An Information-theoretic Framework for the Lossy Compression of Link Streams

Robin Lamarche-Perrin

An Information-theoretic Framework for the Lossy Compression of Link Streams

Robin Lamarche-Perrin

Centre national de la recherche scientifique
Institut des systèmes complexes de Paris Île-de-France
Laboratoire d'informatique de Paris 6

Abstract

Graph compression is a data analysis technique that consists in the replacement of parts of a graph by more general structural patterns in order to reduce its description length. It notably provides interesting exploration tools for the study of real, large-scale, and complex graphs which cannot be grasped at first glance. This article proposes a framework for the compression of temporal graphs, that is for the compression of graphs that evolve with time. This framework first builds on a simple and limited scheme, exploiting structural equivalence for the lossless compression of static graphs, then generalises it to the lossy compression of link streams, a recent formalism for the study of temporal graphs. Such generalisation relies on the natural extension of (bidimensional) relational data by the addition of a third temporal dimension. Moreover, we introduce an information-theoretic measure to quantify and to control the information that is lost during compression, as well as an algebraic characterisation of the space of possible compression patterns to enhance the expressiveness of the initial compression scheme. These contributions lead to the definition of a combinatorial optimisation problem, that is the Lossy Multistream Compression Problem, for which we provide an exact algorithm.

Keywords: Graph compression, link streams, structural equivalence, information theory, combinatorial optimisation.

Table of Definitions

Email address: Robin.Lamarche-Perrin@lip6.fr (Robin Lamarche-Perrin)
URL: https://www-complexnetworks.lip6.fr/~lamarche/ (Robin Lamarche-Perrin)

Contents

Table of Notations

$v \in \boldsymbol{V}$	/	$t \in \boldsymbol{T}$	a vertex / a time instance
$V \in \mathcal{P}(\boldsymbol{V})$	/	$T \in \mathcal{P}(\boldsymbol{T})$	a vertex subset / a time subset
$\mathcal{V} \in \mathfrak{P}(\boldsymbol{V})$	/	$\mathcal{T} \in \mathfrak{P}(\boldsymbol{T})$	a vertex partition / a time partition
$\mathcal{V}(v) \in \mathcal{V}$	/	$\mathcal{T}(t) \in \mathcal{T}$	the vertex subset in $\mathcal{V}$ that contains v / the time instance in $\mathcal{T}$ that contains t

$(v, v', t) \in \boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}$	a multiedge
$V\times V'\times T \in \mathcal{P}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})$	a Cartesian multiedge subset
$\mathcal{V}\times\mathcal{V}\times\mathcal{T} \in \mathfrak{P}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})$	a grid multiedge partition
$\mathcal{V}\mathcal{V}\mathcal{T} \in \mathfrak{P}^{\times}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})$	a Cartesian multiedge partition
$\mathcal{V}\mathcal{V}\mathcal{T}(v, v', t) \in \mathcal{V}\mathcal{V}\mathcal{T}$	the multiedge subset in partition $\mathcal{V}\mathcal{V}\mathcal{T}$ that contains (v, v', t)

$e : \boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T} \to \mathbb{N}$	the edge function of a multistream
$e : \mathcal{P}(\boldsymbol{V})\times\mathcal{P}(\boldsymbol{V})\times\mathcal{P}(\boldsymbol{T}) \to \mathbb{N}$	the additive extension of the edge function
$e(\boldsymbol{V}, \boldsymbol{V}, \boldsymbol{T})$	the total number of edges

$(X, X', X'') \in \boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}$	the observed variable associated with the empirical distribution of edges in a multistream
$\mathcal{V}\times\mathcal{V}\times\mathcal{T}(X, X', X'') \in \mathcal{V}\times\mathcal{V}\times\mathcal{T}$	the compressed variable resulting from the compression of the observed variable (X, X', X'') by a given multiedge partition $\mathcal{V}\times\mathcal{V}\times\mathcal{T}$
$(Y, Y', Y'') \in \boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}$	the external variable which distribution is used to decompress the compressed variable $\mathcal{V}\times\mathcal{V}\times\mathcal{T}(X, X', X'')$
$\mathcal{V}\times\mathcal{V}\times\mathcal{T}_{(Y,Y',Y'')}(X, X', X'') \in \boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}$	the decompressed variable obtained from the decompression of the compressed variable $\mathcal{V}\times\mathcal{V}\times\mathcal{T}(X, X', X'')$ according to the external variable (Y, Y', Y'')
$\mathrm{loss}(\mathcal{V}\times\mathcal{V}\times\mathcal{T})$	the information loss induced from the compression of the observed variable (X, X', X'') by a given multiedge partition $\mathcal{V}\times\mathcal{V}\times\mathcal{T}$ and its decompression according to the external variable (Y, Y', Y'')

$V \in \hat{\mathcal{P}}(\boldsymbol{V})$	/	$T \in \hat{\mathcal{P}}(\boldsymbol{T})$	a feasible vertex subset / a feasible time subset
$\mathcal{H}(\boldsymbol{V})$	/	$\mathcal{I}(\boldsymbol{T})$	a vertex hierarchy / a set of time intervals
$V\times V'\times T \in \hat{\mathcal{P}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})$			a feasible Cartesian multiedge subset
$\mathcal{V}\mathcal{V}\mathcal{T} \in \hat{\mathfrak{P}}^{\times}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})$			a feasible Cartesian multiedge partition

$\hat{\mathfrak{R}}(\mathcal{V}\mathcal{V}\mathcal{T}) \subset \hat{\mathfrak{P}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})$	the set of feasible multiedge partitions that refine $\mathcal{V}\mathcal{V}\mathcal{T}$
$\hat{\mathfrak{C}}(\mathcal{V}\mathcal{V}\mathcal{T}) \subset \hat{\mathfrak{P}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})$	the set of feasible multiedge partitions that are covered by $\mathcal{V}\mathcal{V}\mathcal{T}$

1. Introduction

Graph abstraction is a data analysis technique aiming at the extraction of salient features from relational data to provide a simpler, and hence more useful representation of the graph under study. Such a process generally relies on a controlled information reduction suppressing redundancies or irrelevant parts of the data [1]. Abstraction techniques are hence crucial to the study of real, large-scale, and complex graphs which cannot be grasped at first glance. First, they provide tools for an optimised storage and data treatment by reducing memory requirements and running times of analysis algorithms. Second, and more importantly, they constitute valuable exploration tools for domain experts who are looking for preliminary macroscopic insights about their graphs' topology or, even better, a multiscale representation of their data.

Among abstraction techniques, *graph compression* [2, 3], also known as *graph simplification* or *graph summarisation* [4, 5, 6], consists in replacing parts of the graph by more general structural patterns in order to reduce its description length. For example, one "can replace a dense cluster by a single node, so the overall structure of the network becomes clearer" [1], or more generally replace any frequent subgraph pattern (*e.g.*, cliques, stars, loops) by a label of that pattern. Such techniques hence range from those building on collections of domain-specific patterns, such as *graph rewriting* techniques in which patterns of interest are specified according to expert knowledge [7], to those relying on more generic patterns, such as *power graph* techniques in which any group of vertices with identical interaction profiles is a candidate for summarisation [8, 9]. Because they provide more general approaches to graph analysis, we will focus on the latter.

In this article, we are more particularly interested in the compression of *temporal graphs*, that is the compression of graphs that evolve with time. Many research studies are indeed interested in the dynamics of relations, as for example the evolution of friendship relations in social sciences, or even in the dynamics of interaction events [10], as for example contact or communication networks, such as mail exchanges, financial transactions, physical meetings, and so on. Having to deal with an additional dimension – that is the temporal dimension – challenges compression techniques that have initially been developed for the study of static graphs. A traditional approach to generalise such techniques preliminary consists in the construction of a sequence of static graphs, by slicing the temporal dimension into distinct periods of interest, then in independently applying classical compression schemes to each graph of this sequence. However, such a process introduces an asymmetry in the way structural and temporal information is handled, the latter being compressed prior to – and independently from – the former.

To this extent, recent work on the *link stream* formalism proposes to deal with time as a simple addition to the graph's structural dimensions [11, 12]. Considering temporal graphs and interaction networks as genuine tridimensional data, the arbitrary separation of structure and time is therefore prohibited. Following this line of thinking, the compression scheme we present in this article aims at the natural generalisation of the bidimensional compression of static graphs to the tridimensional compression of link streams, thus participating in the development of this emerging framework. Similar generalisation objectives have been addressed in previous work on graph compression, as for example the application of bidimensional *block models* to multidimensional matrices [13] or the application of *biclustering* to triplets of variables [14], which has then been exploited for the statistical analysis of temporal graphs [15]. The particular interest of such approaches also consists in the fact that they provide a unified compression scheme in which structural and temporal information is simultaneously taken into account.

In order to present our compression framework, this article starts canonical and specific, then increase in generality and in sophistication. Section 2 introduces the *Graph Compression Problem* (GCP), a first compression scheme that relies on a most classical combinatorial problem in graph theory: Finding classes of structurally-equivalent vertices [16] to summarise the adjacency-list and the adjacency-matrix representations of a given graph. This approach to graph compression is canonical in the sense that it only builds on the primary, first-order information that is contained in relational data, that is the information encoded in vertex adjacency. It is also specific in the sense that it only applies to simple graphs (that is graphs for which at most one edge is allowed between two vertices) with no temporal dimension (that is static graphs). Moreover, this first scheme is lossless (it does not allow for any information loss during compression) and its solution space is both strongly constrained (only vertex partitions are considered, whereas edge partitions would allow for much more compression choices) and weakly expressive (any vertex subset is feasible, whereas interesting structural properties preliminarily defined by the expert domain might need to be preserved during compression).

In order to address such limitations, Section 3 consists in a step-by-step generalisation of the GCP to make it suitable for the lossy compression of temporal graphs. First, we show how to deal with the compression of *multigraphs* (that is graphs for which multiple edges are allowed between two vertices) by generalising the notion of structural equivalence to the case of multiple edges (3.1). Second, we allow for a *lossy* compression scheme by formalising a proper measure of information loss building on the entropy of the adjacency information contained in the compressed graph relative to the one contained in the initial graph (3.2). Third, we allow for a less constrained compression scheme by generalising from vertex partitions to edge partitions (3.3). Fourth, we allow for a more expressive scheme by driving compression according to a predefined set of feasible aggregates (3.4). Fifth and last, we generalise the resulting framework to the compression of temporal multigraphs, that is what we later call *multistreams*, by adding a temporal dimension to the compression scheme (3.5). These five contributions finally define a general and flexible scheme for link stream compression, that we call the *Multistream Compression Problem* (MSCP).

Section 4 then presents a combinatorial optimisation algorithm to solve the MSCP. It relies on the reduction of the problem to the better-known *Set Partitioning Problem* (SPP) arising as soon as one wants to organise a set of objects into covering and pairwise disjoint subsets such that an additive objective is minimised [17]. Building on a generic algorithmic framework proposed in previous work to solve special versions of the SPP [18, 19], this article derives an algorithm to the particular case of the MSCP. This algorithm relies on the acknowledgement of a principle of optimality, showing that the problem's solution space has an optimal substructure allowing for the recursive combination of locally-optimal solutions. Applying classical methods of dynamic programming and providing a proper data structure for the MSCP, we finally derive an exact algorithm which is exponential in the worst case, but polynomial when the set of feasible vertex aggregates is assumed to have some particular structure (*e.g.*, hierarchies of vertices and sets of intervals).

Section 5 discusses the outcomes of this new compression scheme and provides some research perspectives, notably to propose in the future tractable approximation algorithms for the lossy compression of large-scale temporal graphs.

2. Starting Point: The Lossless Graph Compression Problem

The starting point to build our compression scheme is a well-known combinatorial problem: Find the quotient set of the structural equivalence relation applying to the vertices of a graph. As the resulting equivalence classes form a partition of the vertex set by grouping together vertices with an identical (first-order) structure – that is with identical neighbourhoods – one can exploit such classes to compress the graph representation, as illustrated in Figure 1. Structural equivalence can thus be used for the lossless compression of static graphs, and we later list the improvements one needs in order to generalise this first simple scheme to the lossy compression of link streams.

2.1. Preliminary Notations

Given a set of vertices $\boldsymbol{V} = \{v_1, \ldots, v_n\}$, we mark:

- $\mathcal{P}(\boldsymbol{V})$ the set of all vertex subsets: $\mathcal{P}(\boldsymbol{V}) = \{V \subseteq \boldsymbol{V}\}$;
- $\mathfrak{P}(\boldsymbol{V})$ the set of all vertex partitions:

$$\mathfrak{P}(\boldsymbol{V}) \;=\; \{\{V_1, \ldots, V_m\} \subseteq \mathcal{P}(\boldsymbol{V}) \;:\; \cup_i V_i = \boldsymbol{V} \;\wedge\; \forall i \neq j,\; V_i \cap V_j = \emptyset\};$$

- Given a vertex $v \in \boldsymbol{V}$ and a vertex partition $\mathcal{V} \in \mathfrak{P}(\boldsymbol{V})$, we mark $\mathcal{V}(v)$ the unique vertex subset in $\mathcal{V}$ that contains v.

More generally, this article uses a consistent system of capitalization and typefaces to properly formalise the compression problem and its solution space:

- Vertices are designated by lowercase letters: v, v', u, u';
- Vertex sets and vertex subsets by uppercase letters: $\boldsymbol{V}$, V, V';
- Vertex partitions and sets of vertex subsets by calligraphic letters: $\mathcal{V}$, $\mathcal{V}'$, $\mathcal{P}(\boldsymbol{V})$, $\mathcal{H}(\boldsymbol{V})$, $\mathcal{I}(\boldsymbol{V})$;
- Sets of vertex partitions by Gothic letters: $\mathfrak{P}(\boldsymbol{V})$, $\mathfrak{H}(\boldsymbol{V})$, $\mathfrak{I}(\boldsymbol{V})$.

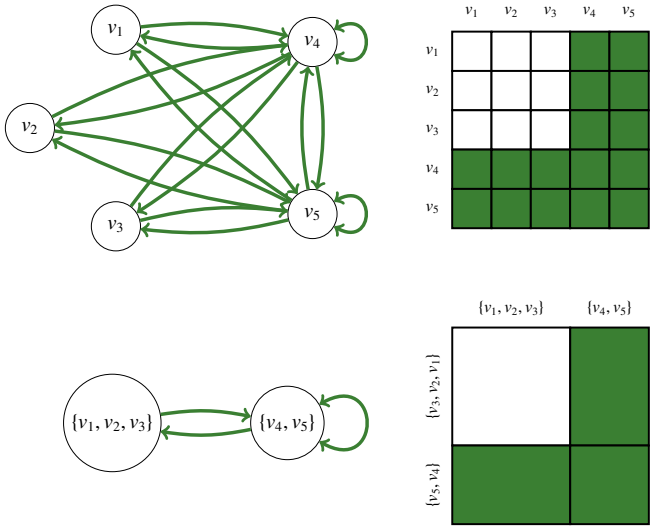


Figure 1: Lossless compression of a 5-vertex, 16-edge graph (above) into a 2-vertex, 3-edge graph (below). The *adjacency-list* representation is given on the left and the *adjacency-matrix* representation on the right.

2.2. The Lossless GCP

To begin with, we consider a simple case: Directed static graphs, with possible self-loops on the vertices.

Definition 1 (Directed Graph)**.**
A directed graph $G = (\boldsymbol{V}, \boldsymbol{E})$ *is characterised by:*

- *A set of* vertices $\boldsymbol{V}$*;*
- *A set of* directed edges $\boldsymbol{E} \subseteq \boldsymbol{V}\times\boldsymbol{V}$.

For all vertex $v \in \boldsymbol{V}$*, we respectively mark* $N_{in}(v) = \{v' \in \boldsymbol{V} : (v', v) \in \boldsymbol{E}\}$ *and* $N_{out}(v) = \{v' \in \boldsymbol{V} : (v, v') \in \boldsymbol{E}\}$ *the* in-coming *and the* out-going neighbourhoods *of* v.

The upper part of Figure 1 gives an example of directed graph made of $|\boldsymbol{V}| = 5$ vertices and $|\boldsymbol{E}| = 16$ edges. It is represented in the form of *adjacency lists* (on the left), where each edge is represented as an arrow going from a source vertex $v \in \boldsymbol{V}$ to a target vertex $v' \in \boldsymbol{V}$, as well as in the form of an *adjacency matrix* (on the right), where edges are represented within a binary matrix of size $|\boldsymbol{V}| \times |\boldsymbol{V}|$.

The combinatorial problem we now formalise builds on the classical relation of *structural equivalence* applying to the vertex set of a graph [16].

Definition 2 (Structural Equivalence)**.**
The structural equivalence relation $\sim \subseteq \boldsymbol{V}^2$ *is defined on directed graphs by the equality of neighbourhoods: Two vertices* $(v, v') \in \boldsymbol{V}^2$ *are* structurally equivalent *if and only if they are connected to the same vertices. Formally:*

$$v \sim v' \quad \Leftrightarrow \quad N_{in}(v) = N_{in}(v') \quad \textit{and} \quad N_{out}(v) = N_{out}(v').$$

A vertex subset $V \in \mathcal{P}(\boldsymbol{V})$ *is* structurally consistent *if and only if all its vertices are structurally equivalent with each others, and a vertex partition* $\mathcal{V} \in \mathfrak{P}(\boldsymbol{V})$ *is* structurally consistent *if and only if all its vertex subsets are structurally consistent. We respectively mark* $\widetilde{\mathcal{P}}(\boldsymbol{V})$ *and* $\widetilde{\mathfrak{P}}(\boldsymbol{V})$ *the sets of structurally-consistent vertex subsets and vertex partitions:*

$$V \in \widetilde{\mathcal{P}}(\boldsymbol{V}) \quad \Leftrightarrow \quad \forall (v, v') \in V^2, \quad v \sim v'.$$
$$\mathcal{V} \in \widetilde{\mathfrak{P}}(\boldsymbol{V}) \quad \Leftrightarrow \quad \forall V \in \mathcal{V}, \quad V \in \widetilde{\mathcal{P}}(\boldsymbol{V}).$$

The lower part of Figure 1 uses the fact that $v_1 \sim v_2 \sim v_3$ and that $v_4 \sim v_5$ to define a structurally-consistent vertex partition $\mathcal{V} = \{V_1, V_2\}$ made of two structurally-consistent vertex subsets $V_1 = \{v_1, v_2, v_3\}$ and $V_2 = \{v_4, v_5\}$.

Because all vertices belonging to a structurally-consistent vertex subset have the exact same neighbourhoods, one can use this structural redundancy to simplify the graph representation. Such a compression first consists in aggregating all vertices in structurally-consistent subsets to form *compressed vertices*, then in aggregating all edges between couples of structurally-consistent subsets to form *compressed edges*. The resulting *compressed graph* provides a smaller, yet complete description of the initial one.

Definition 3 (Compressed Directed Graph)**.**
Given a directed graph $G = (\boldsymbol{V}, \boldsymbol{E})$ and a structurally-consistent vertex partition $\mathcal{V} \in \widetilde{\mathfrak{P}}(\boldsymbol{V})$, the compressed directed graph $\mathcal{V}(G) = (\mathcal{V}, \mathcal{E})$ *is the graph such that:*

- $\mathcal{V}$ *is the set of* (compressed) vertices*;*
- $\mathcal{E} \subseteq \mathcal{V}\times\mathcal{V}$ *is the set of* (compressed) directed edges *such that:*

$$\forall (V, V') \in \mathcal{V}\times\mathcal{V}, \quad (V, V') \in \mathcal{E} \quad \begin{array}{ll} \Leftrightarrow & \forall v \in V,\ \forall v' \in V',\ (v, v') \in \boldsymbol{E} \\ \Leftrightarrow & \exists v \in V,\ \exists v' \in V',\ (v, v') \in \boldsymbol{E}. \end{array}$$

Note that both conditions are equivalent since V and V' are structurally consistent.

The lower part of Figure 1 shows the effect of such a compression on the graph's representations. Regarding adjacency lists, no more than one compressed edge is encoded between two given compressed vertices. Regarding the adjacency matrix, cells are merged into "rectangular tiles" containing only one binary value for each couple of compressed vertices.

These definitions lead to a well-known combinatorial problem that we call here the *Lossless Graph Compression Problem* (Lossless GCP). It simply consists in finding the quotient set of the structural equivalence relation, that is the smallest structurally-consistent partition of $\boldsymbol{V}$.

Definition 4 (The Lossless Graph Compression Problem)**.**
Given a directed graph $G = (\boldsymbol{V}, \boldsymbol{E})$, find a structurally-consistent vertex partition $\mathcal{V}^* \in \widetilde{\mathfrak{P}}(\boldsymbol{V})$ with minimal size $|\mathcal{V}^*|$:

$$\mathcal{V}^* \ = \ \underset{\mathcal{V}\in\widetilde{\mathfrak{P}}(\boldsymbol{V})}{\arg\min}\ |\mathcal{V}|.$$

In Figure 1, the represented structurally-consistent vertex partition is the smallest: One cannot find such another partition that contains fewer vertex subsets. This is hence the most optimal lossless compression of the graph.

2.3. Related Problems

Note that *structural* equivalence is the stricter form of vertex equivalence one might consider for graph analysis [20, 21]. Yet, other equivalence relations are traditionally used in the literature in social sciences for the detection of other kinds of structural patterns, such as *automorphic* equivalence (two vertices are equivalent if there is an isomorphic graph such that these vertices are interchanged) and *regular* equivalence [13] (two vertices are equivalent if they are equally related to other equivalent classes). Because these two latter equivalence relations are less strict, they induce smaller vertex partitions with bigger classes. But more importantly, and contrary to structural equivalence, the resulting compression scheme is not reversible in the sense that one cannot find back the initial graph from the equivalent classes and their compressed edges.

Structural equivalence, and so the GCP, is also related to *community detection* [22], also known as *graph clustering*, that is a classical problem for graph analysis which consists in finding groups of vertices that are strongly connected with each others while being loosely connected to other groups. However, dense and isolated clusters are only particular examples of structurally-consistent classes. They correspond to dense diagonal blocks in the adjacency matrix. The notion of structural equivalence is more generally interested in groups of vertices with similar relational

patterns, that is in any block of equal-density within the adjacency matrix (not necessarily dense and not necessarily on the diagonal, as in other work focusing on *block compression* [6, 2, 3]). Hence, the GCP is more strongly related to the family of *edge compression techniques* [8] such as *modular decomposition*[1] [3], *matching neighbours*, and *power graph analysis* [9]. In the latter, one is searching for groups of vertices that have similar relation patterns of any sort. Because it is more generally interested in the compression of equal-density blocks, the GCP can lastly be seen as a strict instance of *block modelling* [16, 20, 13, 15], another classical method of network analysis that relies on structural equivalence to discover roles and positions in social networks.

2.4. Possible Generalisations

This first formulation of the GCP is restricted to static simple graphs. Moreover, it only allows lossless compression, that is compression of vertices with *identical* neighbourhoods, which is a quite stringent and unrealistic condition for empirical research. In what follows, we list the requirements to formulate a more general and more flexible optimisation problem allowing for the lossy compression of temporal graphs.

From simple graphs to multigraphs (see 3.1). This first version of the GCP is restricted to *simple graphs* (no more than one edge between two given vertices). Yet, it is easily generalisable to *multigraphs* (multiple edges are allowed between two given vertices). Such a generalisation has two advantages. First, multigraphs are strictly more general than simple graphs since simple graphs can be considered as a particular cases of multigraphs. Second, multigraphs are more consistent with the lossy compression scheme later presented since the end result of lossy compression is not necessarily a simple graph (as edges are aggregated into multiedges during compression).

From lossless to lossy compression (see 3.2). This first version of the GCP is *lossless* in the sense that the result of compression contains all the information that is required to errorlessly build back the initial graph. However, such a lossless compression – relying on *exact* equivalence – is quite inefficient in the case of real graphs within which *identical* neighbourhoods are quite unlikely. One hence needs a measure of *information loss* to allow for a more flexible compression scheme.

From vertex to edge partitions (see 3.3). This first version of the GCP consists in finding an interesting *vertex partition* to compress the graph, thus inducing a partition of its edges. This relates to classical approaches such as *modular decomposition* where subsets of vertices (modules) that have similar neighbourhoods are exploited to compressed the graph's structure. However, this can be generalised to the direct search for *edge partitions*, that is the search for interesting edge subsets that do not all necessarily rest on similar vertex subsets. This relates to less known approaches such as *power-graph decomposition* that allows for a more subtle analysis of the graph's structure.

Adding constraints to the set of feasible vertex subsets (see 3.4). In this first version of the GCP, one considers any possible vertex subset as a potential candidate for compression, thus leading to an *unconstrained* compression scheme. However, in order to represent and to preserve *additional constraints* that might apply on the vertex structure, one might want to only consider "feasible" vertex subsets when searching for an optimal partition. This requires to integrate such additional constraints within the compression scheme.

From static graphs to link streams (see 3.5). Our last generalisation step consists in integrating a temporal dimension within the optimisation problem in order to deal with the compression of *link streams*. The structural equivalence relation hence needs to be redefined with respect to this additional dimension and equivalent classes will then be only valid on given time intervals. In this context, one is hence searching for aggregates that partition the Cartesian product of the vertex set and of the temporal dimension.

[1] Not to be confused with modularity-based clustering, which is a form of community detection.

3. Generalisation: From Lossless Static Graphs to Lossy Mutlistreams

3.1. From Graphs to Multigraphs

Most approaches in the domain of graph theory focus on the analysis of *simple graphs*, that is graphs for which at most one edge is allowed between two vertices, thus represented as binary adjacency matrices. This is also the case when it comes to the field of graph compression (see for example [13, 4, 5, 6, 8]). Yet, in the scope of this article, we aim at the compression of *multigraphs*, that is graphs for which multiple edges are allowed between two vertices, thus represented as integer adjacency matrices. As simple graphs are special cases of multigraphs, the resulting approach is necessarily more general.

In some articles on graph compression, the generalisation to multigraphs would be quite straightforward as the result of compression – that is the compressed graph – already is, in fact, a multigraph (see for example [6]). Even if not explicitly formalised, research perspectives in that direction are sometimes provided [9]. Yet, other approaches natively deals with multigraph compression by directly taking into account, within the compression scheme, the presence of multiple edges [2, 3]. Statistical methods for *variable co-clustering* also offers compression frameworks that are designed for numerical (non-binary) matrices [23, 15, 14]. This is the approach we choose here by directly working with multigraphs.

Definition 5 (Directed Multigraph)**.**
A directed multigraph $MG = (\boldsymbol{V}, e)$ *is characterised by:*

- *A set of* vertices $\boldsymbol{V}$*;*
- *A multiset of* directed edges $(\boldsymbol{V}\times\boldsymbol{V}, e)$
  where $e : \boldsymbol{V}\times\boldsymbol{V} \to \mathbb{N}$ *is the* edge function*, that is the multiplicity function counting the number of edges* $e(v, v') \in \mathbb{N}$ *going from a given source vertex* $v \in \boldsymbol{V}$ *to a given target vertex* $v' \in \boldsymbol{V}$*.*

  We also define the additive extension of the edge function on couples of vertex subsets:

$$e : \mathcal{P}(\boldsymbol{V})\times\mathcal{P}(\boldsymbol{V}) \to \mathbb{N} \quad \textit{such that} \quad e(V, V') = \sum_{(v,v')\in V\times V'} e(v, v').$$

  It simply counts the number of edges going from any vertex of a given source subset $V \in \mathcal{P}(\boldsymbol{V})$ *to any vertex of a given target subset* $V' \in \mathcal{P}(\boldsymbol{V})$*. In particular,* $e(\boldsymbol{V}, \boldsymbol{V})$ *is the total number of edges in the multigraph,* $e(\boldsymbol{V}, v)$ *is the in-coming degree of* v *and* $e(v, \boldsymbol{V})$ *its out-going degree.*

The upper part of Figure 2 gives an example of directed multigraph made of $|\boldsymbol{V}| = 5$ vertices, that is $|\boldsymbol{V}\times\boldsymbol{V}| = 25$ multiedges, and $e(\boldsymbol{V}, \boldsymbol{V}) = 40$ edges distributed within $\boldsymbol{V}\times\boldsymbol{V}$. It is represented in the form of *adjacency lists* (on the left), where each multiedge is represented as an arrow which width is proportional to the number of edges $e(v, v')$ going from a source vertex v to a target vertex v', as well as in the form of an *adjacency matrix* (on the right), where the edge function is represented as an integer matrix of size $|\boldsymbol{V}|\times|\boldsymbol{V}|$.

Structural equivalence could then be generalised to multigraphs in order to define, as done previously for simple graphs, a Lossless Multigraph Compression Problem (MGCP). In few words, the structural equivalence relation would be defined on directed multigraphs by the equality of the edge function: Two vertices are hence structurally equivalent if and only if they are each connected the same number of times to the different graph's vertices. However, as we are interested in this article in lossy compression, we directly consider an alternative version of the MGCP that relies on a stochastic relaxation of the structural equivalence relation and on an appropriate measure of information loss.

3.2. From Lossless to Lossy Compression

Information-theoretic compression first requires a stochastic model of the data, that is a model of the multigraph to be compressed. The measure of information loss that we present in this subsection has been previously introduced for the aggregation of geographical data [24] and for the summarisation of execution traces of distributed systems [19]. The first contribution of this article in this regard is the application of this measure to graph compression. Moreover, the underlying stochastic models were not made explicit in previous work. Our second contribution is hence the

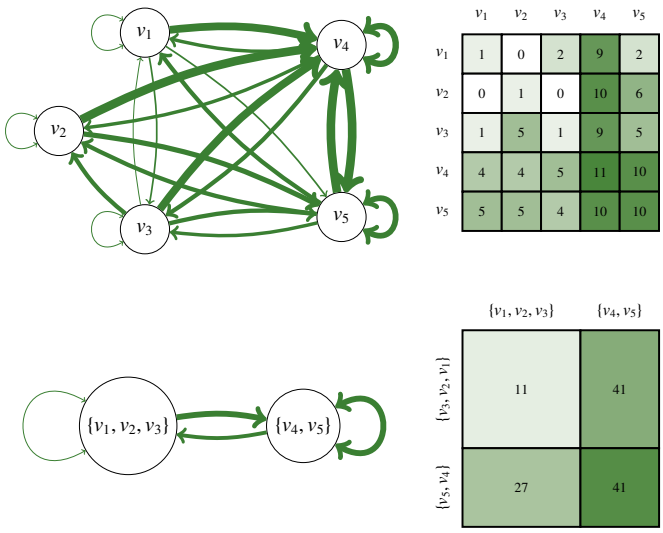


Figure 2: Lossy compression of a 5-vertex, 40-edge multigraph (above) into a 2-vertex, 40-edge multigraph (below). In the *adjacency-list representation* (on the left), the width of arrows is proportional to the edge function, that is to the number of edges going from a source vertex to a target vertex.

thorough formalisation of the graph model we use, in order to properly justify and interpret the resulting measure. In few words, we model a multigraph as a set of edges that are stochastically distributed within the two-dimensional set of multiedges $\boldsymbol{V}\times\boldsymbol{V}$: Each edge has a particular location within this space, characterised by its source vertex and its target vertex. The edge function $e : \boldsymbol{V}\times\boldsymbol{V} \to \mathbb{N}$ hence characterises the *empirical distribution* of the edges within the multigraph, thus allowing to model the data as a discrete random variable (X, X') taking on $\boldsymbol{V}\times\boldsymbol{V}$.

Definition 6 (Observed Variable)**.**
The observed variable $(X, X') \in \boldsymbol{V}\times\boldsymbol{V}$ *associated with a multigraph* $MG = (\boldsymbol{V}, e)$ *is a couple of discrete random variables having the empirical distribution of edges in MG:*

$$\Pr((X, X') = (v, v')) \quad = \quad \frac{e(v, v')}{e(\boldsymbol{V}, \boldsymbol{V})} \quad \stackrel{\text{def}}{=} \quad p_{(X,X')}(v, v').$$

In other terms, $p_{(X,X')}(v, v')$ represents the probability that, if one chooses an edge at random among the $e(\boldsymbol{V}, \boldsymbol{V})$ edges of the multigraph, it will go from the source vertex v to the target vertex v'. For example, matrix (i) in Figure 3 represents the distribution of the observed variable associated with the multigraph of Figure 2.

We then define the edge distribution of a multigraph that have been compressed according to a vertex partition $\mathcal{V} \in \mathfrak{P}(\boldsymbol{V})$ by defining a second random variable taking on the multiedge partition $\mathcal{V}\times\mathcal{V} \in \mathfrak{P}(\boldsymbol{V}\times\boldsymbol{V})$.

Definition 7 (Compressed Variable)**.**
The compressed variable $\mathcal{V}\times\mathcal{V}(X, X') \in \mathcal{V}\times\mathcal{V}$ *associated with an observed variable* $(X, X') \in \boldsymbol{V}\times\boldsymbol{V}$ *and a vertex partition* $\mathcal{V} \in \mathfrak{P}(\boldsymbol{V})$ *is the unique couple of vertex subsets in* $\mathcal{V}\times\mathcal{V}$ *that contains* (X, X')*:*

$$\mathcal{V}\times\mathcal{V}(X, X') \quad = \quad (\mathcal{V}(X), \mathcal{V}(X')) \quad \in \quad \mathcal{V}\times\mathcal{V}.$$

It hence has the following distribution[2]*:*

$$\Pr(\mathcal{V}\times\mathcal{V}(X, X') = (V, V')) \quad = \quad \frac{e(V, V')}{e(\boldsymbol{V}, \boldsymbol{V})} \quad \stackrel{\text{def}}{=} \quad p_{\mathcal{V}\times\mathcal{V}(X,X')}(V, V').$$

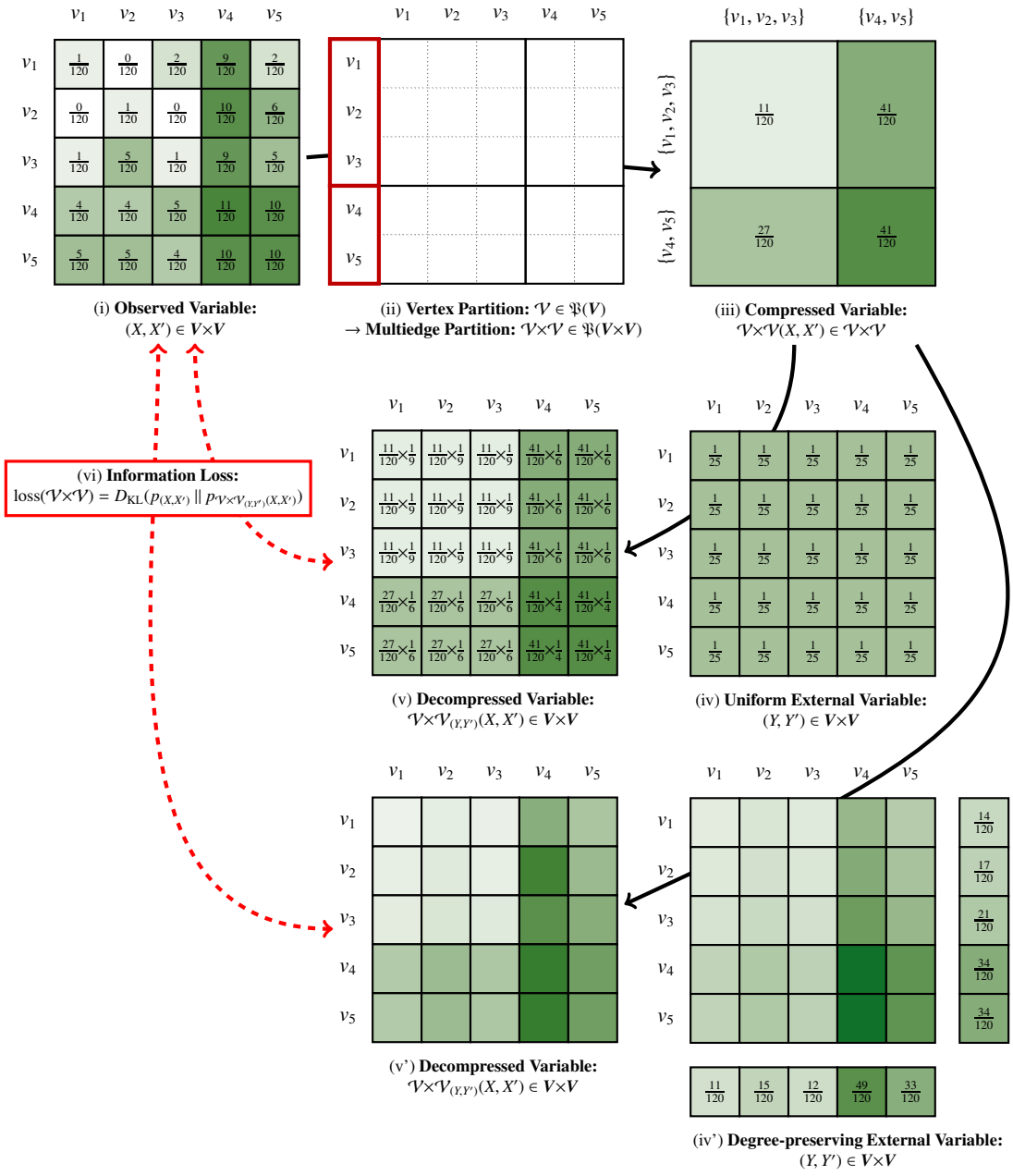


Figure 3: Lossy compression consists in (i) modelling the multigraph as a random variable (X, X') having the empirical distribution of edges in $V\times V$, (ii) choosing a vertex partition $\mathcal{V}$, and so a multiedge partition $\mathcal{V}\times\mathcal{V}$ that is used to compress (X, X'), (iii) computing the distribution of the resulting compressed variable $\mathcal{V}\times\mathcal{V}(X, X')$ by applying partition $\mathcal{V}\times\mathcal{V}$ onto (X, X'), (iv) taking an external variable (Y, Y') (for example (iv) uniformly distributed or (iv') preserving the degree profile of vertices) to project back the distribution of $\mathcal{V}\times\mathcal{V}(X, X')$ into $V\times V$, (v) computing the distribution of the resulting decompressed variable $\mathcal{V}\times\mathcal{V}_{(Y,Y')}(X, X')$ by first choosing (X, X'), then choosing (Y, Y') conditioned on $\mathcal{V}\times\mathcal{V}(Y, Y') = \mathcal{V}\times\mathcal{V}(X, X')$, and (vi) comparing the distribution of (X, X') with the distribution of $\mathcal{V}\times\mathcal{V}_{(Y,Y')}(X, X')$ by using information-theoretic measures such as the entropy of (X, X') relative to $\mathcal{V}\times\mathcal{V}_{(Y,Y')}(X, X')$.

In other terms, $p_{\mathcal{V}\times\mathcal{V}(X,X')}(V,V')$ represents the probability that, if one chooses an edge at random among the $e(\boldsymbol{V},\boldsymbol{V})$ edges of the multigraph, it will go from a vertex of the source subset V to a vertex of the target subset V'. For example, matrix (ii) in Figure 3 represents the multiedge partition $\mathcal{V}\times\mathcal{V}$ induced by the vertex partition $\mathcal{V} = \{\{v_1, v_2, v_3\}, \{v_4, v_5\}\}$ and matrix (iii) then represents the distribution of the resulting compressed variable $\mathcal{V}\times\mathcal{V}(X,X')$.

In order to quantify the information that has been lost during this compression step, we propose to compare the information that is contained in the initial multigraph (that is in the observed variable (X,X')) with the information that is contained in the compressed multigraph (that is in the compressed variable $\mathcal{V}\times\mathcal{V}(X,X')$). To do so, we project back the compressed distribution onto the initial value space $\boldsymbol{V}\times\boldsymbol{V}$ by defining a third random variable, the *external variable* $(Y,Y') \in \boldsymbol{V}\times\boldsymbol{V}$, that models additional information that one might have at his or her disposal when trying to decompress the multigraph. It is hence assumed to have a distribution that is somehow "informative" of the initial distribution. It then induces a fourth variable, the *decompressed variable* $\mathcal{V}\times\mathcal{V}_{(Y,Y')}(X,X')$, that models an approximation of the initial multigraph inferred from the combined knowledge of the compressed variable $\mathcal{V}\times\mathcal{V}(X,X')$ and of the external variable (Y,Y'). This last variable is hence defined according to the distribution of the external variable within the multiedge subsets.

Definition 8 (Decompressed Variable)**.**
The decompressed variable $\mathcal{V}\times\mathcal{V}_{(Y,Y')}(X,X') \in \boldsymbol{V}\times\boldsymbol{V}$ *associated with an observed variable* $(X,X') \in \boldsymbol{V}\times\boldsymbol{V}$*, a vertex partition* $\mathcal{V} \in \mathfrak{P}(\boldsymbol{V})$*, and an external variable* $(Y,Y') \in \boldsymbol{V}\times\boldsymbol{V}$*, is the result of this external variable* (Y,Y') *conditioned by its compression* $\mathcal{V}\times\mathcal{V}(Y,Y')$ *being equal to the result of the compressed variable* $\mathcal{V}\times\mathcal{V}(X,X')$*:*

$$\mathcal{V}\times\mathcal{V}_{(Y,Y')}(X,X') \quad = \quad (Y,Y') \,|\, \mathcal{V}\times\mathcal{V}(Y,Y') = \mathcal{V}\times\mathcal{V}(X,X') \quad \in \quad \boldsymbol{V}\times\boldsymbol{V}.$$

It hence has the following distribution[3]*:*

$$\Pr(\mathcal{V}\times\mathcal{V}_{(Y,Y')}(X,X') = (v,v')) \quad = \quad \frac{e(\mathcal{V}\times\mathcal{V}(v,v'))}{e(\boldsymbol{V},\boldsymbol{V})} \, \frac{p_{(Y,Y')}(v,v')}{p_{\mathcal{V}\times\mathcal{V}(Y,Y')}(\mathcal{V}\times\mathcal{V}(v,v'))} \quad \stackrel{\text{def}}{=} \quad p_{\mathcal{V}\times\mathcal{V}_{(Y,Y')}(X,X')}(v,v').$$

In other terms, $p_{\mathcal{V}\times\mathcal{V}_{(Y,Y')}(X,X')}(v,v')$ represents the probability that, if one (i) chooses an edge (u,u') at random among the $e(\boldsymbol{V},\boldsymbol{V})$ edges of the multigraph, (ii) considers its compressed multiedge subset $\mathcal{V}\times\mathcal{V}(u,u') = (V,V') \in \mathcal{V}\times\mathcal{V}$, and (iii) chooses a source vertex within V and a target vertex within V' according to the distribution $p_{(Y,Y')}$ of the external variable within this multiedge subset, then one will result with an edge going from the source vertex v to the target vertex v'. For example, matrix (iv) in Figure 3 represents a uniformly-distributed external variable $(Y,Y') \in \boldsymbol{V}\times\boldsymbol{V}$ that is used to decompress $\mathcal{V}\times\mathcal{V}(X,X')$ (see *Blind Decompression* below). Matrix (iv') represents another such external

[2]By applying the law of total probability:

$$\begin{aligned}\Pr(\mathcal{V}\times\mathcal{V}(X,X') = (V,V')) \;&=\; \sum_{(v,v')\in\boldsymbol{V}\times\boldsymbol{V}} \underbrace{\Pr(\mathcal{V}\times\mathcal{V}(X,X') = (V,V') \,|\, (X,X') = (v,v'))}_{=1 \text{ if } (v,v')\in V\times V', \text{ and 0 else}} \Pr((X,X') = (v,v')) \\ &=\; \sum_{(v,v')\in V\times V'} \Pr((X,X') = (v,v')) \;=\; \sum_{(v,v')\in V\times V'} \frac{e(v,v')}{e(\boldsymbol{V},\boldsymbol{V})} \;=\; \frac{e(V,V')}{e(\boldsymbol{V},\boldsymbol{V})}\end{aligned}$$

Note that, more generally, compression could be defined for any (possibly stochastic) function of the observed variable (X,X'), thus modelling what is sometimes called a "soft partitioning" of the vertices. The information-theoretic framework presented in this article, along with all the measures it contains, are straightforwardly generalisable to such setting. However, because it is often much easier to interpret the result of compression when it is based on "hard partitioning", especially in the case of vertex partitioning, we focus in this article on this simpler setting.

[3]By applying the law of total probability:

$$\begin{aligned}\Pr(\mathcal{V}\times\mathcal{V}_{(Y,Y')}(X,X') = (v,v')) \;&=\; \Pr((Y,Y') = (v,v') \,|\, \mathcal{V}\times\mathcal{V}(Y,Y') = \mathcal{V}\times\mathcal{V}(X,X')) \\ &=\; \sum_{(V,V')\in\mathcal{V}\times\mathcal{V}} \underbrace{\Pr((Y,Y') = (v,v') \,|\, \mathcal{V}\times\mathcal{V}(Y,Y') = (V,V'))}_{=0 \text{ if } (V,V')\neq\mathcal{V}\times\mathcal{V}(v,v')} \Pr(\mathcal{V}\times\mathcal{V}(X,X') = (V,V')) \\ &=\; \Pr((Y,Y') = (v,v') \,|\, \mathcal{V}\times\mathcal{V}(Y,Y') = \mathcal{V}\times\mathcal{V}(v,v')) \;\Pr(\mathcal{V}\times\mathcal{V}(X,X') = \mathcal{V}\times\mathcal{V}(v,v')) \\ &=\; \frac{\Pr((Y,Y') = (v,v'))}{\Pr(\mathcal{V}\times\mathcal{V}(Y,Y') = \mathcal{V}\times\mathcal{V}(v,v'))} \, \frac{e(\mathcal{V}\times\mathcal{V}(v,v'))}{e(\boldsymbol{V},\boldsymbol{V})}.\end{aligned}$$

variable (see *Degree-preserving Decompression* below). Matrices (v) and (v') then represent the distribution of the resulting uniformly decompressed variable $\mathcal{V}\times\mathcal{V}_{(Y,Y')}(X,X')$.

Blind Decompression. When the external variable (Y,Y') is uniformly distributed on $\boldsymbol{V}\times\boldsymbol{V}$, the decompression step is done without any additional information about the initial edge distribution:

$$p_{(Y,Y')}(v,v') \;=\; \frac{1}{|\boldsymbol{V}\times\boldsymbol{V}|} \quad\Rightarrow\quad p_{\mathcal{V}\times\mathcal{V}_{(Y,Y')}(X,X')}(v,v') \;=\; \frac{e(\mathcal{V}\times\mathcal{V}(v,v'))}{e(\boldsymbol{V},\boldsymbol{V})}\frac{1}{|\mathcal{V}\times\mathcal{V}(v,v')|}.$$

In this case, only the knowledge of the compressed variable hence is exploited. The decompressed variable is hence the result of a uniform trial among the multiedges contained in $\mathcal{V}\times\mathcal{V}(X,X')$, that is a "maximum-entropy sampling" guarantying that no additional information has been injected during decompression.

Reversible Decompression. To the contrary, when the external variable (Y,Y') has the same distribution than the observed variable (X,X'), then the decompression step is done with a full knowledge of the initial edge distribution:

$$p_{(Y,Y')}(v,v') \;=\; p_{(X,X')}(v,v') \quad\Rightarrow\quad \mathcal{V}\times\mathcal{V}_{(Y,Y')}(X,X')(v,v') \;=\; \frac{e(v,v')}{e(\boldsymbol{V},\boldsymbol{V})} \;=\; p_{(X,X')}(v,v').$$

In this case, the decompressed variable also has the same distribution than the observed variable, meaning that one fully restores the initial multigraph when decompressing.

Degree-preserving Decompression. An intermediary example of external information can be derived from the knowledge of the vertex degrees in the initial multigraph:

$$p_{(Y,Y')}(v,v') \;=\; p_X(v)\,p_{X'}(v') \;=\; \frac{e(v,\boldsymbol{V})}{e(\boldsymbol{V},\boldsymbol{V})}\frac{e(\boldsymbol{V},v')}{e(\boldsymbol{V},\boldsymbol{V})}$$

$$\Rightarrow\quad p_{\mathcal{V}\times\mathcal{V}_{(Y,Y')}(X,X')}(v,v') \;=\; \frac{e(\mathcal{V}\times\mathcal{V}(v,v'))}{e(\boldsymbol{V},\boldsymbol{V})}\frac{e(v,\boldsymbol{V})\,e(\boldsymbol{V},v')}{e(\mathcal{V}(v),\boldsymbol{V})\,e(\boldsymbol{V},\mathcal{V}(v'))}.$$

In this case, the decompression step takes into account the initial vertex degrees and the resulting multigraph hence has the same degree profile than the initial one. The corresponding generative model is hence similar to the one of a *configuration model* [25]: Multigraphs are sampled according to the compressed variable, while also preserving the initial degree profile.

Now that we have defined compression and decompression as sequential operations on stochastic variables, we exploit a classical measure of information theory to quantify the information that is lost during such a process. Intuitively, it consists in comparing the initial edge distribution (the one of the observed variable (X,X')) with the approximated edge distribution (the one of the decompressed variable $\mathcal{V}\times\mathcal{V}_{(Y,Y')}(X,X')$). In this article, we propose to do so by using the *relative entropy* of these two distributions – also known as the *Kullback-Leibler divergence* [26, 27] – as it is the most canonical measure of dissimilarity provided by information theory to compare an approximated probability distribution to a real one.

Definition 9 (Information Loss)**.**
The information loss *induced by a vertex partition $\mathcal{V} \in \mathfrak{P}(\boldsymbol{V})$ on an observed variable $(X, X') \in \boldsymbol{V}\times\boldsymbol{V}$, and according to an external variable $(Y, Y') \in \boldsymbol{V}\times\boldsymbol{V}$, is given by the entropy of (X, X') relative to $\mathcal{V}\times\mathcal{V}_{(Y,Y')}(X, X')$:*

$$\begin{aligned}\mathrm{loss}(\mathcal{V}\times\mathcal{V}) &= \sum_{(v,v')\in\boldsymbol{V}\times\boldsymbol{V}} p_{(X,X')}(v,v')\log_2\left(\frac{p_{(X,X')}(v,v')}{p_{\mathcal{V}\times\mathcal{V}_{(Y,Y')}(X,X')}(v,v')}\right)\\ &= \sum_{(v,v')\in\boldsymbol{V}\times\boldsymbol{V}} \frac{e(v,v')}{e(\boldsymbol{V},\boldsymbol{V})}\log_2\left(\frac{e(v,v')}{e(\mathcal{V}\times\mathcal{V}(v,v'))}\Big/\frac{p_{(Y,Y')}(v,v')}{p_{\mathcal{V}\times\mathcal{V}(Y,Y')}(\mathcal{V}\times\mathcal{V}(v,v'))}\right)\end{aligned}$$

Note that information loss is additively decomposable*. It can be expressed as a sum of information losses defined at the subset level instead of at the partition level:*

$$\mathrm{loss}(\mathcal{V}\times\mathcal{V}) = \sum_{(V,V')\in\mathcal{V}\times\mathcal{V}} \mathrm{loss}(V,V') \quad \textit{with}\ \mathrm{loss}(V,V') = \sum_{(v,v')\in V\times V'} \frac{e(v,v')}{e(\boldsymbol{V},\boldsymbol{V})}\log_2\left(\frac{e(v,v')}{e(V,V')}\Big/\frac{p_{(Y,Y')}(v,v')}{p_{\mathcal{V}\times\mathcal{V}(Y,Y')}(V,V')}\right).$$

Intuitively, this measure considers the following reconstruction task: Imagine that all the edges of a multigraph have been "detached" from their vertices and put into a bag. An observer is now taking one edge out of the bag and tries to guess its initial location, that is its source and target vertices. We then compare two peoples trying to do so: One having a perfect knowledge of the distribution $p_{(X,X')}$ of the edges in the initial multigraph (*e.g.*, matrix (i) in Figure 3); The second only having an approximation $p_{\mathcal{V}\times\mathcal{V}_{(Y,Y')}(X,X')}$ of this distribution, obtained through the compression, then the decompression of the initial distribution (*e.g.*, matrices (v) and (v') in Figure 3). Relative entropy then measures the average quantity of *additional* information (in bits per edge) that the second observer needs in order to make a guess that is as informed as the guess of the first observer. In other words, relative entropy quantifies the information that has been lost during compression, that is no longer contained in the compressed graph, and that cannot be retrieved from the knowledge of the external variable.

Blind Decompression. When the external variable is uniformly distributed on $\boldsymbol{V}\times\boldsymbol{V}$, relative entropy simply quantifies the information that has been lost during compression, without the help of any additional information:

$$p_{(Y,Y')}(v,v') = \frac{1}{|\boldsymbol{V}\times\boldsymbol{V}|} \quad\Rightarrow\quad \mathrm{loss}(V,V') = \sum_{vv\in V\times V'} \frac{e(v,v')}{e(\boldsymbol{V},\boldsymbol{V})}\log_2\left(\frac{e(v,v')}{e(V,V')}|V\times V'|\right).$$

Reversible Decompression. When the external variable (Y, Y') has the same distribution than the observed variable (X, X'), compression then induces no information loss – whatever the chosen vertex partition $\mathcal{V}$ – since all the information required to reconstruct the initial multigraph is reinjected during the decompression step:

$$p_{(Y,Y')}(v,v') = p_{(X,X')}(v,v') \quad\Rightarrow\quad \mathrm{loss}(V,V') = 0.$$

Degree-preserving Decompression. In this intermediary context, relative entropy quantifies the information that has been lost during compression, and that cannot be retrieved from the additional knowledge of the vertex degrees in the initial multigraph:

$$p_{(Y,Y')}(v,v') = p_X(v)\,p_{X'}(v') = \frac{e(v,\boldsymbol{V})}{e(\boldsymbol{V},\boldsymbol{V})}\frac{e(\boldsymbol{V},v')}{e(\boldsymbol{V},\boldsymbol{V})}$$
$$\Rightarrow\quad \mathrm{loss}(V,V') = \sum_{(v,v')\in V\times V'} \frac{e(v,v')}{e(\boldsymbol{V},\boldsymbol{V})}\log_2\left(\frac{e(v,v')}{e(V,V')}\Big/\frac{e(v,\boldsymbol{V})\,e(\boldsymbol{V},v')}{e(V,\boldsymbol{V})\,e(\boldsymbol{V},V')}\right).$$

Related Measures. Relative entropy is one among many measures that can be found in the literature to quantify information loss in graph compression. Given an initial multigraph and a decompressed one, which is described by the approximated edge distribution, any measure of weighted graph similarity may be relevant to quantify the impact of compression [1]: *E.g.*, the percentage of edges in common, the size of the maximum common subgraph or of the minimum common supergraph, the edit distance, that is the insertion and removal of vertices and edges needed to

go from one graph to another [28], or any measure aggregating the similarities of vertices from graph to graph (*e.g.*, Jaccard index, Pearson coefficient, cosine similarity on vertex neighbourhoods).

More sophisticated graph-theoretical measures go beyond the mere level of edges by taking into account paths within the two compared graphs [2]: "[T]he best path between any two nodes should be approximately equally good in the compressed graph as in the original graph, but the path does not have to be the same." More generally, query-based measures aim at quantifying the impact of compression on the results of goal-oriented queries regarding the graph structure: *E.g.*, queries about shortest paths, about degrees and adjacency, about centrality and community structures (see for example [29, 30, 31]). The expected difference between the results of queries on the initial graph and the results of queries on the decompressed graph thus provides a reconstruction error that serves as a goal-oriented information loss [6]. To some extent, we are interested in this article in adjacency-oriented queries, that is the most canonical ones, taking the perspective of the weighted adjacency matrices of the two compared graphs: *What is the weight of the multiedge located between two given vertices of the initial multigraph?* Relative entropy measures the expected error when answering this query from the only knowledge of the compressed graph.

More generally, this perspective is related to the density profile of edges within the graph. Hence, density-based measures [5] seems more relevant than other traditional connectivity measures: *E.g.*, the Euclidean distance or the mean squared error between the two density matrices [2, 3], the average variance within matrix blocks [13], and many other measures inherited from traditional block modelling methods [20]. Note that, when it comes to the latter, the stochastic model underlying our compression framework is similar, but not equivalent to the one of block modelling. The compressed matrix describes in our case the parameters of a multinomial distribution from which the graph's edges are sampled, whereas it describes in the case of block modelling the parameters of $|\boldsymbol{V}\times\boldsymbol{V}|$ independent Bernouilli distributions. In other words, our model gives the probability that an edge – taken at random – is located between two given vertices, and not the probability that an edge exists between two given vertices (see for example block model compression in [6]).

Because of this particular stochastic model, we chose in this article an information-theoretic approach to measure information loss. This allows to derive a measure that is clearly in line with the defined model and which can be easily interpreted within the realm of information theory. Among tools provided by this theory, other approaches use the principles of *minimum description length* [4, 6] to compress a graph using the density-based model in an optimal fashion. In the same line of thinking, traditional tools for Bayesian inference propose to interpret the compressed graph as a generative model and the initial graph as observed data, then computes the likelihood of the data given the model as a measure of information loss [15]. A similar Bayesian interpretation of relative entropy could be given, as it measures the difference of likelihood between two generative models of the multigraph: One corresponding to $e(\boldsymbol{V},\boldsymbol{V})$ independent trials with the empirical distribution of the multigraph's edges; The second corresponding to $e(\boldsymbol{V},\boldsymbol{V})$ independent trials with the distribution obtained through compression, and then decompression. Similarly, *co-clustering* [23] interprets the graph's adjacency matrix as the joint probability distribution of two random variables, then finds two vertex partitions that minimise the loss in mutual information between these two variables [27] from the initial graph to the compressed graph. This is shown to be equivalent to minimising the relative entropy between the initial distribution and a decompressed distribution that preserves the marginal values. It is hence equivalent to our measure of information loss in the particular case of a decompression scheme that takes into account additional information regarding the vertex degrees.

3.3. From Vertex to Edge Partitions

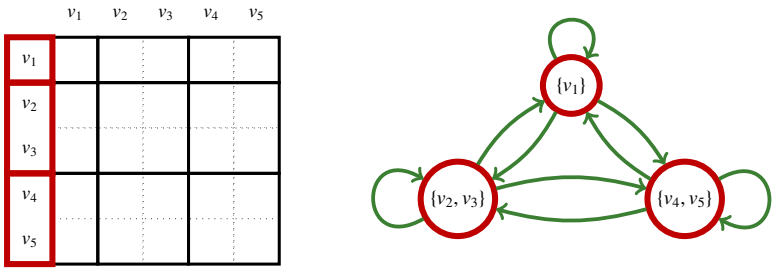
By providing a measure of information loss, previous subsection focuses on the *objective function* of the GCP, that is on the quality measure to be optimised. We now focus on the *solution space* of this problem, that is the set of partitions that one actually consider for compression. The original GCP presented in Section 2 consists in using a vertex partition $\mathcal{V}\in\mathfrak{P}(\boldsymbol{V})$ to then determine a multiedge "grid" partition (see top-left matrix of Figure 4):

$$\mathcal{V}\times\mathcal{V} \quad = \quad \{V\times V' \;:\; V\in\mathcal{V} \;\wedge\; V'\in\mathcal{V}\} \quad \in \quad \mathfrak{P}(\boldsymbol{V}\times\boldsymbol{V}),$$

such that two multiedges (v,v') and (u,u') are in the same multiedge subset $V\times V'$ if and only if their source vertices are both in V and their target vertices are both in V':

$$\mathcal{V}\times\mathcal{V}(v,v') = \mathcal{V}\times\mathcal{V}(u,u') \qquad \Leftrightarrow \qquad \mathcal{V}(v)=\mathcal{V}(u) \quad \text{and} \quad \mathcal{V}(v')=\mathcal{V}(u').$$

Grid multiedge partition $\mathcal{V}\times\mathcal{V} \in \mathfrak{P}(\boldsymbol{V}\times\boldsymbol{V})$
that is the Cartesian product of a vertex partition $\mathcal{V} \in \mathfrak{P}(\boldsymbol{V})$



Cartesian multiedge partition $\mathcal{V}\mathcal{V} \in \mathfrak{P}^{\times}(\boldsymbol{V}\times\boldsymbol{V})$
consisting in Cartesian multiedge subsets $V\times V' \in \mathcal{P}(\boldsymbol{V}\times\boldsymbol{V})$

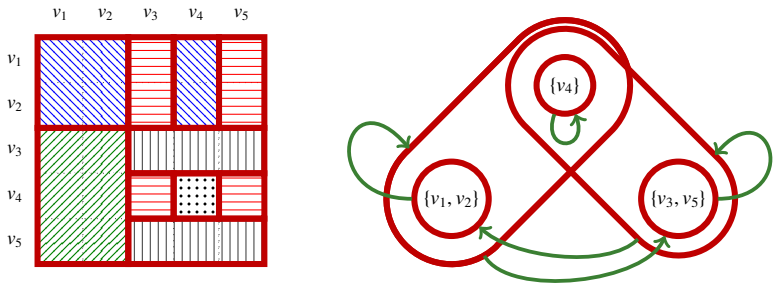


Figure 4: Two partitioning schemes that one might consider to define the solution space of the GCP. Multiedge subsets are represented with hatching when they are not "compact tiles".

In other terms, the induced two-dimensional partitioning of the multiedge set $\boldsymbol{V}\times\boldsymbol{V}$ is the Cartesian product of a one-dimensional partitioning of the vertex set $\boldsymbol{V}$. This first compression scheme is classically used in macroscopic graph models such as block models and community-based representations. One of the reason is that the result of compression can still be represented as a graph, which vertices are the subsets of the vertex partition and which multiedges are the subsets of the multiedge "grid" partition (see top-right graph of Figure 4)

However, this results in a quite constrained solution space for the optimisation problem: Only a small number of partitions of $\boldsymbol{V}\times\boldsymbol{V}$ are actually feasible. One might instead consider a less constrained solution space by allowing a larger number of multiedge partitions to be used for compression, by for example considering partitions of the multiedge set $\boldsymbol{V}\times\boldsymbol{V}$ that are made of Cartesian products of two vertex subsets $V\times V' \subseteq \mathcal{P}(\boldsymbol{V}\times\boldsymbol{V})$.

Definition 10 (Cartesian Multiedge Partitions)**.**
Given a vertex set $\boldsymbol{V}$*, the set of* Cartesian multiedge partitions $\mathfrak{P}^{\times}(\boldsymbol{V}\times\boldsymbol{V}) \subset \mathfrak{P}(\boldsymbol{V}\times\boldsymbol{V})$ *is the set of multiedge partitions that are made of Cartesian products of vertex subsets*[4]*:*

$$\begin{aligned}\mathfrak{P}^{\times}(\boldsymbol{V}\times\boldsymbol{V}) \quad = \quad & \{\{(V_1 \times V'_1), \ldots, (V_m \times V'_m)\} \subseteq \mathcal{P}(\boldsymbol{V})\times\mathcal{P}(\boldsymbol{V}) \\ & : \cup_i (V_i \times V'_i) = \boldsymbol{V}\times\boldsymbol{V} \ \wedge \ \forall i \neq j, \ (V_i \times V'_i) \cap (V_j \times V'_j) = \emptyset\}.\end{aligned}$$

Such Cartesian partitions of $\boldsymbol{V}\times\boldsymbol{V}$ hence contains "rectangular" multiedge subsets[5] (see bottom-left matrix of Figure 4). Hence, although the result of compression can no longer be represented as a graph, it can be represented as

[4]There is an abuse of notation in this definition when writing that "$V\times V' \in \mathcal{P}(\boldsymbol{V})\times\mathcal{P}(\boldsymbol{V})$". We should instead write that "$V\times V' \in \mathcal{P}(\boldsymbol{V}\times\boldsymbol{V})$ such that $(V, V') \in \mathcal{P}(\boldsymbol{V})\times\mathcal{P}(\boldsymbol{V})$", but we prefer the former for notation conciseness.

[5]Note that each of these multiedge subsets is indeed a rectangle in the adjacency matrix, modulo a reordering of its rows and columns. It might happen however that no reordering can make all multiedge subsets look rectangular *at the same time*.

a *directed hypergraph* – or as a *power graph* [8, 9] – that is a graph where edges can join couples of vertex subsets instead of couples of vertices (see bottom-right graph of Figure 4).

Related Work. Most classical approaches for graph compression are based on the discovery of *vertex partitions*. This is for example the case of most work on graph summarisation [5, 6], block modelling [13], community detection [22], and modular decomposition [3]. As previously said, one of the interests of such vertex partitioning is that the result of compression is still a graph (a set of compressed vertices connected by a set of compressed edges) that can hence be represented, analysed, and visualised with traditional tools [4, 2]. However, the number and diversity of feasible solutions – that is the set of vertex partitions – is quite limited when compated to the full space of matrix partitions.

As proposed above, some approaches hence focus on *edge partitions*, instead of vertex partitions, in order to provide more expressive compression schemes, but only on *particular* edge partitions, in order to preserve some fundamental structures of graph data. The most generic framework is formalised by *power graph analysis* [8, 9] and *Mondrian processes* [32], where edge subsets are only required to be the Cartesian product of vertex subsets. Such edge subsets can hence be represented as compressed edges between couples of compressed vertices. But since different edge subsets can lead to overlapping vertex subsets, the resulting compressed graph is no longer a graph, but an *hypergraph* (or a *power graph*). While still interpretable within the broad realm of graph theory, power graphs are more expressive [2] than classical approaches – such as community partitions – since a given vertex might belong to different similarity groups: It might be similar to a given group of vertices with respect to a given part of the graph, and similar to another group of vertices with respect to another part (see examples in [9]).

3.4. Adding Constraints to the Set of Feasible Vertex Subsets

A wide variety of approaches span from generic edge partitions to more constrained schemes that "impose restrictions on the nature of overlap" for example by "fixing the topology of connectivity between overlapping nodes" [9] such as in clique percolation, spin models, mixed-membership block models, latent attribute models, spectral clustering, and link communities. This aims at expressing particular topological properties within the compression scheme in order to produce meaningful graphs with respect to some particular analysis framework. "Frequent patterns possibly reflect some semantic structures of the domain and therefore are useful candidates for replacement" [1].

In the partitioning scheme hereabove presented, the Cartesian product of *any* two vertex subsets $(V, V') \in \mathcal{P}(\boldsymbol{V})\times\mathcal{P}(\boldsymbol{V})$ gives a feasible multiedge subset $V\times V' \in \mathcal{P}(\boldsymbol{V}\times\boldsymbol{V})$. In other terms, the solution space is only limited by this Cartesian principle and by the partitioning constraints (covering and non-overlapping subsets). However, one can also control the shape of feasible partitions by directly specifying the feasible vertex subsets that can be combined to build the "rectangular" multiedge subsets of the Cartesian partition. Such additional constraints might prove to be useful to incorporate domain knowledge about particular vertex structures within the compression scheme: By driving the process, feasibility constraints might indeed facilitate the interpretation of the compression's result according to the expert domain.

Definition 11 (Feasible Multiedge Partitions)**.**
Given a set of feasible vertex subsets $\hat{\mathcal{P}}(\boldsymbol{V}) \subseteq \mathcal{P}(\boldsymbol{V})$, the set of feasible Cartesian multiedge partitions $\hat{\mathfrak{P}}^{\times}(\boldsymbol{V}\times\boldsymbol{V}) \subseteq \mathfrak{P}^{\times}(\boldsymbol{V}\times\boldsymbol{V})$ *is the set of Cartesian multiedge partitions which multiedge subsets are only made of feasible vertex subsets:*

$$\begin{aligned}\hat{\mathfrak{P}}^{\times}(\boldsymbol{V}\times\boldsymbol{V}) \;=\; \{\{(V_1\times V'_1),\ldots,(V_m\times V'_m)\} \in \hat{\mathcal{P}}(\boldsymbol{V})\times\hat{\mathcal{P}}(\boldsymbol{V}) \\ :\; \cup_i(V_i\times V'_i) = \boldsymbol{V}\times\boldsymbol{V} \;\wedge\; \forall i \neq j,\; (V_i\times V'_i)\cap(V_j\times V'_j) = \emptyset\}.\end{aligned}$$

In the most general case, the set of feasible vertex subsets $\hat{\mathcal{P}}(\boldsymbol{V})$ can be composed of all vertex subset in $\mathcal{P}(\boldsymbol{V})$. However, one can use more constrained – and hence more meaningful vertex structures – to drive the compression scheme and its possible applications. A survey about the types of constraints that have been used in many different domains can be found in [33]. We present below some of these structures and their basic combinatorics.

The Complete Case. When no additional constraint applies to vertex subsets, then:

$$\hat{\mathcal{P}}(\boldsymbol{V}) \;=\; \mathcal{P}(\boldsymbol{V}), \quad \text{and so} \quad \hat{\mathfrak{P}}(\boldsymbol{V}) = \mathfrak{P}(\boldsymbol{V}).$$

Such scheme can be used when no additional useful structure is known about the vertex set, for example to model *coalition structures* in multi-agent systems when assuming that every possible group of agents is an adequate candidate to constitute a coalition [34, 35]. It has been shown that, in this case, the number of feasible partitions grows considerably faster than the number of feasible subsets [36]:

$$|\mathcal{P}(\boldsymbol{V})| = 2^n \quad \text{and} \quad |\mathfrak{P}(\boldsymbol{V})| = \omega(n^{n/2}),$$

where $n = |\boldsymbol{V}|$ is the number of vertices in the graph.

Hierarchies of Vertex Subsets. A *hierarchy* $\hat{\mathcal{P}}(\boldsymbol{V}) = \mathcal{H}(\boldsymbol{V})$ is such that any two feasible vertex subsets are either disjoint, or one is included in the other:

$$\forall (V_1, V_2) \in \mathcal{H}(\boldsymbol{V})^2, \quad V_1 \cap V_2 = \emptyset \ \vee \ V_1 \subseteq V_2 \ \vee \ V_2 \subseteq V_1.$$

We mark $\hat{\mathfrak{P}}(\boldsymbol{V}) = \mathfrak{H}(\boldsymbol{V})$ the resulting set of feasible vertex partitions. Such vertex hierarchies can be used to model graphs that are known to have a multilevel nested structure that one wants to preserve during compression. This might include, among others, a *community structure* that have been preliminarily identified through a hierarchical community detection algorithm [37], a sequence of *geographical nested partitions* of the world's territorial units [24]; a *hierarchical communication network* in distributed computers [19].

It has been shown that the number of feasible subsets in a hierarchy is asymptotically bounded from above by the number of objects it contains [33]: $|\mathcal{H}(\boldsymbol{V})| = O(n)$. The resulting number of feasible partitions however depends on the number of levels and branches in the hierarchy. For a complete binary tree, it is asymptotically bounded by an exponential function: $|\mathfrak{H}(\boldsymbol{V})| = \Theta(\alpha^n)$, with $\alpha \approx 1.226$ [38]. Similar results have been found for complete ternary trees (with $\alpha \approx 1.084$ [39]), complete quaternary trees, and so on. Henceforth, for any bounded number of children per node in the hierarchy, the number of feasible partitions exponentially grows with the number of objects.

Sets of Vertex Intervals. Any total order $<$ on the vertex set $\boldsymbol{V}$ induces a *set of vertex intervals* $\hat{\mathcal{P}}(\boldsymbol{V}) = \mathcal{I}(\boldsymbol{V})$ defined as follows:

$$\mathcal{I}(\boldsymbol{V}) \ = \ \{[v_1, v_2] \ : \ (v_1, v_2) \in \boldsymbol{V}^2\} \quad \text{where} \quad [v_1, v_2] \ = \ \{v \in \boldsymbol{V} : v_1 \leq v \leq v_2\}.$$

We mark $\hat{\mathfrak{P}}(\boldsymbol{V}) = \mathfrak{I}(\boldsymbol{V})$ the resulting set of feasible vertex partitions. It can be represented as a "pyramid of intervals" [33] and such partitions are sometimes called *consecutive partitions* [40]. Such sets of intervals naturally apply to vertices having a temporal feature (*e.g.*, events, dates, or time periods are naturally ordered by the "arrow of time") and have hence been exploited for the aggregation of time series [41, 42, 24]. They might also model unidimensional spacial features, such as the geographical ordering of cities on a coast, or on a transport route [43].

The number of intervals of an ordered set of size n is $|\mathcal{I}(\boldsymbol{V})| = \frac{n(n+1)}{2}$. The resulting number of feasible partitions is $|\mathfrak{I}(\boldsymbol{V})| = 2^{n-1}$ [33].

3.5. From Multigraphs to Multistreams

The last step of our generalisation work regards the integration of a temporal dimension within our framework in order to finally deal with the compression of temporal graphs [10]. As argued in the introduction of this article, we build on the *link stream* representation of such graphs [11, 12]. However, because we also want to deal with cases for which multiple edges are allowed between two vertices at a given time instance, we actually generalise from streams to multistreams, as we did in Subsection 3.1 to generalise from graphs to multigraphs.

Definition 12 (Directed Multistream)**.**
A directed multistream $MS = (\boldsymbol{V}, \boldsymbol{T}, e)$ *is characterised by:*

- *A set of* vertices $\boldsymbol{V}$*;*
- *A set of* time instances $\boldsymbol{T} \subseteq \mathbb{R}$*;*
- *A multiset of* directed edges $(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}, e)$
  where $e : \boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T} \to \mathbb{N}$ *is the* edge function*, that is the multiplicity function counting the number of edges* $e(v, v', t) \in \mathbb{N}$ *going from a given source vertex* $v \in \boldsymbol{V}$ *to a given target vertex* $v' \in \boldsymbol{V}$ *at a given time instance* $t \in \boldsymbol{T}$*.*

Figure 5: Multistream compression, that is the Cartesian partitioning of $\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}$, consists in the natural tridimensional generalisation of multigraph compression, that is the Cartesian partitioning of $\boldsymbol{V}\times\boldsymbol{V}$.

In this context, the edge function counts the number of interactions happening between vertices at a given time: $e(v, v', t)$ is the number of edges going from source vertex v to target vertex v' at time t.

As illustrated in Figure 5, this formalism constitutes an elegant solution to generalise our compression scheme since it simply adds a third dimension $\boldsymbol{T} \subseteq \mathbb{R}$ to the multigraph's definition. The GCP is then generalised to this three-dimensional formalism by (i) defining a set of feasible time subsets that preserves the ordering of time instances, that is a set of intervals $\hat{\mathcal{P}}(\boldsymbol{T}) = \mathcal{I}(\boldsymbol{T})$ (see previous subsection), (ii) considering three-dimensional Cartesian multiedge subsets $V\times V'\times T \in \hat{\mathcal{P}}(\boldsymbol{V})\times\hat{\mathcal{P}}(\boldsymbol{V})\times\mathcal{I}(\boldsymbol{T})$, and (iii) computing the information loss on this generalised space in a similar fashion than for the static version of the compression problem. Here is the resulting generalisation in details.

see Definition 5

- Time instances: $t \in \boldsymbol{T} \subseteq \mathbb{R}$;
- Time subsets: $T \in \mathcal{P}(\boldsymbol{T})$;
- Multiedges: $(v, v', t) \in \boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}$;
- Edge function: $e : \boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T} \to \mathbb{N}$;
- Compressed edge function:

$$e : \mathcal{P}(\boldsymbol{V})\times\mathcal{P}(\boldsymbol{V})\times\mathcal{P}(\boldsymbol{T}) \to \mathbb{N} \quad \text{with} \quad e(V, V', T) = \sum_{(v,v')\in V\times V'} \int_{t\in T} e(v, v', t)\, dt;$$

see Def. 11

- Feasible time subsets, that is time intervals: $T = [t_1, t_2] \in \hat{\mathcal{P}}(\boldsymbol{T}) = \mathcal{I}(\boldsymbol{T}) \subset \mathcal{P}(\boldsymbol{T})$;
- Feasible multiedge subsets: $V\times V'\times T \in \hat{\mathcal{P}}(\boldsymbol{V})\times\hat{\mathcal{P}}(\boldsymbol{V})\times\mathcal{I}(\boldsymbol{T}) \subset \mathcal{P}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})$;

see Def. 10

- Feasible Cartesian multiedge partitions:

$$\mathcal{VVT} \in \hat{\mathfrak{P}}^{\times}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}) = \{\{(V_1\times V_1'\times T_1), \ldots, (V_m\times V_m'\times T_m)\} \in \hat{\mathcal{P}}(\boldsymbol{V})\times\hat{\mathcal{P}}(\boldsymbol{V})\times\mathcal{I}(\boldsymbol{T}) \\ : \cup_i(V_i\times V_i'\times T_i) = \boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T} \wedge (V_i\times V_i'\times T_i)\cap(V_j\times V_j'\times T_j) = \emptyset\};$$

see Definition 6

- Observed variable:

$$(X, X', X'') \in \boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T} \quad \text{with} \quad f_{(X,X',X'')}(v, v', t) = \frac{e(v, v', t)}{e(\boldsymbol{V}, \boldsymbol{V}, \boldsymbol{T})};$$

see Definition 7

- Partition function:

$$\mathcal{VVT}(v, v', t) \text{ is the only multiedge subset } V{\times}V'{\times}T \text{ in } \mathcal{VVT} \text{ that contains } (v, v', t);$$

- Compressed variable:

$$\mathcal{VVT}(X, X', X'') \in \mathcal{VVT} \quad \text{with} \quad f_{\mathcal{VVT}(X,X',X'')}(V, V', T) = \frac{e(V, V', T)}{e(\boldsymbol{V}, \boldsymbol{V}, \boldsymbol{T})};$$

see Definition 8

- External variable, in the case of a degree-preserving compression:

$$(Y, Y', Y'') \in \mathcal{VVT} \quad \text{with} \quad f_{(Y,Y',Y'')}(v, v', t) = \frac{e(v, \boldsymbol{V}, \boldsymbol{T})}{e(\boldsymbol{V}, \boldsymbol{V}, \boldsymbol{T})} \frac{e(\boldsymbol{V}, v', \boldsymbol{T})}{e(\boldsymbol{V}, \boldsymbol{V}, \boldsymbol{T})} \frac{e(\boldsymbol{V}, \boldsymbol{V}, t)}{e(\boldsymbol{V}, \boldsymbol{V}, \boldsymbol{T})};$$

- Decompressed variable:

$$\mathcal{VVT}_{(Y,Y',Y'')}(X, X', X'') \in \boldsymbol{V}{\times}\boldsymbol{V}{\times}\boldsymbol{T}$$

$$\text{with} \quad f_{\mathcal{VVT}_{(Y,Y',Y'')}(X,X',X'')}(v, v', t) = \frac{e(V, V', T)}{e(\boldsymbol{V}, \boldsymbol{V}, \boldsymbol{T})} \frac{e(v, \boldsymbol{V}, \boldsymbol{T})\, e(\boldsymbol{V}, v', \boldsymbol{T})\, e(\boldsymbol{V}, \boldsymbol{V}, t)}{e(V, \boldsymbol{V}, \boldsymbol{T})\, e(\boldsymbol{V}, V', \boldsymbol{T})\, e(\boldsymbol{V}, \boldsymbol{V}, T)}$$

where $V{\times}V'{\times}T = \mathcal{VVT}(v, v', t)$;

see Definition 9

- Information loss, decomposed at the level of multiedge subsets:

$$\text{loss}(V, V', T) = \sum_{(v,v')\in V\times V'} \int_{t\in T} \frac{e(v, v', t)}{e(\boldsymbol{V}, \boldsymbol{V}, \boldsymbol{T})} \log_2 \left(\frac{e(v, v', t)}{e(V, V', T)} \Big/ \frac{e(v, \boldsymbol{V}, \boldsymbol{T})\, e(\boldsymbol{V}, v', \boldsymbol{T})\, e(\boldsymbol{V}, \boldsymbol{V}, t)}{e(V, \boldsymbol{V}, \boldsymbol{T})\, e(\boldsymbol{V}, V', \boldsymbol{T})\, e(\boldsymbol{V}, \boldsymbol{V}, T)} \right) dt.$$

The use of *integrals* instead of discrete sums to define the edge function, as well as the use of probability *density function* instead of discrete probability distribution to define the random variables, is due to the fact that the added temporal dimension $\boldsymbol{T}$ is continuous, contrary to the vertex set $\boldsymbol{V}$. A simpler setting for temporal graphs would consist in assuming a discrete representation of time, that would hence only require discrete operators. In this regard, the optimisation algorithm that is introduced in next section to solve the generalised GCP is only provided for the discrete case, for simplicity reasons.

Related Work. As discussed in the introduction of this article, and as illustrated in this subsection, the recent work on the *link stream* formalism [11, 12] proposes to deal with time as a simple addition to the graph's structural dimension. Considering temporal graphs as genuine tridimensional data, the arbitrary separation of structure and time is avoided, thus making the generalisation quite natural. Note that similar generalisation objectives have been addressed in previous work on graph compression, as for example the application of bidimensional *block models* to multidimensional matrices [13] or the application of *biclustering* to triplets of variables [14], which has then been exploited for the statistical analysis of temporal graphs [15]. The particular interest of such approaches also consists in the fact that they provide a unified compression scheme in which structural and temporal information is simultaneously taken into account.

4. Result: The Lossy Multistream Compression Problem

This section integrates all generalisations that have been proposed in previous section to the GCP in order to properly formalise the Lossy Multistream Compression Problem (MSCP). It then proposes an algorithmic solution to this problem by reducing it to the *Set Partitioning Problem*, a well-known combinatorial optimisation problem allowing to exploit state-of-the-art approaches.

4.1. The Lossy MSCP

We first formalise the Lossy MSCP by exploiting all notions introduced in previous section. Note that this instance of the problem has a discrete temporal dimension (for an easier algorithmic solution) and is described for a particular external variable (corresponding to a degree-preserving decompression, see Subsection 3.2).

Definition 13 (The Lossy Multistream Compression Problem)**.**
Given:

- *A directed multistream* $MS = (\boldsymbol{V}, \boldsymbol{T}, e)$*, where* $\boldsymbol{T} = [\![t_\alpha, t_\omega]\!] \subseteq \mathbb{N}$ *and where the edge function* $e : \boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T} \to \mathbb{N}$ *determines the observed variable* $(X, X', X'') \in \boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}$*;*
- *A set of feasible vertex subsets* $\hat{\mathcal{P}}(\boldsymbol{V}) \subseteq \mathcal{P}(\boldsymbol{V})$*, the set of time intervals* $\mathcal{I}(\boldsymbol{T}) \subset \mathcal{P}(\boldsymbol{T})$*, and the set of feasible Cartesian multiedge partitions* $\hat{\mathfrak{P}}^\times(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})$*;*
- *The external variable* $(Y, Y', Y'') \in \boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}$ *corresponding to a degree-preserving decompression, which determines the information function* $\mathrm{loss} : \hat{\mathfrak{P}}^\times(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}) \to \mathbb{R}^+$*, and an information threshold* $\tau \in \mathbb{R}^+$*;*

Find a feasible Cartesian multiedge partition $\mathcal{VVT}^* \in \hat{\mathfrak{P}}^\times(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})$ *with minimal size* $|\mathcal{VVT}^*|$ *such that* $\mathrm{loss}(\mathcal{VVT}^*) \leq \tau$*:*

$$\mathcal{VVT}^* = \underset{\substack{\mathcal{VVT}\in\hat{\mathfrak{P}}^\times(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}) \\ \mathrm{loss}(\mathcal{VVT})\leq\tau}}{\arg\min} |\mathcal{VVT}|.$$

Lagrangian Relaxation. The MSCP is a *constrained* optimisation problem: One wants to minimise the size of the multiedge partition (objective) while guarantying that the information loss it induces stays below a given threshold (constraint). Lagrangian relaxation is a classical relaxation method which approximates the constrained problem by a simpler unconstrained one [44]. It consists in injecting the constraints within the objective, penalising eventual violations of these constraints by using a Lagrange multiplier $\lambda \in \mathbb{R}^+$ that is interpreted as the cost that such violations pose to the objective. It hence results in an unconstrained optimisation problem which objective is parametrised by the Lagrange multiplier. In the case of the MSCP, the Lagrange function that mixes the information-theoretic objective and the size constraint is

$$q_\lambda(\mathcal{VVT}) = |\mathcal{VVT}| + \mathrm{loss}(\mathcal{VVT}),$$

where $\lambda \in \mathbb{R}^+$ is the Lagrange multiplier. The relaxed MSCP then simply consists in minimising this parametrised objective:

$$\mathcal{VVT}^* = \underset{\mathcal{VVT}\in\hat{\mathfrak{P}}^\times(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})}{\arg\min} q_\lambda(\mathcal{VVT}).$$

Intuitively, when $\lambda = 0$, the relaxed MSCP consists in minimising the partition size without any constraint regarding the information loss it induces. This hence corresponds to a virtually infinite information threshold τ for which the *maximal partition* (all multiedges in one subset) is optimal. When $\lambda \to \infty$, the objective becomes negligible in comparison to the cost of constraint violation, so the relaxed MSCP simply consists in minimising the information loss. This hence corresponds to the highest level of constraint ($\tau = 0$) for which the *minimal partition* (each multiedge in its own subset) is optimal. By varying the Lagrange multiplier λ between 0 and ∞, one can then express different levels of constraints and obtain a multiscale sequence of compression results ranging from the minimal to the maximal partition.

4.2. Reducing the Lossy MSCP to the Set Partitioning Problem

The *Set Partitioning Problem* (SPP) is a deeply-studied combinatorial optimisation problem that naturally arises as soon as one wants to organise a set of objects into covering and pairwise disjoint subsets such that an additive objective is minimised [17]. To that matter, the Lossy MSCP can be expressed as a particular case of the SPP, hence allowing the use of state-of-the-art algorithmic results to address it.

Definition 14 (The Set Partitioning Problem)**.**
Given a set of objects $\boldsymbol{X}$*, a set of feasible subsets* $\hat{\mathcal{P}}(\boldsymbol{X}) \subseteq \mathcal{P}(\boldsymbol{X})$*, and a cost function* $c : \hat{\mathcal{P}}(\boldsymbol{X}) \to \mathbb{R}^+$*, the* Set Partitioning Problem *consists in finding a minimal feasible partition of* $\boldsymbol{X}$*, that is a set of feasible subsets* $\mathcal{X} \subseteq \hat{\mathcal{P}}(\boldsymbol{X})$ *that partitions* $\boldsymbol{X}$ *and that minimises the sum of the costs:*

$$\mathcal{X}^* \quad = \quad \underset{\mathcal{X} \in \hat{\mathfrak{P}}(\boldsymbol{X})}{\arg\min} \sum_{X \in \mathcal{X}} c(X),$$

where $\hat{\mathfrak{P}}(\boldsymbol{X}) = \{\{X_1, \ldots, X_m\} \subseteq \hat{\mathcal{P}}(\boldsymbol{X}) \ : \ \cup_i X_i = \boldsymbol{X} \ \wedge \ \forall i \neq j, \ X_i \cap X_j = \emptyset\}$.

The Lossy MSCP can be expressed as special cases of the SPP:

- The objects are the triples consisting in two vertices and one time instance: $\boldsymbol{X} \ = \ \boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}$;
- The feasible subsets are the Cartesian products of any two feasible vertex subsets and a time interval[6]:

$$\hat{\mathcal{P}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}) \ = \ \hat{\mathcal{P}}(\boldsymbol{V}) \times \hat{\mathcal{P}}(\boldsymbol{V}) \times \mathcal{I}(\boldsymbol{T});$$

- The cost function is given by the information loss function defined at the subset level:

$$c : \hat{\mathcal{P}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}) \to \mathbb{R}^+ \quad \text{with} \quad c(V, V', T) \ = \ q_\lambda(V, V', T) \ = \ 1 \ + \ \lambda \, \text{loss}(V, V', T).$$

Solving the Set Partitioning Problem. The SPP is NP-complete in the general case [45]. Hence, one cannot hope for a general-purpose algorithm that can efficiently solve every instance of the SPP (except if P = NP). However, it as been shown that some restrictive, yet meaningful constraints on the set of feasible subsets induce tractable versions of the SPP. Hence, among the many algorithmic strategies that have been proposed in the literature to tackle the problem (see [33] for an extensive review of such work), we are interested in this article in approaches that (i) take advantage of the algebraic properties of partition lattices (see Appendix A) to cleverly search the solution space for an optimal solution, and that moreover (ii) take advantage of the structure of constraints exerting on the set of feasible subsets (see Subsection 3.4) to reduce the size of the solution space and to speed up the search.

Regarding (i), approaches based on dynamic programming have been proposed in the past to solve the SPP in its most general formulation, leading to a $\Omega(2^n)$ and $O(3^n)$ optimisation algorithm [43, 36], where $n = |\boldsymbol{X}|$ is the number of objects. However, regarding (ii), it has been shown that more efficient algorithms are possible when dealing with special versions of the problem, that is when the set of feasible subsets has a more specific structure that can also be exploited by dynamic programming. For example, when the set of feasible subsets forms a hierarchy $\mathcal{H}(\boldsymbol{X})$ (see Subsection 3.4), then an appropriate depth-first search of the tree representing the hierarchy allows for a $O(n)$ optimisation algorithm [37, 24]. When feasible subsets forms a set of intervals $\mathcal{I}(\boldsymbol{X})$, then solving the SPP is equivalent to solving the *shortest path problem* [45] and several $O(n^2)$ dynamic programming algorithms have been proposed in the past for this particular case [40, 46, 43, 41].

In Appendix A, we present a general algorithmic framework that has been proposed in previous work to solve such special versions of the SPP. The key principle of this framework, and of dynamic programming in general, is the following: (i) The solution space is first broken down into smaller covering subspaces; (ii) Then, thanks to a *principle of optimality* that fits with the algebraic structure of the partition set, these subproblems are recursively solved; (iii) Locally-optimal solutions are then compared to globally solve the initial problem. More precisely, the principle of optimality builds on the following observation: For each feasible subset $X \in \hat{\mathcal{P}}(\boldsymbol{X})$, by first computing a feasible optimal partition $\mathcal{X}^* \in \mathfrak{P}(X)$ on this subset, one can use this optimal partition to efficiently evaluate all the partitions that are coarser than X, that is all partitions that contain a subset that contains X. This allows to identify and suppress numerous redundant computations when searching the solution space (see detailed definitions and proofs in Appendix A, in particular for the definition of refinements $\mathfrak{R}$ and covering partitions $\mathfrak{C}$).

[6]There is again an abuse of notation in the following definition when we write that "$\hat{\mathcal{P}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}) \ = \ \hat{\mathcal{P}}(\boldsymbol{V}) \times \hat{\mathcal{P}}(\boldsymbol{V}) \times \mathcal{I}(\boldsymbol{T})$". In principle, we should write that "$\hat{\mathcal{P}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}) \ = \ \{V\times V'\times T : (V, V', T) \in \hat{\mathcal{P}}(\boldsymbol{V}) \times \hat{\mathcal{P}}(\boldsymbol{V}) \times \mathcal{I}(\boldsymbol{T})\}$", but we prefer the former for notation conciseness.

Here is the resulting high-level algorithm:

Algorithm 1 (Recursive Algorithm to Solve the SPP [18, 33])**.**

Global:

- *A set of objects* $\boldsymbol{X}$*;*
- *A set of feasible subsets* $\hat{\mathcal{P}}(\boldsymbol{X}) \subseteq \mathcal{P}(\boldsymbol{X})$*;*
- *A cost function* $c : \hat{\mathcal{P}}(\boldsymbol{X}) \to \mathbb{R}^+$.

Input: *A feasible subset* $X \in \hat{\mathcal{P}}(\boldsymbol{X})$.

Output: *A locally-optimal feasible partition* $\mathcal{X}^* \in \hat{\mathfrak{P}}^*(X)$.

Algorithm:

(step 1) If the algorithm has already been called on X, return the recorded locally-optimal partition and its cost.

(step 2) Compute the set $\hat{\mathfrak{C}}(\{X\})$ *of feasible partitions that are covered by* $\{X\}$.

(step 3) For each such partition $\mathcal{X} \in \hat{\mathfrak{C}}(\{X\})$*, do the following:*

(step 3.a) For each subset $X' \in \mathcal{X}$*, recursively compute a locally-optimal feasible partition* $\mathcal{X}^*_{X'} \in \hat{\mathfrak{P}}^*(X')$*;*

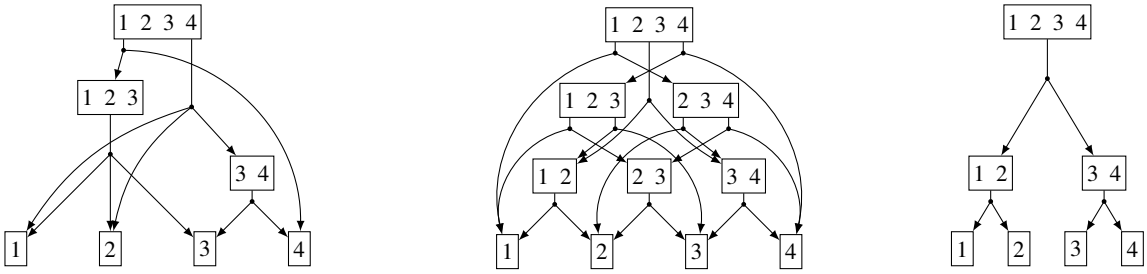
(step 3.b) Compute the union $\mathcal{X}^* = \bigcup_{X' \in \mathcal{X}} \mathcal{X}^*_{X'}$ *of these partitions. The principle of optimality ensures that* $\mathcal{X}^* \in \hat{\mathfrak{R}}^*(\mathcal{X})$.

(step 4) Record and return a partition that minimises the cost function c among the maximal partition $\{X\}$ *and all the partitions* $\mathcal{X}^*$ *that have been computed.*

4.3. Solving the Lossy MSCP

We now precise and apply the high-level SPP algorithm introduced in previous subsection to solve the Lossy MSCP, by giving more details about its specialisation. In particular, the algorithm above does neither precise how the set of feasible subsets $\hat{\mathcal{P}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})$ should be represented in memory, nor how the computation of feasible covered partitions $\hat{\mathfrak{C}}(\{V\times V'\times T\})$ and the computation of the cost function $c(V\times V'\times T)$ should be implemented in practice. In what follows, we provide an efficient data structure to do so and a detailed implementation of the resulting combinatorial optimisation algorithm.

Data structure to represent the set of feasible vertex subsets. In general, one can use a poset structure with one-to-many links to represent a set of feasible vertex subset $\hat{\mathcal{P}}(\boldsymbol{V})$ along with its covering relation $\sqsubset$: Each node of this data structure represents a feasible vertex subset $V \in \hat{\mathcal{P}}(\boldsymbol{V})$ and each link represents a covering relation $\{V\} \sqsupset \{V_1, \ldots, V_k\}$ going from one node V to multiple nodes $\{V_1, \ldots, V_k\} \in \hat{\mathfrak{C}}(\{V\})$, thus representing a feasible vertex partition that is



a. Arbitrary set b. Set of intervals c. Hierarchy

Figure 6: Poset structures representing different sets of feasible vertex subsets $\hat{\mathcal{P}}(\boldsymbol{V})$. Each node represents a feasible vertex subset $V \in \hat{\mathcal{P}}(\boldsymbol{V})$ and each one-to-many link represents a feasible covered partition $\{V_1, \ldots, V_k\} \in \hat{\mathfrak{C}}(\{V\})$.

covered by $\{V\}$. Figure 6 gives an example of such poset structures in the case of (a) an arbitrary set of feasible vertex subsets, (b) a set of vertex intervals, and (c) a hierarchy of vertex subsets (see Subsection 3.4).

In what follows, we mark $|\hat{\mathfrak{C}}(\boldsymbol{V})|$ the sum of the numbers of vertex subsets for all covered partitions encoded in the poset structure, that is the total number of in-coming links to be encoded:

$$|\hat{\mathfrak{C}}(\boldsymbol{V})| \quad = \quad \sum_{\substack{V \in \hat{\mathcal{P}}(\boldsymbol{V}) \\ \mathcal{V} \in \hat{\mathfrak{C}}(\{V\})}} |\mathcal{V}|.$$

Note that, in particular cases such as (b) and (c), more straightforward data structures have been proposed in the literature: For example, using a triangular matrix in the case of a set of intervals and using a simple poset in the case of a hierarchy (with one-to-one links). This would allow to avoid the explicit representation of the covering relation through the use of one-to-many links: *E.g.*, memory requirement in $\Theta(n^2)$ instead of $\Theta(n^3)$ to represent the covering relations within a set of intervals by using a triangular matrix instead of a poset structure [33]. However, as the objective is to be here as general as possible, we do not enter in the details of such specialised implementations.

Data structure to represent the set of feasible multiedge subsets. In the case of a Cartesian multiedge partition $\mathcal{V}\mathcal{V}\mathcal{T} \in \hat{\mathfrak{P}}^{\times}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})$ (see Subsection 3.3), given the data structure hereabove mentioned to represent the set of feasible *vertex subsets* $\hat{\mathcal{P}}(\boldsymbol{V})$ and the set of feasible *time intervals* $\mathcal{I}(\boldsymbol{T})$, one can build a similar data structure to represent the set of feasible *multiedge subsets* $\hat{\mathcal{P}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})$ by generalising to a tridimensional poset structure (see a bidimensional example in Figure 7). Each node in this tridimensional poset structure represents the Cartesian product of three nodes in the unidimensional poset structures:

$$\hat{\mathcal{P}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}) \quad = \quad \hat{\mathcal{P}}(\boldsymbol{V}) \times \hat{\mathcal{P}}(\boldsymbol{V}) \times \mathcal{I}(\boldsymbol{T}).$$

The number of nodes in the tridimensional poset structure is hence the product of the number of nodes in the unidimensional poset structures:

$$|\hat{\mathcal{P}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})| \quad = \quad |\hat{\mathcal{P}}(\boldsymbol{V})| \times |\hat{\mathcal{P}}(\boldsymbol{V})| \times |\mathcal{I}(\boldsymbol{T})|. \tag{1}$$

The one-to-many links representing the covering relations are then obtained by copying, for each node in the tridimensional poset structure, the corresponding links of the unidimensional poset structures *along all dimensions*, that is the links associated with the three nodes of the corresponding Cartesian product:

$$\begin{aligned}\hat{\mathfrak{C}}(\{V\times V'\times T\}) \quad = \quad & \{\{V\times V'\times T : V \in \mathcal{V}\} : \mathcal{V} \in \hat{\mathfrak{C}}(\{V\})\} \\ & \cup \{\{V\times V'\times T : V' \in \mathcal{V}'\} : \mathcal{V}' \in \hat{\mathfrak{C}}(\{V'\})\} \\ & \cup \{\{V\times V'\times T : T \in \mathcal{T}\} : \mathcal{T} \in \hat{\mathfrak{C}}(\{T\})\}.\end{aligned}$$

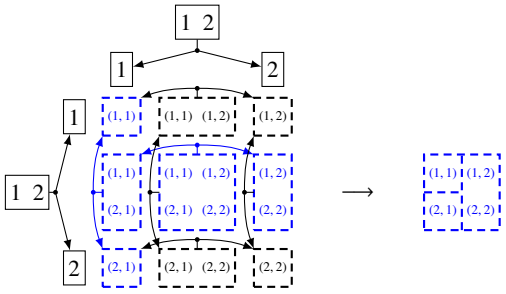


Figure 7: Bidimensional poset structure representing a set of feasible multiedge subsets $\hat{\mathcal{P}}(\boldsymbol{V}\times\boldsymbol{V})$. Solid rectangles on the left and on the top represent the two unidimensional poset structures $\mathcal{P}(\boldsymbol{V}) \times \mathcal{P}(\boldsymbol{V})$. Each dashed rectangle represents a feasible multiedge subset $(V, V') \in \hat{\mathcal{P}}(\boldsymbol{V}\times\boldsymbol{V})$ and each one-to-many link represents a feasible covered partition $\{V_1\times V'_1, \ldots, V_k\times V'_k\} \in \hat{\mathfrak{C}}(\{(V, V')\})$. The nodes and the links in blue form a rooted subposet that represents a feasible multiedge partition (represented on the right).

The number of in-coming links in the tridimensional poset structure is hence given by the following formula:

$$\begin{aligned} |\hat{\mathfrak{C}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})| \quad &= \quad |\hat{\mathfrak{C}}(\boldsymbol{V})|\,|\hat{\mathcal{P}}(\boldsymbol{V})|\,|\mathcal{I}(\boldsymbol{T})| \\ &\quad + |\hat{\mathcal{P}}(\boldsymbol{V})|\,|\hat{\mathfrak{C}}(\boldsymbol{V})|\,|\mathcal{I}(\boldsymbol{T})| \\ &\quad + |\hat{\mathcal{P}}(\boldsymbol{V})|\,|\hat{\mathcal{P}}(\boldsymbol{V})|\,|\hat{\mathfrak{C}}(\boldsymbol{T})|. \end{aligned} \tag{2}$$

Table 1 then exploits Equations 1 and 2, as well as the basic combinatorial analysis presented in Subsection 3.4, to present the size of the resulting unidimensional and tridimensional poset structures (in terms of nodes and in terms of links) depending on the types of feasibility constraints applying on the set of vertex subset.

Representing a feasible multiedge partition. Within such a poset structure, a feasible multiedge partition can be simply represented as a rooted subposet, that is a poset which nodes and links are contained in the original poset and such that its maximal element is the same as the one of the original poset. This amount in expressing the partition as a sequence of "division steps" going from the root of the poset structure (its maximal element) down to the nodes that represent the multiedge subsets belonging to the represented partition. Hence, to do so, one only needs to record, for each node in this sequence, the next link to follow, or no link at all if one arrived at a subset that belongs to the encoded partition. An example of such representation is given in blue in Figure 7 (in the case of a bidimensional poset structure).

Recursive computation of the cost function. In addition of being decomposable from partitions to their subsets (see Definition 9), the information loss function defined in Section 3.2 can also be decomposed into aggregative measures on these subsets, hence allowing for a recursive computation:

$$\mathrm{loss}(V,V',T) \;=\; \mathrm{info} \;-\; \mathrm{sum}\,(V,V',T)\log_2\left(\frac{\mathrm{sum}\,(V,V',T)}{\mathrm{sum}_1\,(V,V',T)\;\mathrm{sum}_2\,(V,V',T)\;\mathrm{sum}_3\,(V,V',T)}\right),$$

where

$$\mathrm{info} \;=\; \sum_{(v,v',t)\in V\times V'\times T} \frac{e(v,v',t)}{e(\boldsymbol{V},\boldsymbol{V},\boldsymbol{T})}\log_2\left(\frac{e(v,v',t)}{e(\boldsymbol{V},\boldsymbol{V},\boldsymbol{T})}\Big/\frac{e(v,\boldsymbol{V},\boldsymbol{T})\,e(\boldsymbol{V},v',\boldsymbol{T})\,e(\boldsymbol{V},\boldsymbol{V},t)}{e(\boldsymbol{V},\boldsymbol{V},\boldsymbol{T})\,e(\boldsymbol{V},\boldsymbol{V},\boldsymbol{T})\,e(\boldsymbol{V},\boldsymbol{V},\boldsymbol{T})}\right)$$

is a constant term that does not depend on (V,V',T) and

$$\begin{aligned} \mathrm{sum}\,(V,V',T) \quad &= \quad \frac{e(V,V',T)}{e(\boldsymbol{V},\boldsymbol{V},\boldsymbol{T})}, \qquad \mathrm{sum}_1\,(V,V',T) \quad = \quad \frac{e(V,\boldsymbol{V},\boldsymbol{T})}{e(\boldsymbol{V},\boldsymbol{V},\boldsymbol{T})}, \\ \mathrm{sum}_2\,(V,V',T) \quad &= \quad \frac{e(\boldsymbol{V},V',\boldsymbol{T})}{e(\boldsymbol{V},\boldsymbol{V},\boldsymbol{T})}, \qquad \mathrm{sum}_3\,(V,V',T) \quad = \quad \frac{e(\boldsymbol{V},\boldsymbol{V},T)}{e(\boldsymbol{V},\boldsymbol{V},\boldsymbol{T})}, \end{aligned}$$

are all additive, in the sense that, when applied to a multiedge subset $V\times V'\times T$, they can be expressed as the sum of measures applied to a set of multiedge subsets that forms a partition of $V\times V'\times T$:

$$\{V_1\times V'_1\times T_1,\ldots,V_k\times V'_k\times T_k\} \;\in\; \mathfrak{P}(V\times V'\times T) \quad\Rightarrow\quad \mathrm{sum}_k\,(V,V',T) \;=\; \sum_i \mathrm{sum}_k(V_i,V'_i,T_i).$$

Hence, by first computing these measures for minimal elements in the aforementioned tridimensional poset structure, then browsing the poset in a bottom-up fashion, one can efficiently compute the values of the cost function for all feasible multiedge subsets it contains. The information loss induced by a multiedge partition is then given by the sum of these values, to which is added the constant term (info) which is pre-computed once and for all.

Resulting optimisation algorithm. The resulting recursive algorithm solving the Lossy MSCP is detailed below. It is obtained by applying high-level Algorithm 1 to the poset structure we just described. Hence, to solve the Lossy MSCP, one first needs (i) to build this data structure, then (ii) to compute the cost of each node it contains by applying the recursive approach above, and finally (iii) to run Algorithm 2 on its maximal element.

Algorithm 2 (Recursive Algorithm to Solve the Lossy MSCP)**.**

Global: *A tridimensional poset structure with one-to-many links such that, given a* `node` *in this structure:*

- `node` *represents a feasible multiedge subset* $V\times V'\times T \in \hat{\mathcal{P}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})$*;*
- `node.links` *is a list of one-to-many links such that, given a* `link` *in this list:*
  - `link` *represents a feasible multiedge partition* $\mathcal{VVT} \in \hat{\mathfrak{C}}(\{V\times V'\times T\})$ *covered by* $\{V\times V'\times T\}$*;*
  - `link.children` *is a list of nodes in the tridimensional poset structure representing the feasible multiedge subsets* $\{V_1\times V_1'\times T_1, \ldots, V_k\times V_k'\times T_k\}$ *in the covered partition* $\mathcal{VVT}$*;*
- `node.cost` *is a positive float value representing the Lagrange version* $q_\lambda(V\times V'\times T) \in \mathbb{R}^+$ *of the information loss induced by* $V\times V'\times T$*;*
- `node.optimal_link` *is one of the links in* `node.links` *encoding a locally-optimal feasible multiedge partition* $\mathcal{VVT}^* \in \hat{\mathfrak{P}}^*(V\times V'\times T)$ *(it is set to* `NULL` *before the algorithm starts);*
- `node.optimal_cost` *is a positive float value representing the Lagrange version* $q_\lambda(\mathcal{VVT}^*) \in \mathbb{R}^+$ *of the information loss induced by the locally-optimal partition* $\mathcal{VVT}^*$ *(it is set to* `NULL` *before the algorithm starts).*

Input:

- *A* `node` *in the tridimensional poset structure.*

Output:

- *Computes a locally-optimal feasible multiedge subset for* `node`*;*
- *Sets* `node.optimal_link` *accordingly;*
- *Returns* `node.optimal_cost`*.*

Algorithm:

```
compute_optimal_partition (node)
    if node.optimal_cost not NULL, then return node.optimal_cost
    node.optimal_link ← NULL
    node.optimal_cost ← node.cost
    for each link in node.links do
        cost ← 0
        for each child in link.children do
            cost ← cost + compute_optimal_partition (child)
        if cost < node.optimal_cost, then
            node.optimal_link ← link
            node.optimal_cost ← cost
    return node.optimal_cost
```

For $\lvert\boldsymbol{V}\rvert = n$ vertices, $\hat{\mathcal{P}}(\boldsymbol{T}) = \mathcal{I}(\boldsymbol{T})$, and $\hat{\mathcal{P}}(\boldsymbol{V}) = \ldots$	**Number of nodes** Number of *vertex* subsets or *multiedge* subsets		**Number of links** Number of *vertex* coverings or *multiedge* coverings	
	$\lvert\hat{\mathcal{P}}(\boldsymbol{V})\rvert$	$\lvert\hat{\mathcal{P}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})\rvert$	$\lvert\hat{\mathfrak{C}}(\boldsymbol{V})\rvert$	$\lvert\hat{\mathfrak{C}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})\rvert$
$\mathcal{P}(\boldsymbol{V})$ (complete set)	$\Theta(2^n)$	$\Theta(4^n)$	$\Theta(3^n)$	$\Theta(6^n)$
$\mathcal{I}(\boldsymbol{V})$ (set of intervals)	$\Theta(n^2)$	$\Theta(n^6)$	$\Theta(n^3)$	$\Theta(n^7)$
$\mathcal{H}(\boldsymbol{V})$ (hierarchy)	$\Theta(n)$	$\Theta(n^4)$	$\Theta(n)$	$\Theta(n^5)$

Table 1: Size of the poset structure in terms of nodes (columns 1 and 2) and in terms of in-coming links (columns 3 and 4) for a unidimensional poset structure (columns 1 and 3) and for a tridimensional poset structure (columns 2 and 4) when different types of feasibility constraints apply to the set of vertex subsets (rows). The set of feasible time subsets is here assumed to be the set of time intervals: $\hat{\mathcal{P}}(\boldsymbol{T}) = \mathcal{I}(\boldsymbol{T}) \;\Rightarrow\; \lvert\hat{\mathcal{P}}(\boldsymbol{T})\rvert = \Theta(n)$ and $\lvert\hat{\mathfrak{C}}(\boldsymbol{T})\rvert = \Theta(n^3)$.

Complexity of the resulting optimisation algorithm. The space complexity of the resulting optimisation algorithm is given in Table 1 by the number of nodes $\lvert\hat{\mathcal{P}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})\rvert$ and by the number of links $\lvert\hat{\mathfrak{C}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})\rvert$ that are encoded in the tridimensional poset structure. It is exponential in the worst case, that is when all vertex subsets are feasible, polynomial of order 7 in the case of a set of vertex intervals, and polynomial of order 5 in the case of a vertex hierarchy.

The unidimensional poset structures, encoding the set of feasible vertex subsets and the set of time intervals, are considered as inputs of the optimisation problem and their building cost is hence not taken into account in the algorithm's complexity, although it is quite cheap and straightforward in the case of sets of intervals and hierarchies. Building the corresponding tridimensional poset structure requires as many operations as there are nodes and in-coming links. Filling it with the values of the cost function requires as many operations as there are in-coming links, thanks to the recursive decomposition of costs. Hence, the time complexity to build the overall data structure is equivalent to its space complexity.

Finally, when applied to the maximal element, that is to the multiedge set $\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T}$, Algorithm 2 is then recursively applied once to all feasible multiedge subsets $V\times V'\times T \in \hat{\mathcal{P}}(\boldsymbol{V}\times\boldsymbol{V}\times\boldsymbol{T})$. The bottleneck is then the summation of costs that are retrieved by the recursive calls, for each multiedge subset of each covered partition. Hence, here again, there are as many such operations as they are in-coming links in the tridimensional poset structure, so the overall time complexity of the approach is equivalent to the space complexity of the data structure we presented.

5. Conclusion

This article presents a formal framework for the compression of temporal graphs. By summarising homogeneous parts of the graph and replacing them with more general structural patterns, compression allows to reduce its description length while preserving its information content. This framework first builds on a simple and limited combinatorial problem, that we call the *Lossless Graph Compression Problem*, which exploits the (most classical) structural equivalence relation between vertices for the exact compression of simple graphs. Among the proposed generalisations to address the more complex *Lossy Multistream Compression Problem*, dealing with the approximated compression of temporal multigraphs, three main contributions are worth mentioning:

- The definition of an information-theoretic measure, relying on a proper formalisation of a multigraph stochastic model, to quantify and to control the information that is lost during compression, while also taking into account additional information that might be reinjected during the decompression step;
- The enhancement of the solution space of the initial problem (i) by defining a less constrained partitioning of the graph working at the multiedge level instead of the vertex level, and (ii) by allowing to express and to preserve additional vertex structures during compression;
- The generalisation from static graphs to temporal graphs by exploiting the *link stream* representation, which consists in the extension of the set of multiedges by a third dimension representing the temporal evolution of these edges, thus allowing the natural extension of all notions that have been previously introduced.

Building on a previous algorithmic framework to solve special versions of the *Set Partitioning Problem*, an exact algorithm is finally introduced for the Lossy MSCP. While it is exponential in the worst case, it is showed to be polynomial when particular vertex structures are assumed. Yet, in order to achieve the compression of large-scale temporal graphs, future research would need to work on heuristics for the approximate solving of the Lossy MSCP. This would require the definition of adequate operators on multiedge partitions to browse the solution space, taking into account its particular algebraic structure to efficiently evaluate slight modifications of the considered partitions, and to thus proceed to a greedy search for a local optima. Another improvement, regarding the current implementation of the optimisation algorithm, would consists in the acknowledgement that most link streams that are considered in empirical research are quite sparse, meaning that the support of the edge function is quite small. Hence, the data structures proposed in this article would benefit from a sparse representation of the data to decrease computation resources in real-case applications of this framework.

Acknowledgement

The author sincerely thanks Yves Demazeau, Damien Dosimont, Matthieu Latapy, Léonard Panichi, Hindol Rakshit, Fabrice Rossi, Mridul Seth, Catherine Matias, Lucas Mello Schnorr, Lionel Tabourier, Fabien Tarissan, Tiphaine Viard, and Jean-Marc Vincent, for their valuable feedback. He also gratefully thanks all the members of Prof. Grasset's team for their warm welcome within their premises. This work has been partially funded by the European Commission H2020 FETPROACT 2016-2017 program under grant 732942 (ODYCCEUS) and by the French National Agency of Research (ANR) under grant ANR-15-CE38-0001 (AlgoDiv).

References

References

[1] F. Zhou, S. Mahler, H. Toivonen, Review of Network Abstraction Techniques, in: Workshop on Explorative Analytics of Information Networks at ECML PKDD, 2009, pp. 50–63.
[2] H. Toivonen, F. Zhou, A. Hartikainen, A. Hinkka, Compression of Weighted Graphs, in: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'11), ACM, New York, NY, USA, 2011, pp. 965–973.
[3] P. Serafino, Speeding Up Graph Clustering via Modular Decomposition Based Compression, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC'13), ACM, New York, NY, USA, 2013, pp. 156–163.
[4] S. Navlakha, R. Rastogi, N. Shrivastava, Graph Summarization with Bounded Error, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD'08), ACM, New York, NY, USA, 2008, pp. 419–432.
[5] N. Zhang, Y. Tian, J. M. Patel, Discovery-Driven Graph Summarization, in: Proceedings of the 26th International Conference on Data Engineering (ICDE'2010), 2010, pp. 880–891.
[6] K. LeFevre, E. Terzi, GraSS: Graph Structure Summarization, in: Proceedings of the SIAM International Conference on Data Mining (SDM'2010), 2010, pp. 454–465.
[7] B. Pinaud, G. Melançon, J. Dubois, PORGY: A Visual Graph Rewriting Environment for Complex Systems, Computer Graphics Forum 31 (3) (2012) 1265–1274.
[8] T. Dwyer, N. H. Riche, K. Marriott, C. Mears, Edge Compression Techniques for Visualization of Dense Directed Graphs, IEEE Transactions on Visualization and Computer Graphics 19 (12) (2013) 2596–2605.
[9] S. E. Ahnert, Generalised power graph compression reveals dominant relationship patterns in complex networks, Scientific Reports 4 (4385).
[10] P. Holme, J. Saramäki (Eds.), Temporal Networks, Understanding Complex Systems, Springer-Verlag Berlin Heidelberg, 2013.
[11] T. Viard, M. Latapy, C. Magnien, Computing maximal cliques in link streams, Theoretical Computer Science 609 (2016) 245252.
[12] M. Latapy, T. Viard, C. Magnien, Stream Graphs and Link Streams for the Modeling of Interactions over Time, arXiv:1710.04073.
[13] S. P. Borgatti, M. G. Everett, Regular blockmodels of multiway, multimode matrices, Social Networks 14 (1) (1992) 91–120.
[14] N. Narmadha, R. Rathipriya, Triclustering: An Evolution of Clustering, in: Proceedings of the Online International Conference on Green Engineering and Technologies (IC-GET'16), 2016, pp. 1–4.
[15] R. Guigourès, M. Boullé, F. Rossi, A Triclustering Approach for Time Evolving Graphs, in: Co-clustering and Applications International Conference on Data Mining Workshop, IEEE, Brussels, Belgium, 2012, pp. 115–122.
[16] F. Lorrain, H. C. White, Structural equivalence of individuals in social networks, The Journal of Mathematical Sociology 1 (1) (1971) 49–80.
[17] E. Balas, M. W. Padberg, Set Partitioning: A Survey, SIAM Review 18 (4) (1976) 710–760.
[18] R. Lamarche-Perrin, Y. Demazeau, J.-M. Vincent, A Generic Algorithmic Framework to Solve Special Versions of the Set Partitioning Problem, in: A. Andreou, G. A. Papadopoulos (Eds.), Proceedings of the 2014 IEEE 26th International Conference on Tools with Artificial Intelligence (ICTAI'14), IEEE Computer Society, 2014, pp. 891–897.
[19] D. Dosimont, R. Lamarche-Perrin, L. M. Schnorr, G. Huard, J.-M. Vincent, A Spatiotemporal Data Aggregation Technique for Performance Analysis of Large-scale Execution Traces, in: M. S. Pérez, G. Antoniu, K. Keahey (Eds.), Proceedings of the 2014 IEEE International Conference on Cluster Computing (CLUSTER'14), IEEE Computer Society, 2014, pp. 149–157.

[20] V. Batagelj, A. Ferligoj, P. Doreian, Direct and indirect methods for structural equivalence, Social Networks 14 (1) (1992) 63–90, special Issue on Blockmodels.
[21] R. A. Hanneman, M. Riddle, Introduction to social network methods, University of California, Riverside, CA, 2005.
[22] S. Fortunato, Community detection in graphs, Physics Reports 486 (3) (2010) 75–174.
[23] I. S. Dhillon, S. Mallela, D. S. Modha, Information-theoretic Co-clustering, in: Proceedings of the Ninth ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'03), ACM, New York, NY, USA, 2003, pp. 89–98.
[24] R. Lamarche-Perrin, Y. Demazeau, J.-M. Vincent, Building Optimal Macroscopic Representations of Complex Multi-agent Systems. Application to the Spatial and Temporal Analysis of International Relations through News Aggregation, in: N. T. Nguyen, R. Kowalczyk, J. M. Corchado, J. Bajo (Eds.), Transactions on Computational Collective Intelligence, Vol. XV of LNCS 8670, Springer-Verlag Berlin, Heidelberg, 2014, pp. 1–27.
[25] R. v. d. Hofstad, Configuration Model, Vol. 1, Cambridge University Press, 2016, Ch. III.7, p. 216255.
[26] S. Kullback, R. A. Leibler, On Information and Sufficiency, The Annals of Mathematical Statistics 22 (1) (1951) 79–86.
[27] T. M. Cover, J. A. Thomas, Elements of Information Theory, 2nd Edition, John Wiley & Sons, Inc., Hoboken, NJ, 2006.
[28] H. He, A. K. Singh, Closure-Tree: An Index Structure for Graph Queries, in: 22nd International Conference on Data Engineering (ICDE'06), 2006, pp. 38–38.
[29] T. Feder, R. Motwani, Clique Partitions, Graph Compression and Speeding-Up Algorithms, Journal of Computer and System Sciences 51 (2) (1995) 261–272.
[30] C. Hernández, G. Navarro, Compressed Representation of Web and Social Networks via Dense Subgraphs, in: L. Calderón-Benavides, C. González-Caro, E. Chávez, N. Ziviani (Eds.), String Processing and Information Retrieval, Vol. 7608 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 264–276.
[31] M. Dhabu, P. S. Deshpande, S. Vishwakarma, Partition based graph compression, International Journal of Advanced Computer Science and Applications 4 (9) (2013) 7–12.
[32] D. M. Roy, T. Y. Whye, The Mondrian Process, in: Advances in Neural Information Processing Systems, Vol. 21, Curran Associates, Inc., 2009, pp. 1377–1384.
[33] R. Lamarche-Perrin, Y. Demazeau, J.-M. Vincent, A Generic Algorithmic Framework to Solve Special Versions of the Set Partiioning Problem, Tech. Rep. MIS-Preprint 105/2014, Max Planck Institute for Mathematics in the Sciences, Leipzig, Germany (2014).
[34] T. Sandholm, K. Larson, M. Andersson, O. Shehory, F. Tohmé, Coalition structure generation with worst case guarantees, Artificial Intelligence 111 (1-2) (1999) 209–238.
[35] T. Rahwan, N. R. Jennings, Coalition Structure Generation: Dynamic Programming Meets Anytime Optimisation, in: Proceedings of the Twenty-third Conference on Artificial Intelligence, AAAI, 2008, pp. 156–161.
[36] T. Sandholm, Algorithm for optimal winner determination in combinatorial auctions, Artificial Intelligence 135 (2002) 1–54.
[37] P. Pons, M. Latapy, Post-processing hierarchical community structures: Quality improvements and multi-scale view, Theoretical Computer Science 412 (8-10) (2011) 892–900. doi:http://dx.doi.org/10.1016/j.tcs.2010.11.041.
[38] B. Cloitre, Sequence A003095, in: The On-Line Encyclopedia of Integer Sequences, `http://oeis.org/A003095`, 2002.
[39] G. McGarvey, Sequence A135361, in: The On-Line Encyclopedia of Integer Sequences, `http://oeis.org/A135361`, 2007.
[40] S. Anily, A. Federgruen, Structured Partitioning Problems, Operations Research 39 (1) (1991) 130–149.
[41] B. Jackson, J.D. Scargle, D. Barnes, S. Arabhi, A. Alt, *et al.*, An algorithm for optimal partitioning of data on an interval, IEEE Signal Processing Letters 12 (2) (2005) 105–108.
[42] G. Pagano, D. Dosimont, G. Huard, V. Marangozova-Martin, J.-M. Vincent, Trace Management and Analysis for Embedded Systems, in: Proceedings of the 7th International Symposium on Embedded Multicore SoCs (MCSoC'13), IEEE Computer Society Press, 2013, pp. 119–122.
[43] M. H. Rothkopf, A. Pekeč, R. M. Harstad, Computationally Manageable Combinational Auctions, Management Science 44 (8) (1998) 1131–1147.
[44] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, Network Flows: Theory, Algorithms, and Applications, Prentice-Hall, Inc., 1993, Ch. Lagrangian Relaxation and Network Optimization, pp. 598–648.
[45] A. K. Chakravarty, J. B. Orlin, U. G. Rothblum, A partitioning problem with additive objective with an application to optimal inventory groupings for joint replenishment, Operations Research 30 (5) (1982) 1018–1022.
[46] R. V. V. Vidal, Optimal Partition of an Interval – The Discrete Version, in: R. V. V. Vidal (Ed.), Applied Simulated Annealing, Vol. 396 of Lecture Notes in Economics and Mathematical Systems, Springer Berlin Heidelberg, 1993, pp. 291–312.
[47] B. Davey, H. Priestley, Introduction to Lattices and Order, 2nd Edition, Cambridge University Press, (2002).

Appendix A. Solving the Set Partitioning Problem

This appendix provides a detailed description of the algebraic properties of the *Set Partitioning Problem* (SPP), leading to a principle of optimality that we exploit to derive the algorithm presented in Subsection 4.2 of this article. This part of our work has been previously published in the proceedings of the 2014 IEEE International Conference on Tools with Artificial Intelligence (ICTAI'14) [18] and detailed in a research report of the Max Planck Institute for Mathematics in the Sciences [33].

Algebraic Structure of the Solution Space. Given a set of objects $\boldsymbol{X} = \{x_1, \ldots, x_n\}$ and a set of feasible subsets $\hat{\mathcal{P}}(\boldsymbol{X}) \subseteq \mathcal{P}(\boldsymbol{X})$, the resulting set of feasible partitions $\hat{\mathfrak{P}}(\boldsymbol{X}) \subseteq \mathfrak{P}(\boldsymbol{X})$ is structured by an essential algebraic relation, usually referred to as the *refinement relation* $\subseteq$ [47].

Definition 1 (Refinement Relation)**.**
Partition $\mathcal{X} \in \hat{\mathfrak{P}}(\boldsymbol{X})$ refines *partition $\mathcal{Y} \in \hat{\mathfrak{P}}(\boldsymbol{X})$, and we mark $\mathcal{X} \subseteq \mathcal{Y}$, if and only if each subset in $\mathcal{X}$ is a subset of a subset in $\mathcal{Y}$:*

$$\mathcal{X} \subseteq \mathcal{Y} \quad \Leftrightarrow \quad \forall X \in \mathcal{X}, \quad \exists Y \in \mathcal{Y}, \quad X \subseteq Y$$

Given a partition $\mathcal{X} \in \hat{\mathfrak{P}}(\boldsymbol{X})$, we define $\hat{\mathfrak{R}}(\mathcal{X}) \subseteq \hat{\mathfrak{P}}(\boldsymbol{X})$ as the set of feasible partitions refining *$\mathcal{X}$:*

$$\hat{\mathfrak{R}}(\mathcal{X}) \;=\; \{\mathcal{Y} \in \hat{\mathfrak{P}}(\boldsymbol{X}) : \mathcal{Y} \subseteq \mathcal{X}\}$$

As this binary relation is reflexive, antisymmetric, and transitive, it defines a partial order on the partition set $\hat{\mathfrak{P}}(\boldsymbol{X})$ that consequently forms a poset and can be represented as a Hasse diagram [47]. In particular, if the *minimal partition* $\mathcal{X}_\perp = \{\{x_1\}, \ldots, \{x_n\}\}$ is feasible, then it refines all feasible partitions:

$$\forall x \in \boldsymbol{X},\ \{x\} \in \hat{\mathcal{P}}(\boldsymbol{X}) \qquad \Rightarrow \qquad \forall \mathcal{X} \in \hat{\mathfrak{P}}(\boldsymbol{X}), \quad \mathcal{X}_\perp \in \hat{\mathfrak{R}}(\mathcal{X})$$

and if the *maximal partition* $\mathcal{X}_\top = \{\{x_1, \ldots, x_n\}\}$ is feasible, then it is refined by all feasible partitions:

$$\boldsymbol{X} \in \hat{\mathcal{P}}(\boldsymbol{X}) \qquad \Rightarrow \qquad \hat{\mathfrak{R}}(\mathcal{X}_\top) = \hat{\mathfrak{P}}(\boldsymbol{X})$$

The *covering relation* $\sqsubset$ is the transitive reduction of the refinement relation, that is the binary relation which holds between immediate "neighbours" with respect to $\subseteq$.

Definition 2 (Covering relation)**.**
Partition $\mathcal{X} \in \hat{\mathfrak{P}}(\boldsymbol{X})$ is covered *by partition $\mathcal{Y} \in \hat{\mathfrak{P}}(\boldsymbol{X})$, and we mark $\mathcal{X} \sqsubset \mathcal{Y}$, if and only if $\mathcal{X}$ and $\mathcal{Y}$ are different, $\mathcal{X}$ refines $\mathcal{Y}$, and there is no other feasible partition "in-between":*

$$\mathcal{X} \sqsubset \mathcal{Y} \quad \Leftrightarrow \quad \mathcal{X} \subsetneq \mathcal{Y} \quad \text{and} \quad \nexists \mathcal{Z} \in \hat{\mathfrak{P}}(\boldsymbol{X}),\ \mathcal{X} \subsetneq \mathcal{Z} \subsetneq \mathcal{Y}$$

Given a partition $\mathcal{X} \in \hat{\mathfrak{P}}(\boldsymbol{X})$, we define $\hat{\mathfrak{C}}(\mathcal{X})$ as the set of feasible partitions covered by *$\mathcal{X}$:*

$$\hat{\mathfrak{C}}(\mathcal{X}) \;=\; \{\mathcal{Y} \in \hat{\mathfrak{P}}(\boldsymbol{X}) : \mathcal{Y} \sqsubset \mathcal{X}\}$$

As it is shown in what follows, these two relations give essential algebraic tools to cleverly search for optimal partitions in $\hat{\mathfrak{P}}(\boldsymbol{X})$.

Principle of Optimality. In dynamic programming, finding a principle of optimality consists in showing that the solution space has an optimal substructure, that is that the solution to the optimisation problem can be obtained by recursively combining locally-optimal solutions to several subproblems. Intuitively, in the case of the SPP, one can rely on the fact that *the union of optimal partitions of subsets forms an interesting candidate partition of the union of these subsets* [33]. Hence, by appropriately decomposing the initial set into subsets, one might provide a computationally efficient procedure to recursively build such an optimal solution (see Figure A.8 for an illustration of the following principle).

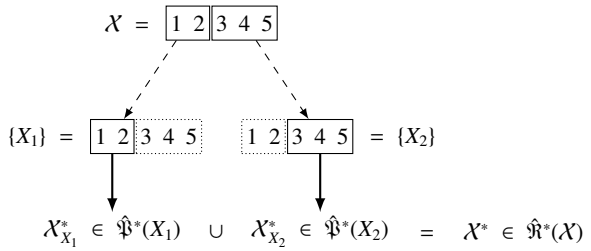


Figure A.8: Recursively solving the SPP by applying the principle of optimality on a partition $\mathcal{X}$ to find an optimal partition among its refinements (see Equation A.1)

Theorem 1 (Principle of Optimality [33])**.**
Let $\boldsymbol{X}$ be a set of objects, $\hat{\mathcal{P}}(\boldsymbol{X})$ a set of feasible subsets, $\hat{\mathfrak{P}}(\boldsymbol{X})$ the resulting set of feasible partitions, and $c : \hat{\mathcal{P}}(\boldsymbol{X}) \to \mathbb{R}^+$ a cost function defining partition optimality. For any feasible partition $\mathcal{X} \in \hat{\mathfrak{P}}(\boldsymbol{X})$, the union of locally-optimal feasible partitions of the subsets in $\mathcal{X}$ is optimal among the feasible refinements of $\mathcal{X}$:

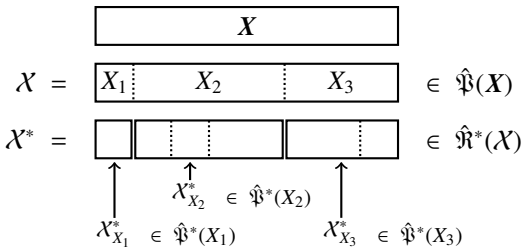
$$\forall X \in \mathcal{X},\ \mathcal{X}^*_X \in \hat{\mathfrak{P}}^*(X) \quad \Rightarrow \quad \left(\bigcup_{X\in\mathcal{X}} \mathcal{X}^*_X\right) \in \hat{\mathfrak{R}}^*(\mathcal{X}), \tag{A.1}$$

where $\hat{\mathfrak{P}}^*(X)$ is the set of optimal feasible partitions of X and $\hat{\mathfrak{R}}^*(\mathcal{X})$ is the set of optimal feasible partitions refining $\mathcal{X}$.

Proof. Let $\mathcal{X} \in \hat{\mathfrak{P}}(\boldsymbol{X})$ be a feasible partition of $\boldsymbol{X}$ and, for all $X \in \mathcal{X}$, let $\mathcal{X}^*_X \in \hat{\mathfrak{P}}^*(X)$ be a locally-optimal feasible partition of X, meaning that

$$\forall \mathcal{X}'_X \in \hat{\mathfrak{P}}(X), \quad c(\mathcal{X}^*_X) \;=\; \sum_{X^*\in\mathcal{X}^*_X} c(X^*) \;\leq\; \sum_{X'\in\mathcal{X}'_X} c(X') \;=\; c(\mathcal{X}'_X). \tag{α}$$

In the following, we mark $\mathcal{X}^* = \bigcup_{X\in\mathcal{X}} \mathcal{X}^*_X$. Here is an example of such setting:



First, since any subset $X^* \in \mathcal{X}^*_X$ is a feasible subset of $X \in \mathcal{X}$, then $\mathcal{X}^*$ is a feasible refinement of $\mathcal{X}$: $\mathcal{X}^* \in \hat{\mathfrak{R}}(\mathcal{X})$. Second, let $\mathcal{X}' = \bigcup_{X\in\mathcal{X}} \mathcal{X}'_X$ with $\mathcal{X}'_X \in \hat{\mathfrak{P}}(X)$ be another feasible refinement of $\mathcal{X}$. We then have, applying Equation α:

$$c(\mathcal{X}^*) \;=\; \sum_{X\in\mathcal{X}} c(\mathcal{X}^*_X) \;\leq\; \sum_{X\in\mathcal{X}} c(\mathcal{X}'_X) \;=\; c(\mathcal{X}'). \tag{β}$$

Therefore, $\mathcal{X}^*$ is optimal among the feasible refinements of $\mathcal{X}$: $\mathcal{X}^* \in \hat{\mathfrak{R}}^*(\mathcal{X})$. □

Branching the Solution Space. Given a feasible subset $X \in \hat{\mathcal{P}}(\boldsymbol{X})$ for which one wants to compute a locally-optimal feasible partition $\mathcal{X}^* \in \hat{\mathfrak{P}}^*(X)$, a *branching* first consists in building subspaces $\hat{\mathfrak{P}}_i(X) \subseteq \hat{\mathfrak{P}}(X)$ that cover the solution space:

$$\hat{\mathfrak{P}}_1(X) \cup \ldots \cup \hat{\mathfrak{P}}_k(X) \quad = \quad \hat{\mathfrak{P}}(X).$$

Then, once one has found locally-optimal partitions within these subspaces $\mathcal{X}_1^* \in \hat{\mathfrak{P}}_1^*(X), \ldots, \mathcal{X}_k^* \in \hat{\mathfrak{P}}_k^*(X)$, one can easily solve the global optimisation problem by simply choosing among these local solutions one that minimises the objective:

$$\underset{\mathcal{X} \in \{\mathcal{X}_1^*, \ldots, \mathcal{X}_k^*\}}{\arg\min} \; c(\mathcal{X}) \quad \subseteq \quad \hat{\mathfrak{P}}^*(X). \tag{A.2}$$

For that purpose, the covering relation provides "elementary steps" to branch the solution space, each branch corresponding to a direction to go down in the partition poset. First, assuming that the maximal partition $\{X\}$ is feasible[7], we know that all feasible partitions of X refine the maximal partition $\{X\}$:

$$\hat{\mathfrak{P}}(X) \quad = \quad \hat{\mathfrak{R}}(\{X\}).$$

Second, for any such feasible partition $\mathcal{X} \in \hat{\mathfrak{P}}(X)$, a refining partition of $\mathcal{X}$ is either *the partition $\mathcal{X}$ itself*, or *a partition that refines a partition covered by $\mathcal{X}$*. Hence, the solution space can be branched the following way (see Figure A.9 for an illustration of such branching):

$$\hat{\mathfrak{P}}(X) \quad = \quad \{\{X\}\} \cup \left(\bigcup_{\mathcal{X} \in \hat{\mathfrak{C}}(\{X\})} \hat{\mathfrak{R}}(\mathcal{X}) \right) \tag{A.3}$$

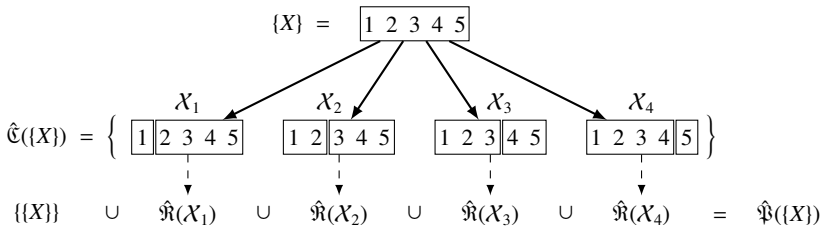


Figure A.9: Branching the solution space according to the refinement and the covering relations (see Equation A.3)

Recursive Algorithm. Computing an optimal partition of $\boldsymbol{X}$ thus consists in computing locally-optimal partitions among the ones that are refining the partitions covered by $\{\boldsymbol{X}\}$. Thanks to the principle of optimality, such a computation can be recursively performed by applying the same principle to the subsets of these covered partitions. Hence, the branching and recursion equations (see Equations A.1, A.2, and A.3) allow to define a divide and conquer algorithm that computes a locally-optimal partition $\mathcal{X}^* \in \hat{\mathfrak{P}}^*(X)$ for any $X \in \hat{\mathcal{P}}(\boldsymbol{X})$ by applying the following recursive formula:

$$\underset{\mathcal{X} \in \{\{X\}\} \cup \left(\bigcup_{\mathcal{X} \in \hat{\mathfrak{C}}(\{X\})} \left\{ \bigcup_{X' \in \mathcal{X}} \mathcal{X}^*_{X'} \right\} \right)}{\arg\min} \quad c(\mathcal{X}) \quad \subseteq \quad \hat{\mathfrak{P}}^*(X) \tag{A.4}$$

where $\mathcal{X}^*_X$ designates an optimal feasible partition of X, that is $\mathcal{X}^*_X \in \hat{\mathfrak{P}}^*(X)$.

Moreover, in the dynamic programming paradigm, recursive algorithms can be easily improved by *memoization*, that is by recording the results of time-consuming recursive calls [43, 36]. For each subset on which the algorithm is once applied, by keeping trace of the resulting locally-optimal partition, one can immediately return this result when posterior calls occur on the same subset. This way, the algorithm is applied only once to each feasible subset $X \in \hat{\mathcal{P}}(\boldsymbol{X})$.

The resulting algorithm is provided in Subsection 4.2 of this article.

[7] If this is not the case for the initial set $\boldsymbol{X}$, the following approach can easily be generalised by sequentially applying the algorithm to all maximal partitions, that is maximal elements in the poset of partitions induced by the refinement relation.