ -\frac{2}{5} & -\frac{1}{10} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} == \begin{pmatrix} \frac{1}{3} \\ \frac{5}{6} \\ \frac{1}{10} \\ \frac{1}{2} \end{pmatrix}$; Solution: $\begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} = \begin{pmatrix} \frac{5}{12} \\ \frac{11}{12} \\ \frac{43}{120} \\ \frac{1}{2} \end{pmatrix}$

Non optimal schedule for: $\{1, 3\}$ reduced cost: $-\frac{197}{120}$ for introducing $\{4, 1\}$ as basic

Reachable states from $\{\{2, 6\}\}$ in the current coupling

$\{1, 3\} \to$

	2	3
2	$\frac{1}{3}$	0
3	0	$\frac{1}{3}$
4	0	$\frac{1}{6}$
6	$\frac{1}{6}$	**0**

$\{1, 4\} \to$

	1	2	3
2	$\frac{1}{3}$	**0**	0
3	0	$\frac{1}{3}$	**0**
4	0	0	$\frac{1}{6}$
6	0	0	$\frac{1}{6}$

$\{2, 6\} \to$

	1	2	4	6
3	$\frac{2}{5}$	0	0	0
4	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{5}$	**0**
6	0	0	0	$\frac{1}{5}$

$\{3, 4\} \to$

	1	2	3
2	$\frac{1}{3}$	$\frac{1}{6}$	0
3	0	$\frac{1}{6}$	$\frac{1}{3}$

Variables detected to be zero: $\{\}$

Reduced Linear System: $\begin{pmatrix} 1 & 0 & -\frac{1}{6} & -\frac{1}{6} \\ 0 & 1 & 0 & -\frac{1}{6} \\ -\frac{2}{5} & -\frac{1}{10} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$\begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} == \begin{pmatrix} 0 \\ \frac{5}{6} \\ \frac{1}{10} \\ \frac{1}{2} \end{pmatrix}$; Solution: $\begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} = \begin{pmatrix} \frac{83}{672} \\ \frac{11}{12} \\ \frac{27}{112} \\ \frac{1}{2} \end{pmatrix}$

Non optimal schedule for: {1, 4} reduced cost: -2 for introducing {4, 1} as basic

Reachable states from {{2, 6}} in the current coupling

{1, 3} →

	2	3
2	$\frac{1}{3}$	0
3	0	$\frac{1}{3}$
4	0	$\frac{1}{6}$
6	$\frac{1}{6}$	**0**

{1, 4} →

	1	2	3
2	**0**	$\frac{1}{3}$	0
3	$\frac{1}{6}$	0	$\frac{1}{6}$
4	0	0	$\frac{1}{6}$
6	$\frac{1}{6}$	0	0

{2, 6} →

	1	2	4	6
3	$\frac{2}{5}$	0	0	0
4	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{5}$	**0**
6	0	0	0	$\frac{1}{5}$

{3, 4} →

	1	2	3
2	$\frac{1}{3}$	$\frac{1}{6}$	0
3	0	$\frac{1}{6}$	$\frac{1}{3}$

Variables detected to be zero:{}

Reduced Linear System: $\begin{pmatrix} 1 & 0 & -\frac{1}{6} & -\frac{1}{6} \\ -\frac{1}{6} & 1 & 0 & -\frac{1}{6} \\ -\frac{2}{5} & -\frac{1}{10} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$\begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} == \begin{pmatrix} 0 \\ \frac{1}{6} \\ \frac{1}{10} \\ \frac{1}{2} \end{pmatrix}$; Solution: $\begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} = \begin{pmatrix} \frac{15}{134} \\ \frac{18}{67} \\ \frac{23}{134} \\ \frac{1}{2} \end{pmatrix}$

Non optimal schedule for: {1, 4} reduced cost: $-\frac{23}{67}$ for introducing {3, 1} as basic

Reachable states from {{2, 6}} in the current coupling

{1, 3} →

	2	3
2	$\frac{1}{3}$	0
3	0	$\frac{1}{3}$
4	0	$\frac{1}{6}$
6	$\frac{1}{6}$	**0**

{1, 4} →

	1	2	3
2	0	$\frac{1}{3}$	**0**
3	0	0	$\frac{1}{3}$
4	$\frac{1}{6}$	0	0
6	$\frac{1}{6}$	0	**0**

{2, 6} →

	1	2	4	6
3	$\frac{2}{5}$	0	0	0
4	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{5}$	**0**
6	0	0	0	$\frac{1}{5}$

{3, 4} →

	1	2	3
2	$\frac{1}{3}$	$\frac{1}{6}$	0
3	0	$\frac{1}{6}$	$\frac{1}{3}$

Variables detected to be zero:{}

Reduced Linear System: $\begin{pmatrix} 1 & 0 & -\frac{1}{6} & -\frac{1}{6} \\ 0 & \frac{5}{6} & 0 & 0 \\ -\frac{2}{5} & -\frac{1}{10} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$\begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} == \begin{pmatrix} 0 \\ \frac{1}{6} \\ \frac{1}{10} \\ \frac{1}{2} \end{pmatrix}$; Solution: $\begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} = \begin{pmatrix} \frac{31}{280} \\ \frac{1}{5} \\ \frac{23}{140} \\ \frac{1}{2} \end{pmatrix}$

Non optimal schedule for: {3, 4} reduced cost: $-\frac{529}{280}$ for introducing {2, 1} as basic

Reachable states from {{2, 6}} in the current coupling

{1, 3} →

	2	3
2	$\frac{1}{3}$	0
3	0	$\frac{1}{3}$
4	0	$\frac{1}{6}$
6	$\frac{1}{6}$	**0**

{1, 4} →

	1	2	3
2	0	$\frac{1}{3}$	**0**
3	0	0	$\frac{1}{3}$
4	$\frac{1}{6}$	0	0
6	$\frac{1}{6}$	0	**0**

{2, 6} →

	1	2	4	6
3	$\frac{2}{5}$	0	0	0
4	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{5}$	**0**
6	0	0	0	$\frac{1}{5}$

{3, 4} →

	1	2	3
2	$\frac{1}{6}$	$\frac{1}{3}$	0
3	$\frac{1}{6}$	0	$\frac{1}{3}$

Variables detected to be zero:{}

Reduced Linear System: $\begin{pmatrix} 1 & 0 & -\frac{1}{6} & -\frac{1}{6} \\ 0 & \frac{5}{6} & 0 & 0 \\ -\frac{2}{5} & -\frac{1}{10} & 1 & 0 \\ -\frac{1}{6} & 0 & 0 & 1 \end{pmatrix}$

$\begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} == \begin{pmatrix} 0 \\ \frac{1}{6} \\ \frac{1}{10} \\ \frac{1}{6} \end{pmatrix}$; Solution: $\begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} = \begin{pmatrix} \frac{43}{815} \\ \frac{1}{5} \\ \frac{23}{163} \\ \frac{143}{815} \end{pmatrix}$

Non optimal schedule for: {1, 4} reduced cost: $-\frac{4}{163}$ for introducing {3, 3} as basic

Reachable states from {{2, 6}} in the current coupling

{1, 3} →

	2	3
2	$\frac{1}{3}$	0
3	0	$\frac{1}{3}$
4	0	$\frac{1}{6}$
6	$\frac{1}{6}$	**0**

{1, 4} →

	1	2	3
2	**0**	$\frac{1}{3}$	0
3	0	0	$\frac{1}{3}$
4	$\frac{1}{6}$	0	**0**
6	$\frac{1}{6}$	0	0

{2, 6} →

	1	2	4	6
3	$\frac{2}{5}$	0	0	0
4	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{5}$	**0**
6	0	0	0	$\frac{1}{5}$

{3, 4} →

	1	2	3
2	$\frac{1}{6}$	$\frac{1}{3}$	0
3	$\frac{1}{6}$	0	$\frac{1}{3}$

Variables detected to be zero:{}

Reduced Linear System: $\begin{pmatrix} 1 & 0 & -\frac{1}{6} & -\frac{1}{6} \\ 0 & \frac{5}{6} & 0 & 0 \\ -\frac{2}{5} & -\frac{1}{10} & 1 & 0 \\ -\frac{1}{6} & 0 & 0 & 1 \end{pmatrix}$

$\begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} == \begin{pmatrix} 0 \\ \frac{1}{6} \\ \frac{1}{10} \\ \frac{1}{6} \end{pmatrix}$; Solution: $\begin{pmatrix} d(1,3) \\ d(1,4) \\ d(2,6) \\ d(3,4) \end{pmatrix} = \begin{pmatrix} \frac{43}{815} \\ \frac{1}{5} \\ \frac{23}{163} \\ \frac{143}{815} \end{pmatrix}$

Total solved TPs: 5

Execution Time: 0.132869 sec

Out[271]= $\left\{\{1, 4\} \to \frac{1}{5}, \{2, 6\} \to \frac{23}{163}, \{3, 4\} \to \frac{143}{815}\right\}$

Another way to limit the exploration of the MC is to give estimates for some distances. For instance, we have seen that for computing the distance d(3,4) we need to solve exactly
the problem for the pair (2,6). Assume we have an over-approximation of the exact value for d(2,6), we can use this information in order to further reduce the exploration of the MC.

In[272]:=
```
ComputeDistances[mcPaper, 1, {{3, 4}},
  Verbose → True, Estimates → {{2, 6} -> 25/163}]
```

Pairs to compute: {{3, 4}}

Picked pair: {{3, 4}}

Reachable states from {{3, 4}} in the current coupling

{3, 4} →

	1	2	3
2	$\frac{1}{3}$	$\frac{1}{6}$	0
3	0	$\frac{1}{6}$	$\frac{1}{3}$

Variables detected to be zero:{}

Reduced Linear System: (1)(d(3,4)) == $\left(\frac{1}{2}\right)$; Solution:(d(3,4)) = $\left(\frac{1}{2}\right)$

Non optimal schedule for: {3, 4} reduced cost: -1 for introducing {2, 1} as basic

Reachable states from {{3, 4}} in the current coupling

{1, 3} →

	2	3
2	$\frac{1}{3}$	0
3	$\frac{1}{6}$	$\frac{1}{6}$
4	0	$\frac{1}{6}$
6	0	$\frac{1}{6}$

{3, 4} →

	1	2	3
2	$\frac{1}{6}$	$\frac{1}{3}$	0
3	$\frac{1}{6}$	0	$\frac{1}{3}$

Variables detected to be zero:{}

Reduced Linear System: $\begin{pmatrix} 1 & -\frac{1}{6} \\ -\frac{1}{6} & 1 \end{pmatrix} \begin{pmatrix} d(1,3) \\ d(3,4) \end{pmatrix} == \begin{pmatrix} \frac{1}{3} \\ \frac{1}{6} \end{pmatrix}$; Solution: $\begin{pmatrix} d(1,3) \\ d(3,4) \end{pmatrix} = \begin{pmatrix} \frac{13}{35} \\ \frac{8}{35} \end{pmatrix}$

Non optimal schedule for: {1, 3} reduced cost: $-\frac{301}{163}$ for introducing {4, 1} as basic

Reachable states from {{3, 4}} in the current coupling

{1, 3} →

	2	3
2	$\frac{1}{3}$	0
3	0	$\frac{1}{3}$
4	0	$\frac{1}{6}$
6	$\frac{1}{6}$	**0**

{3, 4} →

	1	2	3
2	$\frac{1}{6}$	$\frac{1}{3}$	0
3	$\frac{1}{6}$	0	$\frac{1}{3}$

Variables detected to be zero:{}

Reduced Linear System: $\begin{pmatrix} 1 & -\frac{1}{6} \\ -\frac{1}{6} & 1 \end{pmatrix} \begin{pmatrix} d(1,3) \\ d(3,4) \end{pmatrix} == \begin{pmatrix} \frac{25}{978} \\ \frac{1}{6} \end{pmatrix}$; Solution: $\begin{pmatrix} d(1,3) \\ d(3,4) \end{pmatrix} = \begin{pmatrix} \frac{313}{5705} \\ \frac{1003}{5705} \end{pmatrix}$

Total solved TPs: 2

Execution Time: 0.031808 sec

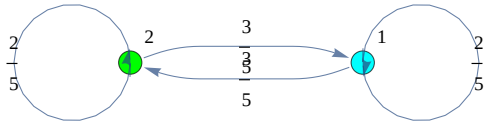
Out[272]= $\left\{ \{3, 4\} \to \frac{1003}{5705} \right\}$

Using the estimate $\frac{25}{163}$ for the distance d(2,6) we considerably reduced the number of steps for computing a good over-approximation for d(3,4).

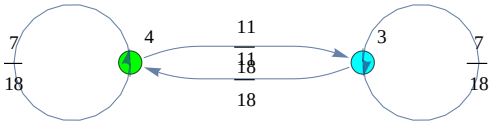
■ Unfair Tossing Coins

In this example we model two unfair flipping coins.

In[273]:=
```
EdgesCoin[e_] := {Edge[1, 1 / 2 - e, 1],
   Edge[1, 1 / 2 + e, 2], Edge[2, 1 / 2 - e, 2], Edge[2, 1 / 2 + e, 1]};
labelCoin := {1, 2};
(mcCoin = JoinMC[MC[TransitionMatrix[2, EdgesCoin[1 / 10]], labelCoin],
    MC[TransitionMatrix[2, EdgesCoin[1 / 9]], labelCoin]]) // PlotMC
```



Out[275]=



In[276]:=
```
ComputeDistances[mc, 1, All, Verbose → True]
```

Pairs to compute: {{1, 3}, {2, 4}}

Picked pair: {{1, 3}}

Reachable states from {{1, 3}} in the current coupling

{1, 3} →

	3	4
1	$\frac{7}{18}$	$\frac{1}{90}$
2	0	$\frac{3}{5}$

{2, 4} →

	3	4
1	$\frac{3}{5}$	0
2	$\frac{1}{90}$	$\frac{7}{18}$

Variables detected to be zero:{}

Reduced Linear System: $\begin{pmatrix} \frac{11}{18} & -\frac{3}{5} \\ -\frac{3}{5} & \frac{11}{18} \end{pmatrix} \begin{pmatrix} d(1,3) \\ d(2,4) \end{pmatrix} == \begin{pmatrix} \frac{1}{90} \\ \frac{1}{90} \end{pmatrix}$; Solution: $\begin{pmatrix} d(1,3) \\ d(2,4) \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

Total solved TPs: 0

Execution Time: 0.009969 sec

Out[276]=
```
{{1, 1} → 0, {1, 2} → 1, {1, 3} → 1, {1, 4} → 1, {2, 1} → 1,
 {2, 2} → 0, {2, 3} → 1, {2, 4} → 1, {3, 1} → 1, {3, 2} → 1,
 {3, 3} → 0, {3, 4} → 1, {4, 1} → 1, {4, 2} → 1, {4, 3} → 1, {4, 4} → 0}
```

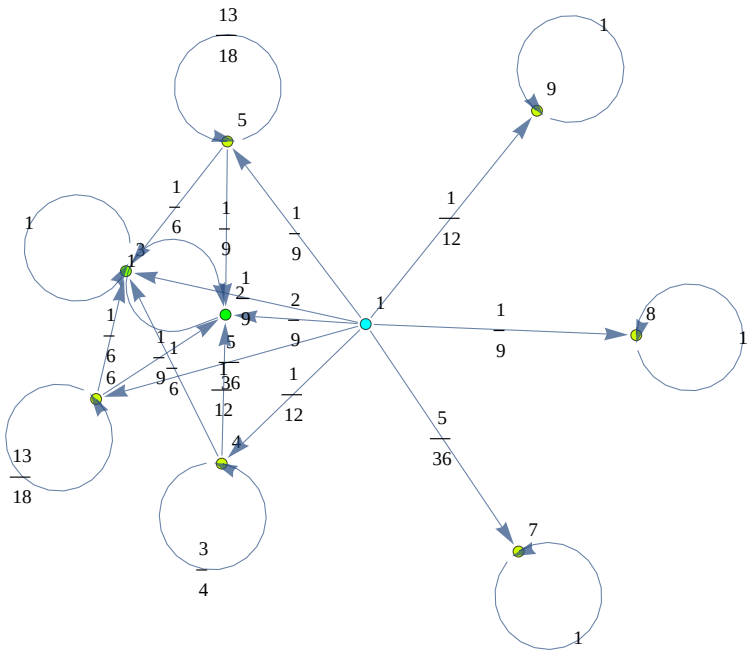
■ The Craps Game

Here we present two different MCs modeling two slightly different versions for the “craps game”. These models have both been taken from the book Principles of Model Checking, Examples 10.4 and 10.23 respectivelty. We compare the two models by means of the distance beteween their initial states.

In[277]:=
```
start = 1; won = 2; lost = 3; other = 4; (* labels *)
labelsGame1 = {start, won, lost, other, other, other, other, other, other};
edgeGame1 =
  {Edge[1, 1 / 12, 4], Edge[1, 1 / 12, 9], Edge[1, 1 / 9, 5], Edge[1, 1 / 9, 8],
   Edge[1, 5 / 36, 6], Edge[1, 5 / 36, 7], Edge[1, 2 / 9, won], Edge[1, 1 / 9, lost],
   Edge[won, 1, won], Edge[lost, 1, lost], Edge[4, 3 / 4, 4],
   Edge[4, 1 / 12, won], Edge[4, 1 / 6, lost], Edge[9, 1, 9],
   Edge[5, 13 / 18, 5], Edge[5, 1 / 9, won], Edge[5, 1 / 6, lost], Edge[8, 1, 8],
   Edge[6, 13 / 18, 6], Edge[6, 1 / 9, won], Edge[6, 1 / 6, lost], Edge[7, 1, 7]};
```

In[280]:=
```
(game1 = MC[TransitionMatrix[Length@labelsGame1, edgeGame1], labelsGame1]) //
 PlotMC
```

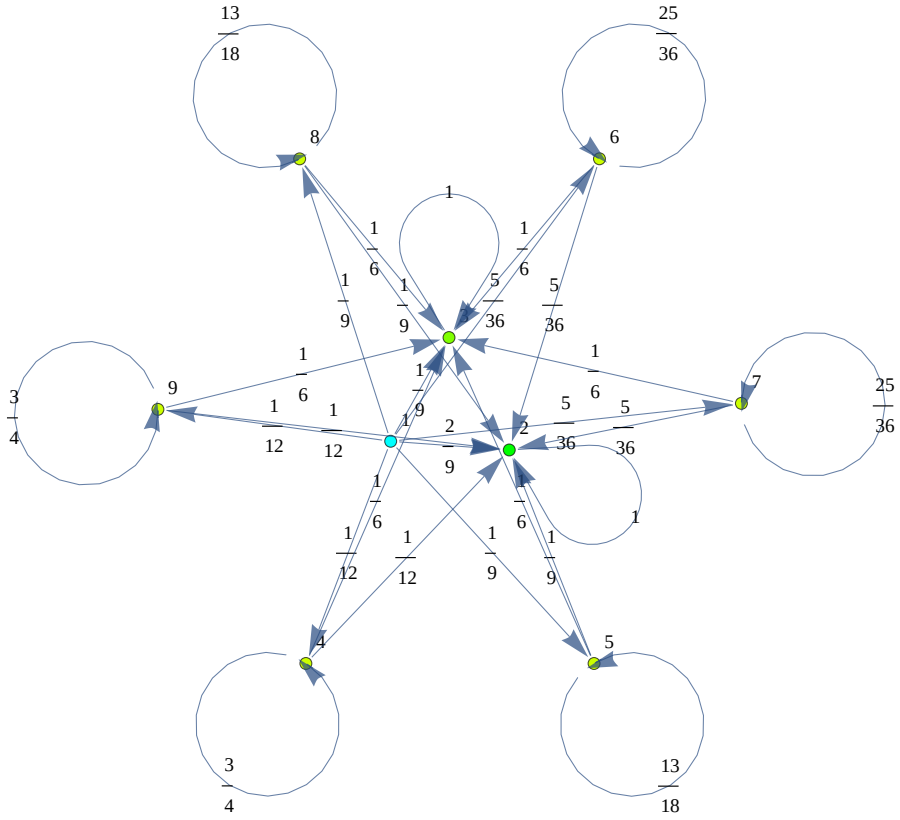
Out[280]=



In[281]:=
```
labelsGame2 = labelsGame1;
edgeGame2 =
  {Edge[1, 1 / 12, 4], Edge[1, 1 / 12, 9], Edge[1, 1 / 9, 5], Edge[1, 1 / 9, 8],
   Edge[1, 5 / 36, 6], Edge[1, 5 / 36, 7], Edge[1, 2 / 9, won], Edge[1, 1 / 9, lost],
   Edge[won, 1, won], Edge[lost, 1, lost],
   Edge[4, 3 / 4, 4], Edge[4, 1 / 12, won], Edge[4, 1 / 6, lost],
   Edge[9, 3 / 4, 9], Edge[9, 1 / 12, won], Edge[9, 1 / 6, lost],
   Edge[5, 13 / 18, 5], Edge[5, 1 / 9, won], Edge[5, 1 / 6, lost],
   Edge[8, 13 / 18, 8], Edge[8, 1 / 9, won], Edge[8, 1 / 6, lost],
   Edge[6, 25 / 36, 6], Edge[6, 5 / 36, won], Edge[6, 1 / 6, lost],
   Edge[7, 25 / 36, 7], Edge[7, 5 / 36, won], Edge[7, 1 / 6, lost]};
```

In[283]:=
```
(game2 = MC[TransitionMatrix[Length@labelsGame2, edgeGame2], labelsGame2]) //
 PlotMC
```

Out[283]=



In[284]:=
```
ComputeDistances[JoinMC[game1, game2], 1, {{1, 10}}, Verbose → True]
```

Pairs to compute: {{1, 10}}

Picked pair: {{1, 10}}

Reachable states from {{1, 10}} in the current coupling

{1, 10} →	11	12	13	14	15	16	17	18
2	$\frac{2}{9}$	**0**	0	0	0	0	0	0
3	0	$\frac{1}{9}$	**0**	0	0	0	0	0
4	0	0	$\frac{1}{12}$	**0**	0	0	0	0
5	0	0	0	$\frac{1}{9}$	**0**	0	0	0
6	0	0	0	0	$\frac{5}{36}$	**0**	0	0
7	0	0	0	0	0	$\frac{5}{36}$	**0**	0
8	0	0	0	0	0	0	$\frac{1}{9}$	**0**
9	0	0	0	0	0	0	0	$\frac{1}{12}$

{2, 11} →	11
2	**1**

{3, 12} →	12
3	**1**

$\{4, 13\} \to$

	11	12	13
2	$\frac{1}{12}$	**0**	0
3	0	$\frac{1}{6}$	**0**
4	0	0	$\frac{3}{4}$

$\{5, 14\} \to$

	11	12	14
2	$\frac{1}{9}$	**0**	0
3	0	$\frac{1}{6}$	**0**
5	0	0	$\frac{13}{18}$

$\{6, 15\} \to$

	11	12	15
2	$\frac{1}{9}$	0	0
3	$\frac{1}{36}$	$\frac{5}{36}$	0
6	0	$\frac{1}{36}$	$\frac{25}{36}$

$\{7, 16\} \to$

	11	12	16
7	$\frac{5}{36}$	$\frac{1}{6}$	$\frac{25}{36}$

$\{8, 17\} \to$

	11	12	17
8	$\frac{1}{9}$	$\frac{1}{6}$	$\frac{13}{18}$

$\{9, 18\} \to$

	11	12	18
9	$\frac{1}{12}$	$\frac{1}{6}$	$\frac{3}{4}$

Variables detected to be zero:{d(2,11), d(3,12), d(4,13), d(5,14)}

Reduced Linear System: $\begin{pmatrix} 1 & -\frac{5}{36} & -\frac{5}{36} & -\frac{1}{9} & -\frac{1}{12} \\ 0 & \frac{11}{36} & 0 & 0 & 0 \\ 0 & 0 & \frac{11}{36} & 0 & 0 \\ 0 & 0 & 0 & \frac{5}{18} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} \end{pmatrix}$

$\begin{pmatrix} d(1,10) \\ d(6,15) \\ d(7,16) \\ d(8,17) \\ d(9,18) \end{pmatrix} == \begin{pmatrix} 0 \\ \frac{1}{18} \\ \frac{11}{36} \\ \frac{5}{18} \\ \frac{1}{4} \end{pmatrix}$; Solution: $\begin{pmatrix} d(1,10) \\ d(6,15) \\ d(7,16) \\ d(8,17) \\ d(9,18) \end{pmatrix} = \begin{pmatrix} \frac{71}{198} \\ \frac{2}{11} \\ 1 \\ 1 \\ 1 \end{pmatrix}$

Non optimal schedule for: {1, 10} reduced cost: $-\frac{42}{11}$ for introducing {1, 6} as basic

Reachable states from {{1, 10}} in the current coupling

$\{1, 10\} \to$

	11	12	13	14	15	16	17	18
2	$\frac{2}{9}$	0	0	0	0	**0**	0	0
3	0	$\frac{1}{9}$	0	0	0	**0**	0	0
4	0	0	$\frac{1}{12}$	0	0	**0**	0	0
5	0	0	0	$\frac{1}{9}$	0	**0**	0	0
6	0	0	0	0	$\frac{5}{36}$	**0**	0	0
7	0	0	0	0	0	$\frac{5}{36}$	**0**	0
8	0	0	0	0	0	0	$\frac{1}{9}$	**0**
9	0	0	0	0	0	0	0	$\frac{1}{12}$

$\{6, 15\} \to$

	11	12	15
2	$\frac{1}{9}$	0	0
3	$\frac{1}{36}$	$\frac{5}{36}$	0
6	0	$\frac{1}{36}$	$\frac{25}{36}$

$\{7, 16\} \to$

	11	12	16
7	$\frac{5}{36}$	$\frac{1}{6}$	$\frac{25}{36}$

$\{8, 17\} \to$

	11	12	17
8	$\frac{1}{9}$	$\frac{1}{6}$	$\frac{13}{18}$

$\{9, 18\} \to$

	11	12	18
9	$\frac{1}{12}$	$\frac{1}{6}$	$\frac{3}{4}$

Variables detected to be zero: $\{\}$

Reduced Linear System:
$$\begin{pmatrix} 1 & -\frac{5}{36} & -\frac{5}{36} & -\frac{1}{9} & -\frac{1}{12} \\ 0 & \frac{11}{36} & 0 & 0 & 0 \\ 0 & 0 & \frac{11}{36} & 0 & 0 \\ 0 & 0 & 0 & \frac{5}{18} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} \end{pmatrix}$$

$$\begin{pmatrix} d(1,10) \\ d(6,15) \\ d(7,16) \\ d(8,17) \\ d(9,18) \end{pmatrix} == \begin{pmatrix} 0 \\ \frac{1}{18} \\ \frac{11}{36} \\ \frac{5}{18} \\ \frac{1}{4} \end{pmatrix}; \quad \text{Solution:} \begin{pmatrix} d(1,10) \\ d(6,15) \\ d(7,16) \\ d(8,17) \\ d(9,18) \end{pmatrix} = \begin{pmatrix} \frac{71}{198} \\ \frac{2}{11} \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Non optimal schedule for: $\{6, 15\}$ reduced cost: -1 for introducing $\{3, 1\}$ as basic

Reachable states from $\{\{1, 10\}\}$ in the current coupling

$\{1, 10\} \to$

	11	12	13	14	15	16	17	18
2	$\frac{2}{9}$	0	0	0	0	**0**	0	0
3	0	$\frac{1}{9}$	0	0	0	**0**	0	0
4	0	0	$\frac{1}{12}$	0	0	**0**	0	0
5	0	0	0	$\frac{1}{9}$	0	**0**	0	0
6	0	0	0	0	$\frac{5}{36}$	**0**	0	0
7	0	0	0	0	0	$\frac{5}{36}$	**0**	0
8	0	0	0	0	0	0	$\frac{1}{9}$	**0**
9	0	0	0	0	0	0	0	$\frac{1}{12}$

$\{6, 15\} \to$

	11	12	15
2	$\frac{1}{9}$	0	0
3	**0**	$\frac{1}{6}$	0
6	$\frac{1}{36}$	0	$\frac{25}{36}$

$\{7, 16\} \to$

	11	12	16
7	$\frac{5}{36}$	$\frac{1}{6}$	$\frac{25}{36}$

$\{8, 17\} \to$

	11	12	17
8	$\frac{1}{9}$	$\frac{1}{6}$	$\frac{13}{18}$

$\{9, 18\} \to$

	11	12	18
9	$\frac{1}{12}$	$\frac{1}{6}$	$\frac{3}{4}$

Variables detected to be zero: $\{\}$

Reduced Linear System:
$$\begin{pmatrix} 1 & -\frac{5}{36} & -\frac{5}{36} & -\frac{1}{9} & -\frac{1}{12} \\ 0 & \frac{11}{36} & 0 & 0 & 0 \\ 0 & 0 & \frac{11}{36} & 0 & 0 \\ 0 & 0 & 0 & \frac{5}{18} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} \end{pmatrix}$$

$$\begin{pmatrix} d(1,10) \\ d(6,15) \\ d(7,16) \\ d(8,17) \\ d(9,18) \end{pmatrix} == \begin{pmatrix} 0 \\ \frac{1}{36} \\ \frac{11}{36} \\ \frac{5}{18} \\ \frac{1}{4} \end{pmatrix}; \quad \text{Solution:} \begin{pmatrix} d(1,10) \\ d(6,15) \\ d(7,16) \\ d(8,17) \\ d(9,18) \end{pmatrix} = \begin{pmatrix} \frac{137}{396} \\ \frac{1}{11} \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Total solved TPs: 2

Execution Time: 0.127852 sec

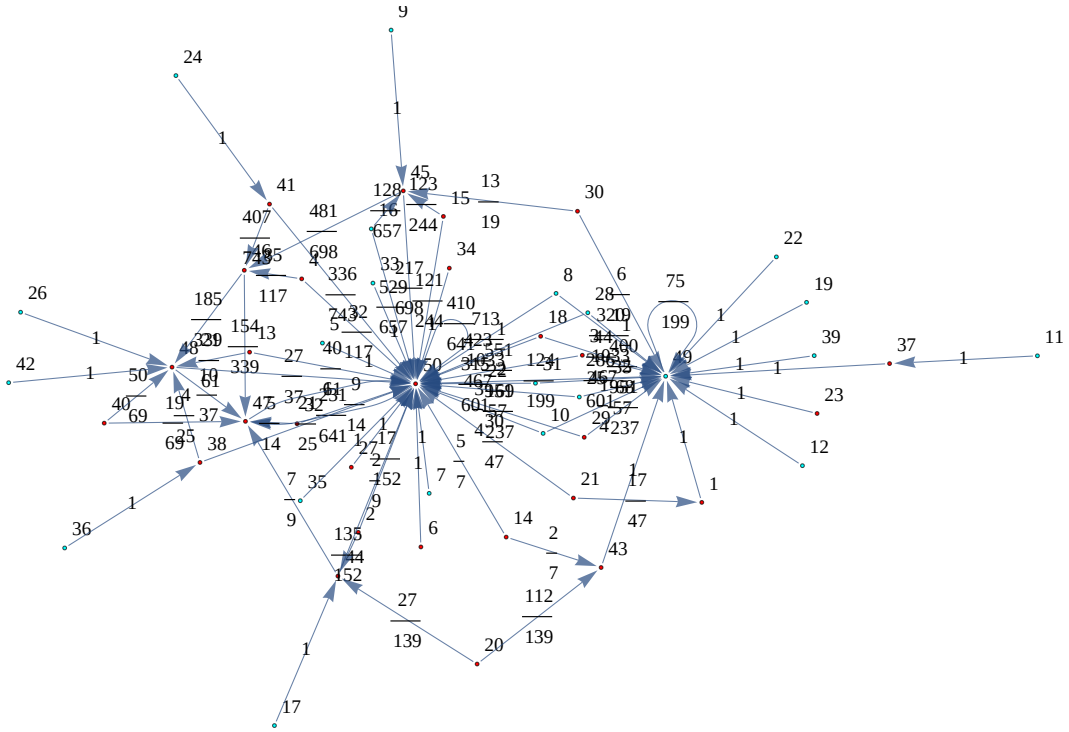
Out[284]= $\left\{\{1, 10\} \to \frac{137}{396}\right\}$

■ Bisimilarity Distance and Bisimilarity Quotient

Let $\mathcal{M}$ be a (randomly generated) Markov chain with 50 states and outer-degree 2

In[366]:=
```
numStates = 50; outerDegree = 2;
mc = RandomMarkovChain[numStates, outerDegree];
PlotMC@mc
```

Out[368]=



the bisimilarity pseudometric can also be used in order to detect all the bisimilarity classes, i.e. the equivalence classes of the set of state w.r.t probabilistic bisimilarity

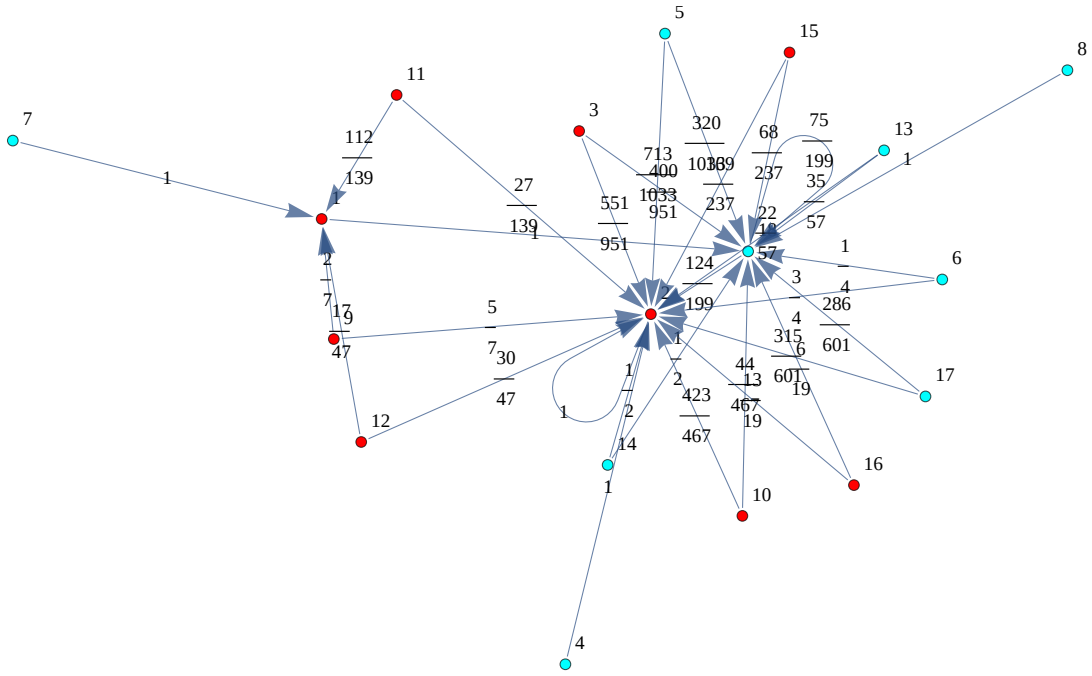
In[369]:=
```
BisimClasses[mc]
```

Out[369]=
```
{{1, 23, 37, 43}, {2, 4, 6, 13, 15, 27, 32, 34, 38, 40, 41, 44, 45, 46, 47, 48, 50},
 {3}, {5, 7, 9, 16, 17, 24, 26, 33, 35, 36, 42}, {8}, {10}, {11},
 {12, 19, 22, 39}, {14}, {18}, {20}, {21}, {25}, {28}, {29}, {30}, {31}, {49}}
```

so that this can be used to construct an MC $\mathcal{M}$' bisimilar to $\mathcal{M}$ with all the states that are bisimilar in $\mathcal{M}$ lumped together. $\mathcal{M}$' is also known as the bisimilar quotient of $\mathcal{M}$.

In[370]:= **BisimQuotient[mc] // PlotMC**

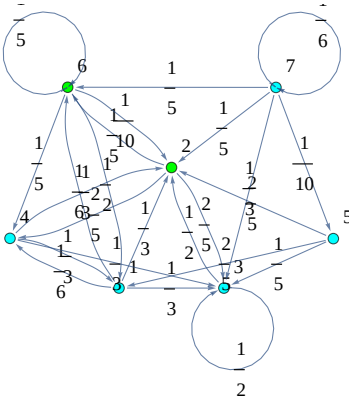
Out[370]=

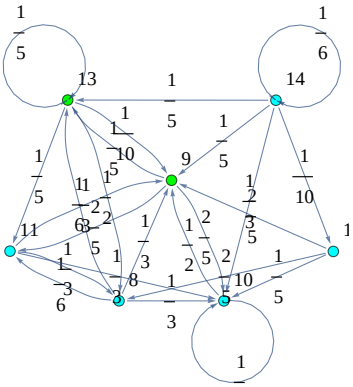


Obvousy, for any MC $\mathcal{M}$, the bisimilarity quotient of two copies of $\mathcal{M}$ joined together correspons to $\mathcal{M}$ itselfs

In[371]:= **JoinMC[mcPaper, mcPaper] // PlotMC**

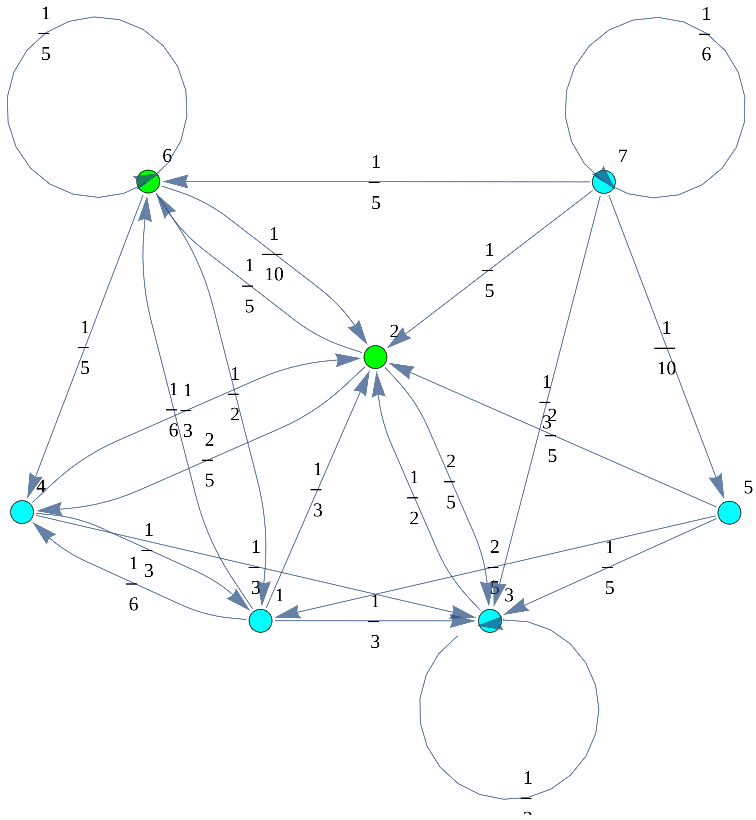
Out[371]=





In[372]:=
```
BisimQuotient[JoinMC[mcPaper, mcPaper]] // PlotMC
```

Out[372]=



Implementation

■ Utilities

In[292]:=
```
DiffDist[d1_, d2_] := Max[Flatten[Abs[d1 - d2]]];
```

In[293]:=
```
ZeroMatrix[n_] := SparseArray[{}, {n, n}];
```

In[294]:=
```
TestAll[list_, test_] := And @@ (test /@ list);
```

In[295]:=
```
DebugPrint[s___] := Print[s] /; OptionValue[ComputeDistances, Verbose];
```

In[296]:=
```
PrintActiveProblems[dist_, problems_, queryLst_] :=
  Module[{ActivePrint, active = ReachableProblems[dist, problems, queryLst]},
    ActivePrint[{u_, v_} → actv_] :=
     Print[{u, v} → PrPrint[problems⟦u, v⟧]] /; actv;
    Print["Reachable states from ", queryLst, " in the current coupling"];
    Scan[ActivePrint, ArrayRules[active]]
    (*DebugPrint[MatrixForm@MapIndexed[ActivePrint,problems,{2}]]*)
   ] /; OptionValue[ComputeDistances, Verbose];
```

In[297]:=
```
PrintProblems[problems_] :=
  DebugPrint[MatrixForm@Map[PrPrint, problems, {2}]];
```

■ Labelled Markov Chains (LMCs)

In[298]:=
```
AllPairs[MC[_, labels_]] :=
  Module[{size = Length@labels}, Join @@ (Table[{i, j}, {i, size}, {j, size}])];
```

In[299]:=
```
SameLabelsPairs[mc : MC[_, labels_]] :=
  Select[AllPairs@mc, Equal @@ (labels[[#]]) &];
```

In[300]:=
```
MarkovChainQ[MC[TM_, labels_]] :=
  Module[{size, a, b},
    size = Length@labels; {a, b} = Dimensions[TM];
    (* check if the sizes are consistent,
    and if TM is a probability distribution matrix *)
    size == a ⋀ a == b ⋀ TestAll[(Plus @@ # &) /@ TM, # == 1 &]
   ] /; MatchQ[labels, {__Integer}] ⋀ (* labels is a list of integers *)
    MatrixQ[TM, MatchQ[#, _Rational] ⋁ IntegerQ[#] &]
    (* TM is a matrix of rationals *);
MarkovChainQ[_] := False;
```

Given a list of edges of the form `Edge[s,p,t]` indicating that there exists a transition from the state `s` to `t` with probability p,
`TransitionMatrix[|S|,edgeList]` returns the transition matrix π induced by them.

In[302]:=
```
TransitionMatrix[numStates_, edgeList_] :=
  Module[{f},
   f[Edge[x_, p_, y_]] := {x, y} → p;
   SparseArray[f /@ edgeList, {numStates, numStates}]
  ];
```

`JoinMC` returns a graphical representation of a given Markov chain

In[303]:=
```
JoinMC[seq__MC] := Fold[JoinMCAux, First@{seq}, Rest@{seq}];
JoinMCAux[MC[tm1_, l1_], MC[tm2_, l2_]] :=
  Module[{n = Length@l1, m = Length@l2},
   MC[ArrayFlatten[
     {{tm1, SparseArray[{}, {n, m}]}, {SparseArray[{}, {m, n}], tm2}}],
    Join[l1, l2]]
  ];
```

`PlotMC` returns a graphical representation of a given Markov chain

In[305]:=
```
PlotMC[edgeList : {___Edge}] :=
  Module[{f},
   f[Edge[x_, p_, y_]] := Labeled[x → y, p];
   Graph[f /@ edgeList, VertexLabels → "Name", VertexSize → 0.1]
  ];
PlotMC[MC[tm_, labels_]] :=
  Module[{f, g, edgelist},
   f[Edge[x_, p_, y_]] := Labeled[x → y, p];
   g[lbl_, {vertex_}] := vertex → Hue[1 / (lbl + 1)];
   edgelist = Edge[#[[1, 1]], #[[2]], #[[1, 2]]] & /@ Most@ArrayRules[tm];
   Graph[f /@ edgelist, VertexLabels → "Name",
    VertexSize → 0.1, VertexStyle → MapIndexed[g, labels]]
  ];
```

■ Random generation of LMCs

```
In[307]:= RandomRational[rmax_, n_] :=
           (RandomInteger[{1, #}] / #) & /@ RandomInteger[{1, rmax}, n]
```

RandomProbDistr gives a pseudorandom list of rationals $r_1, \ldots, r_n$ such that $\sum_{i=1}^{n} r_i = 1$

```
In[308]:= RandomProbDistr[rmax_, n_] :=
           Module[{rnd = RandomRational[rmax, n], total},
            total = Plus @@ rnd;
            (# / total) & /@ rnd
           ]

In[309]:= RandomTransitionMatrix[numStates_, maxDegree_] :=
            Module[{MakeRow},
              MakeRow[degree_] := Shuffle@
                Join[Table[0, {numStates - degree}], RandomProbDistr[numStates, degree]];
              SparseArray@Table[MakeRow[RandomInteger[{1, maxDegree}]], {numStates}]
             ] /; numStates ≥ maxDegree;
          RandomTransitionMatrix[numStates_, maxDegree_] :=
            (Message[RandomTransitionMatrix::nnarg, numStates, maxDegree]; Abort[]);

In[311]:= RandomTransitionMatrix::nnarg =
            "The argument `1` is not greater than or equal to `2`.";

In[312]:= RandomMarkovChain[numStates_, maxDegree_] :=
            MC[RandomTransitionMatrix[numStates, maxDegree],
             RandomInteger[{0, 1}, numStates]];

In[313]:= Shuffle[{}] := {};
          Shuffle[lst_] :=
            Module[{left, right, pick = RandomInteger[{1, Length@lst}]},
             left = Take[lst, pick]; right = Drop[lst, pick];
             Join[{lst[[pick]]}, Shuffle[Most@left], Shuffle[right]]
            ];
```

■ Transportation Problem

Given two ϵ-numbers x and y, EpsilonLeq[x,y] determines if $x \le y$ provided that $\epsilon \to +0$.

```
In[315]:= EpsilonLeq[x_, y_] :=
            If[EpsilonRemoval[x - y] == 0, (* check if x-y depends only on ϵ*)
             EpsilonFix[x - y, 1] ≤ 0, (* set ϵ > 0 and check again *)
             EpsilonRemoval[x - y] ≤ 0 (* set e[]= 0 and check again *)
            ];

In[316]:= EpsilonMin[x_, y_] := If[EpsilonLeq[x, y], x, y];

In[317]:= EpsilonPerturbation[πs_, πt_] :=
            Module[{AddEpsilon},
             AddEpsilon[k_][{a_, id_}] := {a + k e[], id};
             {AddEpsilon[1] /@ πs, Append[Most[πt], AddEpsilon[Length@πs][Last@πt]]}
            ];

In[318]:= EpsilonFix[x_, ϵvalue_] := x /. e[] → ϵvalue;

In[319]:= EpsilonRemoval[x_] := EpsilonFix[x, 0];

In[320]:= indexed[lst_] := Thread[{lst, Table[i, {i, 1, Length@lst}]}];
```

In[321]:=
```
GuessVertexAux[_][πs_, πt_] := {} /; πs == {} || πt == {};
GuessVertexAux[strategy_][πs_, πt_] :=
  Module[{G},
   G[{i_, j_}] :=
    Module[{rec},
     rec = If[EpsilonLeq[πs[[i, 1]], πt[[j, 1]]], (*πs[[i,1]] ≤ πt[[j,1]]*)
       GuessVertexAux[strategy][
        Drop[πs, {i}], ReplacePart[πt, {j, 1} → πt[[j, 1]] - πs[[i, 1]]]],
       GuessVertexAux[strategy][ReplacePart[πs, {i, 1} → πs[[i, 1]] - πt[[j, 1]],
        Drop[πt, {j}]]];
     Prepend[rec, {πs[[i, 2]], πt[[j, 2]]} → EpsilonMin[πs[[i, 1]], πt[[j, 1]]]]
    ];
   G[strategy[πs, πt]]
  ];
```

In[323]:=
```
NorthWestStrategy[_, _] := {1, 1};
```

In[324]:=
```
ToMatrix[πs_, πt_, vertex_] :=
  Module[{mapS, revmapS, mapT, revmapT, toMap, mapIndex, mappedvertex},
   toMap[indexlst_] :=
    Module[{index = Table[i, {i, 1, Length@indexlst}] },
     {Thread[Rule[index, indexlst]], Thread[Rule[indexlst, index]]}];
   mapIndex[{s_, t_} → val_] := {s /. revmapS, t /. revmapT} → val;

   {{mapS, revmapS}, {mapT, revmapT}} = toMap[#[[All, 2]] & /@ {πs, πt};
   mappedvertex = mapIndex /@ vertex;
   IndexedVertex[mapS, mapT, mappedvertex[[All, 1]],
    SparseArray[mappedvertex, Length /@ {πs, πt}]]
  ];
```

In[325]:=
```
GuessVertex[πs_, πt_] :=
  Module[{ϵπs, ϵπt},
   {ϵπs, ϵπt} = EpsilonPerturbation[πs, πt];
   ToMatrix[ϵπs, ϵπt, GuessVertexAux[NorthWestStrategy][ϵπs, ϵπt]]
  ];
```

`AddtoBase[d, vertex] == {reducedcost, non-basic cell}`
returns the non-basic cell which has the minimal reduced cost, if the vertex has no non-basic cells, then it returns $\{\infty, \{0,0\}\}$.

In[326]:=
```
AddtoBase[dist_, IndexedVertex[mapS_, mapT_, basicCells_, V_]] :=
 Module[{sizeS = Length@mapS, sizeT = Length@mapT,
   t, getCost, ReducedCost, uv, makeRow, base},

  makeRow[{1, j_}] := {SparseArray[{{1, sizeS + j} → 1}, {1, sizeS + sizeT}]};
  makeRow[{i_, j_}] :=
   {SparseArray[{{1, i} → 1, {1, sizeS + j} → 1}, {1, sizeS + sizeT}]};

  getCost[{i_, j_}] := getDistValue[dist, {i /. mapS, j /. mapT}];

  ReducedCost[_, {i_, j_}] :=
   Module[{reducedcost = getCost[{i, j}] - uv⟦i⟧ - uv⟦sizeS + j⟧},
    If[base⟦1⟧ > reducedcost,
     base = {reducedcost, {i, j}}
     (* update the current least reduced cost *)
    ]; reducedcost];

  (* u-v modifiers (u1,...,usizeS,v1,...,vsizeT) *)
  uv = LinearSolve[
    Normal@ArrayFlatten[makeRow /@ basicCells], getCost /@ basicCells];
  (*DebugPrint["u-v Problem:",MatrixForm@ArrayFlatten[makeRow/@basicCells],
    MatrixForm@uv,MatrixForm[getCost/@basicCells]];*)
  base = {∞, {0, 0}}; (* init cell *)
  (* scan the cells computing their reduced cost *)
  MapIndexed[ReducedCost, V, {2}];
  base (* return the cell with the least reduced cost *)
 ];
```

`SteppingStone[non-basic cell, vertex]` returns the vertex obtained by inserting the given non-basic cell into the base

In[327]:=
```
SteppingStone[{s_, t_}, vertex : IndexedVertex[mapS_, mapT_, basicCells_, V_]] :=
 Module[{M, d, oldBasicCell, newBasicCells},
  {oldBasicCell, M} = SteppingStonePath[{s, t}, vertex];
  (*DebugPrint[MatrixForm@M];*)
  (*DebugPrint["old basic cell: ",
    oldBasicCell, ", new basic cell: ",{s,t}];*)
  d = Extract[V, oldBasicCell];
  (*DebugPrint["ammount moved: ",d];*)
  newBasicCells = basicCells /. oldBasicCell → {s, t};
  (*DebugPrint["Old basic cells:",
    basicCells,", new basic cells",newBasicCells];*)
  IndexedVertex[mapS, mapT, newBasicCells, SparseArray[V + d M]]
 ];
```

In[328]:=
```
SteppingStonePath[{s_, t_},
  vertex : IndexedVertex[mapS_, mapT_, basicCells_, V_]] :=
 Module[{neighbors, visit, parent, cell, path, getMatrix, visited, f},

  visited[_] := False;
  neighbors[{u_, True}] := {#[[2]], False} & /@
    (Select[basicCells, #[[1]] == u ⋀ Not[visited[{#[[2]], False}]] &]);
  neighbors[{v_, False}] := {#[[1]], True} & /@
    (Select[basicCells, #[[2]] == v ⋀ Not[visited[{#[[1]], True}]] &]);

  f[{{a_, True}, {b_, False}}] := {a, b};
  f[{{a_, False}, {b_, True}}] := {b, a};

  visit[node_] :=
   (visited[node] = True;
    Scan[(parent[#] = node; visit@#) &, neighbors[node]]
   );

  getMatrix[{}, sign_] := {};
  getMatrix[lst_, sign_] :=
   (If[sign < 0 ⋀ EpsilonLeq[Extract[V, First@lst] , Extract[V, cell]],
     cell = First@lst]; (* update cell *)
    Append[getMatrix[Rest@lst, - sign], First@lst → sign]);
  (* compute arrayrules *)

  (*DebugPrint[PrPrint@vertex];*)

  (* search for an alternating path from s to t *)
  parent[{s, True}] = {}; (* {s,True} is the root of the tree *)
  visit[{s, True}]; (* compute the spanning tree rooted in {s,True} *)

  (* get the alternating path from {t,False} to {s,True} *)
  path = Append[f /@ Partition[
      (Most@NestWhileList[parent, {t, False}, # != {} &]), 2, 1], {s, t}];
  cell = First@path; path = getMatrix[path, -1];

  {cell, SparseArray[path, Dimensions@V]}
 ];
```

In[329]:=
```
BestVertex[dist_, {redCost_, nbCell_}, vertex_] :=
 Module[{currVertex = vertex, currRedCost = redCost, currNbCell = nbCell},
  While[currRedCost < 0,
   currVertex = SteppingStone[currNbCell, currVertex];
   (* move to a better vertex *)
   (*DebugPrint[PrPrint@currVertex];*)
   (* update the current least reduced cost *)
   {currRedCost, currNbCell} = AddtoBase[dist, currVertex];
  ];
  currVertex
 ];
```

```
In[330]:= PrPrint[IndexedVertex[mapS_, mapT_, basicCells_, V_]] :=
            Module[{basic},
             basic[value_, index_] :=
              Style[EpsilonRemoval@value, Red, Bold] /; MemberQ[basicCells, index];
             basic[value_, _] := value;
             TableForm[MapIndexed[basic, V, {2}],
              TableHeadings → {ToString[#[[2]]] & /@ mapS, ToString[#[[2]]] & /@ mapT}]
            ];
          PrPrint[x_] := x;
```

■ Iterative Algorithm

Given the numer of states `|S|`, `BottomDist[|S|]` returns the bottom distance

```
In[332]:= BottomDist[size_] := Table[0, {size}, {size}];
```

```
In[333]:= TopDist[size_] := Table[1, {size}, {size}];
```

The funtion Δ implements the distance transformer

```
In[334]:= Δ[MC[TM_, labels_], λ_][dist_] :=
            Module[{size = Length@labels, Marginal, TP, NewValue},
             Marginal[state_] := Select[indexed[TM[[state, All]]], #[[1]] > 0 &];
             TP[s_, t_] := Module[{vertex = GuessVertex[Marginal[s], Marginal[t]]},
               BestVertex[dist, AddtoBase[dist, vertex], vertex]];

             NewValue[s_, t_] := 1 /; labels[[s]] ≠ labels[[t]];
             NewValue[s_, t_] :=
              Module[{cellCost, optSchedule = TP[s, t]},
               cellCost[IndexedVertex[mapS_, mapT_, _, V_]][{i_, j_}] :=
                getDistValue[dist, {i /. mapS, j /. mapT}] * EpsilonRemoval[V[[i, j]]];
               λ * Plus @@ (cellCost[optSchedule] /@ optSchedule[[3]])
              ];
             (* compute the new values for each pair *)
             Table[NewValue[s, t], {s, size}, {t, size}]
            ];
```

```
In[335]:= IterativeAlg[mc : MC[_, labels_], λ_,
             OptionsPattern[{StoppingTime -> 4, Verbose → False}]] :=
            Module[{t = 0, iterCount = 0, size = Length[labels], dt, old, curr},
             SetOptions[ComputeDistances, Verbose → OptionValue[Verbose]];
             (* initialization *)
             old = TopDist[Length[labels]];
             curr = BottomDist[Length[labels]];
             (* iteratively apply Δ, until the extact result is reached or
              the StoppingTime has been exceeded  *)
             While[t ≤ OptionValue[StoppingTime] ⋀ curr ≠ old, (*  *)
              old = curr;
              {dt, curr} = Timing[Δ[mc, λ][old]];
              t += dt; iterCount++;
             ];
             {t, R[iterCount, Length[SameLabelsPairs[mc]], curr]}
            ];
```

■ On-the-fly Algorithm for computing Bisimilarity Pseudometric

Given as input an indexed vertex v, SubProblems[v] returns the list of active subproblems. This is simply done by selecting the (non-degenerate) basic cells of ω.

In[336]:=
```
SubProblems[IndexedVertex[mapS_, mapT_, basicCells_, V_]] :=
  DeleteDuplicates[(Sort[{#[[1]] /. mapS, #[[2]] /. mapT}] &) /@
    Select[basicCells, EpsilonRemoval[Extract[V, #]] > 0 &]];
```

In[337]:=
```
FixedDistance[dist_, {i_, j_}] := MatchQ[dist[[i, j]], _Fixed] ;
(* says if the distance is fixed or not *)
```

In[338]:=
```
KnownProblem[dist_, problems_, {i_, j_}] :=
  (* decide whether the problem (i,j) has already been considered *)
  FixedDistance[dist, {i, j}] || MatchQ[problems[[i, j]], _IndexedVertex];
```

In[339]:=
```
AddProblem[mc : MC[_, _], dist_][problems_, {i_, j_}] :=
  problems /; KnownProblem[dist, problems, {i, j}];
AddProblem[mc : MC[TM_, labels_], dist_][problems_, {i_, j_}] :=
  Module[{Marginal, newvertex},
   Marginal[state_] := Select[indexed[TM[[state, All]]], #[[1]] > 0 &];
   newvertex = GuessVertex[Marginal[i], Marginal[j]];
   (* guess a fesible transposrtation schedule *)
   UpdateProblems[mc, dist][problems, {i, j}, newvertex]
   (* recursively update the problems matrix *)
  ];
```

In[341]:=
```
AddProblems[mc : MC[_, _], dist_][problems_, queryLst_] :=
  Fold[AddProblem[mc, dist], problems, queryLst];
```

In[342]:=
```
UpdateProblems[mc : MC[_, _], dist_][problems_, {i_, j_}, newvertex_] :=
  AddProblems[mc, dist][
   ReplacePart[problems, {i, j} → newvertex], SubProblems[newvertex]];
```

Compute the coefficients $\text{VertexCoeffs}\left[\left(\omega\left((s, t), (u, v)\right)\right)_{(u,v)\in S?}, \sum_{l(u)\neq l(v)} \omega\left((s, t), (u, v)\right)\right]$ for the given (vertex) schedule

In[343]:=
```
VertexCoeffs[MC[_, labels_], dist_,
  IndexedVertex[mapS_, mapT_, basicCells_, V_]] :=
 Module[{coeffValue, gatherFun, aggr},
  coeffValue[{i_, j_}] :=
   getDistValue[dist, {i /. mapS, j /. mapT}] * EpsilonRemoval[V[[i, j]]];

  gatherFun[{i_, j_}] := 0 /; FixedDistance[dist, Sort[{i /. mapS, j /. mapT}]];
  (* constant *)
  gatherFun[{i_, j_}] := Sort[{i /. mapS, j /. mapT}]; (* coefficient *)

  aggr[VCoeffs[unknown_, const_], lst : {x_, ___}] :=
   VCoeffs[unknown, Fold[#1 + coeffValue[#2] &, const, lst]] /;
    gatherFun[x] == 0;
  aggr[VCoeffs[unknown_, const_], lst : {x_, ___}] :=
   VCoeffs[Append[unknown,
     gatherFun[x] → Fold[#1 + EpsilonRemoval[Extract[V, #2]] &, 0, lst]], const];

  Fold[aggr, VCoeffs[{}, 0],
   GatherBy[Select[basicCells, coeffValue[#] > 0 &], gatherFun]]
 ]
```

ActiveProblems maps each active problem to its coefficients
$(s, t) \mapsto \text{VertexCoeffs}\left[\left(\omega \left((s, t), (u, v) \right) \right)_{(u,v) \in S?}, \sum_{l(u) \neq l(v)} \omega \left((s, t), (u, v) \right) \right]$

```
In[344]:= ActiveProblems[mc : MC[_, labels_], dist_, problems_, queryLst_] :=
            Most@ArrayRules@Fold[ActiveProblemsAux[mc, dist, problems],
               ZeroMatrix[Length@labels], queryLst];
          ActiveProblemsAux[MC[_, labels_], dist_, _][active_, {s_, t_}] :=
            active /; FixedDistance[dist, {s, t}] || MatchQ[active[[s, t]], _VCoeffs];
          ActiveProblemsAux[mc : MC[_, _], dist_, problems_][active_, {s_, t_}] :=
            Fold[ActiveProblemsAux[mc, dist, problems],
             ReplacePart[active, {s, t} → VertexCoeffs[mc, dist, problems[[s, t]]]],
             SubProblems[problems[[s, t]]]];

In[347]:= ReachableProblems[dist_, problems_, queryLst_] :=
            Fold[DFS[dist, problems],
             SparseArray[{}, Dimensions@dist, False], queryLst];
          DFS[dist_, problems_][visited_, {s_, t_}] :=
            visited /; visited[[s, t]] ⋁ FixedDistance[dist, {s, t}];
          DFS[dist_, problems_][visited_, {s_, t_}] :=
            Fold[DFS[dist, problems],
             ReplacePart[visited, {s, t} → True], SubProblems[problems[[s, t]]]];

In[350]:= UpdateWithDiscrepancy[λ_, mc : MC[_, labels_], dist_, problems_, queryLst_] :=
            Module[
             {aggr, V0 = {}, V = {}, AdjList = Table[{}, {Length@labels}, {Length@labels}],
              eqrules, DFS, notFree, zeroVars, freeVariables, RemoveFree, PrintD, d,
              toBind, makeRow, A, b, sol, eqNumber, varMap, invVarMap, constMap},
             PrintD[{x_, y_}] := "d(" <> ToString[x] <> "," <> ToString[y] <> ")";

             aggr[id_ -> VCoeffs[lst_, n_]] :=
              (If[n == 0, V0 = Append[V0, id], V = Append[V, id]];
               AdjList = Fold[ReplacePart[#1, #2 → Append[Extract[#1, #2], id]] &,
                 AdjList, lst[[All, 1]]]);
             DFS[visited_, id_] := visited /; Extract[visited, id];
             DFS[visited_, id_] :=
              Fold[DFS, ReplacePart[visited, id → True], Extract[AdjList, id]];
             RemoveFree[id_ -> VCoeffs[lst_, n_]] :=
              id -> VCoeffs[Select[lst, Extract[notFree, First@#] &], n];

             toBind[index_ → VCoeffs[_, const_], {n_}] := {index → n, n → index, n → const};
             makeRow[index_ → VCoeffs[rules_, _]] :=
              {SparseArray[{1, index /. varMap} → 1, {1, eqNumber}] -
                λ SparseArray[({1, #[[1]] /. varMap} → #[[2]] &) /@ rules, {1, eqNumber}]};

             eqrules = ActiveProblems[mc, dist, problems, queryLst];
             (* compute coefficients of active problems *)
             (* LOOK FOR POSSIBLE FREE-VARIABLES *)
             Scan[aggr, eqrules]; (* construct the inverse graph *)

             notFree = Fold[DFS, SparseArray[{}, Dimensions@dist, False], V];
             (* detect non-free variables *)
             zeroVars = Select[V0, Not@Extract[notFree, #] &];
             (* detect zero-variables *)
             DebugPrint["Variables detected to be zero:", PrintD /@ zeroVars];
             (* "GAUSSIAN ELIMINATION" *)
```

```
    eqrules = RemoveFree /@ Select[eqrules, Extract[notFree, First@#] &];
    (* remove free variables from eqrules *)
    sol =
     If[(eqNumber = Length@eqrules) > 0,
      (* CONSTRUCT THE SINGULAR EQUATION SYSTEM *)
      {varMap, invVarMap, constMap} = Transpose[MapIndexed[toBind, eqrules]];
      A = ArrayFlatten[makeRow /@ eqrules];
      b = λ SparseArray[constMap, {eqNumber}];
      sol = LinearSolve[Normal@A, Normal@b];

      d = MatrixForm[(PrintD[# /. invVarMap]) & /@ Table[i, {i, Length@sol}]];
      DebugPrint["Reduced Linear System: ",
       MatrixForm@A, d, "==", MatrixForm@b,
       ";  Solution:", d, "=", sol // MatrixForm];

      sol,
      DebugPrint["Reduced Linear System: it is empty"];
      {}
     ];

    ReplacePart[dist,
     Join[# → Fixed[0] & /@ zeroVars, (* fix free variables to zero *)
      MapIndexed[(#2〚1〛 /. invVarMap) → #1 &, sol]]
      (* new values for the non-free variables *)
    ]
   ];
```

In[351]:=
```
getDistValue[dist_, index : {_, _}] :=
  Module[{getValue},
   getValue[Fixed[v_]] := v;
   getValue[v_] := v;

   getValue[Extract[dist, Sort@index]]
  ];
```

In[352]:=
```
InitDistances[MC[_, labels_], estimates_] :=
  Module[{numStates = Length@labels, setDist},
   setDist[s_, t_] := Fixed[0] /; s == t;
   setDist[s_, t_] := Fixed[1] /; labels〚s〛 ≠ labels〚t〛;
   setDist[s_, t_] :=
    Fixed[{s, t} /. estimates] /; MemberQ[estimates, {s, t} → _];
   setDist[s_, t_] := 1;

   Table[setDist[s, t], {s, numStates}, {t, numStates}]
  ];
```

`VertexToChange[dist, {s,t}] == {{i,j}, {redCost, nbcell}}` if the problem `(i,j)` can be further minimized, otherwise it returns `{}`

In[353]:=
```
VertexToChange[problems_, dist_, queryLst_] :=
  Module[{Q = queryLst, i, j, reducedcost, cell, neighbours, toVisit,
    visited = SparseArray[{}, Dimensions@problems, False]},

   toVisit[{u_, v_}] := Not[visited[[u, v]] ⋁ FixedDistance[dist, {u, v}]];
   (* a new problem should not be visited if it has been visited already or
    its corresponding value is fixed *)
   While[Length@Q > 0,
    (* take a problem from the queue and set it as already visited *)
    {i, j} = First@Q; visited = ReplacePart[visited, {i, j} → True];
    (*DebugPrint["Visited problem:",{i,j}];*)
    (* compute the least reduced cost for the current problem *)
    {reducedcost, cell} = AddtoBase[dist, problems[[i, j]]];
    If[reducedcost < 0,
     (* the first encoutered vertex to be changed *)
     DebugPrint["Non optimal schedule for: ", {i, j}, " reduced cost: ",
      reducedcost, " for introducing ", cell, " as basic"];
     Return[{{i, j}, {reducedcost, cell}}],
     (* otherwise look forward *)
     neighbours = Select[SubProblems[problems[[i, j]]], toVisit];
     Q = DeleteDuplicates@Join[Rest@Q, neighbours]
    ]];
   (* none of the reached problems can be further decreased *)
   {}
  ];
```

In[354]:=
```
Options[ComputeDistances] =
  {Verbose → False, ConsistencyCheck → True, Estimates → {}};
ComputeDistances::query = "The following queries are ill-formed: `1`.";
ComputeDistances::MarkovChain =
  "The 1-st argument is not a well-formed rational-valued
    probabilistic Markov Chain.";
ComputeDistances::DiscountFactor = "The second argument, namely
    `1`, must be a rational in the interval (0,1].";
ComputeDistances::MultipleEstimates =
  "There must be at most one estimate for each pair";
ComputeDistances::WrongEstimates =
  "Estimates must be a rationals in the interval (0,1]";

ComputeDistances[mc : MC[_, _], λ_, All, options : OptionsPattern[
     {Verbose -> False, ConsistencyCheck → True, Estimates → {}}]] :=
  ComputeDistances[mc, λ, AllPairs@mc, options];
ComputeDistances[mc : MC[_, _], λ_, queryLst_, OptionsPattern[
    {Verbose -> False, ConsistencyCheck → True, Estimates → {}}]] :=
  (If[OptionValue[ConsistencyCheck], CheckInput[mc, λ, queryLst]];
   (* Set Options for ComputeDistance *)
   SetOptions[ComputeDistances, Verbose → OptionValue[Verbose]];
   SetOptions[ComputeDistances,
    ConsistencyCheck → OptionValue[ConsistencyCheck]];
   SetOptions[ComputeDistances, Estimates → OptionValue[Estimates]];
   (* Start computing the dλ(s,t) *)
   ComputeDistancesAux[mc, λ, queryLst]
  );
```

In[362]:=
```
CheckInput[mc : MC[_, labels_], λ_, queryLst_] :=
  Module[{BadQuery, badqueries, estimates, BadEstimate},
   BadQuery[size_][pair : {_Integer, _Integer}] :=
    Not[And @@ (IntervalMemberQ[Interval[{1, size}], #] & /@ pair)];
   BadQuery[_][_] := True;
   BadEstimate[size_][pair_ → value_] := BadQuery[size][pair] ⋁
     value < 0 ⋁ value > 1 ⋁ Not@MatchQ[value, _Rational | 1];

   estimates = OptionValue[ComputeDistances, Estimates];
   Which[
    Not@MarkovChainQ[mc],
        Message[ComputeDistances::MarkovChain]; Abort[],
    λ ≤ 0 ⋁ λ > 1 ⋁ Not@MatchQ[λ, _Rational | 1],
         Message[ComputeDistances::DiscountFactor, λ]; Abort[],
    Length@estimates >
     Length@DeleteDuplicates[estimates, First@#1 == First@#2 &],
        Message[ComputeDistances::MultipleEstimates]; Abort[],
    Select[estimates, BadQuery[Length@labels]] > 0,
        Message[ComputeDistances::WrongEstimates]; Abort[],
    (badqueries = Select[queryLst, BadQuery[Length@labels]]) > 0 ,
        Message[ComputeDistances::pair, badqueries]; Abort[]
   ];
  ];
```

In[363]:=
```
ComputeDistancesAux[mc : MC[TM_, labels_], λ_, queryLst_] :=
  Module[{sortedQuery, query, FixDistance, currProblems, time,
    currDist = InitDistances[mc, OptionValue[ComputeDistances, Estimates]],
    vtc, prb, rcCell, newvertex, iterations = 0},

   FixDistance[dist_, {s_, t_}] := dist /; FixedDistance[dist, {s, t}];
   FixDistance[dist_, index : {s_, t_}] :=
    Fold[FixDistance,
     ReplacePart[dist, index → Fixed[getDistValue[dist, index]]],
     SubProblems[currProblems[[s, t]]]];

   (* Initialization *)
   time = First@Timing[
      sortedQuery = DeleteDuplicates[
        Sort /@ Select[queryLst, Not@FixedDistance[currDist, #] &]];
      While[sortedQuery ≠ {},
       DebugPrint["Pairs to compute: ", sortedQuery];
       (* pick at random a pair from the query *)
       query = sortedQuery[[RandomInteger[{1, Length@sortedQuery}, 1]]];
       DebugPrint["Picked pair: ", query];
       currProblems =
        AddProblems[mc, currDist][ZeroMatrix@Length@labels, query];
       PrintActiveProblems[currDist, currProblems, query];
       currDist =
        UpdateWithDiscrepancy[λ, mc, currDist, currProblems, query];
       (*DebugPrint[MatrixForm@currDist];*)
       (* look for a non-optimal transportation schedule *)
       vtc = VertexToChange[currProblems, currDist, query];
       While[vtc ≠ {},
        iterations++;
        {prb, rcCell} = vtc; (* consider the schedule to be optimized *)
```

```
        (* move to a better coupling *)
        newvertex =
         BestVertex[currDist, rcCell, Extract[currProblems, prb]];
        currProblems = UpdateProblems[mc, currDist][
          currProblems, prb, newvertex];
        PrintActiveProblems[currDist, currProblems, query];
        (* update the current
         distance with the (locally) computed discrepancy *)
        currDist = UpdateWithDiscrepancy[λ, mc, currDist,
          currProblems, query];
        (*DebugPrint[MatrixForm@currDist];*)
        (* look for the next non-optimal transportation schedule *)
        vtc = VertexToChange[currProblems, currDist, query];
       ];
       (* Set the current active problems as fixed *)
       currDist = FixDistance[currDist, First@query];
       (*DebugPrint[MatrixForm@currDist];*)
       (* update the query list *)
       sortedQuery = Select[sortedQuery, Not@FixedDistance[currDist, #] &];
      ]];
    DebugPrint["There are no more pairs to process."];
    DebugPrint["Total solved TPs: ", iterations];
    DebugPrint["Execution Time: ", time, " sec"];
    (# → getDistValue[currDist, #] &) /@ queryLst
   ];
```

■ Bisimilarity

BisimQuotient[mc] returns an MC with all bisimilar states lumped together

In[364]:=
```
BisimQuotient[mc : MC[tm_, labels_]] :=
  Module[{Rate, Quo},
   Quo = BisimClasses@mc;
   Rate[class1_, class2_] := Plus @@ (tm[[First@class1, #]] & /@ class2);
   MC[Table[Rate[i, j], {i, Quo}, {j, Quo}], labels[[First@# & /@ Quo]]
  ];
```

BisimClasses[mc] returns the quotient w.r.t. bisimilarity

In[365]:=
```
BisimClasses[mc : MC[_, _]] :=
  FixedPoint[(Union @@ # &) /@ Gather[#, Intersection[#1, #2] ≠ {} &] &,
   Select[ComputeDistances[mc, 1, All, Verbose → False], #[[2]] == 0 &][[All, 1]]];
```