State Space Reduction without Information Loss

Markus Mayerwieser

February 5, 2014

Bachelor Thesis
Advisor: Dipl. Ing. Bernhard Geiger

Contents

Abstract

Based on the theoretical works [Gei12] and [Gei13], a Matlab© implementation for an information preserving state space reduction of Markov chains was carried out. The reduction takes place by a combination of states to a cluster, the cluster combinations are represented by partitions. These partitions are generated and encoded as proposed in [Ich82]. The resulting Markov chain of 2nd order is proven to still have the same entropy rate as the original Markov chain of 1st order.

---

1 Introduction

Markov chains represent stochastic processes for many different approaches in engineering. Due to performance reasons it is relevant to keep the state space of Markov chains as small as possible. The whole information still should be provided by the Markov representation, therefore some kind of lossless compression of the states is needed.

If there are special structures in the Markov chain, we can combine states to clusters and still keep the whole model information, but the resulting Markov chain has its order increased from 1st to 2nd order [Gei13].

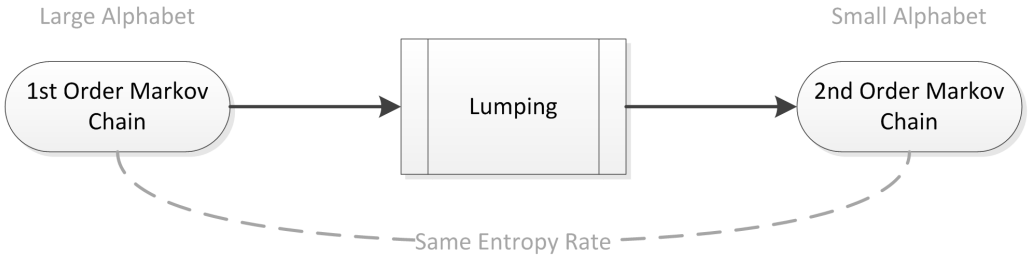

Figure 1: State Space Lumping

In this project, an algorithm was implemented that performs a combination of states in a given state space and therefore provides a clustering of it. The clustering is determined with a specific set of rules on the structure of the transition graph, namely a "Single Entry State" [1] structure. If the conditions for this structure are met, states can be combined without losing the information in this states. In other words, we can reduce the state space in an information preserving manner.

In this thesis, this "Single Entry State" structure and the related structural rules are explained. We will also have a look at the implementation of these rules and the generation of the possible partitions. To provide a prove of this concept, the calculation of the entropy rate of the partitions of the state space are explained and there will be a concrete example to demonstrate the functionality of this approach.

[1] In [Gei12] and [Gei13] the Single Entry State structure is called SFS(2). The Single Entry property of [Gei12] is a related, but different property

2 Entropy of Higher Order Markov Chains

The algorithm comes up with all possible "Single Entry State" clusterings as a result. These results have to be checked to prove the functionality of the clustering. The easiest way to do this is to check if the entropy rate of the model has changed due to the clustering. Therefore we need to have a look at the calculation of the entropy rate of Markov chains, and further on the entropy rate of higher order Markov chains.

First we give some definitions about the original Markov chain X. P is the transition matrix and μ the stationary probability vector:

$$P_{ij} = \mathbb{P}(X_1 = j | X_0 = i) \tag{1}$$
$$\mu_i = \mathbb{P}(X_0 = i) \tag{2}$$

The Markov property tells us: The conditional probability distribution of a future state is just depended on the present state and is independent from the state sequence before the present state. For a stationary 2nd order Markov chain the following holds:

$$\begin{aligned}
&\mathbb{P}(Y_n = y_n | Y_{n-1} = y_{n-1}, ..., Y_1 = y_1) \\
&= \mathbb{P}(Y_n = y_n | Y_{n-1} = y_{n-1}, ..., Y_{n-2} = y_{n-2}) && (3)\\
&= \mathbb{P}(Y_2 = y_2 | Y_1 = y_1, Y_0 = y_0) && (4)\\
&= \frac{\mathbb{P}(Y_2 = y_2, Y_1 = y_1, Y_0 = y_0)}{\mathbb{P}(Y_1 = y_1, Y_0 = y_0)} && (5)
\end{aligned}$$

where:

$$\mathbb{P}(Y_1 = y_1, Y_0 = y_0) = \sum_{\substack{x_0 \\ x_1 \in g^{-1}\binom{y_0}{y_1}}} \underbrace{\mathbb{P}(X_1 = x_1, X_0 = x_0)}_{P_{x_0,x_1} \cdot \mu_{x0}} \tag{6}$$

and:

$$\mathbb{P}(Y_2 = y_2, Y_1 = y_1, Y_0 = y_0) = \sum_{\substack{x_0 \\ x_1 \in g^{-1}\binom{y_0}{y_1} \\ x_2}} \underbrace{= \mathbb{P}(X_2 = x_2, X_1 = x_1, X_0 = x_0)}_{\spadesuit} \tag{7}$$

$$\begin{aligned}
\spadesuit &= \mathbb{P}(X_2 = x_2 | X_1 = x_1, X_0 = x_0) \cdot \mathbb{P}(X_1 = x_1 | X_0 = x_0) \cdot \mathbb{P}(X_0 = x_0) && (8)\\
&= \mathbb{P}(X_2 = x_2 | X_1 = x_1) \mathbb{P}(X_1 = x_1 | X_0 = x_0) \mathbb{P}(X_0 = x_0) && (9)\\
&= P_{x1,x2} \cdot P_{x0,x1} \cdot \mu_{x_0} && (10)
\end{aligned}$$

Now that we have determined the conditional form of the Markov 2nd order probability, we can write the entropy rate as:

$$\begin{aligned}
\bar{H}(Y) = H(Y_2 | Y_1, Y_0) = \sum_{y_0, y_1, y_2} & \mathbb{P}(Y_2 = y_2, Y_1 = y_1, Y_0 = y_0) \\
& \cdot \log \mathbb{P}(Y_2 = y_2 | Y_1 = y_1, Y_0 = y_0)
\end{aligned} \tag{11}$$

---

3 Single Entry State Lumpability

To achieve an information preserving state clustering, the state space has to fulfill the condition to consist of "Single Entry States". First of all we have to consider that a state that does not belong to any cluster can still be seen as a cluster of size 1.
A "Single Entry State" is defined as a cluster of states, that is only reachable from another cluster via one single state. That means, just one state in the destination cluster can be the entry state for all member states of the source cluster. If another cluster checks the "Single Entry States" condition, it is allowed that this cluster enters the destination cluster via another state. So the "Single Entry State" condition does not mean that the same entry state defines the connection to any other cluster, it just means that there is only one entry state between the two checked clusters. So the entry state does not have to be unique to all other clusters.

Let us have a look at an example to explain this definition. Figure 2 shows the given state space where we want to cluster in a "Single Entry State" way. Note that this is not the complete transition graph of the Markov chain, but just a relevant part for the clustering purpose.

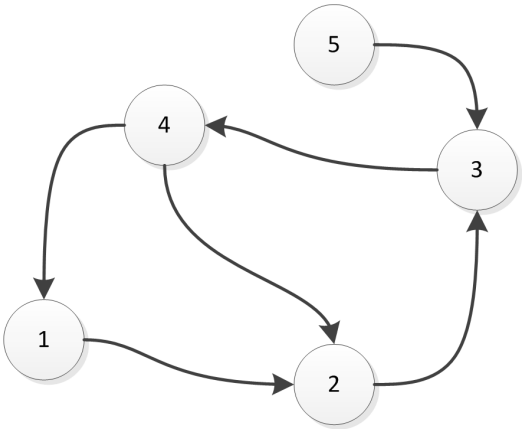

Figure 2: State Space Example

The first approach of combining states to a cluster shows a case where the "Single Entry State" condition is not fulfilled, see figure 3 for the given situation. If we want to combine states 1 and 2, we have to check all incoming connections from other clusters (which are all clusters of size 1 at this moment). We can see that state 4 has entry connections to state 1 and also to state 2. So if we would combine states 1 and 2 to a cluster, there would be not a single entry state.

Now we want to check if we could combine states 2 and 3, as shown in figure 4. Here the states 1 and 4 both have an entry connection to state 2, but the only entry condition to state 3 is the one from state 2. Due to the fact that we want to see state 2 and 3 as a cluster, the connections inside this cluster also have to be single entry. As we can see, the cluster can "enter" itself just over state 2. We can combine states 2 and 3 to cluster (2 3), figure 5 shows the new representation. From now on we can write the two states in one symbol, but we still have to keep all the connections to the included states in mind.

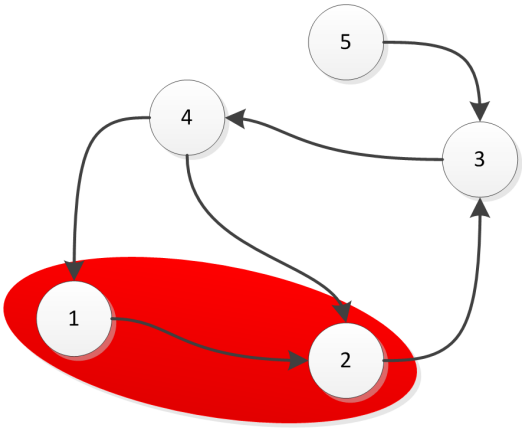

Figure 3: State Space Example, Cluster fail 1

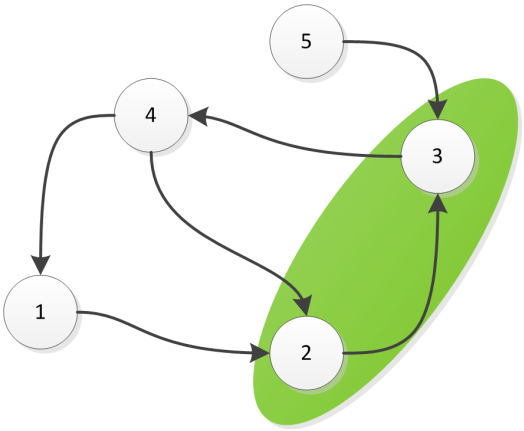

Figure 4: State Space Example, Cluster success 1

The attempt to combine the states 4 and 5 (shown in figure 6) looks fine at the first glance , but keep one thing in mind: The combination of states 2 and 3 leads to a cluster that includes both states and the connections to them. State 2 was entered from state 4 and state 3 was entered from state 5. A combination of states 4 and 5 would hurt the "Single Entry State" condition.

A combination of states 1 and 4 is the next to check, figure 7 shows this attempt. The only entry into this cluster comes from the (2 3) cluster (better

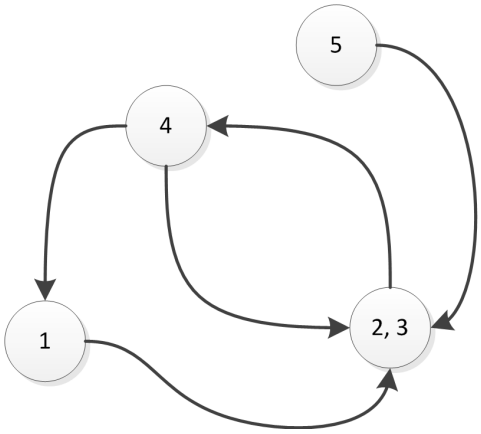

Figure 5: State Space Example, State Combination

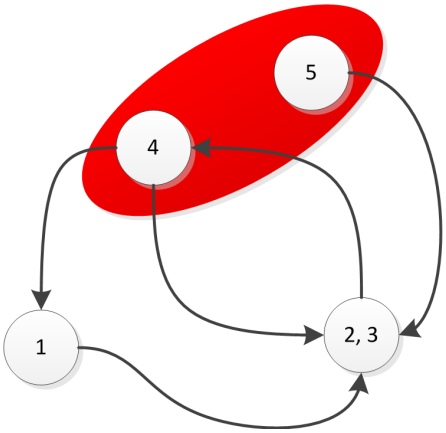

Figure 6: State Space Example, Cluster fail 2

to say from the state 3 inside the cluster). This connection remains the only entry for this cluster connection. Now we have to check if the single entry condition still holds with the cluster (2 3). The states 1 and 4 where counted as separate clusters while we checked for the clustering of 2 and 3. Now that we want to cluster 1 and 4, we cannot count them separately and we have to reconsider this decision. Since both states 1 and 4 enter the cluster (2 3) via the internal state 2, the condition still holds (as shown in figure 8).

For more detailed information about the theoretical background of Markov chain lumpability, please check [Gei12] and [Gei13].

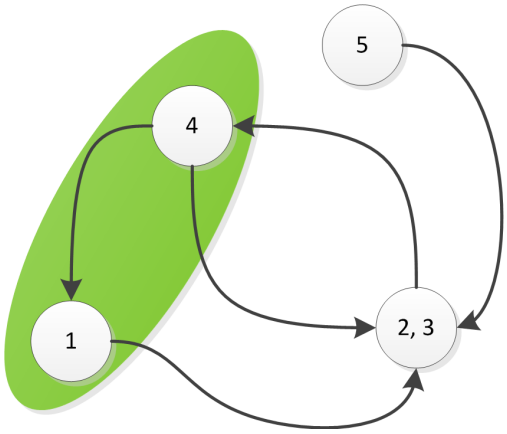

Figure 7: State Space Example, Cluster success 2

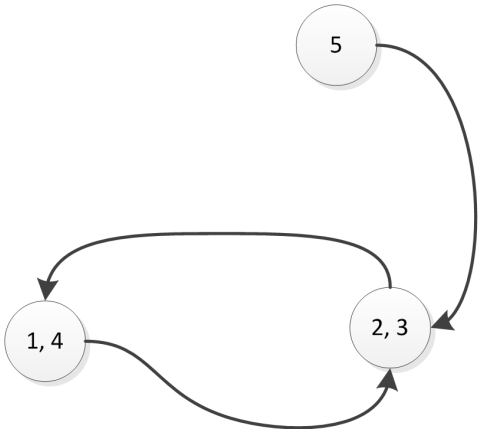

Figure 8: State Space Example, Cluster success 2

---

4 Partitioning

In order to find out which partitions satisfy the "Single Entry State" condition, all candidate partitions have to be generated and tested. To do this in an efficient way, the conclusion of a paper [Ich82] was used.

A partition of a give state space $\{1, 2, 3, ..., n\}$ is described as a combination of several clusters of states. The 5 possible partitions of the state space 1, 2, 3 are:

- (1)(2)(3)
- (12)(3)
- (13)(2)
- (1)(23)
- (123)

To identify the state clustering inside a partition, some representation is needed. A complex data structure would be a solution, but is not very efficient in terms of generation, usability in algorithms and storage. Therefore an encoding of the state combinations is introduced.
States that belong together in terms of a state cluster share an index. The index of the clusters always starts with 1, the highest number in the partition represents the number of clusters in the partition. To illustrate this, let us have a look at the example for the state space with 3 different states:

Partitioning	Encoding
(1)(2)(3)	1 2 3
(12)(3)	1 1 2
(13)(2)	1 2 1
(1)(23)	1 2 2
(123)	1 1 1

The structure of the partitions brings the idea of using a rooted tree where partitions are leaves. The partitions are created with an algorithm that uses this fact and traverses this tree from level to level as shown in figure 9.

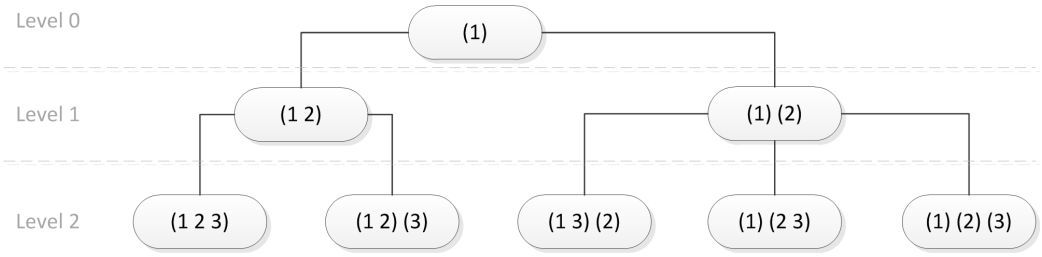

Figure 9: Tree Structure of State Space Partitions

For details on the generation algorithm or the representation of partitions, have a look at the referenced paper [Ich82].

---

5 Algorithm

The implementation of the used algorithms was fully performed in Matlab© due to reasons of simplicity.

Figure 10 gives an overview of the implementation. As an input,a transition matrix representation of a state space should be chosen. The algorithm is programmed general enough to work also with an adjacency matrices, but for the calculation of the entropy rate we need a transition matrix. If the input is valid in terms of content and dimensions, the main part is started where the partitions are generated and tested. The output consists of an array with the possible partitions.

By zooming into the main calculation block (see figure 11), we see that the given transition matrix is used to determine the size of the required tree. The partitions are generated one by one and checked just after creation. If a partition meets the "Single Entry State" condition, it gets saved to the output.

A closer look inside the checking block (see figure 12) shows how the test is performed. One partition and the whole transition matrix are the inputs to this block. Each combination of cluster pairs is checked separately. This means a check for all states that belong to the tested cluster pair. If all pairs match the condition of "Single Entry State", the Partitions are okay and ready to be saved.

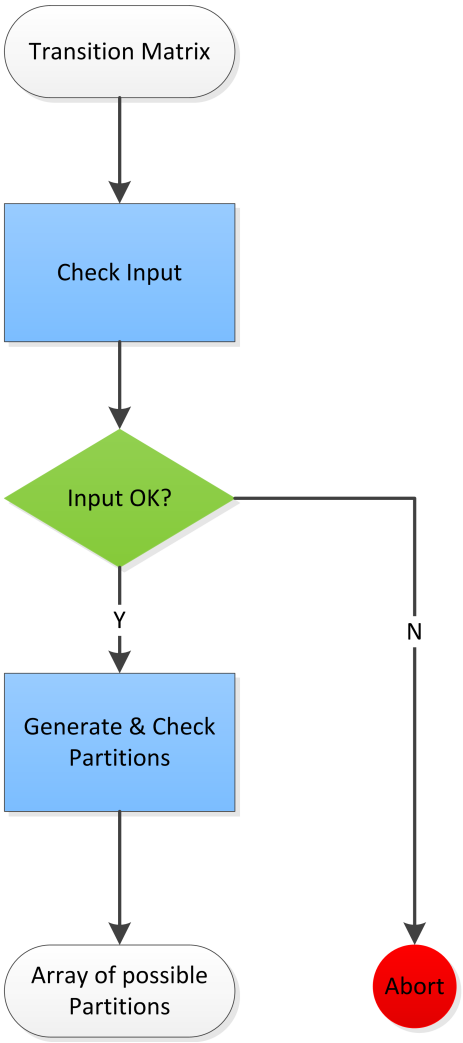

Figure 10: Overall view of Algorithm

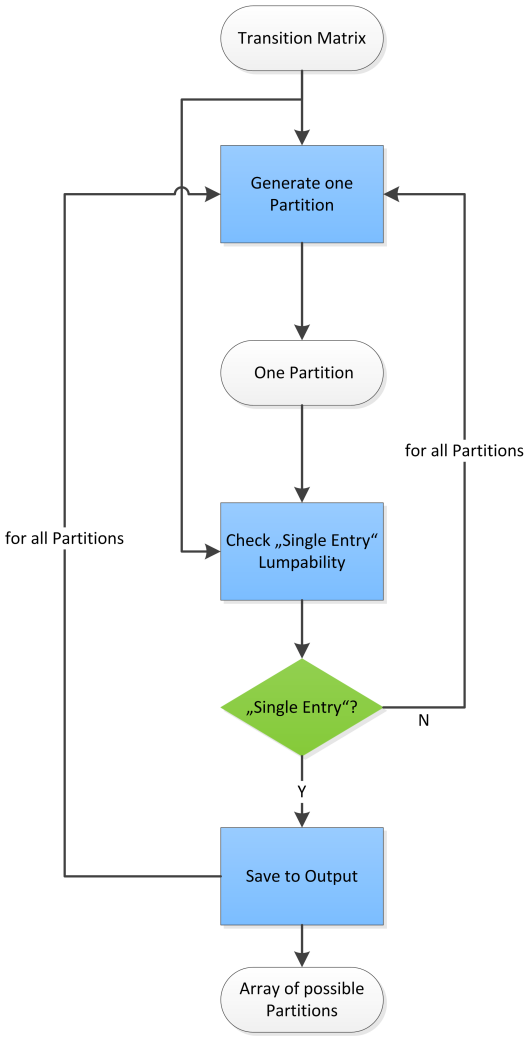

Figure 11: Inside the Generate and Check Block

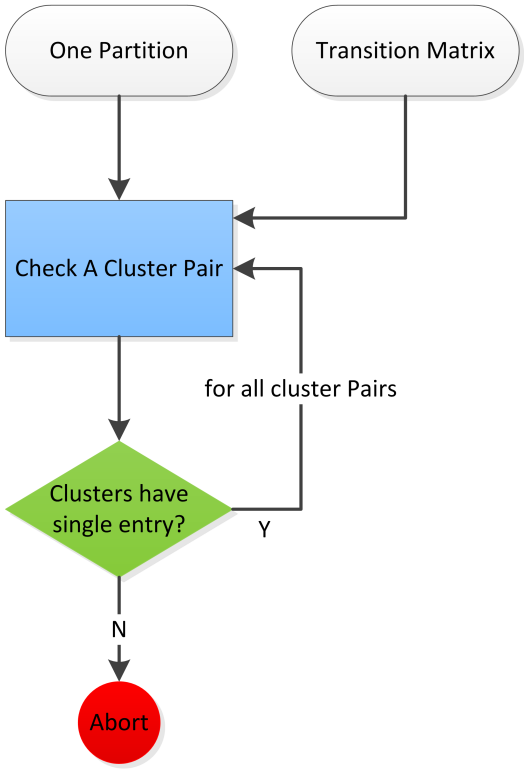

Figure 12: Inside the Single Entry Check Block

6 Example and Results

As an example we look at the given state space, as shown in figure 13.

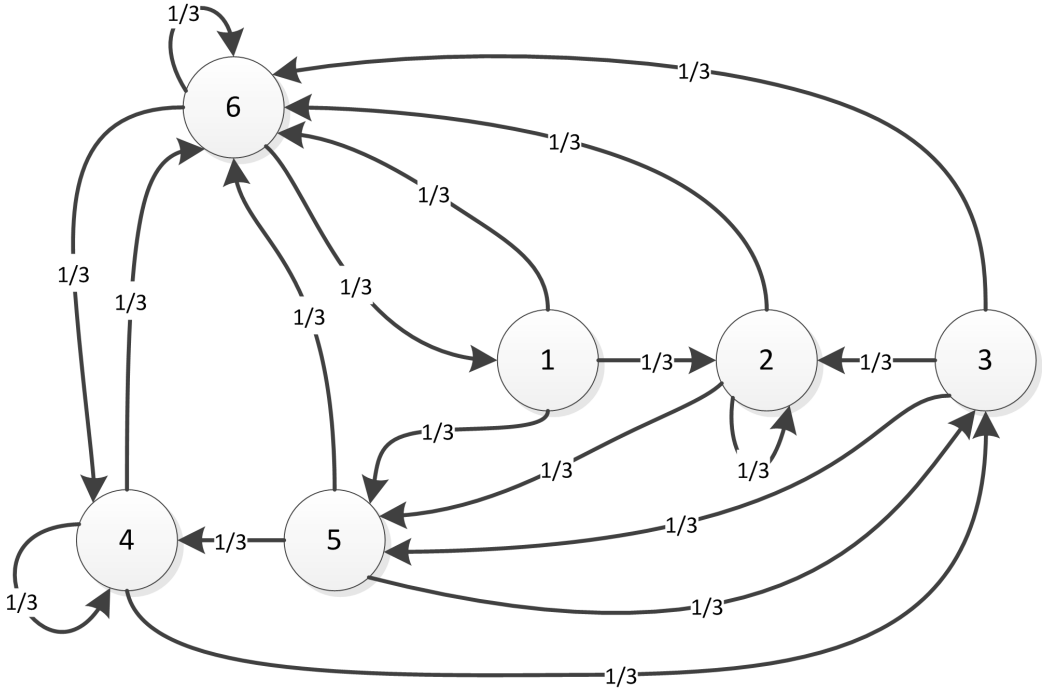

Figure 13: Example State Space

The transition matrix for this state space is:

$$P = \begin{pmatrix} 0 & 1/3 & 0 & 0 & 1/3 & 1/3 \\ 0 & 1/3 & 0 & 0 & 1/3 & 1/3 \\ 0 & 1/3 & 0 & 0 & 1/3 & 1/3 \\ 0 & 0 & 1/3 & 1/3 & 0 & 1/3 \\ 0 & 0 & 1/3 & 1/3 & 0 & 1/3 \\ 1/3 & 0 & 0 & 1/3 & 0 & 1/3 \end{pmatrix}$$

There are 11 possible partitions for this input transition matrix:

Possible Partitions						
1.)	1	1	1	2	2	3
2.)	1	1	1	2	3	4
3.)	1	1	2	3	3	4
4.)	1	1	2	3	4	5
5.)	1	2	1	3	3	4
6.)	1	2	1	3	4	5
7.)	1	2	2	3	3	4
8.)	1	2	2	3	4	5
9.)	1	2	3	4	1	5
10.)	1	2	3	4	4	5
11.)	1	2	3	4	5	6

Now we choose number 1.) of the results (see figure 14) to take a closer look at the entropy rate:

If we calculate the entropy rate of the original Markov chain and one of the possible partitions (let us say we take line 1: (1 1 1 2 2 3)), we get an entropy rate of $\overline{H}(X) = \overline{H}(Y) = 1.5850$. Therefore we see that the resulting 2nd order Markov chains still has the same entropy rate as the input Markov chain of 1st order.

---

7 Conclusion

The combination of states in a given State Space can take place without the loss of information, if the combined states are in the described special "Single Entry State" structure. A look at the calculation of the entropy rates of the original Markov chain and the 2nd order Markov chain with modified state space verifies this expectation.

There are many new approaches for the future that leave room for improvement: The generation of the partitions works quite efficiently, but we

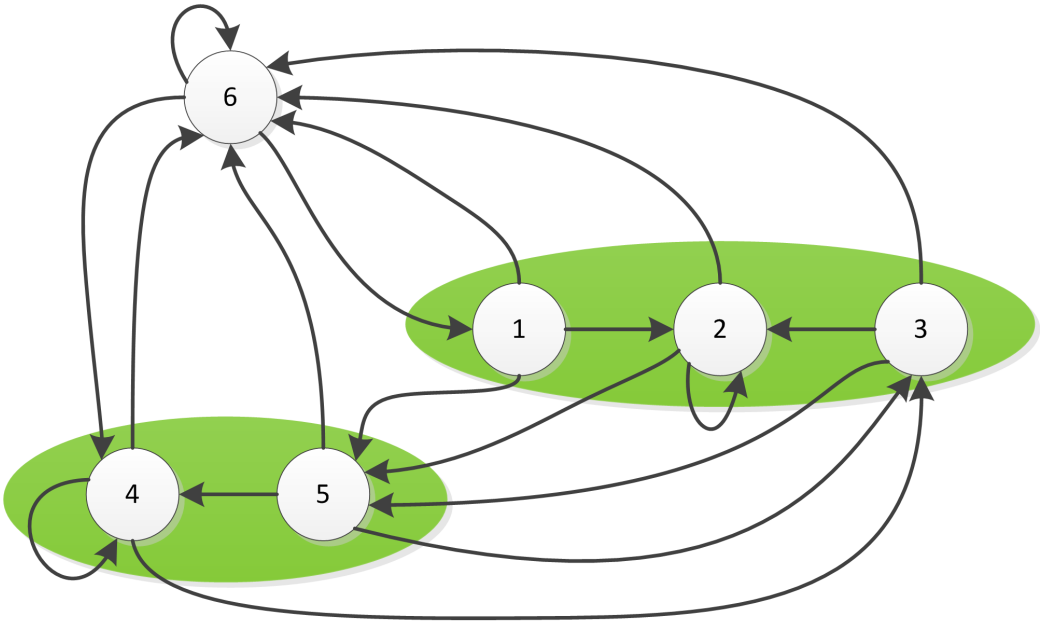

Figure 14: Resulting Partition (1 of 11)

have to take into account that not all partitions are required. If the combination of two states is not a possible option for the structure of the given state space, a combination of these two states and a third one makes no sense (as explained in [Gei13]). An efficient way of excluding unnecessary state combinations would result in a great performance increase for the calculation.
While this project focused on one type of special state space structure for the rules of lumpability, there are more different types of rule sets. Each of this rule sets would feature a different approach of partitioning of state spaces and therefore needs different implementations of the partition generation and the state combination algorithms.

References

[Gei12] Geiger, Bernhard C. and Temmel, Christoph: *Lumpings of Markov chains, entropy rate preservation, and higher-order lumpability.* Dezember 2012. – accepted in J. Appl. Prob.; preprint available: `arXiv:1212.4375 [cs.IT]`

[Gei13] Geiger, Bernhard C. and Temmel, Christoph: Information-Preserving Markov Aggregation. In: *Proc. IEEE Information Theory Workshop (ITW).* Seville, Spain, September 2013, S. 258–262. – extended version: `arXiv:1304.0920 [cs.IT]`

[Ich82] Ichiro, Semba: *An Efficient Algorithm for Generating all Partitions of the Set {1, 2, ..., n}.* Mai 1982. – Journal of Information Processing, Vol.7, No. 1, 1984