Induced Markov Chains for the Encoding of Graph-based Data

Stefan P. Müller
Institute of Mathematics
Humboldt-Universität zu Berlin
Email: stefan.mueller@math.hu-berlin.de

***Abstract*—Graph-based data occurs in various applications, e.g. finite-element simulations and computer-generated imagery. There are several techniques to compress these data sets with prediction methods and encoding of the residual. The focus of these methods is almost always on the prediction rather than on encoding. We present a new encoding scheme based on induced Markov chains (iMc) reflecting the underlying distribution of graph-based data. For this purpose, we define transition probabilities between the occurring values that are dependent on the topology of the graph. The basic idea is to transform a topological relation into a value-based one. The transition probabilities along with an initial distribution can be interpreted as a Markov chain, the so-called iMc. The topology combined with the transition probabilities can be used as side information for an encoder. Additionally we combine the iMc encoding scheme with tree and time differences as prediction methods, since some correlations cannot be entirely removed neither by the prediction methods, nor by the iMc by its own.**

I. Introduction

Data compression is nowadays an essential component for the storage and transmission of all kinds of data. Our motivating application field is the simulation of car crashes, which began in the early 1980s [1]. It is a central component in the development and quality management of automakers. Due to safety regulations and quality management, a portion of the simulations is stored for some months up to several years. This leads, on the one hand, to a huge number of stored datasets. On the other hand, the simulation results increase in size as the engineers want to improve the accuracy of the simulation results, e.g. by refining the model. Overall, this leads to an steadily increasing need of storage and bandwidth.

The main requirements on a compression method of simulation results is that the assigned mesh has to be stored losslessly and of course the decompression has to be fast. The first requirement prohibits the use of remeshing methods and we can therefore assume that the mesh exists in the decompression stage.

The simulation results consist of various data sets like header, initial components like connectivities, and time dependent data like coordinates, velocity, and stresses. The time dependent data will be calculated numerically, and thus it is only precise up to a certain accuracy. This justifies a lossy compression up to a respective precision. Moreover, these data sets are usually defined on elements of a mesh, e.g. nodes, edges, or finite elements. Hence, we may assume that the correlation of the data can be expressed by the topology of the underlying graph, see [2].

We investigate how to encode the values that are associated with the nodes of a finite graph in a performant way. If the node values are floating point numbers, we apply a quantization as we want to ensure that the node values are integers in the iMc encoding process. In the case of quantization, we will apply a certain absolute quantization with precision q. Additionally, it is possible that some variables are multidimensional, like e.g. the coordinates. In this case, we only consider one coordinate direction at a time. Furthermore we suppose that the investigated graph is connected. Otherwise we handle each connected component by its own. Hence, it is overall sufficient to confine ourself to the case of one dimensional integer node values that are defined on a connected graph.

After some statistical basics in Section II, we introduce the model of induced Markov chains for graph-based data in Section III. We call this technique induced Markov chain (iMc) and will use its statistics as side information for an arithmetic encoder. We propose the application of the iMc as the statistical model in an encoder in the following way:

- Determine a tree that contains all nodes of the graph.
- Determine an iMc based on the observed node values v and the spanning tree.
- Arithmetic coding of the values v regarding the statistics.
- Encode the statistics needed for the decoding stage.

We will use a deterministic way to find a tree only on base of the topology of the graph. Therefore, we do not have to store the tree.

With this setup, the iMc encoding scheme fits well in the method "topological surgery" of Taubin and Rossignac [3] as they establish a spanning tree for the prediction of the node positions.

The strategy to use the topology of a general graph for a specialized encoding scheme on node values is otherwise only considered in Markov random fields (MRF). But in the MRF case the time complexity of generating the statistics prevents the application in situations, where the alphabet is large, see Section VI and [4].

In a car crash simulation, we are often confronted with a high redundancy in the data, e.g. when we consider the sheet metal parts, so-called shell-elements. For a good compression rate, it is crucial to eliminate these redundancies. This task can be tackled in two ways. The first one is to predict the

data in a way that the resulting variables are independent or at least close to independent. The second one is to find and use dependencies in the data, which will be done by the iMc. A prediction usually modifies the distribution of a data set. We will investigate a combination of these two approaches, which will lead to the best compression rates in our test case, see Section VII. For the prediction part, we will use an iterative application of tree differences as well as time differences, which will be specified in Section IV. Regarding the remaining dependencies, we propose a two-pass universal method based on coding and sending empirical data measurements, see Section V, rather than a coding method based on an a priori probability model.

II. Statistical model

In this section, we will introduce the transition probabilities of node values defined on a connected graph and motivate their application by identifying the node values as a results of a random walk with certain transition probabilities. Afterwards, we use the Kullback-Leibler divergence [5] to measure, if the distribution of the underlying graph and the one of the random walk fit together.

Usually, a graph of a finite element simulation as well as in computer-generated imagery is undirected. Since we are interested in how probable it is to be at a node with a certain value and go to a neighbor with a possible different value, it is necessary to know the direction of the edge. If we have an undirected connected graph $G_u = (N, E_u)$, we will construct a directed graph $G = (N, E)$ by duplicating each edge of E_u and orient them both ways. Let the number of nodes be $n = \#N$ and the number of directed edges be $e = \#E$. Furthermore, let the finite set of node values be $\mathcal{A} = \{a_1, ..., a_m\}$, and a node value vector $v = (v_1, ..., v_n) \in \mathcal{A}^n$ be given. We call $\mathcal{A}$ the *alphabet* and assume that all elements of $\mathcal{A}$ will be part of the node value vector v. This will not be necessary for the algorithm, but simplifies the notation afterwards.

Let $(N, \mathcal{P}(N), \mathbb{P}_N)$ be the *Laplace probability space of order* n [6], where all n elementary events are equiprobable, i.e. $\mathbb{P}_N[n_i] = \frac{1}{n}$. As $\mathcal{A}$ is finite, the tuple $(\mathcal{A}, \mathcal{P}(\mathcal{A}))$ is a measurable space. We define a random variable $Y : N \to \mathcal{A}$ with $Y : n_i \mapsto a_l$ that maps a node to the value of the alphabet, which is defined by an entry of the node value vector v. We identify v as a result of the random process and assume for now that the empirical probabilities fit to the underlying distribution. The *probability mass function* (pmf) for the random variable is defined by $\mathbb{P}_Y = \mathbb{P}_N \circ Y^{-1}$ and so

$$\begin{aligned}\mathbb{P}_Y[a_l] &= \mathbb{P}_N[\{n_i \in N : Y(n_i) = a_l\}] \\ &= \frac{1}{n}\#\{n_i \in N : Y(n_i) = a_l\}. \end{aligned} \tag{1}$$

We obtain the empirical probabilities as the pmf of the random variable Y. The distribution coincides with the relative frequencies of the node values in $\mathcal{A}$.

We define a second Laplace probability space of order e as $(E, \mathcal{P}(E), \mathbb{P}_E)$. Furthermore we define a two-dimensional random variable $X : E \to \mathcal{A} \times \mathcal{A}$ with $X : e_i \mapsto (a_k, a_l)$. The pmf for the random variable is defined by $\mathbb{P}_X = \mathbb{P}_E \circ X^{-1}$ and so

$$\begin{aligned}\mathbb{P}_X[(a_k, a_l)] &= \mathbb{P}_E[\{e_i \in E : X(e_i) = (a_k, a_l)\}] \\ &= \frac{1}{e}\#\{e_i \in E : X(e_i) = (a_k, a_l)\}. \end{aligned} \tag{2}$$

With the help of the two distributions $\mathbb{P}_Y$ and $\mathbb{P}_X$, we can imagine the statistical model in the following way. We start with a graph with no node values and pick one at random with an uniform distribution. Then we assign this node a value from the alphabet dependent on the relative frequency of the node values, cp. (1). In the next step, we will start a random walk at the selected node with the chosen node value. For a random walk on a connected graph, every edge is equiprobable if the first edge was chosen equiprobable. If this is not the case, the distribution converges with the number of steps of the random walk to an uniform distribution [7]. Therefore, we also use a Laplace probability space for the transitions from one node value to another. In the following we assign each node a value dependent on the transition probabilities of the given graph G, which will be specified next.

Based on the structure of the graph, we can determine how often a node will be reached in a random walk on the graph and so how often a certain value of the alphabet will be reached. For a random walk on a graph with equiprobable edges the probability to be on node n_i is $\mathbb{P}[n_i] = \frac{deg(n_i)}{e}$, whereby $\deg(n_i)$ represents the *degree* of the node n_i [7]. Hence, the probability for a certain value to be the first entry of a directed edge value tuple is

$$\begin{aligned}\mathbb{P}_Z[a_k] &= \frac{1}{e}\sum_{n_i : Y(n_i) = a_k} \deg(n_i) \\ &= \sum_{a_l \in \mathcal{A}} \mathbb{P}_X[a_k, a_l]. \end{aligned} \tag{3}$$

If the measure $\mathbb{P}_X$ is seen as a joint distribution of two times the random variable Y with pmf $\mathbb{P}_Y$, the equation (3) also represents the marginal distribution for a certain starting node and all possible end node values in $\mathcal{A}$.

Based on the joint distribution (2) and the marginal distribution (3) we can determine the conditional probabilities of a certain value of an ending node while starting at a node whose value is known:

$$\begin{aligned}\mathbb{P}_{X|Z}[(a_k, a_l)|a_k] &= \frac{\mathbb{P}_X[(a_k, a_l)]}{\mathbb{P}_Z[a_k]} \\ &= \frac{\#\{(i,j) \in E | v_i = a_k \wedge v_j = a_l\}}{\#\{(i, \cdot) \in E | v_i = a_k\}}. \end{aligned} \tag{4}$$

We introduce the *transition probabilities* based on a graph with node values by the relative frequency of letter pairs as:

$$P_{kl} := \mathbb{P}_{X|Z}[(a_k, a_l)|a_k] \tag{5}$$

for all $k, l \in \{1, ..., m\}$. For an example, see Figure 1. We store the probabilities P_{kl} in the transition probability matrix P. The *initial distribution* ν is defined by the relative frequency of the

node values:

$$\nu_i := \mathbb{P}_Y[a_i] = \frac{\#\{k \in N | v_k = a_i\}}{n}. \quad (6)$$

After the definition of the transition probabilities, we want to investigate if the distribution of the random walk and the one of the underlying graph fit together. For that purpose we compare the distribution of the node values induced by the transition probabilities (6) with the relative frequencies (1) of the given graph using the Kullback-Leibler divergence (KLD):

$$KL(\mathbb{P}_Y, \mathbb{P}_Z) = \sum_{a \in \mathcal{A}} \mathbb{P}_Y[a] \log_2 \left(\frac{\mathbb{P}_Y[a]}{\mathbb{P}_Z[a]} \right) \quad (7)$$

In general $\mathbb{P}_Z$ and $\mathbb{P}_Y$ are different and so the KLD is greater than zero. In the special case of a k-regular graph, where each node has k neighbours, the distributions are identical.
The KLD of these two distributions can be interpreted as a measure for additional bits per symbol required to encode values distributed by $\mathbb{P}_Y$ but using a code based on $\mathbb{P}_Z$. If the distributions are very similar, the transition probabilities will depict the distribution of the graph very well. We can interpret $\mathbb{P}_Y$ as the initial distribution and $\mathbb{P}_Z$ as the stationary distribution for a random walk on a graph with node values in $\mathcal{A}$.
In this section, we introduced the underlying statistical model more generally for a connected graph instead of confining to a tree in the first step. One reason behind this approach is that the generality is needed for some applications, which will be listed in a short outlook in Section VIII. Furthermore, we stated the formula of the KLD to compare $\mathbb{P}_Y$ and $\mathbb{P}_Z$, and use it in Section VII to show that these two distributions are similar at least for the investigated example.

III. Induced Markov chains

In this section, we will define the iMc.
Let $X_0, X_1, ...$ be a sequence of random variables with values in S, which is a finite set with m elements $s_1, ..., s_m$, the so-called *state space*. A *discrete Markov chain* M is a sequence of these X_i satisfying the *Markov property* for all $t_0, ..., t_k \in S$:

$$\mathbb{P}[X_{k+1} | X_0 = t_0, ..., X_k = t_k] = \mathbb{P}[X_{k+1} | X_k = t_k], \forall k \in \mathbb{N}.$$

A Markov chain M is called *time homogeneous*, if for all $k \in \mathbb{N}$ and all $s, t \in S$, it holds:

$$\mathbb{P}[X_k = s | X_{k-1} = t] = \mathbb{P}[X_1 = s | X_0 = t].$$

A *stochastic matrix* is a matrix whose rows are distributions. The *transition probability matrix* $P = (P_{ij})_{i,j=1,...,m}$ with $P_{ij} = \mathbb{P}[X_1 = s_j | X_0 = s_i]$ is a special case of a stochastic matrix.

Lemma 3.1 ([8]): The tuple (P, ν) of a stochastic matrix P and a distribution ν defines a finite state space, time homogeneous Markov chain M with transition probabilities given in P and an initial distribution ν.

$$\mathbb{P}[X_0 = s_i] = \nu_i, \ \mathbb{P}[X_{k+1} = s_j | X_k = s_i] = P_{ij}, \ \ \forall s, t \in \mathcal{A}.$$

We introduce the *induced Markov chain* (iMc) as the Markov chain from Lemma 3.1 with transition probabilities defined by Formula (5) and initial distribution defined by (6). For an example, see again Figure 1.

IV. Tree and time differences

In this section, we want to explain shortly the prediction we will use to decorrelate the data.
Let $T = (N, E_t)$ be a tree with each edge oriented in the direction pointing to the leafs. We predict all values v_i for all nodes $i \in \{n_1, ..., n_n\} \backslash \{n_{\text{root}}\}$ by the value of its predecessor. This reduces the correlation of the data regarding the tree. Hereby, the value of the root cannot be predicted. [3]
When we have the data of several points of time in a simulation result, we can also predict the data in time. The easiest way is to predict a time step with its previous one. Again the first time step cannot be predicted, and therefore can only be decorrelated by the tree differences.
We exclude the non-predicted values, i.e. the root nodes as well as the first time step, from the statistics on the one hand to reduce the size of the statistics and on the other hand to adapt them as good as possible to the underlying data set. Of course, there are more sophisticated prediction methods possible.

V. The iMc encoder

In this section, we introduce the iMc encoder.
The modeling of graph-based data as an iMc can be used in an arithmetic encoder that uses the connectivities and statistics as side information. A possible implementation is to establish the graph, then define a tree, and use the transition probabilities while we walk from the root to the leafs. When we confine the graph to a tree, we can consider the tree as a directed graph with the direction of all edges from the root to the leafs. We can determine the transition probabilities on basis of the directed tree and the distribution is optimal for encoding regarding the chosen tree.
In the case of only a few time steps, we store the root value for every time step without encoding. For non-root elements of the tree, we have the information about its predecessor, and therefore, we can apply the transition probabilities that are specialized for the given tree, cp. Figure 2. This leads to an encoding scheme, whereby the applied distribution is depending on the current node value and therefore will generally change from one node to another. Arithmetic encoders can handle this situation [9].
There are several possibilities to find a tree of a graph, like e.g. breadth-first search and depth-first search (DFS) [10]. Since we use a deterministic strategy, we will not store the structure of the tree.
Assuming that the mesh and the node values are available, the main steps of the encoding algorithm are:

1) Construct the adjacency matrix in CRS format.
2) Finding a tree by DFS.
3) Build the adjacency matrix of the tree.

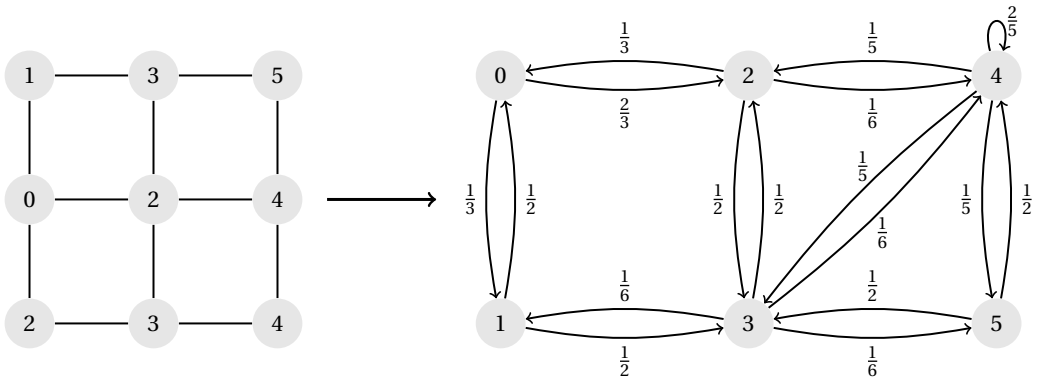

Fig. 1. Mesh with node values and the induced Markov chain with transition probabilities calculated by Formula (5).

4) Set root values and apply tree and time differences (optional).
5) Determine transition frequencies.
6) Cumulate the transition frequencies (for the application in an arithmetic coder).
7) Determine a map matrix (for a faster access due to the CRS format).
8) Arithmetic coding.
9) Encoding of the transition frequency matrix.

The first two steps of encoding and decoding procedures are the same. Assuming that the coded values are available, the main steps of the decoding algorithm are:

3) Decode transition frequency matrix.
4) Cumulate transition frequencies.
5) Arithmetic decoding.
6) Tree and time sums (if necessary).

We compress the iMc statistics losslessly by exploiting that in an arithmetic encoder we use integers, i.e. the frequencies, to describe the probabilities. For the structure of the transition probability matrix, we use the *Compressed Sparse Row* (CSR) format. All integer arrays are decorrelated and encoded with the zlib [12] afterwards.

When we save the distribution, we do not use the knowledge

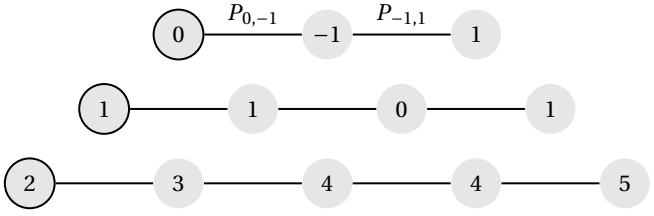

Fig. 2. Snippet of a tree for the example from Figure 1 with two tree differences. The bold-framed values, and the two transition probabilities have to be provided to an iMc encoder as side information.

about the transitions we have already applied in the encoding phase as well as in the decoding phase. We can use this knowledge by adapting the distribution by deleting the already processed transitions from the distribution. This is an opposing application of the adaptive arithmetic coding in [9].

TABLE I
TIME COMPLEXITY OF THE IMC ENCODER AND DECODER.

Initialization	
1. Adjacency matrix of graph	$\mathcal{O}(e+n)$
2. Finding tree	$\mathcal{O}(n+l)$
Compression	
3. Adjacency matrix of tree	$\mathcal{O}(n)$
4. Prediction	$\mathcal{O}(n \cdot r)$
5. Transition frequencies	$\mathcal{O}(e \cdot r + g \cdot log_2(g))$
6. Cumulate frequencies	$\mathcal{O}(p)$
7. Construct map matrix	$\mathcal{O}(p)$
8. Arithmetic encoding	$\mathcal{O}(e \cdot r \cdot \log_2(d))$
9. Encoding statistics	$\mathcal{O}(n+p)$
Decompression	
3. Decode Statistics	$\mathcal{O}(n+p)$
4. Cumulate frequencies	$\mathcal{O}(p)$
5. Arithmetic decoding	$\mathcal{O}(e \cdot r \cdot \log_2(d))$
6. Reverse prediction	$\mathcal{O}(n \cdot r)$

VI. TIME COMPLEXITY OF IMC GENERATION

In this section, we estimate the time complexity of the iMc encoder introduced in the last section and compare the effort of generating the statistics with those of the MRF.

Let the maximal degree of a node in G be denoted by d, the number of nodes by n, the number of edges by e, the number of nonzero elements of the adjacency matrix by l, the number of nonzero elements of the transition probability matrix by p, the number of points in time stored in the simulation result file by r, the maximum number of entries per line in the transition probability matrix by g, and the size of the compressed dataset by c. The time complexity for the encoding stage is

$$\mathcal{O}((e \cdot r \cdot \log_2(d)) + n \cdot r + 2p + g \log_2(g))$$

and for the decoding stage, it is

$$\mathcal{O}((e \cdot r \cdot \log_2(d)) + n \cdot r + p + c),$$

see Table I. For a part with shell elements, the average number of neighbors is approximately four. The maximal degree of a node d is usually be bounded by ten. Thus, d can be neglected. The effort of generating the statistics in the iMc case can be found as the fifth point "Transition frequencies" in Table I. If we only consider one point in time, the time complexity can

be bound by $\mathcal{O}(e + g \cdot \log_2(g))$, whereby the variable g is by construction at most m. Therefore, iMc encoding is applicable in the case of a large alphabet, especially when the number of possible transitions from one value to another one is small. When we consider an entropy encoding scheme with MRFs on a spanning tree for one variable and one point in time, the time complexity of generating the statistics is $\mathcal{O}(n \cdot m^2)$ [4]. Thus, it is mostly used in situations where the size of the alphabet is very small, e.g. black/white images.

The comparison of the time complexities shows that, in opposite to the MRF case, the iMc can be applied in cases where the alphabet is large.

VII. Results

In this section, we list and discuss the results of applying the iMc encoder on graph-based data. We compare the results on the one hand with those regarding to encoding with gzip 1.6 [11] and on the other hand with those using the relative frequencies of Equation (6) as statistics for an arithmetic encoder. We call the second approach *relative frequency coding* (rfc). For the iMc as well as the rfc case we encode unsigned 32 bit integers as opposed to the unsigned 8 bit encoding of gzip. Due to the 32 bit coding, we store the statistics in the rfc case and do not use an adaptive arithmetic coder like in [9].

In the gzip case, we apply a quantization and the same prediction steps like in the iMc case, i.e. time and tree differences. Afterwards, we use gzip 1.6 on the predicted data set. For the iMc case, we have to store the transition frequency matrix P for the statistics as side information, the root values for the prediction, and the encoded data.

To compare the three methods, we investigate the coordinates of the biggest part in a variation of the Dodge Neon model [13] with a refined mesh. It is a underbody part with cardan tunnel, has a shell element mesh with 78869 nodes, and the ranges for the x, y, and z coordinates are $[-4016, -1900]$, $[697, 629]$, and $[108, 550]$ millimeter (mm), respectively. The model was simulated with Pam-Crash [14], where 25 points in time were recorded.

In Table II, III, and IV, we list the bit per symbol (bps) dependent on the precision of the quantization in mm and on the number of applied tree differences. The bps were averaged over the x, y, and z coordinate.

Table II shows that the iMc generates better compression rates than gzip for the first time step, especially for a coarse quantization. Interestingly, the rfc case performs better with two tree differences than with one and outperforms the iMc encoding for the finest investigated case of 0.01mm. Since the results suggest to use one tree difference in the iMc case, this combination will be better than rfc encoding for 0.1mm and 1mm quantization. In Table III, the compression of the second to 25th time step is stated, which are predicted with time differences. Again, for all combinations the iMc encoder generates distinct better compression rates than the gzip encoding. Moreover, all compression rates are smaller for the iMc encoder compared to the rfc encoder.

TABLE II
Compression rates (bps) for the iMc encoder, rfc encoder, and gzip for the first point in time of the Dodge Neon model.

Absolute precision	Number of tree differences	iMc (bps)	rfc (bps)	gzip (bps)
0.01	0	14.41	17.81	17.01
	1	8.28	9.29	11.00
	2	8.76	7.21	10.72
0.1	0	8.04	12.95	12.47
	1	4.97	5.93	7.36
	2	5.19	5.13	7.71
1.0	0	3.42	11.75	6.41
	1	2.64	5.05	3.99
	2	2.76	4.56	4.53

TABLE III
Compression rates (bps) for the iMc encoder, rfc encoder, and gzip for the second to 25th point in time of the Dodge Neon model.

Absolute precision	Number of tree differences	iMc (bps)	rfc (bps)	gzip (bps)
0.01	0	3.04	10.14	6.20
	1	2.53	3.05	4.09
	2	2.61	2.96	4.54
0.1	0	0.89	6.82	1.95
	1	0.84	0.97	1.40
	2	0.91	1.19	1.79
1.0	0	0.17	3.67	0.42
	1	0.18	0.19	0.33
	2	0.20	0.30	0.45

TABLE IV
Compression rates (bps) for the iMc encoder, rfc encoder, and gzip for the first point in time combined with the time-predicted second to 25th point in time of the Dodge Neon model.

Absolute precision	Number of tree differences	iMc (bps)	rfc (bps)	gzip (bps)
0.01	0	3.50	10.44	6.64
	1	2.76	3.30	4.37
	2	2.86	3.13	4.78
0.1	0	1.17	7.07	2.37
	1	1.00	1.17	1.64
	2	1.09	1.35	2.02
1.0	0	0.30	3.89	0.66
	1	0.27	0.30	0.47
	2	0.30	0.42	0.61

Table IV shows that the iMc encoding generates better compression rates than gzip for the combination of the first time step and the time predicted second to 25th time step. Again, the iMc encoder outperforms the rfc encoder in all investigated cases. The combination of time differences and an initial time step is a possible strategy for the compression of a whole simulation result. The compression rate for a realistic precision of 0.1mm, time differences, and one tree difference is about 1 bps for the iMc case compared to 1.17 bps for rfc encoding and 1.64 bps for gzip encoding, respectively.

After the short discussion of the compression rates, we want to investigate how good the Markov chain approach fits to the compression of node values. For this purpose, we stated the Kullback-Leibler divergence (KLD), see (7). Table V contains the values of the KLD for the first time step of the x coordinate

TABLE V
KULLBACK-LEIBLER DIVERGENCE FOR THE FIRST TIME STEP, x COORDINATE, AND ONE TREE DIFFERENCE.

	Absolute Precision	$KL(\mathbb{P}_Y, \mathbb{P}_Z)$	$KL(\mathbb{P}_Z, \mathbb{P}_Y)$
Graph	0.01	3.50E-04	3.58E-04
	0.1	1.39E-04	1.41E-04
	1	8.62E-05	8.77E-05
Tree	0.01	–	1.68E-03
	0.1	1.51E-04	1.54E-04
	1	1.42E-05	1.48E-05

that are predicted by one tree difference. Here, we distinguish the case where we calculate the stationary distribution $\mathbb{P}_Z$ on base of the tree and on base of the whole graph, respectively. Since we want to compress the node values and no random walk itself, we assume that $\mathbb{P}_Y$ is the distribution we want to approximate by $\mathbb{P}_Z$. The elements of Table V can be interpreted as a measure for additional bps required to code values distributed by $\mathbb{P}_Y$ but using a code based on $\mathbb{P}_Z$. Table V shows that the expense is very small. For an absolute precision of 0.01 and the stationary distribution determined for a tree, the KLD is not defined as at least one value $a_i \in \mathcal{A}$ exists with $\mathbb{P}_Z[a_i] = 0$ and $\mathbb{P}_Y[a_i] \neq 0$. This is the case if a_i only occurs on leafs of the tree. This does not influence the coding algorithm but prevents to compare the two distributions. Therefore, we listed additionally the KLD $KL(\mathbb{P}_Z, \mathbb{P}_Y)$ to show that the influence of the leaf values is small.

VIII. CONCLUSION

We investigated the compression of graph-based data using a new strategy, the so-called iMc encoding. For that, we defined a Markov chain on base of transition probabilities that are dependent on the node values of neighbors in the graph and interpret the node values as a result of a random walk on the graph. The size of the Kullback-Leibler divergence showed, that the stationary of the iMc, which reflects the probability to be at a certain value in the k-th step, $k \gg 1$, of a random walk, is very similar to the initial distribution for the investigated model. Additionally, we could determine the number of bits we give away by using the transition probabilities instead of the unknown underlying distribution. Therefore, at least for this case, the iMc can reasonably be applied.

The results show that even after the application of tree and time differences, the data is not yet fully decorrelated. This can be exploited by an encoder that uses the connectivities and the iMc statistics as side information. Especially, if the number of different values can be reduced by time differences or a coarse quantization, this leads to significantly better compression rates. Since we assume that the connectivities have to be stored in the compressed file we can combine the iMc with a connectivity compression method like topological surgery [3], the TG coder [2], or the Cut Border Machine [15]. One big advantage of the iMc is, that it can be employed on general graphs and is not limited to a certain regularity of the mesh.

The statistics of the iMc encoder cause a certain overhead. This has to be compensated by a better encoding. Therefore the application of iMc encoding is suggested for graphs with at least 500 nodes.

There are several additional strategies to use the iMc statistics. In [16], it was shown that in some cases, it can be useful to store the tree. In this case we can determine the transition probabilities of the whole graph and identify each edge $e_j = (n_s, n_t) \in E$ of G with an edge weight $w_j = -\log_2(P_{kl})$, whereby the node value on n_s is a_k and on n_t is a_l. Afterwards we can determine the arborescence of the weighted graph, e.g. by the Edmonds' algorithm [17].

Another way of applying the iMc is to use a general directed acyclic graph, which can consist of one half of all directed edges of the graph, instead of just a tree. Moreover, with the iMc, we can calculate the entropy for a data set and determine if it can reasonably be compressed by the iMc encoder.

A possible way to improve the speed of encoding and decoding is to encode the points of time in parallel. If we do not use the opposing adaptive approach, we expect a good scaling as the procedures are independent for each time step.

ACKNOWLEDGMENT

The author would like to thank Rudolph Lorentz, Martin Weiser, Caren Tischendorf, Lennart Jansen, and Matthew Reyes for many productive discussions and the Berlin Mathematical School for conference travel support.

REFERENCES

[1] E. Haug, *Engineering safety analysis via destructive numerical experiments*, EUROMECH 121, Polish Academy of Sciences, Engineering Transactions 29(1), p. 3949, 1981.
[2] C. Touma, C. Gotsman, *Triangle mesh compression*, Proc. Graphics Interface '98, p. 26-34, 1998.
[3] G. Taubin, J. Rossignac, *Geometric compression through topological surgery*, ACM Trans. Graph., vol. 17, no. 2, p. 84-115, 1998.
[4] M. Reyes, *Cutset Based Processing and Compression of Markov Random Fields*, Ph.D. dissertation, University of Michigan, 2010.
[5] S. Kullback, R. Leibler, *On Information and Sufciency*, The Annals of Mathematical Statistics, vol. 22, no. 1, p. 79-86, 1951.
[6] H. Bauer, *Probability theory*, de Gruyter, Berlin, Germany, 1996.
[7] L. Lovász, *Random Walks on Graphs: A Survey*, Combinatorics, Paul Erdös is Eighty, János Bolyai Mathematical Society, p. 353-398, Budapest, Hungary, 1996.
[8] D. Stroock, *An Introduction to Markov Processes*, Springer, Graduate texts in mathematics, Berlin, Germany, 2005.
[9] D. Salomon, *Data Compression*, Springer, London, England, 2007.
[10] T. Cormen, C. Leiserson, R. Rivest, C. Stein, *Introduction to algorithms*, The MIT Press, Cambridge, Massachusetts, 2001.
[11] J. Gailly, M. Adler *gzip 1.6*, http://www.gzip.org, 2013.
[12] G. Roelofs, J. Gailly, M. Adler *zlib 1.2.8*, http://www.zlib.net, 2013.
[13] National Crash Analysis Center, *Dodge Neon, Detailed model (272,485 elements)*, http://www.ncac.gwu.edu/vml/models.html, 2006.
[14] ESI Group, *PAM-Crash*, http://virtualperformance.esi-group.com/applications-structural-crash, 2014.
[15] S. Gumhold, W. Straßer, *Real time compression for triangle mesh connectivity*, SIGGRAPH '98, p. 133-140, 1998.
[16] M. Rettenmeier, *Data Compression for Fluid Dynamics on Irregular Grids*, Logos Verlag Berlin, Germany, 2012.
[17] H. Gabow, Z. Galil, T. Spencer, R. Tarjan, *Efficient algorithms for finding minimum spanning trees in undirected and directed graphs*, Combinatorica, vol. 6, no. 2, Springer-Verlag, p. 109-122, 1986.