
UNIVERSITÀ
DI SIENA
1240

Department of Engineering and Mathematical Sciences

Ph.D Course in
INFORMATION ENGINEERING AND SCIENCE
Thematic Area: Discrete Mathematics and Computing Science

# Algorithms for identifying clusters in temporal graphs and realising distance matrices by unicyclic graphs.

**Supervisor:**
**Prof.** ***Cristiano Bocci***

**Co-Supervisor:**
**Prof.** ***Kitty Meeks***

**Candidate:**
***Chiara Capresi***

**Cycle XXXIV**

# Abstract

In this thesis I present some results, which mainly concern finding algorithms for graphs problems, to which I worked on during my Ph.D with the supervision of my supervisors. In particular, as a first theme, it is presented a new temporal interpretation of the well studied Cluster Editing which we called Editing to Temporal Cliques (ETC). In this regard, in this work, it is proved that ETC is NP-Complete even if we restrict the possible inputs to the class of temporal graphs with a path as their underlying graph. Furthermore, it is presented a result that shows that ETC is instead tractable in polynomial time if the underlying graph is a path and the maximum number of appearances allowed for each of the edges of that path is fixed. Taking in mind the known key observation that a static graph is a cluster graph if and only if it does not contain any induced $P_3$, it is presented a local characterisation for cluster temporal graphs. This characterisation establishes that a temporal graph is a cluster temporal graph if and only if every subset of at most five vertices induces a cluster temporal graph. Using this characterisation, we obtain an FPT algorithm for ETC parameterised simultaneously by the number of modifications and the lifetime (number of timesteps) of the input temporal graph. Furthermore, it is shown via a counterexample, that a cluster temporal graph can not be properly characterised by sets of at most four vertices.

In the last Chapter of this thesis, at first it is proven a result on the realisation of distance matrices by $n-$cycles and then it is developed an algorithm that allows to establish if a given distance matrix $D$ can or cannot be realised by a weighted unicyclic graph or at least by a weighted tree. In case of affirmative answer, a second part of the algorithm reconstructs that graph. The algorithm takes $\mathcal{O}(n^4)$. Furthermore, it is shown that if the algorithm returns a unicyclic graph as a realisation of $D$, then this realisation is optimal.

# Contents

# Introduction

Graphs are powerful combinatorial structures which can be used to model many real settings where it is crucial to represent the connections that exist between pairs of elements of a given set. Hence, they are used to represent problems in many different scientific areas, such as Epidemiology, Computer Science or Phylogenetics. Furthermore, graphs turn out to be very adaptable instruments, so much that they allow to approach to a given problem from various points of view, depending on its characteristics and on the target to reach. Cluster graphs, for instance, are useful structures to represent sets whose elements are divided into disjoint and fully connected subsets. Nevertheless, for certain real world problems, such as in the Epidemiology context as in many other settings, it is crucial for a proper analysis of the problem also to be able to specify a sort of order at which contacts between pairs of elements actually happens. In this case, are necessary more complex structures than graphs or cluster graphs. In particular, would be required kind of graphs that admit and take in consideration discrete changes over time in the interactions between pairs of elements. More precisely, structures that turns out to be more suitable to this purpose are for instance temporal graphs and temporal cluster graphs.

In other settings instead, such as in Phylogenetics, it can be useful to start from a quantitative estimate of the similarities between pairs of elements that can be collected by means of a distance matrix and then ask to be able to represent this problem situation by means of a weighted graph (so one for which to each edge it is associated a positive number that estimates the similarity existing between the two endpoints).

In this thesis, I will start recalling some of the most important concepts about Graph Theory in Chapter 1, together with an introduction to the more recent field of Temporal Graphs. In the same Chapter, I will also give a brief citation of known theoretical facts about Complexity Classes and Parameterised Complexity which turn out to be of a certain interest for the contents of this thesis, in addition to a brief discussion about the algorithmic techniques of dynamic programming and bounded search tree algorithms.

The first two main topics that I will discuss in this thesis, in Chapter 2 and Chapter 3 respectively, both concern the coming up studying area of temporal graphs, that is to say graphs that change their structure at different discrete instants of time. In particular, together with my

supervisors and the post-doctoral research associate Dr. John Sylvester (University of Glasgow), I worked on two different problems, both related in some way to temporal cluster graphs; that is to say, temporal graphs which can be partitioned into clusters pairwise independent in time. These particular temporal graphs that will be formally defined in Chapter 2, are a natural generalisation of cluster graphs - static vertex-disjoint unions of complete graphs. These structures can be significantly useful to study the density of contacts that happen among the elements of a given set whose interactions are subject to discrete changes over time and where the order at which these contacts take place turns out to be particularly relevant for a correct analysis of the situation. In particular, in Chapter 2 we formulate a generalisation of the well studied Cluster Editing problem (CE) in the setting of temporal graphs. Given as input a graph $G$ and a positive integer $k$, Cluster Editing asks whether it is possible to obtain a cluster graph, by at most $k$ edge modifications. Each of these edge modifications can be either an edge deletion or an edge addition. This problem is known to be NP-Complete in general [4, 17, 33, 42] and we refer to the "Overview and related works" section in Chapter 2 for more details about complexity results existing for it. Our generalisation of CE to temporal graphs is not the first attempt that have been done in this direction.

A first example is due to Chen, Molter, Sorge and Suchy [14], who recently introduced the Temporal Cluster Editing problem. In this work, the authors' aim is that of obtaining a temporal graph where the graph that appears at each *snapshot* - discrete unit of time in a temporal graph - is a cluster graph, imposing restrictions both on the number of modifications that are allowed to be made at each snapshot and to the differences between cluster graphs created at consecutive snapshots. Another version of the problem, which has recently been studied by Luo, Molter, Nichterlein and Niedermeier [35], is named Dynamic Cluster Editing. Here, together with a solution to a certain instance, it is given also a second instance and the problem asks to find a solution for the second instance that does not "differ too much" from the first.

In parallel with the static case, where clusters are essentially complete graphs also called cliques, our approach is more oriented in the direction of giving a temporal definition of what a complete temporal graph should be. To this end we refer to definitions of *temporal cliques* that already exist in literature. The definition we use for temporal cliques is essentially given in terms of a set of vertices and a time interval that must respect the requirement that each pair of vertices must be connected with a certain frequency, even if not necessarily continuously, during the specified interval. This definition is essentially the same definition given by Viard et al. in [48], while differs a bit from that one given in [27] because we do not impose a minimum length for the interval. Under these assumptions, we define a *temporal cluster graph* as a union of pairwise *independent* temporal cliques. It will not be a surprise the fact that in the temporal setting the concept of independence between temporal cliques is much more

complicated than the simpler "vertex disjoint" requirement of the static case. In particular, we will consider two temporal cliques to be independent either if their vertex sets are disjoint or if the two time intervals that respectively define them are enough separated (similar to the notion of independence used to define temporal matchings [39]).

Our temporal interpretation of Cluster Editing, which is introduced in Chapter 2, takes the name of Editing to Temporal Cliques (ETC) and asks for a given temporal graph $\mathcal{G}$ and a positive integer $k$ if it is possible applying at most $k$ time-edge modifications (additions or deletions of time-edges) to obtain a cluster temporal graph. In the same chapter, in addition to the proof of some useful results on temporal graphs that will be widely used throughout this thesis, we show that ETC is NP-Complete even when the underlying graph is a path. We will prove this fact through a reduction from the ($\Delta$-)Temporal Matching problem with restrictions on the parameter $\Delta$. Despite this, we also show that ETC is solvable in polynomial time if the maximum number of possible appearances of any edge of the underlying graph during the lifetime of the given temporal graph $\mathcal{G}$ is bounded. It follows immediately from our hardness reduction that the variant of the problem in which we are only allowed to delete, but not add, edge appearances, is also hard in the same setting.

Still remaining in the ambit of clusters, in the static case, a quite known fact, also reported in [9], is that a cluster graph is uniquely identified by the class of all the graphs that contain no induced copy of the three-vertex path $P_3$. In Chapter 3 we prove that for temporal cluster graph, instead, any local characterisation that involves only sets of at most four vertices cannot properly characterise them. Furthermore, we prove that a temporal graph is a cluster temporal graph if and only if every subset of at most five vertices induces a cluster temporal graph. Using this characterisation, we obtain an FPT algorithm for ETC parameterised simultaneously by the number of modifications and the lifetime (# of timesteps) of the input temporal graph.

In Chapter 4 it is presented a further result about the problem of realisation of distance matrices by unicyclic weighted graphs. This problem turns out to be widely known and discussed for matrices that realise trees. Among the main results that can be found for the problem of realisation of distance matrices through trees we mention the Tree Metric Theorem [11] which is based on a necessary and sufficient condition for a matrix to be realised by a tree, known as *four-point condition*. Even if relevant, results of realisations of distance matrices by trees risk not to be very much useful for real data since they are rarely modelisable by such simple structures of graph. For this reason many authors, are trying to extend this result to more complex classes of graphs. The problem of realisation for some specific classes of graphs, such as paths, caterpillars, cycles, bipartite, complete, planar and cactus graphs are studied for example in [2] and [26]. Our contribution, which I discuss in the final chapter of this thesis, is that of giving a general result on realisation of $n$-cycles and proposing an $O(n^4)$-time algorithm that in

a first part (*Analysis* algorithm), takes in input a distance matrix $D$ and, through an iterative process of compaction and analysis of the compaction vector and reduction matrix step by step obtained, answers to the question if the given matrix is or not realisable by a unicyclic graph. At the end of the process, in case of positive answer, a second part of the algorithm (*Reconstruction* algorithm) returns a draw of a weighted graph that realises $D$. In Proposition 4.11, we also show that if the algorithm outputs a unicyclic graph realising $D$, then this realisation is optimal. Note that the first algorithm will return an affirmative answer even if the matrix $D$ realises a tree. In which case, obviously at the end of the process a tree will be drawn.

# Chapter 1

# Preliminaries

In this Chapter we are going to recall a few fundamental concepts that will be extensively used throughout this thesis. In particular, we will start gathering together a few known fundamentals of Graph Theory and mentioning some basic results from Computational Complexity and Parameterised Complexity Theory. Most of these contents are discussed in more detail in [24, 32]. We will introduce in this Chapter also a section entirely dedicated to temporal graphs (see also [40]), that will be of crucial interest in Chapter 2 and in Chapter 3. First of all, let us introduce some notation that will be used through all the sections. Let $\mathbb{N}$ denote the natural numbers (with zero) and $\mathbb{Z}^+$ denote the positive integers. We refer to a set of consecutive natural numbers $[i, j] = \{i, i + 1, \ldots, j\}$ for some $i, j \in \mathbb{N}$ with $i \leq j$ as an *interval*, and to the number $j - i + 1$ as the *length* of the interval.

## 1.1 Graphs. Notation and Definitions

Graphs are fundamental combinatorial structures which results to be very useful for representing systems where it is crucial to specify the connection between pairs of elements of a given set. In this section we are going to recall a few basic definitions and results about graphs. For a more detailed discussion of this field refer to [18, 21]. Recalling that given two positive integers $0 \leq k \leq n$, the *binomial coefficient* $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ represents the number of ways of choosing $k$ objects out of $n$ without regard to order, we start giving the following basic definition.

**Definition 1.1.** *A graph is a pair $G = (V, E)$ of sets such that $V$ is the set of vertices of the graph and $E \subseteq \binom{V}{2}$ is the set of its edges.*

We will write $V(G)$ and $E(G)$ for the vertex set and the edge set of a given graph if the graph $G$ we are referring to is not clear from the context, otherwise, we will refer to them simply as $V$ and $E$ respectively.

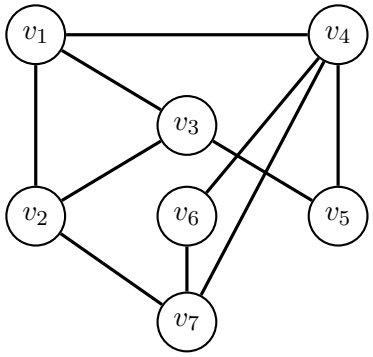



Figure 1.1: A graph $G$ with vertex set $V = \{v_1, \ldots, v_7\}$.

A graph with vertex set $V$ is said to be a graph *on* $V$. Given a graph $G = (V, E)$, an *edge* $e \in E$ is an unordered pair $\{u, v\}$ of vertices in $V$ and $u$ and $v$ are called *endpoints* of $e$. Throughout, we will use a different notation to indicate an edge, namely $e = uv$ with $u, v \in V$. This notation is actually a slight abuse, since it seems to imply that between the two vertices is established a sort of order. However, since it is a quite standard notation in Graph Theory, we choose to adopt it. It is worthwhile to mention the fact that there actually exist graphs where it is specified an order between the endpoints of the edges and these are called *directed* graphs. This kind of graphs will not be considered in this work. Throughout we will only deal with *undirected* graphs. If the edge $uv \in E$, then we will say that the vertices $u$ and $v$ are *incident* to the edge $uv$. Two or more edges that are incident to the same pair of vertices in a graph, are called *multiple edges*. A *loop* is an edge of the kind $uu$ with $u \in V$ where both the endpoints coincide. A graph that neither contains loops nor multiple edges is said a *simple* graph. Throughout, we will only refer to simple graphs.

If $u$ and $v$ are incident to the same edge in $G$, then we will say that they are *neighbours*. More generally, given any vertex $v \in V$, the set of its *neighbours* in $G$, namely the set $N(v)$, is the set of all the vertices $u \in V \setminus \{v\}$ such that the edge $uv \in E$. We will say that two different edges $uv, vw \in E$ which shares a vertex $v \in V$ are *adjacent* in $G$, otherwise will say that they are *disjoint*. Let $v \in V$ be an arbitrary vertex of $G$, then the number of edges which are incident with $v$ in $G$ is called the *degree* of the vertex $v$ and we will write $d(v)$ for it. In other words, $d(v) = |N(v)|$. A vertex with degree 0 is called *isolated* vertex, while a vertex of degree 1 is called a *leaf*. The maximum among the degrees of all the vertices $v \in V$ is called the *maximum degree* of the graph $G$. A graph where all vertices have the same degree $k$ is said to be *k-regular*. In a given undirected graph, a set $M$ of pairwise disjoint edges is called a *matching*. In particular, a matching which consists of $k$ edges is called a $k-$matching. The blue edges in Figure 1.2 represent an example of a 3-matching in the graph $G$.

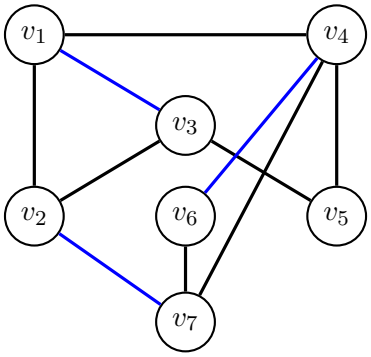



Figure 1.2: The blue edges represent an example of a matching in $G$.

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. These graphs are said to be *isomorphic* if there exists a bijection $\phi : V \to V'$ such that for every $u, v \in V$ it holds that $uv \in E$ if and only if $\phi(u)\phi(v) \in E'$. So, any graph $G$ can be represented graphically in apparently different ways up to isomorfism. In particular, a graph $G$ is *planar* if it can be drawn on the plan in such a way that its edges intersect only at their endpoints. This requirement is a special case of the well known topological property of *embedding* of a graph $G$ on a surface $\Sigma$, which, in this case, is the plan. For the union and the intersection of two graphs, we consider respectively $G \cup G' = (V \cup V', E \cup E')$ and $G \cap G' = (V \cap V', E \cap E')$. The graph $G' = (V', E')$ where $V' \subseteq V$ and $E' \subseteq E$ is said a *subgraph* of the original graph $G$. In this case, we will write $G' \subseteq G$. For a given set $U \subseteq V$, the *subgraph* of $G$ *induced* by $U$ is the graph $G[U]$ which has the set $U$ as vertex set and as edge set, the set of all the edges contained in $E(G)$ with both endpoints in $U$. For a given set of vertices $U \subseteq V(G)$, the graph $G - U$ is the graph $G[V \setminus U]$ obtained from $G$ by deleting from $V$ all the vertices which are also in $U$ and from $E$ all the edges which are incident with at least one vertex in $U$. Given a set of edges $F \subseteq E(G)$, then $G - F$ is the graph $(V, E \setminus F)$ obtained from $G$ by deleting from $E$ all the edges contained in $F$ but leaving the vertex set $V$ unchanged.

**Paths, Cycles, Cliques and Cluster Graphs** A *walk* in a graph $G$ is a sequence of not necessarily distinct vertices $v_1, \ldots, v_k$ such that the edges $v_i, v_{i+1} \in E(G)$ for $i = 1, \ldots, n-1$. For every $n \in \mathbb{N}$, a *path* $P_n$ is a walk on $n$ distinct vertices. In other words, we can say that a path $P_n$ is a graph with vertex set $V(P_n) = \{v_1, \ldots, v_n\}$ and edge set $E(P_n) = \{\{v_i, v_{i+1}\} : i = 1, ..., n-1\}$ where $v_1$ and $v_n$ are called the *endpoints of the path.* A *cycle* $C_n$ on $n$ vertices is a closed path, so one for which $v_n = v_1$.

A graph $K_n$ on $n$ vertices is said to be *complete* if for any pair of vertices $u, v \in V(K_n)$ the edge $uv \in E(K_n)$. Throughout, we will refer to complete graphs as to *cliques* on $n$ vertices.

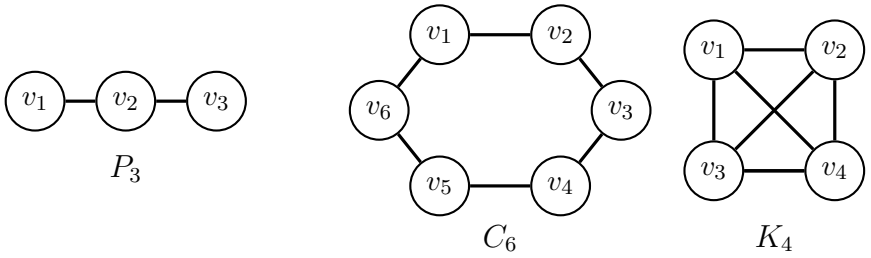



Figure 1.3: In this picture there are examples of a path $P_3$ on 3 vertices, a cycle $C_6$ on 6 vertices and a clique $K_4$ on 4 vertices.

Examples of these kind of graphs are shown in Figure 1.3. A given graph $G$ is a *cluster graph* if it is a vertex-disjoint union of cliques.

A non-empty graph $G$ is *connected* if for any pair of its vertices $u, v \in V$, there exists a path in $G$ with $u$ and $v$ as endpoints. A *connected component* of a graph $G$ is any of its maximal connected subgraphs $G_i \subseteq G$, where maximal means that it does not exist any other connected subgraph $G_j \subseteq G$ such that $G_i \subset G_j$.

**Adjacency Matrix:** From a more algebraic point of view, any given graph $G$ with vertex set $V$ with $|V| = n$ can be represented uniquely up to isomorfisms with an $n \times n$ symmetric matrix that specify which pairs of vertices in $V$ are neighbours and which are not.

**Definition 1.2.** *Given a graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$, the adjacency matrix $A(G)$ of $G$ is a square matrix $n \times n$, such that $A(i, j) = 1$ if the edge $v_i v_j \in E(G)$ and $A(i, j) = 0$ otherwise.*

For instance, the graph in Figure 1.1 has the following adjacency matrix $A(G)$.

$$A(G) = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}.$$

In the adjacency matrix each $i$-row and $i$-column correspond to one vertex $v_i \in V$. The adjacency matrix is symmetric, that is to say that $A(i, j) = A(j, i)$ for any $i, j \in [n]$. This is because, being the graph simple, then for any pair of vertices $u, v \in V$, either there exists one

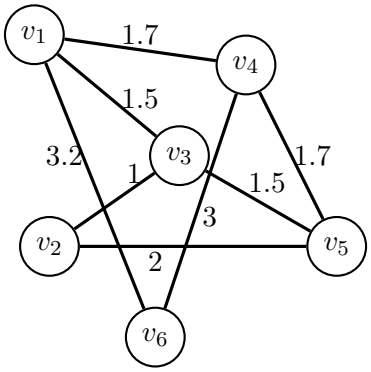



Figure 1.4: An example of a weighted graph. For the graph in figure, at each edge is associated a real number which determines its weight.

edge $uv \in E$ or it does not. Furthermore, $A(i,i) = 0$ for any $i \in [n]$ since simple graphs does not contain any loop.

**Weighted Graphs and Weighted Adjacency Matrix:** A *weighted graph* is a graph $G$ to which is associated a labelling function $W : E(G) \to \mathbb{R}^+$. These weights can represent costs, lengths or capacities assigned to each single edge depending on the problem under investigation. Any path in this kind of graphs is called a *weighted path*, because it is a path where to each edge a weight is associated. The length of a path in a weighted graph is the sum of the weights of all the edges that form that path. For example, let us consider in the graph in Figure 1.4, the paths $v_1v_4v_5v_2$ and $v_1v_3v_2$ are respectively of length 5.9 and 2.5. In the following we define the *weighted adjacency matrix* of a weighted graph $G$ as a matrix where to each entry $A_W(i,j)$ correspond the minimum length of any path in $G$ with endpoints $v_i$ and $v_j$. For example, the weighted adjacency matrix of the graph in Figure 1.4 is the following.

$$A_W(G) = \begin{pmatrix} 0 & 2.5 & 1.5 & 1.7 & 3 & 3.2 \\ 2.5 & 0 & 1 & 3.7 & 2 & 5.7 \\ 1.5 & 1 & 0 & 3.2 & 1.5 & 4.7 \\ 1.7 & 3.7 & 3.2 & 0 & 1.7 & 3 \\ 3 & 2 & 1.5 & 1.7 & 0 & 4.7 \\ 3.2 & 5.7 & 4.7 & 3 & 4.7 & 0 \end{pmatrix}.$$

We observe that, for simple graphs, also the weighted adjacency matrix is symmetric and with null entries on the diagonal, as in the unweighted case.

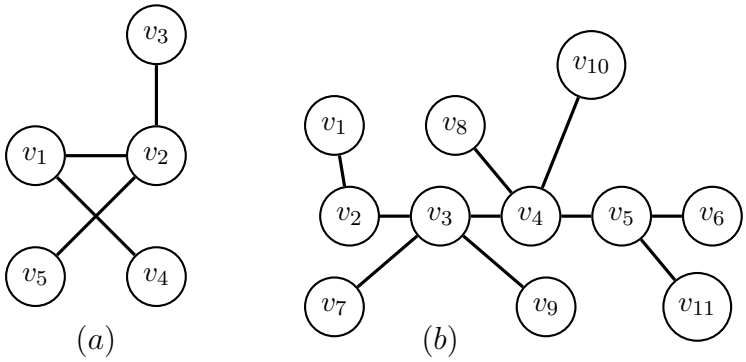



Figure 1.5: In $(a)$ there is an example of a tree on 5 vertices. In $(b)$ it is represented a caterpillar.

### 1.1.1 Other Particular Graphs

It is often useful to be able to recognise graphs with very particular structures. To these graphs are usually even assigned a special name. We already have mentioned *paths*, *cycles* and *cliques*. In the following, we are going to give some other important examples.

**Trees and Caterpillars:** Any undirected connected graph $T$ that does not contain cycles as subgraphs is called a *tree*. See graph $(a)$ in Figure 1.5. for an example. A graph where every connected component is a tree is called a *forest*. A tree is a *caterpillar* if assuming that $L \subseteq V(P)$ is the set of its leafs, the graph $G - F$ is a path. An example of a caterpillar is shown in Figure 1.5.$(b)$.

**Bipartite Graphs, Stars and Cactus Graphs:** Let $r \geq 2$ be an integer. A graph $G = (V, E)$ is said *r-partite* if there exists a partition of $V$ into $r$ classes such that every edge has its endpoints in different classes. If $r = 2$, so $V$ can be partitioned into exactly two classes $\{V_1, V_2\}$, then we say that the graph $G = (V, E)$ is *bipartite*. A bipartite graph is said to be *complete* if every vertex $u \in V_1$ is adjacent to every vertex $v \in V_2$. Complete bipartite graphs are indicated by $K_{|V_1|,|V_2|}$. In Figure 1.6.$(a)$ is shown a graph $K_{3,3}$ as an example. A complete bipartite graph where one of the two sets of the partition has cardinality 1 is clearly a tree and it is called a *star-tree* or simply *star*. The graph shown in Figure 1.6.$(b)$ is an example of a star. Finally, a *cactus* graph is a graph $G$ where each edge belongs to at most one cycle. See Figure 1.6.$(c)$ for an example. A graph that only contains one cycle is called, instead, *unicyclic* graph (Figure 1.6.$(d)$).

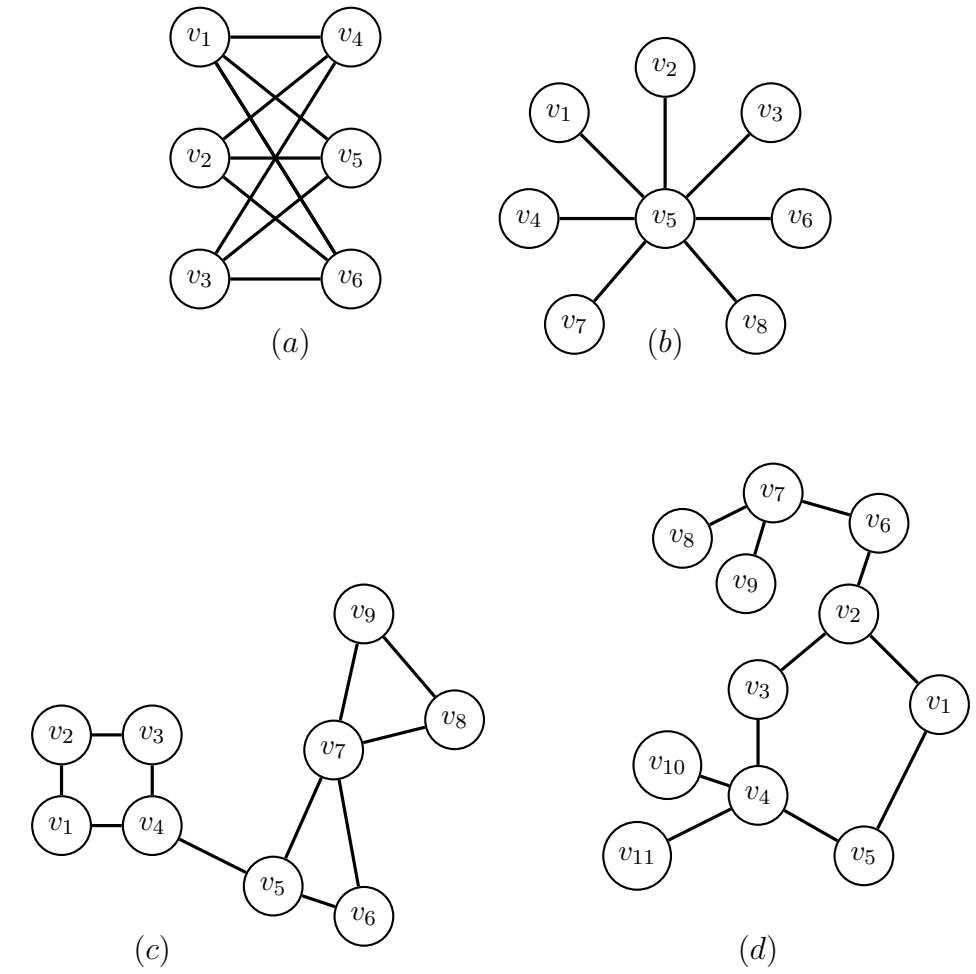



Figure 1.6: In $(a)$ it is shown a complete bipartite graph $K_{3,3}$ with $\{\{v_1, v_2, v_3\}, \{v_4, v_5, v_6\}\}$ as partition of the vertex set. In $(b)$ there is an example of a star on 8 vertices centered in $v_5$. So, a bipartite graph with the vertex set partitioned in the two subsets $\{\{v_1, v_2, v_3, v_4, v_6, v_7, v_8\}, \{v_5\}\}$. In $(c)$ it is represented an example of a cactus graph. Finally, in $(d)$ there is an example of a unicyclic graph.

## 1.2 Temporal graphs

Graphs are computational structures that turn out to be very suitable for modeling various problems which can interest many different scientific domains such as Bioinformatics, Mathematics or Computer Science. However, static graphs can simply represent sets whose elements can be pairwise linked by a certain relation. Nevertheless, in the modern real life there exist problems that could potentially be represented through graphs but such that for a proper description, they need to take into consideration the variable of time. For example, if one wants to study the development of an epidemic considering for any fixed unity of time which pairs of people have been in contact, then a static graph is not anymore enough for describing this particular situation. This is because static graphs inevitably can only describe contacts between pairs of people happened in a specific time. While, in order to have a correct idea of how an epidemic develops, taking in consideration the order at which these contacts happen is crucial. Indeed, a person $A$ infected by a contact with a sick person $B$ at a certain time $t$ can spread the contagion meeting people for a certain period of time after time $t$, but he cannot before. For this reason, these kind of problems needs a more general structure than that of the static graphs to be modeled. In particular, they need graphs that allow discrete changes over time. This necessity leads to the introduction of more complicate structures which are conventionally called *temporal graphs* [40]. See [19] for an example of a temporal problem applied to epidemics. The concept of temporal graphs is of a quite recent introduction in time. For this reason, it is still possible to find works in this field which use slightly different definitions for them. However, by now, it is quite common to adopt the following formal definition given by Kempe et al. in [30].

**Definition 1.3.** *A temporal graph $\mathcal{G} = (G, \mathcal{T})$ is a pair consisting of a static (undirected) graph $G = (V, E)$ and a labeling function $\mathcal{T} : E \to 2^{\mathbb{Z}^+} \setminus \{\emptyset\}$.*

The static graph $G$ takes the name of *underlying graph*. The function $\mathcal{T}$ specifies which edge *appears* at what time. For temporal graphs still does not exist a uniform notation. In the following we will specify the notation we are going to use throughout. For any static edge $e \in E$, $\mathcal{T}(e)$ is the set of time appearances of $e$ in $\mathcal{G}$, so the set of *time instants* at which $e$ appears in $\mathcal{G}$. With $\mathcal{E}(\mathcal{G}) := \{(e, t) \mid e \in E \text{ and } t \in \mathcal{T}(e)\}$ we denote the set of edge appearances in a temporal graph $\mathcal{G}$. Any pair of the kind $(e, t)$ where $e \in E$ and $t \in \mathcal{T}(e)$ is called a *time-edge* of $\mathcal{G}$. We call the interval $L(\mathcal{G}) = [\min\{t \in \mathcal{T}(e) \mid e \in E\}, \max\{t \in \mathcal{T}(e) \mid e \in E\}]$ the *lifetime* of the temporal graph $\mathcal{G}$. With $T(\mathcal{G})$ we indicate the length of the interval $L(\mathcal{G})$, which will be often denoted simply by $T$, when the temporal graph $\mathcal{G}$ we are referring to is clear from the context. For simplicity, we will refer both to $L(\mathcal{G})$ and to its length $T$ as the "lifetime" of the temporal graph. Throughout, we will consider only temporal graphs with *finite lifetime* - so for which the length of $L(\mathcal{G})$ is finite - and we will assume w.l.o.g. that $\min\{t \in \mathcal{T}(e) \mid e \in E\} = 1$. For

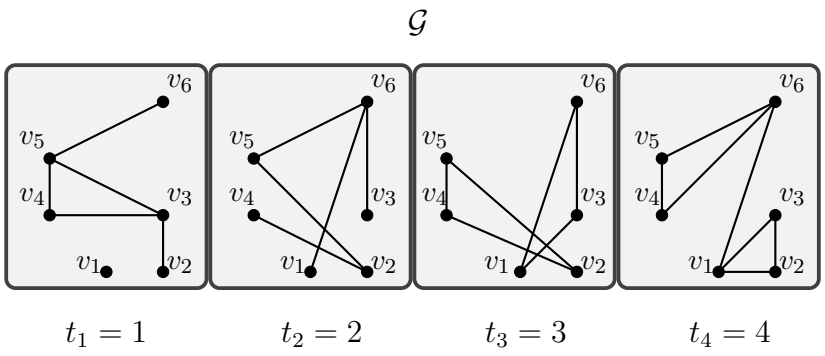



Figure 1.7: This is an example of a temporal graph $\mathcal{G}$ with lifetime $T = 4$ and vertex set $\{v_1, \ldots, v_6\}$. Its temporal edge set is $\mathcal{E}(\mathcal{G}) = \{(v_2v_3, t_1), (v_3v_4, t_1), (v_3v_5, t_1), (v_4v_5, t_1), (v_5v_6, t_1), (v_1v_6, t_2), (v_2v_4, t_2), (v_2v_5, t_2), (v_3v_6, t_2), (v_5v_6, t_2), (v_1v_3, t_3), (v_1v_6, t_3), (v_3v_6, t_3), (v_2v_4, t_3), (v_2v_5, t_3), (v_4v_5, t_3), (v_1v_2, t_4), (v_1v_3, t_4), (v_2v_3, t_4), (v_1v_6, t_4), (v_4v_5, t_4), (v_4v_6, t_4), (v_5v_6, t_4)\}$. Instead, for example, we can say that for the edge $v_1v_2$, $\mathcal{T}(v_1v_2) = \{t_4\}$, while for the edge $v_3v_6$, $\mathcal{T}(v_3v_6) = \{t_2, t_3\}$. Furthermore, for this graph, $L(\mathcal{G}) = [1, 4]$.

this reason throughout we can consider $T(\mathcal{G}) := \max\{t \in \mathcal{T}(e) \mid e \in E\}$. Figure 1.7 shows an example of a temporal graph. The *snapshot* of $\mathcal{G}$ *at time* $t$ is the static graph on $V$ with edge set $\{e \in E \mid t \in \mathcal{T}(e)\}$. Note that, every time instant $t \leq T$ identifies a "snapshot" in $\mathcal{G}$ and the set of all the time-edges that appear in $\mathcal{G}$ at time $t$ can also be empty, in which case clearly follows that the corresponding "snapshot" is empty too. Given temporal graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, let $\mathcal{G}_1 \triangle \mathcal{G}_2$ be the set of time-edges appearing in exactly one of $\mathcal{G}_1$ or $\mathcal{G}_2$.

## 1.3 Basic Complexity Classes

In this section we are going to recall a few basic concepts of Computational Complexity. For doing so, we will take as reference [3, 32].

### 1.3.1 NP-Hardness

In computability theory, a decision problems is one posed in such a way that only answers of the kind "yes" or "no" are allowed. For many of these combinatorial problems there exist known polynomial time algorithms that solve them. For many others instead, even if it is not possible to prove that any such polynomial time algorithm does not exist, it is in practice not known. In this section, we will discuss about the subclass of these last mentioned problems that, together with those for which there exists a polynomial time algorithm, belong to the so called NP class.

Before mentioning a more formal definition of the NP class we recall the following basic definition.

**Definition 1.4.** *An alphabet is a finite set with at least two elements, not containing the special "blank" symbol* $\sqcup$*. Given an alphabet* $\Sigma$ *and assuming that* $\Sigma^n$ *are the strings on that alphabet of length* $n$*, we denote with* $\Sigma^* = \bigcup_{n\in\mathbb{Z}_+} \Sigma^n$ *the set of all the finite strings of symbols contained in* $\Sigma$*. Moreover, we define a language over* $\Sigma$ *as a subset of* $\Sigma^*$*. If* $x \in \Sigma^n$*, then we write* $size(x) := n$ *for the length of the string.*

It is quite common to consider the alphabet $\Sigma$ equal to the set $\{0, 1\}$ and consequently, the set $\{0, 1\}^*$ as the set of all the finite binary strings on that alphabet. The components of a $0 - 1$-string are often called *bits*. We also recall that saying that a language is decidable in polynomial time means that it is decidable in time comparable to the length of the input string.

**Definition 1.5.** *A decision problem is a pair* $\mathcal{P}(X, Y)$ *where* $X$ *is a language decidable in polynomial time and* $Y \subseteq X$*. The elements of* $X$ *are called instances of* $\mathcal{P}$*, the elements of* $Y$ *are called YES-instances and those of* $X \setminus Y$ *are the NO-instances.*

An *algorithm* for a decision problem $(X, Y)$ is an algorithm computing the function $f : X \to \{0, 1\}$ defined by $f(x) = 1$ for $x \in Y$ and $f(x) = 0$ for $x \in X \setminus Y$.

The class NP is commonly identified as that one including all the decision problems having efficiently verifiable solutions. Before giving the well known formal definition, we recall that a very simple and quite used theoretical model for computations is represented by the Turing Machines.

A Turing Machine is said to be *non-deterministic* if when in some given situation, more then one possible actions are allowed. Otherwise, it is said to be *deterministic*.

**Definition 1.6.** *A language* $L \subseteq \{0, 1\}^*$ *is in* NP *if there exists a polynomial* $p : \mathbb{N} \to \mathbb{N}$ *and a polynomial time Turing Machine* $M$ *such that for every* $x \in \{0, 1\}^*$*,*

$$x \in L \iff \exists u \in \{0, 1\}^{p(|x|)} \quad such \quad that \quad M(x, u) = 1.$$

*If* $x \in L$ *and* $u \in \{0, 1\}^{p(|x|)}$ *satisfy* $M(x, u) = 1$*, then we call* $u$ *a certificate for* $x$ *(with respect to the language* $L$ *and machine* $M$*).*

In other words, a decision problem is in NP if, given a $YES$-instance $x$ and a string $u$, it is possible to verify in polynomial time if $u$ is or not a certificate of the original problem for the input $x$, using a deterministic Turing Machine. In this case, we will say that there exists a *solution* of polynomial size in the input for $x$ that *certifies* that it is indeed a $YES$-instance. We can also say that the class NP is the class of the decision problems that can be solved in polynomial time by a *non-deterministic* Turing Machine.

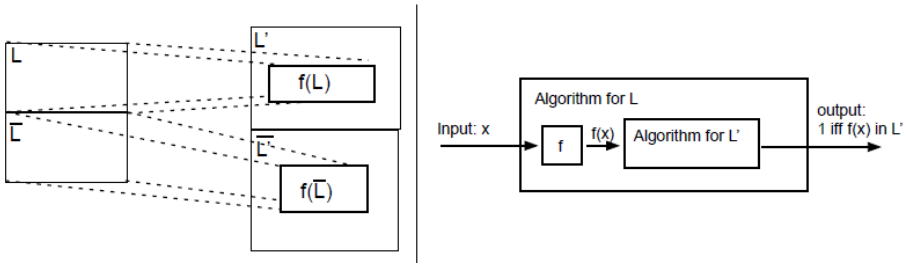



Figure 1.8: A Karp reduction from $L$ to $L'$ is a polynomial time function $f$ that maps strings in $L$ to strings in $L'$ and strings in $\bar{L} = \{0,1\}^* \setminus L$ to strings in $\bar{L}'$.

An example of a graph problem which is in NP is the INDEPENDENT SET, which taken as input a graph $G$ and a positive integer $k$, decides if there is an independent subset of $V(G)$ with cardinality at most $k$. Here, the certificate is the list of $k$ vertices forming an independent set. Another NP problem is the SUBSET SUM that, given a list of $n$ numbers $A_1, \dots A_n$ and a number $S$, decides if there exists a subset of the given numbers that sum up to $S$. In this case, the certificate is the list of numbers in that subset.

### 1.3.2 NP-Hardness and NP-Completeness

Intuitively, an NP-*Hard* problem is one which is "at least as hard as" any other language in NP. To formalise this concept, we first need to recall the definition of *reduction*. Take Figure 1.8 as a reference for the following definition.

**Definition 1.7.** *A language $A \in \{0,1\}^*$ is polynomial-time Karp reducible to a language $B \in \{0,1\}^*$ (sometimes shortened to polynomial-time reducible) denoted by $A \leq_p B$ if there is polynomial time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*$, $x \in A$ if and only if $f(x) \in B$.*

*We say that $B$ is NP$-$Hard if $A \leq_p B$ for every $A \in$ NP. We say that $B$ is NP-Complete if $B$ is NP-Hard and $B \in$ NP.*

Some of the main properties of Karp reductions are formulated in the following theorem whose proof can be found in its entirety in [3].

**Theorem 1.8.** *Given three languages $A, B, C \in \{0,1\}^*$, the following holds:*

1. *(Transitivity). If $A \leq_p B$ and $B \leq_p C$, then $A \leq_p C$.*
2. *If $A$ is NP-Hard and $A \in$ P, then P = NP.*
3. *If $A$ is NP-Complete, then $A \in$ P if and only if P = NP.*

In particular, the second point of Theorem 1.8 formalises the intuitive concept that any NP-Hard problem is "at least as hard" as any other which is in NP.

Let us note that, even if it not a certainty, it is commonly considered unlikely that every NP-Hard problem can also be solvable in polynomial time. Indeed, this fact would implicitly imply that any problem which is in NP is actually solvable in polynomial time; so that P = NP. However, this fact is confuted by the widely esteemed conjecture which asserts that P ≠ NP.

### 1.3.3 Size of Graphs and Temporal Graphs

For the purposes of computation, throughout this work, we will say that $b_n = \mathcal{O}(a_n)$ if there exist constants $C$ and $n_0$ such that $|b_n| \leq Ca_n$ for $n \geq n_0$. We work in the word RAM model of computation, so that arithmetic operations on integers represented using a number of bits logarithmic in the total input size can be carried out in time $\mathcal{O}(1)$. We end this section by giving an intuition of how to calculate the size of graphs or temporal graphs. As we mentioned in Sections 1.1 and 1.2, there can be different ways we can approach to graphs or to temporal graphs. Depending on which of them is more suitable for describing the problem under investigation, it could imply a different size for the input. For example, let us consider a static graph $G = (V, E)$. Generally speaking, such a graph can be represented by an adjacency list whose elements are of the form $(v, N(v))$ where $v \in V$ and its size can be expressed as $\mathcal{O}(|V| + |E|)$. However, for specific graphs can be enough to express their size only by the dimension of one of these two sets. For example, if $G$ is connected, then by construction there cannot be many more vertices than edges, so the size of a connected graph is simply given by $\mathcal{O}(|E|)$, so by the maximum number of its edges. The same holds for example for graphs even disconnected, but that does not admit isolated vertices. In Section 1.1 we also recalled that another way to represent a static graph is by its adjacency matrix $n \times n$ where $n = |V|$. In this case, the size of the graph is determined by the dimension of its adjacency matrix, so it is $\mathcal{O}(|V|^2) = \mathcal{O}(n^2)$.

In Section 1.2, we mentioned that a temporal graph $\mathcal{G}$ can be identified by a set of time edges, that is to say a list of (static) edges together with the list of times $\mathcal{T}(e)$ at which each static edge appears. So, one way to calculate the size of a temporal graph is $\mathcal{O}(\max\{|\mathcal{E}|, |V|\}) = \mathcal{O}(|V|^2 T)$.

## 1.4 Parameterised Complexity

In the standard Computational Complexity Theory, the fact that a given decision problem is NP-Hard means that it is not tractable. In fact, for any of these kind of problems it is not known a polynomial time algorithm capable of solving it.

Parameterised Complexity is a special branch of Computational Complexity Theory that allows a finer classification of NP-Hard problems. In particular, in this context the complexity

of the problem is not anymore only measured as a function of the size of the input, but also as function of some given parameters which figure in the expression of the input itself. In other words, one aim of Parameterised Complexity is to identificate those NP-Hard problems for which the running time function even if understandably not polynomial, has an exponential part only depending on the parameter. In the following we are going to recall a few fundamental definitions of parameterised Complexity referring to [16].

**Definition 1.9.** *A parameterised problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$ where $\Sigma$ is a fixed finite alphabet. For an instance $(x, k) \in \Sigma \times \mathbb{N}$, $k$ is called the parameter.*

Just to give an example, we can recall the formulation of the $k$-VERTEX COVER problem, where $k \in \mathbb{Z}^+$ is a parameter which represents the maximum allowed cardinality for the vertex cover set.

$k$-VERTEX COVER:
*Input:* A graph $G = (V, E)$ and a positive integer $k \in \mathbb{Z}^+$.
*Question:* Does there exist a subset $S \subseteq V$ such that $|S| \leq k$ and $G \setminus S$ is edge-less?

### 1.4.1 FPTAlgorithms

In Parameterised Complexity, algorithms with running time $f(k)n^c$, where $c$ is a constant independent both from $k$ and $n$, $k$ is the parameter and $n$ is the size of the input, are called FPT-*algorithms*. More precisely we recall the following definition.

**Definition 1.10.** *A parameterised problem $L \subseteq \Sigma^* \times \mathbb{N}$ is called fixed-parameter tractable* (FPT) *if there exists an algorithm $A$, a computable function $f : \mathbb{N} \to \mathbb{N}$ and a constant $c$ such that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, the algorithm $A$ correctly decides whether $(x, k) \in L$ in time bounded by $f(k)|(x, k)|^c$, where $|(x, k)|$ denotes the size of the input. The complexity class containing all the fixed-parameter tractable problems is called* FPT.

One aim of parameterised Complexity is that of finding FPTalgorithms in such a way that both $f(k)$ and the constant $c$ are as small as possible.

If this is not possible, also other less efficient classes of algorithms can be defined. One example is the class of the XP-algorithms, also called *slice-wise polynomial* algorithms, which are those with a running time of the form $f(k)n^{g(k)}$ for some function $f$ and $g$ of the parameter $k$.

**Definition 1.11.** *A parameterised problem $L \subseteq \Sigma^* \times \mathbb{N}$ is called slice-wise polynomial* (XP) *if there exists an algorithm $A$ and two computable functions $f, g : \mathbb{N} \to \mathbb{N}$ such that, given*

$(x,k) \in \Sigma^* \times \mathbb{N}$, *the algorithm* $A$ *correctly decides whether* $(x,k) \in L$ *in time bounded by* $f(k)|(x,k)|^{g(k)}$. *The complexity class containing all the slice-wise polynomial problems is called* XP.

## 1.5 Dynamic Programming

Dynamic programming is a recursive algorithmic technique especially intended for treating optimisation problems. From an intuitive point of view, this technique can be seen as exhaustive search procedure designed in such a way to avoid repeating the same calculation for similar structures. This approach typically leads in the best-case scenario to polynomial time algorithms or often at least to FPT ones. In both the cases it would make the problem more tractable with respect to a more expected exponential time algorithm, where the exponent does not depend only on the parameter. The main idea under this technique, which usually starts from some basic states, is that of finding an optimal solution for the original problem, by dividing it into simpler sub-problems and joining together optimal solutions obtained for them, avoiding recomputation of redundant sub-problems calculations.

### 1.5.1 Dynamic Programming on graphs problems

In this section we will mention the sketch of a dynamic programming algorithm for a problem related to graphs, which is entirely showed in [16]. In particular, this example regards the WEIGHTED INDEPENDENT SET problem on trees. The main idea, in the following example, is that of defining sub-problems breaking up the original graph into subgraphs each of them separated from the rest of the graph by small separators, in such a way that the partial solutions of the problem for the two subgraphs identified by the separator, only interact by means of the separator itself. So, for checking whether these two partial solution are compatible it is enough to analyse the situation at the separator.

First of all, in the following, we recall how this problem is defined.

WEIGHTED INDEPENDENT SET:
*Input:* A tree $T$ with a vertex weight function $w : V(T) \to \mathbb{R}_{\geq 0}$.
*Question:* Find an independent set in $T$ with maximum total weight.

In this case, every vertex of the tree separates the sub-tree rooted at it from the rest of the graph. Consequently, the decisions made by any solution of the problem to that specific sub-tree are independent from the rest of the tree too. For a more accurate treatment of this proof we refer to [16]. What we are instead interested in mentioning here is that the dynamic

programming algorithm works as follows. It scan through every rooted sub-tree of $T$ starting from a leaf and proceeding backwards up to $T$ itself. Assuming $t$ to be the root of any rooted sub-tree $T_t$ of $T$, at every step it calculates a function that computes the maximum possible weight of an independent set of $T_t$ both in the case where $t$ belongs to the independent set and in the case where it does not, considering, consequently, as a given data the value of the maximum possible weight of any independent set of $T_t \setminus \{t\}$ calculated at the previous step of the algorithm. The algorithm ends when $T_t = T = T_r$.

## 1.6 Bounded Search Trees

In this section, we are going to outline the main aspects of the well known algorithmic technique of bounded search trees which we will use in particular in Section 3.1. Again we will use as a reference for this [16]. The idea under this kind of algorithms is that of constructing a feasible solution for a certain input instance $I$ of the original problem by dividing it into sub-problems $\{I_1, \ldots, I_\ell\}$ obtained by making decisions on whether to include a certain element in the solution or not. Clearly the size of these sub-problems is, by construction, lower than that of $I$. Furthermore, finding the maximum number $\ell$ of these sub-problems must take polynomial time and this number is required to be of small size, meaning that it must be either a constant or at most a function of the parameter $k$. In case of a $YES$-instance at least one of these branchings must lead to a feasible solution. Every feasible solution $S_i$ of $I_i$ corresponds to a feasible solution $f(S_i)$ of the original input instance and at least one among all the possible $f(S_i)$ with $i \in [\ell]$ is optimum for $I$. If the depth of the search tree is bounded by a function of the parameter $k$, and every step requires polynomial time, then the original problem is in $\mathsf{FPT}$, because the branching algorithm runs itself in $\mathsf{FPT}$ time.

For example, let us assume to consider the VERTEX COVER problem, which given as input a graph $G$ and a positive integer $k$, asks whether there exists a set $X$ of at most $k$ vertices of $G$ such that $G - X$ is edgeless. Given an input instance $(G, k)$ of the problem, a possible search tree algorithm reported in [16], starts selecting a vertex $v \in V(G)$ of maximum degree. If the maximum degree of $G$ is 1, then every connected component is either an isolated vertex, or an edge, so the instance is trivially a "$YES$"$-$instance. Otherwise, the algorithm branches the problem into two sub-problems once including $v$ in the solution and another including instead $N(v)$. The number of sub-problems in which at each step the algorithm branches the current instance limits the degree of the search tree. While, the depth of the tree is bounded by a function of the parameter $k$ which, at each step can decrease from a minimum of 1 up to maximum of $k$. Clearly, when $k$ falls to zero, the algorithm ends and in case it has not been reached any feasible solution, then it means that $(G, k)$ was a "$NO$"$-$instance. The running time of such

algorithm is bounded by the product between the number of nodes in the search tree times the time taken at each node. In the case of a VERTEX COVER, the time required at each node is clearly bounded by $\mathcal{O}(|V|)$, while the number of its leaves is proved to be bounded by $1.4656^k$. Note that, because for any search tree $T$, assuming that $T$ has $d$ leaves, then it can have at most $2d - 1$ nodes, then bounding the number of leaves can be enough. So, admitting that it has been applied at the beginning a kernelisation algorithm with running time $\mathcal{O}(|V|\sqrt{|E|})$ to reduce the size of the input, this search tree algorithm requires $\mathcal{O}(|V|\sqrt{|E|} + 1.4656^k k)$ time. So, this is ultimately an FPT algorithm.

# Chapter 2

# Editing to Temporal Cliques

In this chapter we are going to present a natural interpretation of the well known Cluster Editing problem in the setting of temporal graphs.

The problem of partitioning a certain set into clearly independent and homogeneous clusters, taking in consideration the fact that the interactions between pairs of elements of the original set can change over time, find application in many real world situation. One for all, can be the study of the interactions between pairs of users on a same social network during a fixed period of time, for example one year. Here, the users of that social network can be seen as the vertex set of a temporal graph, whose time-edges represent the interactions between pairs of users that happen at discrete intervals of time, for examples days or weeks (units of time that we will call snapshots). For example, any user $A$ can interact with each of its contacts at different times and with different regularity over time. Clearly, if $A$ interacts with one of its contacts $B$ almost every week and with another user $C$ on average only once a year, we can, for example, consider the interaction between $A$ and $C$ as negligible with respect to that one between $A$ and $B$. In this context, asking if it is possible to partition the temporal graph that models all these interactions into temporal independent clusters, is useful to have a subdivision of all the users of that social network depending on how frequently they interact one another.

Before entering in the details, we are going to give a brief description of the static Cluster Editing citing a few facts from [10].

## 2.1 Cluster Editing

Cluster Editing asks whether it is possible to transform any given static graph $G = (V, E)$ into the union of static cliques pairwise vertex-disjoint, using the minimum number of edge modifications; i.e. additions or deletions. The set $\Sigma$ of such modified edges takes the name of *modification set*. The set of edges of the new graph obtained once the additions/deletions have

been applied is essentially $E\Delta\Sigma$, where here $\Delta$ indicates the symmetric difference between two sets of edges. More precisely Cluster Editing can be enunciated as follows.

Cluster Editing(CE):
*Input:* A graph $G = (V, E)$ and a positive integer $k \in \mathbb{Z}^+$.
*Question:* Does there exist a set $\Sigma$ of edge modifications, of cardinality at most $k$, such that the modified graph is a cluster graph?

Cluster Editing formulation finds potential application in several different fields, for example in Bioinformatics and Computer Science. In addition, it has also drawn upon for inspiration for heuristic clustering algorithms such as in [5, 25, 43].

### 2.1.1 Overview and related works

Complexity results and heuristic approaches about Cluster Editing are discussed in a survey article [9]. It is quite a known fact that Cluster Editing is NP-Complete as many authors have proved independently [4, 17, 33, 42]. However, limiting the choice of the input to specific classes of graphs, Cluster Editing becomes tractable in polynomial time. Examples of these classes are *unit interval graphs* [37] or graphs with maximum degree two (namely graphs where every connected component is either a *path* or a *cycle*) [8]. On the other hand, Cluster Editing turns out to be NP-Hard even if at most four edge modifications incident to each vertex are allowed or on graphs with maximum degree six [31]. Instead, it is still an open problem if Cluster Editing is or not tractable on graphs with maximum degree between three and five.

Cluster Editing has received substantial attention from the parameterised complexity community, and many results focusing on the natural parameterisation by the number $k$ of permitted modifications have been studied. A first intuitive deduction that deserves to be mentioned is the fact that the fixed-parameter tractability of Cluster Editing with respect to $k$, easily follows, via a search tree argument, from the quite known result that a graph is a cluster graph if and only if it contains no induced copy of the three-vertex path $P_3$. This approach has been refined repeatedly in non-trivial ways, culminating in an algorithm running in time $\mathcal{O}(1.76^k + m + n)$ for graphs with $n$ vertices and $m$ edges [13]. A different recent approach has been that of considering as a parameter the number of modifications permitted above the lower bound implied by the number of modification-disjoint copies of $P_3$ (copies of $P_3$ such that no two share either an edge or a non-edge) [34]. However, currently also other parameters, such as the number of clusters [20] and a lower bound on the allowed size of each cluster [1], have been taken into consideration for studying Cluster Editing.

We conclude this paragraph by noting that there exists two straightforward variants of Cluster Editing obtained respectively if we decide to admits to modify edges either only by addition (Cluster Addition) or only by deletion (Cluster Deletion). It is quite intuitive to observe that the Cluster Addition problem is tractable in polynomial time. In fact, in that case, it is enough to add all the missing edges to complete a clique from every connected component of the original input graph. Cluster Deletion, instead, results to be NP-Complete [41] for general graphs. However, it is proven to be solvable in polynomial time on graphs with maximum degree three, even if it is still NP-Hard even on 4-regular graphs [31].

## 2.2 Temporal parameters $\Delta_1$ and $\Delta_2$ and fundamental definitions

Before formulating our temporal interpretation of Cluster Editing, we need to introduce a few further fundamental concepts and definitions about temporal graphs. As we mentioned at the beginning, it is of special interest, for our purposes in this work, the subdivision of a given temporal graph into clusters. In the static case, these clusters are essentially static cliques; that is to say, graphs for which there exists one edge for any pair of vertices $u, v \in V(G)$. So, what we need before defining a generalisation of Cluster Editing to the temporal setting, which is the main aim of this work, is a corresponding definition of clusters (that is to say of complete graphs) which takes in consideration the time variable. Defining what can be seen as a clique in a temporal graph results to be surprisingly difficult for various reasons. In fact, any edge $uv \in E(G)$ of the underlying graph of a given temporal graph $\mathcal{G}$, can appear during the lifetime of $\mathcal{G}$ from a minimum of 1 up to a maximum of $T$ times and two consecutive appearances of it are allowed to be very close as well as very distant in time. For these reasons, while in the static case it was enough to say that any edge with both end-vertices in $V(C)$ must belong to $E(C)$ for considering a graph $C$ as a clique, in the temporal setting we should be more specific in what we mean for an edge to be part of a specific temporal clique. In fact, if two appearances of a same edge $uv \in E(G)$ are very distant in time, then it might not make sense to consider both these two appearances of $uv$ as part of a same cluster. Think for instance to the users $A$ and $C$ of a certain social network that interact each other only once a year as supposed in the example at the beginning of this Chapter. So, in the time setting, it is necessary to introduce two main parameters. One specifies how much "dense" an edge $uv \in E(G)$ must be in a given interval in order to be considered as part of a temporal clique defined in that interval. Another, limits the maximum number of snapshots that can separate two consecutive appearances of a same edge $uv \in E(G)$ for allowing to consider these two appearances in some way "dependent" in time. To these goals we will use the two parameters $\Delta_1, \Delta_2 \in \mathbb{Z}^+$ respectively.

For what concerns the first matter, taking as a model the definition of $\Delta$-cliques given by Viard, Latapy and Magnien in [49], we introduce the concept of $\Delta_1$-density as follows. Let $\Delta_1 \in \mathbb{Z}^+$, $e \in E(G)$ any edge of the underlying graph of a given temporal graph $\mathcal{G} = (G, V)$ and $[a, b]$ an interval, then we say that $e$ is $\Delta_1$-*dense* in $[a, b]$ if for all $\tau \in [a, \max\{a, b-\Delta_1+1\}]$ there exists a $t \in \mathcal{T}(e)$ with $t \in [\tau, \tau+\Delta_1-1]$. We define a *template* to be a pair $C = (X, [a, b])$ where $X$ is a set of vertices and $[a, b]$ is an interval. For a set $S$ of time-edges we let $V(S)$ denote the set of all endpoint of time-edges in $S$, and the *lifetime* $L(S) = [s, t]$, where $s = \min\{s : (e, s) \in S\}$ and $t = \max\{t : (e, t) \in S\}$. In this case, we will say that $S$ *generates* the template $(V(S), L(S))$. A set $S \subset \mathcal{E}(\mathcal{G})$ forms a $\Delta_1$-*temporal clique* if for every pair of vertices $x, y \in V(S)$ the edge $xy$ is $\Delta_1$-dense in $L(S)$. Our definition is quite similar to that given in [49], with the only differences that we give it in terms of templates generated by a set of time-edges. Note that, since we gave the definition of $\Delta_1$-temporal cliques in terms of time-edges, then we only admit temporal cliques with at least two vertices.

For the second parameter there already exist definitions of temporal independence between time-edges, such as in [39]. However, as before, we adapt the known definition giving it in terms of templates as follows. Let be $\Delta_2 \in \mathbb{Z}^+$. We say that two templates $(X, [a, b])$ and $(Y, [c, d])$ are $\Delta_2$-*independent* if

$$X \cap Y = \emptyset \quad \text{or} \quad \min_{s\in[a,b],t\in[c,d]} |s - t| \geq \Delta_2.$$

Let $\mathfrak{T}(\mathcal{G}, \Delta_2)$ denote the class of all collections of pairwise $\Delta_2$-independent templates on $V(\mathcal{G})$ with lifetimes $[a, b]$ satisfying $1 \leq a \leq b \leq T(\mathcal{G})$. Two sets $S_1, S_2$ of time-edges are $\Delta_2$-*independent* if the templates they generate are $\Delta_2$-independent. As a special case of this, two time-edges $(e, t)$, $(e', t')$ are $\Delta_2$-*independent* if $e \cap e' = \emptyset$ or $|t - t'| \geq \Delta_2$. This leads to the following fundamental definition that clarify what we mean throughout for a temporal graph to be a cluster graph in the temporal sense.

**Definition 2.1.** *A temporal graph $\mathcal{G}$ realises a collection $\{(X_i, [a_i, b_i])\}_{i\in[k]} \in \mathfrak{T}(\mathcal{G}, \Delta_2)$ of pairwise $\Delta_2$-independent templates if*

- *for each $(xy, t) \in \mathcal{E}$ there exists $i \in [k]$ such that $x, y \in X_i$ and $t \in [a_i, b_i]$,*
- *for each $i \in [k]$ and $x, y \in X_i$, the edge $xy$ is $\Delta_1$-dense in $[a_i, b_i]$.*

*If there exists some $\mathcal{C} \in \mathfrak{T}(\mathcal{G}, \Delta_2)$ such that $\mathcal{G}$ realises $\mathcal{C}$ then we call $\mathcal{G}$ a $(\Delta_1, \Delta_2)$-cluster temporal graph.*

We will show in Lemma 2.9 that it is possible to verify in time $\mathcal{O}(|\mathcal{E}|^3|V|)$ if a given temporal graph is or not a $(\Delta_1, \Delta_2)$-cluster temporal graph. Throughout we assume that $\Delta_2 > \Delta_1$, since if $\Delta_2 \leq \Delta_1$ then one $\Delta_1$-temporal clique can realise many different sets of $\Delta_2$-independent

templates. For example, if $\Delta_2 = \Delta_1$ then the two time-edges $(e, t)$ and $(e, t + \Delta_1)$ are $\Delta_2$-independent but also $e$ is $\Delta_1$-dense in the interval $[t, t + \Delta_1]$.

A set of pairwise $\Delta_2$-independent time-edges takes that name of $\Delta_2$-*temporal matching*. The *size* of a given $\Delta_2$-temporal matching is the number of its time-edges. For example, if we fix $\Delta_2 = 2$, a possible 2-temporal matching of size 5 in the graph $\mathcal{G}$ in Figure 1.7 is the set $\{(v_3v_6, t_1), (v_2v_4, t_2), (v_1v_6, t_3), (v_2v_3, t_4), (v_4v_5, t_4)\}$.

## 2.3 Editing to Temporal Cliques: ETC

We now introduce a possible interpretation of Cluster Editing into the temporal setting. In particular we define the new decision problem Editing to Temporal Cliques such that, given as input a temporal graph $\mathcal{G}$ and natural numbers $k, \Delta_1, \Delta_2 \in \mathbb{Z}^+$, asks whether it is possible to transform $\mathcal{G}$ into a $(\Delta_1, \Delta_2)$-cluster temporal graph by applying at most $k$ time-edges modifications (addition or deletion). In this context, the set $\Pi$ of time-edges which are added or deleted from $\mathcal{G}$ is called the *temporal modification set*. It is notable that the temporal modification set $\Pi$ can be defined, so as in the static case, as the symmetric difference between the temporal edge set $\mathcal{E}(\mathcal{G})$ of the input graph and that one of the final graph after the modifications have been applied. More formally, our problem can be formulated as follows.

Editing to Temporal Cliques(ETC):
*Input:* A temporal graph $\mathcal{G} = (G, \mathcal{T})$ and positive integers $k, \Delta_1, \Delta_2 \in \mathbb{Z}^+$.
*Question:* Does there exist a set $\Pi$ of time-edge modifications, of cardinality at most $k$, such that the modified temporal graph is a $(\Delta_1, \Delta_2)$-cluster temporal graph?

Clearly ETC is in NP, because for any input instance $(\mathcal{G}, \Delta_1, \Delta_2, k)$, a non-deterministic algorithm (if one exists), can guess the temporal modification set $\Pi$ and, using Lemma 2.9, verify that the modified temporal graph is a $(\Delta_1, \Delta_2)$-cluster temporal graph in polynomial time. One of the main topics we will focus on in the rest of this chapter is about the hardness of ETC even under strong limitations on the input instance. To do that, we start asserting, with an easy proof, that we can hope to gain tractability for ETC only in settings where the static version is tractable too. However, in the following, we shall see that ETC is hard even on temporal graphs with paths as their underlying graphs, and thus the converse is false. As we just mentioned, we start formulating the following Proposition that links, in some way, the tractability of certain versions of ETC with its corresponding versions in the static setting.

**Proposition 2.2.** *Let $\mathcal{C}$ be a class of graphs on which* Cluster Editing *is* NP*-complete. Then* ETC *is* NP*-complete on the class of temporal graphs $\{(G, \mathcal{T}) : G \in \mathcal{C}\}$.*

*Proof.* We obtain a reduction from Cluster Editing to ETC as follows. Given an instance $(G, k)$ of Cluster Editing, where $G \in \mathcal{C}$, we construct an instance $(\mathcal{G}, 1, 1, k)$ of ETC by setting $\mathcal{G} = (G, \mathcal{T})$ where $\mathcal{T}(e) = \{1\}$ for all $e \in E(G)$. It is straightforward to verify that a set $\Pi$ of time-edge modifications makes $\mathcal{G}$ a $(1, 1)$-cluster temporal graph if and only if the set of edge modifications $\{e : (e, 1) \in \Pi\}$ modifies $G$ into a cluster graph. □

So, for example, from Proposition 2.2 and the observations from subsection 2.1.1 it follows that ETC is NP-Hard for temporal graphs whose underlying graph has maximum degree six.

### 2.3.1 Basic Observations and properties of temporal graphs

In this section we collect a few fundamental results on the structure of temporal graphs which we will need later. To this end, we start introducing a few useful definitions on sets of time-edges.

Let $S$ be a set of time-edges and $A$ be a set of vertices, then we set $S[A] = \{(xy, t) \in S : x, y \in A\}$ to be the set of all the time-edges in $S$ *induced* by $A$. Similarly, given a temporal graph $\mathcal{G}$ and $A \subset V$, we set $\mathcal{G}[A]$ to be the temporal graph with vertex set $A$ and temporal edges set $\mathcal{E}[A] = \{(xy, t) \in \mathcal{E}(\mathcal{G}) : x, y \in A\}$. For an interval $[a, b]$ we let $\mathcal{G}|_{[a,b]}$ be the temporal graph on the original vertex set $V(G)$ restricted to the sub-interval $[a, b]$ of its lifetime. In other words, it can be seen itself as a temporal graph with set of time-edges $\{(e, t) \in \mathcal{E}(\mathcal{G}) : t \in [a, b]\}$ and lifetime $[a, b]$. We will say that a set $S$ of time-edges is $\Delta_2$*-indivisible* if there does not exist a pairwise $\Delta_2$-independent collection $\{S_1, \ldots, S_k\}$ of time-edge sets satisfying $\cup_{i\in[k]} S_i = S$. A $\Delta_2$-indivisible set $S$ is said to be $\Delta_2$*-saturated in* $\mathcal{G}$ if after including any other time-edge of $\mathcal{E}(\mathcal{G})$ it would cease to be $\Delta_2$-indivisible.

We start proving the following technical Lemma that will be widely used in the following proofs.

**Lemma 2.3.** *If two $\Delta_2$-indivisible sets $S_1$ and $S_2$ of time-edges satisfy $S_1 \cap S_2 \neq \emptyset$, then $S_1 \cup S_2$ is $\Delta_2$-indivisible.*

*Proof.* Let us assume by contradiction that $S_1 \cup S_2$ is not $\Delta_2$-indivisible. Thus there exist a partition of $S_1 \cup S_2$ into at least two $\Delta_2$-independent subsets. Now, because $S_1$ and $S_2$ are respectively $\Delta_2$-indivisible, then each of them must belong entirely to a same subset of the partition. However, since $S_1 \cap S_2 \neq \emptyset$, then $S_1$ and $S_2$ must belong both to the same set of the partition. Thus, this contradicts our assumption that the partition contains more than one set. □

In the following, we want to show that, according to the way we decided to intend a cluster temporal graph, the set of its $\Delta_2$-independent $\Delta_1$- temporal cliques is essentially a partition of $\Delta_2$-saturated subsets of its time-edge set. In particular, this fact will derive from Lemma 2.8. To this end, with the following two lemmas, we start proving that there exists a unique way to partition any given temporal graph into $\Delta_2$-saturated subsets and that this partition can be found in polynomial time.

**Lemma 2.4.** *For any $\Delta_2 \in \mathbb{Z}^+$, any temporal graph $\mathcal{G}$ has a unique decomposition of its time-edges into $\Delta_2$-saturated subsets.*

*Proof.* Let us consider any temporal graph $\mathcal{G}$. We will first argue that it is always possible to find a partition of $\mathcal{E}(\mathcal{G})$ into sets $\Delta_2$-saturated in $\mathcal{G}$. Note that, given any time-edge, we can greedily build a $\Delta_2$-saturated set that contains it. We now claim that any time-edge $(e,t) \in \mathcal{E}(\mathcal{G})$ can belong to exactly one $\Delta_2$-saturated subset of $\mathcal{E}(\mathcal{G})$. To see this, suppose that $P$ and $P'$ are two distinct $\Delta_2$-saturated sets with $(e,t) \in P \cap P'$. By Lemma 2.3, we then have $P \cup P'$ is also $\Delta_2$-indivisible, contradicting the maximality of $P$ and $P'$. Thus, if $\{P_1, \ldots, P_h\}$ is any collection of $\Delta_2$-saturated subsets such that $\bigcup_{i=1}^{h} P_i = \mathcal{E}(\mathcal{G})$, we know that for $P_i \neq P_j$ we have $P_i \cap P_j = \emptyset$ and hence $\mathcal{P} = \{P_1, \ldots, P_h\}$ is a partition of the time-edges into $\Delta_2$-saturated sets.

We now prove uniqueness of this decomposition. In fact, if there was a different partition $\mathcal{P}'$ of the time-edges of $\mathcal{G}$ into $\Delta_2$-saturated sets, it would imply that there exists at least one time-edge $(e,t) \in \mathcal{E}(\mathcal{G})$ which belongs to different $\Delta_2$-saturated sets in $\mathcal{P}$ and $\mathcal{P}'$. But in this case, $(e,t)$ belongs to two different $\Delta_2$-saturated subsets of $\mathcal{E}(\mathcal{G})$ which we already proved to be impossible. □

The next result shows that this partition can be found in polynomial time. This result, strengthened by the assertion of Lemma 2.8, will be useful in the proof of Lemma 2.9 that states that it is possible to verify in polynomial time if a given temporal graph is or not a temporal cluster graph.

**Lemma 2.5.** *Let $\mathcal{G} = (G = (V,E), \mathcal{T})$ be a temporal graph, and let $\mathcal{E} = \{(e,t) : e \in E, t \in \mathcal{T}(e)\}$ be the set of time-edges of $\mathcal{G}$. Then, there is an algorithm which finds the unique partition of $\mathcal{E}$ into $\Delta_2$-saturated subsets in time $\mathcal{O}(|\mathcal{E}|^3|V|)$.*

*Proof.* Our algorithm proceeds as follows. We maintain a list of disjoint subsets of time-edges, and make multiple passes through the list until it contains precisely the $\Delta_2$-saturated subsets of $\mathcal{E}$. Suppose that $\mathcal{E} = \{e_1, \ldots, e_r\}$; we initialise $\mathcal{L}_0$ as the list $(\{e_i\})_{i=1}^{r}$; for convenience we will also store the template generated by each element of the list. In a single pass through the list $\mathcal{L}_i$, we remove the first element $E_1$ of $\mathcal{L}_i$ and consider each remaining element of the list in turn: as soon as we find a set $E_j$ such that the templates $(X_1, [a_1, b_1])$ and $(X_j, [a_j, b_j])$ generated by $E_1$

and $E_j$ respectively are not $\Delta_2$-independent, we remove $E_j$ from $\mathcal{L}_i$, add the set $E_1 \cup E_j$ to $\mathcal{L}_{i+1}$ (noting that we also compute and store the template generated by this set of time-edges), and move on to $E_2$; if we find no such set $E_j$, we add $E_1$ to $\mathcal{L}_{i+1}$. Now proceed in the same way with the first element of the modified list $\mathcal{L}_i$; we continue until $\mathcal{L}_i$ is empty. Assuming we combined at least one pair of sets when moving from $\mathcal{L}_i$ to $\mathcal{L}_{i+1}$, we now repeat the entire procedure with $\mathcal{L}_{i+1}$. When we reach an index $\ell$ such that we complete a pass through $\mathcal{L}_\ell$ without carrying out any merging operations, we halt and claim that $\mathcal{L}_\ell$ contains precisely the $\Delta_2$-saturated sets of $\mathcal{E}$.

To see that this claim holds, we first argue that the following invariant holds throughout the execution of the algorithm: for each $i$, every set of time-edges in $\mathcal{L}_i$ is $\Delta_2$-indivisible.

This invariant is clearly holds for $\mathcal{L}_0$, since each element of $\mathcal{L}$ contains only a single time-edge. Suppose now that the invariant holds for $\mathcal{L}_i$; we will argue that it must also hold for $\mathcal{L}_{i+1}$. Let $E_r$ and $E_s$ be two sets that are merged during the pass through $\mathcal{L}_i$. It suffices to demonstrate that $E_r \cup E_s$ is $\Delta_2$-indivisible. We already know, from the fact that the invariant holds for $\mathcal{L}_i$, that there is no way to partition $E_1$ (respectively $E_2$) into two or more pairwise $\Delta_2$-independent sets; thus, the only possible way to partition $E_1 \cup E_2$ into pairwise $\Delta_2$-independent sets is with the partition $(E_1, E_2)$. However, by construction, since the algorithm merged $E_1$ and $E_2$, we know that $E_1$ and $E_2$ cannot be $\Delta_2$-independent. We conclude that the invariant also holds for $\mathcal{L}_{i+1}$.

By the termination condition for the algorithm, we know that $\mathcal{L}_\ell$ gives a partition of $\mathcal{E}$ into pairwise $\Delta_2$-independent sets. Moreover, the invariant tells us that each of these sets is $\Delta_2$-indivisible. Maximality and hence saturation of the sets follows immediately from the fact that the edges are partitioned into pairwise $\Delta_2$-independent sets, so we conclude that $\mathcal{L}_\ell$ does indeed give a partition of $\mathcal{E}$ into pairwise $\Delta_2$-independent $\Delta_2$-saturated sets. Finally, by Lemma 2.4, we know that this partition is unique.

It remains only to bound the running time of the algorithm. Note that a single pass (in which we create $\mathcal{L}_{i+1}$ from $\mathcal{L}_i$), requires us to consider $\mathcal{O}(|\mathcal{L}_i|^2)$ pairs of sets; since we store the templates generated by each set, each such comparison can be carried out in time $\mathcal{O}(V)$ as this will be dominated by the time required to verify disjointness or otherwise of the vertex sets. (Note that we can compute the template for each new merged set in constant time.) Since $|\mathcal{L}_0| = |\mathcal{E}|$ and the length of the list decreases by at least one with each pass, we see that the overall running time is bounded by $\mathcal{O}(|\mathcal{E}|^3|V|)$, as required. □

The next three lemmas are useful for relating indivisible sets to $\Delta_1$-temporal cliques to clusters. These results will be used especially in Chapter 3, however we position them here because they are preparatory for the proof of Lemma 2.9 where we will show that it is possible to verify in polynomial time if a given temporal graph is or not a cluster temporal graph.

**Lemma 2.6.** *Let $\mathcal{G}$ be a temporal graph, $S \subseteq \mathcal{E}(\mathcal{G})$ be a $\Delta_2$-saturated set of time-edges, and $\mathcal{K}$ a $\Delta_1$-temporal clique such that $\mathcal{K} \subseteq \mathcal{E}(\mathcal{G})$ and $\mathcal{K} \cap S \neq \emptyset$. Then, $\mathcal{K} \subseteq S$.*

*Proof.* Because $\mathcal{K}$ is a $\Delta_1$-temporal clique it forms an $\Delta_2$-indivisible set. Then as $S$ is $\Delta_2$-indivisible and has non-empty intersection with $\mathcal{K}$ we see that $\mathcal{K} \cup S$ is $\Delta_2$-indivisible by Lemma 2.3. If $\mathcal{K} \not\subseteq S$ then this contradicts assumption that $S$ is a maximal $\Delta_2$-indivisible set. □

**Lemma 2.7.** *Let $\mathcal{G}$ be any $(\Delta_1, \Delta_2)$-cluster temporal graph. Then, any $\Delta_2$-indivisible set $S \subseteq \mathcal{E}(\mathcal{G})$ must be contained within a single $\Delta_1$-temporal clique.*

*Proof.* Let $C_1, \ldots, C_k$ be the $\Delta_1$-temporal cliques in the $(\Delta_1, \Delta_2)$-cluster temporal graph $\mathcal{G}$ in the decomposition of $\mathcal{G}$ into $\Delta_1$-temporal cliques, such that $C_i \cap S \neq \emptyset$. Now for each $i \in [k]$ we let $S'_i = S \cap C_i$. Observe that since $S'_i \subseteq C_i$ then the set $\{S'_i\}_{i\in[k]}$ are pairwise $\Delta_2$-independent and $\cup_{i\in[k]} S'_i = S$, this contradicts the indivisibility of $S$. □

Since any temporal graph has a unique decomposition into $\Delta_2$-saturated sets by Lemma 2.4, and using the fact that any pair of $\Delta_2$-saturated sets is $\Delta_2$-independent by definition, we obtain the following corollary to Lemma 2.7.

**Lemma 2.8.** *A temporal graph $\mathcal{G}$ is a $(\Delta_1, \Delta_2)$-cluster temporal graph if and only if every $\Delta_2$-saturated set of time-edges forms a $\Delta_1$-temporal clique.*

*Proof.* [ $\Longrightarrow$ ] Let us assume that $\mathcal{G}$ is a $(\Delta_1, \Delta_2)$-cluster temporal graph and let $\mathcal{P}_\mathcal{G}$ be the decomposition of its time-edges into $\Delta_2$-saturated subsets. Because any $\Delta_2$-saturated set is also $\Delta_2$-indivisible by definition, then by Lemma 2.7 it follows that for any of these $\Delta_2$-saturated sets $S$ there exists a $\Delta_1$-temporal clique $\mathcal{K}$ in $\mathcal{G}$ which contains $S$ in its time-edge set. This means that $\mathcal{K} \cap S \neq \emptyset$, so by Lemma 2.6, $\mathcal{K} \subset S$. So, $S$ is itself a $\Delta_1$-temporal clique.

[ $\Longleftarrow$ ] By Lemma 2.4 there exists a unique decomposition $\mathcal{P}_\mathcal{G}$ of $\mathcal{G}$ into $\Delta_2$-saturated sets. We claim that these $\Delta_2$-saturated sets are pairwise $\Delta_2$-independent. To see this, suppose, for a contradiction, that two $\Delta_2$-saturated sets $X$ and $Y$ are not $\Delta_2$-independent. By indivisibility of $X$ and $Y$, the only possible way in which $X \cup Y$ could be partitioned into pairwise $\Delta_2$-independent subsets is with the partition $(X, Y)$, but by assumption $X$ and $Y$ are not $\Delta_2$-independent, so we can conclude that $X \cup Y$ is in fact $\Delta_2$-indivisible. This contradicts maximality of $X$ and $Y$, giving the contradiction to the assumption that these are $\Delta_2$-saturated sets. Hence, if all of these sets are $\Delta_1$-temporal cliques, trivially follows that $\mathcal{G}$ is a $(\Delta_1, \Delta_2)$-cluster temporal graph. □

As already mentioned, Lemmas 2.5 and 2.8 allow us to deduce the following result about the computational time that requires to verify if any given temporal graph is or not a $(\Delta_1, \Delta_2)$-cluster temporal graphs.

**Lemma 2.9.** *Let $\mathcal{G} = (G = (V, E), \tau)$ be a temporal graph, and let $\mathcal{E} = \{(e, t) : e \in E, t \in \mathcal{T}(e)\}$ be the set of time-edges of $\mathcal{G}$. Then, we can determine in time $\mathcal{O}(|\mathcal{E}|^3|V|)$ whether $\mathcal{G}$ is a $(\Delta_1, \Delta_2)$-cluster temporal graph.*

*Proof.* We know by Lemma 2.8 that $\mathcal{G}$ is a $(\Delta_1, \Delta_2)$-cluster temporal graph if and only if every $\Delta_2$-saturated set of edges forms a $\Delta_1$-temporal clique. By Lemma 2.5 we can find the unique partition of the $\mathcal{E}$ into $\Delta_2$-saturated subsets in time $\mathcal{O}(|\mathcal{E}|^3|V|)$; our algorithm to do this can easily be adapted so that it also outputs the template generated by each of the edge-sets. To determine whether each edge-set forms a $\Delta_1$-temporal clique, it suffices to consider each template $(X, [a, b])$ in turn, and for each pair $x, y \in X$ to verify whether the edge $xy$ is $\Delta_1$-dense in the interval $[a, b]$. It is clear that we can do this for all templates in time $\mathcal{O}(|\mathcal{E}|)$. □

The next Lemmas concern induced structures in cluster temporal graphs.

The following result proves that the restriction of a temporal graph to any template of a given collection $\mathcal{C}$ that minimises $|\mathcal{G} \triangle \mathcal{G}_{\mathcal{C}}|$ induces a temporal graph whose underlying graph is connected. This property, even if could not appear of deep interest, it will be actually extensively used in the following proofs.

**Lemma 2.10.** *Let $\mathcal{G} = (G, \mathcal{T})$ be a temporal graph, and let $\mathcal{C} \in \mathfrak{T}(\mathcal{G}, \Delta_2)$ minimising*

$$\min_{\mathcal{G}_{\mathcal{C}} \text{ realises } \mathcal{C}} |\mathcal{G} \triangle \mathcal{G}_{\mathcal{C}}|.$$

*Then, for any template $C = (X, [a, b]) \in \mathcal{C}$, the static graph underlying graph of $\mathcal{G}[X]|_{[a,b]}$ is connected.*

*Proof.* Let us assume that for some $C = (X, [a, b]) \in \mathcal{C}$, the graph $G[X]|_{[a,b]}$ is disconnected. This implies that there exists a set $Y \subset V(G)$ such that $Y \neq \emptyset$, $Y \subset X$ and for any $x \in X \backslash Y$ and $y \in Y$, the static edge $xy$ does not belong to the underlying graph $G[X]|_{[a,b]}$. Because $X$ is the vertex set of the template $C = (X, [a, b])$, which by definition is induced by a $\Delta_1$-temporal clique in $\mathcal{G}_{\mathcal{C}}$, the set

$$\Pi = \{(xy, t) : x \in X \backslash Y,\ y \in Y,\ t \in [a, b]\} \subseteq \mathcal{E}(\mathcal{G}) \triangle \mathcal{E}(\mathcal{G}_{\mathcal{C}}),$$

satisfies $|\Pi| \geq \max\{1, \lfloor \frac{b-a}{\Delta_1} \rfloor\} \geq 1$.

We now construct two new templates $C^Y = (Y, [a, b])$ and $C^{X \backslash Y} = (X \setminus Y, [a, b])$ and define the collection $\mathcal{C}' = (\mathcal{C} \backslash C) \cup C^Y \cup C^{X \backslash Y}$. Observe that

$$\mathcal{E}\left(\mathcal{G}_{\mathcal{C}}[X]|_{[a,b]}\right) = \mathcal{E}\left(\mathcal{G}_{\mathcal{C}'}[Y]|_{[a,b]}\right) \cup \mathcal{E}\left(\mathcal{G}_{\mathcal{C}'}[X \backslash Y]|_{[a,b]}\right) \cup \Pi,$$

where the union is disjoint and all other edges in $\mathcal{G}_{\mathcal{C}'}$ are also contained in $\mathcal{G}_{\mathcal{C}}$. Thus,

$$|\mathcal{E}(\mathcal{G}) \triangle \mathcal{E}(\mathcal{G}_{\mathcal{C}'})| < |\mathcal{E}(\mathcal{G}) \triangle \mathcal{E}(\mathcal{G}_{\mathcal{C}})|,$$

contradicting the minimality of $\mathcal{C}$. □

The following result establishes a relationship between the lifetimes of a given temporal graph $\mathcal{G}$ and a $(\Delta_1, \Delta_2)$-cluster temporal graph obtained from $\mathcal{G}$ with minimum edit cost.

**Lemma 2.11.** *For any temporal graph $\mathcal{G}$, there exists a $(\Delta_1, \Delta_2)$-cluster temporal graph $\mathcal{G}'$, minimising the edit distance $|\mathcal{G} \triangle \mathcal{G}'|$ between $\mathcal{G}$ and $\mathcal{G}'$, such that the lifetime of $\mathcal{G}'$ is a subset of the lifetime of $\mathcal{G}$.*

*Proof.* Let $\widetilde{\mathcal{G}}$ be any $(\Delta_1, \Delta_2)$-cluster temporal graph such that $|\widetilde{\mathcal{G}} \triangle \mathcal{G}|$ is minimum. Fix the lifetime of $\mathcal{G}$ to be $T$. It suffices to demonstrate that we can transform $\widetilde{\mathcal{G}}$ into a $(\Delta_1, \Delta_2)$-cluster temporal graph $\mathcal{G}'$ whose lifetime is contained in $[T]$, without increasing the edit distance. Specifically, if $\mathcal{C} = \{C_1, \ldots, C_m\}$ is the set of clique templates generated by $\widetilde{\mathcal{G}}$, we will demonstrate that, for any template $C_i = (X_i, [a_i, b_i])$, we can modify $\widetilde{\mathcal{G}}$ so that its restriction to $X_i$ and $[a_i, b_i]$ is a clique generating the template $(X_i, [a_i, b_i] \cap [T])$, without increasing the edit distance. Notice that modifying each clique in this way cannot violate the $\Delta_2$-independence of the cliques, since we only decrease the lifetime of each clique.

Suppose now that $(X_i, [a_i, b_i]) \in \mathcal{C}$, and $[a_i, b_i] \not\subseteq [T]$. Set $\mathcal{D}_1$ to be the set of time-edges in $\widetilde{\mathcal{G}}[X_i]|_{[a_i,0]}$, and $\mathcal{D}_2$ to be the set of time-edges in $\widetilde{\mathcal{G}}[X_i]|_{[T+1,b_i]}$; by assumption at least one of $\mathcal{D}_1$ and $\mathcal{D}_2$ is non-empty. Note that no time-edges in $\mathcal{D}_1 \cup \mathcal{D}_2$ appear in $\mathcal{G}$. Now set $\mathcal{G}''$ to be the temporal graph obtained from $\widetilde{\mathcal{G}}$ by deleting all time-edges in $\mathcal{D}_1 \cup \mathcal{D}_2$ and adding the set of time-edges

$$\{(e, 1) : \exists t' \text{ with } (e, t') \in \mathcal{D}_1\} \cup \{(e, T) : \exists t' \text{ with } (e, t') \in \mathcal{D}_2\}.$$

Note that we certainly have $|\mathcal{G}'' \triangle \mathcal{G}| \leq |\widetilde{\mathcal{G}} \triangle \mathcal{G}|$, so the edit distance does not increase, and moreover that $\mathcal{G}''[X_i]|_{[a_i,b_i]}$ does not contain any edge-appearances at times not in $[T]$. It remains only to demonstrate that the edges of $\mathcal{G}''[X_i]|_{[a_i,b_i]\cap[T]}$ form a $\Delta_1$-temporal clique, that is, that for every pair of vertices $x, y \in X_i$ the edge $xy$ is $\Delta_1$-dense in $[a_i, b_i] \cap [T]$. Since we did not remove any edge-appearances in this interval, the only way this could happen is if $xy$ did not appear at all in this interval in $\widetilde{\mathcal{G}}$; in this case, we must have $\min\{b_i, T\} - \max\{1, a_i\} < \Delta_1 - 1$, and so a single appearance of the edge $xy$ in this interval – which is guaranteed by construction – is enough to give the required $\Delta_1$-density. □

## 2.4 ETC on paths

In this section we are going to present two results for ETC on inputs limited to temporal graphs with a path as underlying graph. In particular we will show that this problem is NP-Complete even on this restricted class of allowed input graphs. In addition to that, we will also prove that for this same class of temporal graphs, the problem is instead polynomial time solvable if the maximum number of possible appearances of each edge is bounded. Throughout

$P_n$ will denote the path on $V(P_n) = \{v_1, \ldots, v_n\}$ with $E(P_n) = \{v_i v_{i+1} : 1 \leq i < n\}$. Define $\mathfrak{F}_n$ be the set of all temporal graphs $\mathcal{P}_n = (P_n, \mathcal{T})$ on $n$ vertices which have a path $P_n$ as the underlying static graph. For the remainder of this section we will consider only temporal graphs $\mathcal{P}_n \in \mathfrak{F}_n$ which have a path $P_n$ as underlying graph unless it is specified otherwise.

The following theorem states the hardness of ETC even if we restrict the class of allowed inputs to a temporal graph in $\mathfrak{F}_n$.

**Theorem 2.12.** ETC *is* NP*-Complete, even if the underlying graph $G$ of the input temporal graph $\mathcal{G}$ is a path.*

To prove this result we are going to construct a reduction to ETC from the NP-Complete problem Temporal Matching. We recall that for a fixed $\Delta \in \mathbb{Z}^+$, a $\Delta$-*temporal matching* $\mathcal{M}$ of a temporal graph $\mathcal{G}$ is a set of time-edges of $\mathcal{G}$ which are pairwise $\Delta$-independent. It is easy to note that if $\mathcal{G} = \mathcal{M}$, then $\mathcal{G}$ is a $(\Delta_1, \Delta)$-cluster temporal graph for any value of $\Delta_1 \geq 1$, because then each time-edge in $\mathcal{G}$ can be considered as a $\Delta_1$-temporal clique with unit time interval, and these temporal cliques are by definition $\Delta$-independent. We can now state the Temporal Matching problem formally.

Temporal Matching(TM):
*Input:* A temporal graph $\mathcal{G} = (G, \mathcal{T})$ and two positive integers $k, \Delta \in \mathbb{Z}^+$.
*Question:* Does $\mathcal{G}$ admit a $\Delta$-temporal matching $\mathcal{M}$ of size $k$?

It was shown in [39] that Temporal Matching is NP-Complete even if $\Delta = 2$ and the underlying graph $G$ is a path.

***Proof of Theorem 2.12.*** As TM is NP-Hard for temporal graphs with a path as underlying graph and $\Delta = 2$ by [39], we can assume our input has this form. Thus, let $\mathcal{I} = (\mathcal{P}_n, 2, k)$ denote a feasible input instance of TM, where $\mathcal{P}_n = (P_n, \mathcal{T}) \in \mathfrak{F}_n$ and $k \in \mathbb{N}$. From this, we are going to define an instance $\mathcal{I}' = (\mathcal{P}'_n, 1, 5, k')$ of ETC, which is a *yes*-instance if and only if $(\mathcal{P}_n, 2, k)$ is a *yes*-instance for TM. Here $\mathcal{P}'_n = (P_n, \mathcal{T}') \in \mathfrak{F}_n$ is a new temporal graph which has the same path $P_n$ as $\mathcal{P}_n$ as the underlying graph, and we set $k' := |\mathcal{E}(\mathcal{P}_n)| - k$. A key property of our construction is that any pair of consecutive snapshots of $\mathcal{P}_n$ is separated in $\mathcal{P}'_n$ by three empty snapshots which we refer to as 'filler' snapshots; we will refer to all the other snapshots in $\mathcal{P}'_n$ as 'non-filler' snapshots. Formally, our new temporal graph $\mathcal{P}'_n$ has the same vertex set as $\mathcal{P}_n$, and its set of time-edges is

$$\mathcal{E}(\mathcal{P}'_n) = \{(e, 4t - 3) : (e, t) \in \mathcal{E}(\mathcal{P}_n)\}.$$

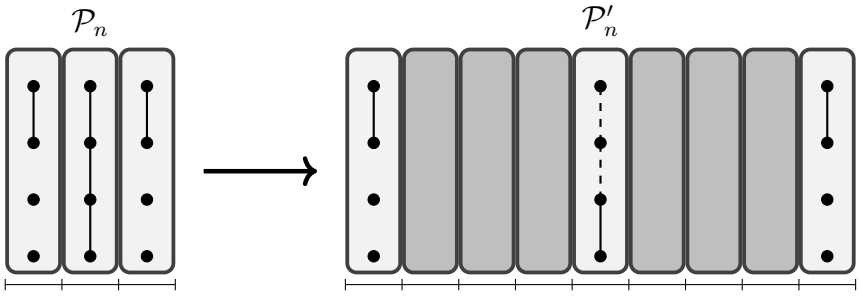



Figure 2.1: The instance $\mathcal{P}$ to the temporal matching problem is shown on the left and the stretched graph $\mathcal{P}'_n$ on which we solve Editing to Temporal Cliques is on the right. Non-filler snapshots are shown in white and filler snapshots are grey. Dotted edges show edges that were removed to leave a $(\Delta_1, \Delta_2)$-cluster temporal graph (which is also a temporal matching $\mathcal{M}'$).

We can assume that the first time-edge of $\mathcal{P}_n$ occurs at time 1 and so this also holds for $\mathcal{P}'_n$. The construction of $\mathcal{P}'_n$ from $\mathcal{P}_n$ is illustrated in Figure 2.1.
[$\Rightarrow$] Let us suppose $(\mathcal{P}_n, 2, k)$ is a *yes*-instance for TM. This means that there exists at least one 2-temporal matching $\mathcal{M}$ of size $k$ in $\mathcal{P}_n$. As we noticed before, $\mathcal{M}$ can be seen as a set of pairwise 2-independent 1-temporal cliques each of which consists of a single time-edge.

**Claim 2.13.** *For any pair $(e,t),(e',t') \in \mathcal{M}$ such that $e \cap e' \neq \emptyset$, we have that $(e, 4t-3)$ and $(e', 4t'-3)$ are 5-independent.*

*Proof of claim.* [2.13]Because $(e,t)$ and $(e',t')$ both belong to $\mathcal{M}$, they must be 2-independent; since $e_1$ and $e_2$ are not disjoint, it follows that $|t - t'| \geq 2$. We therefore deduce that

$$|(4t' - 3) - (4t - 3)| = 4|t' - t| \geq 8, \tag{2.1}$$

so certainly the time-edges $(e, 4t-3)$ and $(e', 4t'-3)$ are 5-independent. $\Diamond$

Now, we set $\Pi := \{(e, 4t-3) \in \mathcal{E}(\mathcal{P}'_n) \colon (e,t) \in \mathcal{E}(\mathcal{P}_n) \setminus \mathcal{M}\}$ be the set of time-edges to be removed from $\mathcal{P}'_n$. By construction it is clear that $|\Pi| = k'$. Thus, after deleting at most $k'$ time-edges from the temporal graph $\mathcal{P}'_n$ we obtain the matching $\mathcal{M}' = \{(e, 4t-2) \colon (e,t) \in \mathcal{M}\}$, which by Claim 2.13 is a $(1,5)$-cluster temporal graph.

[$\Leftarrow$] For the converse direction, suppose $(\mathcal{P}'_n, 1, 5, k')$ with $\mathcal{P}'_n = (P_n, \mathcal{T}') \in \mathfrak{F}_n$ is a *yes*-instance of ETC. We shall now prove three claims about the structure of cliques in optimal solutions to ETC on $\mathcal{P}'_n$.

**Claim 2.14.** *Let $\mathcal{C}$ be any 1-temporal clique on $X$ in an optimal solution to* ETC *on $\mathcal{P}'_n$. Then the end-points of $L(\mathcal{C})$ must be non-filler snapshots of $\mathcal{P}'_n$.*

*Proof of claim.* [2.14] Assume for a contradiction that at least one endpoint of $L(\mathcal{C})$ is a filler snapshot. Filler snapshots do not contain any edges of $\mathcal{P}'_n$, thus not adding the time-edges at this end-point filler snapshot, when constructing $\mathcal{C}$, results in a $\Delta_1$-temporal clique $\mathcal{C}'$ with strictly fewer time-edges than $\mathcal{C}$ whose lifetime is strictly contained within the life time of $\mathcal{C}$. Replacing $\mathcal{C}$ with $\mathcal{C}'$ clearly gives a temporal graph at lesser edit distance from $\mathcal{P}'_n$ than $\mathcal{C}$, so it remains only to verify that this new temporal graph is in fact a $(1,5)$-cluster temporal graph. To see this, note that if $\mathcal{C}$ is 5-independent from all other cliques then so is any subset of $\mathcal{C}$, so we cannot violate the independence condition. We therefore obtain a contradiction to the hypothesis that $\mathcal{C}$ is part of an optimal solution. ◊

**Claim 2.15.** *Let $\mathcal{C}$ be any 1-temporal clique on vertex set $X$ in a solution to* ETC *on $\mathcal{P}'_n$ that contains two time-edges in distinct non-filler snapshots. Then there exists a non-empty set $\mathcal{D}$ of pairwise 5-independent 1-temporal cliques on the vertex set $X$ such that replacing $\mathcal{C}$ with $\mathcal{D}$ results in a $(1,5)$-cluster temporal graph at strictly lower total edit cost from $\mathcal{P}'_n$. Furthermore, for all $\mathcal{D}' \in \mathcal{D}$, we have $L(\mathcal{D}') \subseteq L(\mathcal{C})$, and there exists $\mathcal{D}_1 \in \mathcal{D}$ such that $L(\mathcal{D}_1) = \{t\}$, where $t$ is a non-filler snapshot of $\mathcal{P}'_n$.*

*Proof of claim.* [2.15] Let $\tau = \{t_1, \dots\}$ be the set of non-filler snapshots of $\mathcal{P}'_n$. Let

$$\ell = \min_{i \in \mathbb{Z}^+} \{i : \mathcal{C} \text{ contains an edge at time } t_i \in \tau\}$$
$$r = \max_{i \in \mathbb{Z}^+} \{i : \mathcal{C} \text{ contains an edge at time } t_i \in \tau\},$$

be the leftmost and rightmost (respectively) non-filler snapshots containing a time-edge in $\mathcal{C}$. Note that, because $\Delta_1 = 1$ and $\mathcal{C}$ contains at least two time-edges which correspond to distinct snapshots in $\mathcal{P}_n$, it must be that $\mathcal{C}$ consists of a copy of the static clique on $X$ at every snapshot in its lifetime. Let $|X| = m$. Observe that $\mathcal{C}$ has lifetime $[t_\ell, t_r]$, by Claim 2.14, and so by hypothesis there must be a copy of a clique on $X$ at least in the interval $[t_\ell, t_{\ell+1}]$. Let $\mathcal{C}_1$ be the 1-temporal clique on $X$ with lifetime $\{t_\ell\}$ and if $t_r > t_{\ell+1}$ then let $\mathcal{C}_2$ be the 1-temporal clique on $X$ with lifetime $[t_{\ell+2}, t_r]$, otherwise $\mathcal{C}_2$ is not defined. We will now prove that replacing $\mathcal{C}$ with the set of cliques $\mathcal{D}$ where $\mathcal{D} = \{\mathcal{C}_1, \mathcal{C}_2\}$ if $\mathcal{C}_2$ is defined, and $\{\mathcal{C}_1\}$ otherwise, will result in a $(1,5)$-cluster temporal graph with smaller edit distance from $\mathcal{P}'_n$.

To begin observe that, since $\mathcal{C}$ contains time-edges at least in $t_\ell$ and $t_{\ell+1}$, a copy of the clique on $X$ must be added to each snapshot in $[t_\ell + 1, t_{\ell+1} - 1]$ to form $\mathcal{C}$. Since these edges are not present in $\mathcal{C}_1$ or $\mathcal{C}_2$, not adding them saves $3\binom{m}{2}$ edge additions. Note that if $t_r > t_{\ell+1}$ then additional edge additions in the interval $[t_{\ell+1} + 1, t_{\ell+2} - 1]$ will be saved. When forming

$\mathcal{C}_1$ we must remove all time-edges with both endpoints in $X$ from the snapshot $t_{\ell+1}$, since the underlying graph is a path there are at most $m-1$ such edges. At all snapshots other than $t_\ell+1,\ldots,t_{\ell+2}-1$ we make the same changes as when forming $\mathcal{C}$ and thus there is no net change in the number of edits. Thus at least

$$3\cdot\binom{m}{2}-(m-1)>0,$$

fewer edits are required to form $\mathcal{C}_1$ (and $\mathcal{C}_2$ if $t_r > t_{\ell+1}$) over forming $\mathcal{C}$.

Finally, we can note that even if $\mathcal{C}_2$ was non-empty, then $\mathcal{C}_1$ and $\mathcal{C}_2$ would be still 5-independent by construction. Moreover, since the time-edges of both $\mathcal{C}_1$ and (if it is defined) $\mathcal{C}_2$ are subsets of the time-edges of $\mathcal{C}$, both cliques must be 5-independent from any other temporal clique in the original solution. ◊

Using the two claims above we can establish the following claim.

**Claim 2.16.** *Each clique $\mathcal{C}$ in any optimal solution to* ETC *on $\mathcal{P}'_n$ satisfies $L(\mathcal{C})=\{t\}$, where $t$ is a non-filler snapshot of $\mathcal{P}'_n$.*

*Proof of claim.* [ 2.16]Assume for a contradiction that some 1-temporal clique $\mathcal{C}$ in an optimal solution to ETC has a lifetime greater than one. By Claim 2.14 we can deduce that $L(\mathcal{C})$ contains at least two consecutive non-filler snapshots. It follows immediately from Claim 2.15 that our solution is not optimal, since we know that we can obtain another solution with lesser total edit cost by replacing $\mathcal{C}$ with a different set $\mathcal{D}$ of temporal cliques. This gives the required contradiction. ◊

The next claim is key to the reduction.

**Claim 2.17.** *There exists an optimal solution to* ETC *on $\mathcal{P}'_n$ where each 1-temporal clique consists of a single time-edge $(e,t)$, where $t$ is a non-filler snapshot of $\mathcal{P}'_n$. Furthermore, this solution can be obtained using deletions only.*

*Proof of claim.* [2.17]Let $\{\mathcal{C}_i : 1\le i\le s\}$ be an optimal solution to our instance of ETC. First of all let us note that, by Claim 2.16, each 1-temporal clique $\mathcal{C}_i$ consists of a collection of time-edges that all appear at a single non-filler snapshot $t$ of $\mathcal{P}'_n$. We are going to prove that any of these 1-temporal cliques can be replaced set of pairwise 5-independent single time-edges, all of which are present in the input temporal graph, without increasing the total edit cost.

Let us fix a clique $\mathcal{C}_i$ in our optimal solution, and suppose that $L(\mathcal{C}_i)=\{t\}$, where $t$ is a non-filler snapshot of $\mathcal{P}'_n$. Since the underlying graph is a path, the set of edges appearing at time $t$ consists of either a single path or a disjoint collection of paths. By Lemma 2.10, we may

assume that we may assume that the time-edges of $\mathcal{C}_i$ which also belong to $\mathcal{P}'_n$ form a single path. Suppose that this path has $\ell$ vertices.

First of all we notice that if $\ell \in \{1, 2\}$, then it is already a 1-temporal clique with at most one edge, so there is no modification to apply.

If $\ell = 3$, then $P_3$ consists of two time-edges $\{(uv, t), (vw, t)\}$, so either deleting one of these time-edges, or adding the missing time-edge $(uw, t)$ would create a 1-temporal clique. Since the cost of doing either is the same, we can assume, without loss of generality, that an edge is deleted to obtain a 1-temporal clique consisting of a single time-edge.

Let us now assume $\ell > 3$. Then, deleting $\lfloor \frac{\ell}{2} \rfloor$ time-edges we would obtain a set of 1-temporal cliques which consists of single time-edges and which are pairwise disjoint in the underlying graph, and so are pairwise 5-independent. We now consider the cost of constructing a 1-temporal clique from $P_\ell$ that contains all of its vertices. In this case we will need to add exactly $\frac{\ell(\ell-1)}{2} - (\ell - 1) = \frac{1}{2}(\ell - 1)(\ell - 2)$ time-edges in $\mathcal{P}'_n$ at time $t$. Thus since $\ell > 3$ the number of edges saved by using deletions only is

$$\frac{(\ell-1)(\ell-2)}{2} - \left\lfloor \frac{\ell}{2} \right\rfloor \geq \frac{(\ell-1)(\ell-2)}{2} - \frac{\ell-1}{2} = \frac{(\ell-1)(\ell-3)}{2} > 0.$$

Thus, for any $\ell \geq 1$ there exists a solution with minimum cost which consists of pairwise 5-independent time-edges and can be obtained using only deletion. ◊

By Claim 2.17 above we know that there exists an optimal solution to ETC on $\mathcal{P}'_n$ consisting of pairwise 5-independent time-edges that is obtained using deletions only. Since $(\mathcal{P}'_n, 1, 5, k')$ is a *yes*-instance, we know that there exists such a solution in which the set $\Pi$ of time-edges deleted satisfies $|\Pi| \leq k'$. Let $\mathcal{M}'$ be the set of time-edges in the resulting temporal graph, and note that every element of $\mathcal{M}$ appears in a non-filler snapshot.

**Claim 2.18.** $\mathcal{M} := \{(e, (t+3)/4) \colon (e, t) \in \mathcal{M}'\}$ *is a* 2*-temporal matching.*

*Proof of claim.* [ 2.18] It suffices to demonstrate that, for every pair of time-edges $(e_1, t_1)$ and $(e_2, t_2)$ in $\mathcal{M}$, either $e_1 \cap e_2 = \emptyset$ or $|t_2 - t_1| \geq 2$. Suppose that $e_1 \cap e_2 \neq \emptyset$. By construction of $\mathcal{M}$, we know that $(e_1, 4t_1 - 3)$ and $(e_2, 4t_2 - 3)$ belong to $\mathcal{M}'$. Since $\mathcal{M}'$ is a set of single time-edges which form a $(1, 5)$-cluster temporal graph, we know that the temporal cliques consisting of the single time-edges $(e_1, 4t_1 - 3)$ and $(e_2, 4t_2 - 3)$ are 5-independent; as we are assuming that $e_1 \cap e_2 \neq \emptyset$, this implies that $|(4t_2 - 3) - (4t_1 - 3)| \geq 5$. We can therefore deduce that $4|t_2 - t_1| \geq 5$, implying that $|t_2 - t_1| > 1$; since $t_1, t_2 \in \mathbb{Z}^+$, it follows that $|t_2 - t_1| \geq 2$, as required. ◊

Thus by Claim 2.18 if $(\mathcal{P}'_n, 1, 5, k')$ is a *yes*-instance of ETC problem with $k' = |\mathcal{E}(\mathcal{P})| - k$ then $(\mathcal{P}'_n, 2, k)$ is a *yes*-instance of the TM problem since $|\mathcal{M}| \geq |\mathcal{E}(\mathcal{P})| - k' = k$. □

### 2.4.1 Bounding the number of edge appearances

From the previous section, we know that ETC is NP-hard even on temporal graphs $\mathcal{P}_n \in \mathfrak{F}_n$ which have a path as their underlying graph. We now show that, if additionally the number of appearances of each edge in $\mathcal{P}_n$ is bounded, then ETC can be solved in time polynomial in the size of the input temporal graph. We prove this fact using a fairly standard dynamic programming approach, where we go along the underlying path $P_n$ uncovering one vertex in each step. In particular, at the $i^{\text{th}}$ vertex we try to extend the current set of templates on the first $i-1$ vertices of the path to an optimal set of templates also including the $i^{\text{th}}$ vertex.

In particular, we formulate the following result.

**Theorem 2.19.** *Let $(\mathcal{P}_n, \Delta_1, \Delta_2, k)$ be any instance of* ETC *where $\mathcal{P}_n \in \mathfrak{F}_n$ and there exists a natural number $\sigma$ such that $|\mathcal{T}(e)| \leq \sigma$ for any $e \in E(P_n)$. Then,* ETC *on $(\mathcal{P}_n, \Delta_1, \Delta_2, k)$ is solvable in time $\mathcal{O}(T^{4\sigma}\sigma^2 \cdot n^{2\sigma+1})$.*

We now introduce a few concepts that will be relevant in the proof Theorem 2.19, the main result of this subsection.

Let $\mathcal{C}^i = (C_1, \ldots, C_r)$ be a (potentially empty) collection of templates, where each template $C_j$ with $1 \leq j \leq r$ has vertex set $X_j \subseteq \{v_1, \ldots, v_i\}$ where $|X_j| \geq 2$. For any vertex $v_i \in V(P_n)$ with $i \geq 2$, we will say that $\mathcal{C}^i$ is *feasible for* $v_i$ if it is pairwise $\Delta_2$-independent and respects the following:

(i) for each $C_j = (X_j, [a_j, b_j])$, we have $X_j = \{v_{\ell_j}, v_{\ell_j+1}, \ldots, v_i\}$ for some $\ell_j < i$,

(ii) for each $j$, for any $k \in \{\ell_j, \ldots, i-1\}$ there exists $t \in [a_j, b_j]$ such that $t \in \mathcal{T}(v_k v_{k+1})$.

For the case $i = 1$ we define the empty collection $\mathcal{C} = \emptyset$ to be the only collection feasible for $v_1$. Let $\Gamma^i$ be the class of collections of templates feasible for $v_i$.

The following result shows how collections of templates feasible for vertices relates with collections of templates which are realised by a $(\Delta_1, \Delta_2)$-cluster temporal graph obtained from $\mathcal{P}_n \in \mathfrak{F}_n$ with minimum the edit cost.

**Lemma 2.20.** *Let $n \in \mathbb{Z}^+$, $\mathcal{P}_n \in \mathfrak{F}_n$, and $\mathcal{C} \in \mathfrak{T}(\mathcal{G}, \Delta_2)$ be a collection minimising*

$$\min_{\mathcal{G}_\mathcal{C} \text{ realises } \mathcal{C}} |\mathcal{G} \triangle \mathcal{G}_\mathcal{C}|.$$

*Then, for any set $\mathcal{C}^n \subseteq \mathcal{C}$ of templates that contains $v_n$ we have $\mathcal{C}^n \in \Gamma^n$.*

*Proof.* Given an arbitrary non-empty template $C_j = (X_j, [a_j, b_j]) \in \mathcal{C}^n$ we shall show that it satisfies Properties (i) and (ii) in the definition of feasibility for $v_n$.

*Property* (i): By construction and because $v_n \in X_j$ is and end-vertex of the underlying path $P_n$, we can assume that $C_j$ has as a vertex set a set $X_j$ of consecutive vertices in $P_n$ such that

$\{v_{l_j}, v_{l_j+1}, \ldots, v_n\} \subseteq X_j$ and $v_{l_j-1} \notin X_j$ for some $l_j \leq n$. At least one such set there exists since $|X_j| \geq 2$ and $v_n \in X_j$. We now claim that $X_j \subseteq \{v_{l_j}, v_{l_j+1}, \ldots, v_n\}$. To see this, note that if there exists a further vertex $v \in X_j \setminus \{v_{l_j}, v_{l_j+1}, \ldots, v_n\}$ such that $v \neq v_{l_j-1}$, then $P_n[C_j]$ would be disconnected, which contradicts Lemma 2.10.

*Property* (ii): Now, as we have shown that $C_j$ satisfies property (i), we can assume $X_j = \{v_{l_j}, \ldots, v_n\}$. If we assume that there exists a $k \in \{l_j, \ldots, n-1\}$ such that $[a_j, b_j] \cap \mathcal{T}(v_k v_{k+1}) = \emptyset$ then $C_j$ does not induce a static connected subgraph in $\mathcal{P}_n$. This is in contradiction with Lemma 2.10.

The result then holds for any $C_j \in \mathcal{C}$ since they were arbitrary. □

Let $\mathcal{C}^{i-1} = \{C_1^{i-1}, ..., C_d^{i-1}\}$ and $\mathcal{C}^i = \{C_1^i, ..., C_r^i\}$ be collections of templates which are feasible for $v_{i-1}$ and $v_i$ in $V(P_n)$ respectively. Then, we say that $\mathcal{C}^i$ *extends* $\mathcal{C}^{i-1}$ if for all $C_k^i \in \mathcal{C}^i$ where $|C_k^i| > 2$, there exists a $j \in \{1, .., d\}$ such that $V(C_j^{i-1}) \cup \{v_i\} = V(C_k^i)$. We denote this by $\mathcal{C}^{i-1} \prec \mathcal{C}^i$. The formal connection between these two collections of templates becomes clearer with the following Lemma.

**Lemma 2.21.** *Let $n \in \mathbb{Z}^+$, $\mathcal{P}_n \in \mathfrak{F}_n$, and $\mathcal{C}^i$ be any collection of feasible templates for $v_i \in V(P_n)$. Then the collection $\mathcal{C}^{i-1}$ given by*

$$\mathcal{C}^{i-1} = \left\{(X \backslash \{v_i\}, [a, b]) : (X, [a, b]) \in \mathcal{C}^i,\ |X| > 2\right\},$$

*is feasible for $v_{i-1} \in V(P_n)$ and $\mathcal{C}^{i-1} \prec \mathcal{C}^i$.*

*Proof.* We shall prove this directly by constructing $\mathcal{C}^{i-1}$ from $\mathcal{C}^i$. Let $C_j = (X_j, I_j)$ be any template in $\mathcal{C}^i$. If $X_j = \{v_{i-1}, v_i\}$, then there is nothing to prove and we set $C'_j = \emptyset$. Let us assume that $|X_j| > 2$. By the definition of feasibility $X_j = \{v_{\ell_j}, \ldots, v_i\}$ for some $\ell_j < i$. Let us consider the new template $C'_j$ given by $C'_j = (X_j \setminus \{v_i\}, I_j)$. Then, $C'_j$ satisfies property 1 of feasibility for $v_{i-1} \in V(P_n)$. Furthermore, because $C_j$ satisfies property 2 for $v_i$, then by construction also $C'_j$ satisfies that property for $v_{i-1}$. Thus, given $\mathcal{C}^i$, if we let $\mathcal{C}^{i-1} = \{C'_j : C'_j \in \mathcal{C}^i\}$, then we have $\mathcal{C}^{i-1} \prec \mathcal{C}^i$. □

Let $i \geq 2$ and $\mathcal{C}^i$ be any collection that is feasible for $v_i$, where $C_j^i = (X_j^i, [a_j, b_j])$ for each $C_j^i \in \mathcal{C}^i$. Then, using the functions $g_i, h_i$ and $f_i$ given by (2.3), (2.4) and (2.5) below, we define the function $\vartheta_i[\mathcal{C}^i]$ as follows:

$$\vartheta_i[\mathcal{C}^i] = \min_{\mathcal{C}^{i-1} \prec \mathcal{C}^i} \left[\vartheta_{i-1}[\mathcal{C}^{i-1}] + g_i(\mathcal{C}^i) + h_i(\mathcal{C}^i) + f_i(\mathcal{C}^i)\right], \tag{2.2}$$

and $\vartheta_1[\mathcal{C}^1] = 0$, where $\mathcal{C}^1 = \emptyset$ is the only collection feasible for $v_1$. We now state the functions $f_i, g_i, h_i$.

- The function $g_i : \Gamma^i \to \mathbb{N}$ counts the total number of time-edges from $v_i$ to vertices $u \neq v_{i-1}$ which appear in the $\Delta_1$-temporal cliques which realise all the $C_j^i \in \mathcal{C}^i$ with more than two vertices. This function is given by

$$g_i(\mathcal{C}^i) = \sum_{j=1}^{|\mathcal{C}^i|} \left(|X_j^i| - 2\right) \cdot \max\left(\left\lfloor \frac{|b_j - a_j|}{\Delta_1} \right\rfloor, 1\right). \tag{2.3}$$

- The function $h_i : \Gamma^i \to \mathbb{N}$ counts the number of appearances of $v_{i-1}v_i$ that do not lie in any $\Delta_1$-temporal clique which realises any template in $\mathcal{C}^i$. This function is given by

$$h_i(\mathcal{C}^i) = \left| \mathcal{T}(v_{i-1}v_i) \setminus \bigcup_{j=1}^{|\mathcal{C}^i|} [a_j, b_j] \right|. \tag{2.4}$$

- The function $f_i : \Gamma^i \to \mathbb{N}$ counts the total number of time-edges from $v_i$ to $v_{i-1}$ which appear in the $\Delta_1$-temporal cliques realising each $C_j^i \in \mathcal{C}^i$. Firstly, for each template $C_k^i = (X_k^i, [a_k, b_k])$ we define $\mathcal{T}_k^i = \mathcal{T}(v_{i-1}v_i) \cap [a_k, b_k]$. Let $t_0 = a_k$, $t_{|\mathcal{T}_k^i|+1} = b_k$, and $(t_j)_{j=1}^{|\mathcal{T}_k^i|}$ be the elements of $\mathcal{T}_k^i$ ordered increasingly. Then,

$$f_i(\mathcal{C}^i) = \sum_{k=1}^{|\mathcal{C}^i|} \sum_{j=0}^{|\mathcal{T}_k^i|+1} \left\lfloor \frac{t_{j+1} - t_j - 1}{\Delta_1} \right\rfloor. \tag{2.5}$$

The next lemma shows that the recurrence given by (2.2) counts what we need. Note that with the notation $\mathcal{P}_i = \mathcal{P}_n[v_1, \ldots, v_i]$ we means the temporal graph $\mathcal{P}_n$ restricted to the first $i$ vertices of the underlying path.

**Lemma 2.22.** *Let $i, n \in \mathbb{Z}^+$ where $i \leq n$, $\mathcal{P}_n \in \mathfrak{F}_n$, and $\mathcal{C}^i = \{C_1^i, ..., C_r^i\}$ be any collection of templates feasible for the vertex $v_i \in V(P_n)$. Define $\mathrm{Opt}(\mathcal{C}^i)$ be the minimum number of modifications required to transform $\mathcal{P}_i = \mathcal{P}_n[v_1, \ldots, v_i]$ into a $(\Delta_1, \Delta_2)$-cluster temporal graph which realises a collection $\mathcal{Q}^i$ of pairwise $\Delta_2$-independent templates such that $\{Q \in Q^i : v_i \in Q\} = \mathcal{C}^i$. Then,*

$$\vartheta_i[\mathcal{C}^i] = \mathrm{Opt}(\mathcal{C}^i).$$

*Proof.* We will prove this equality by induction on $i \leq n$, the number of vertices in the induced path.

For the base case ($i = 1$), by the definition of feasibility, the only collection $\mathcal{C}^1$ of templates feasible for $v_1$ is the empty one. As $\vartheta_1[\mathcal{C}^1] = 0$ and any temporal graph on one vertex is a $(\Delta_1, \Delta_2)$-cluster temporal graph the base case follows.

The following claim will be used to prove the inductive step.

**Claim 2.23.** *For any $\mathcal{C}^i \in \Gamma^i$ the minimum number of additions and deletions of time-edges of the form $(v_x v_i, t)$, where $x < i$, required to modify $\mathcal{P}^i$ into a $(\Delta_1, \Delta_2)$-cluster temporal graph realising a collection $\mathcal{Q}^i$ such that $\{Q \in Q^i : v_i \in Q\} = \mathcal{C}^i$ is*

$$f_i(\mathcal{C}^i) + g_i(\mathcal{C}^i) + h_i(\mathcal{C}^i).$$

*Proof of claim.* [ 2.23] First of all, let us note that at step $i-1$ of this algorithm we already have selected a collection $\mathcal{C}^{i-1} \in \Gamma^{i-1}$ as in Proposition 2.21, such that $\mathcal{C}^{i-1} \prec \mathcal{C}^i$. So, in order to extend $\mathcal{C}^{i-1}$ to $\mathcal{C}^i$ we need to count the total number of modifications that we need to apply to any edge of the form $v_x v_i$ with $x < i$ in such a way that it results to be $\Delta_1$-dense exactly in the lifetimes of each template in $\mathcal{C}^i$. We need to count: (i) the minimum time-edges from $v_i$ to $v_j$, where $j < i-1$, we need to add to make each template a $\Delta_1$-temporal clique, (ii) time-edge from $v_i$ to $v_{i-1}$ outside the lifetime of any template that we must remove, and finally (iii) the minimum time-edges from $v_i$ to $v_{i-1}$ we need to add to make each template a $\Delta_1$-temporal clique. Since our input temporal graph has a path as the underlying graph, this covers all possible time-edges of the form $(v_i v_x, t)$.

To begin we must count, for each $C^i_j \in \mathcal{C}^i$, the minimum number of time-edges $(v_z v_i, t)$ where $v_z \in X^i_j \setminus \{v_{i-1}, v_i\}$ and $t \in [a_j, b_j]$, that we need to add in order to link the new vertex $v_i$ to all the other vertices in $v \in X^i_j \setminus \{v_{i-1}, v_i\}$, in such a way every edge $vv_i$ is $\Delta_1$-dense in $[a_j, b_j]$. This number is given by $\left(|X^i_j| - 2\right) \cdot \max\left(\left\lfloor \frac{|b_j - a_j|}{\Delta_1} \right\rfloor, 1\right)$. Summing over $j \in \{1, \ldots, |\mathcal{C}^i|\}$, the minimum number of edges of the kind $v_z v_i$, where $z \leq i-2$, we need to add is given by

$$g_i(\mathcal{C}^i) = \sum_{j=1}^{|\mathcal{C}^i|} \left(|X^i_j| - 2\right) \cdot \max\left(\left\lfloor \frac{|b_j - a_j|}{\Delta_1} \right\rfloor, 1\right).$$

Then, one must consider the number appearances of the edge $v_{i-1}v_i$ which lie outside of the lifetime of any template $C^i_j = (X_j, [a_j, b_j])$ contained in $\mathcal{C}^i$, and thus must be deleted. This value is accounted by the function

$$h_i(\mathcal{C}^i) = \left| \mathcal{T}(v_{i-1}v_i) \setminus \bigcup_{j=1}^{|\mathcal{C}^i|} [a_j, b_j] \right|.$$

Finally we should calculate the minimum number of appearances of the edge $v_{i-1}v_i$ that should be added to $\mathcal{T}(v_{i-1}v_i)$ so that $v_{i-1}v_i$ results to be $\Delta_1$-dense within each template. To do so, let us consider any $C^i_k = (X_k, [a_k, b_k]) \in \mathcal{C}^i$ and let $\mathcal{T}^i_k = \mathcal{T}(v_{i-1}v_i) \cap [a_k, b_k]$ be the set of all the time appearances of the edge $v_{i-1}v_i$ in the interval $[a_k, b_k]$. Now, let $(t_1, ...., t_{|\mathcal{T}^i_k|})$ be the ordered list of all the time appearances of $v_{i-1}v_i$ in $[a_k, b_k]$, and denote $t_0 = a_k$ and $t_{|\mathcal{T}^i_k|+1} = b_k$. Then, the sum

$$\sum_{j=0}^{|\mathcal{T}^i_k|+1} \left\lfloor \frac{t_{j+1} - t_j - 1}{\Delta_1} \right\rfloor \tag{2.6}$$

counts exactly the minimum number of appearances of $v_{i-1}v_i$ that are missing in $[a_k, b_k]$ for any pair $(t_j, t_{j+1})$ of consecutive appearances of $v_{i-1}v_i$ such that $|t_{j+1} - t_j| > \Delta_1$. Summing over all the possible $C_k^i \in \mathcal{C}^i$ gives

$$f_i(\mathcal{C}^i) = \sum_{k=1}^{|\mathcal{C}^i|} \sum_{j=0}^{|\mathcal{T}_k^i|+1} \left\lfloor \frac{t_{j+1} - t_j - 1}{\Delta_1} \right\rfloor. \tag{2.7}$$

The claim now follows from summing $f_i(\mathcal{C}^i), g_i(\mathcal{C}^i)$ and $h_i(\mathcal{C}^i)$. ◊

We will now prove the inductive step by showing that the inequality holds in both the directions for an arbitrary collection $\mathcal{C}^i$ of templates feasible for $v_i$.

We first prove that $\vartheta_i[\mathcal{C}^i] \leq \mathrm{Opt}(\mathcal{C}^i)$**:** for the collection $\mathcal{C}^i$, let $\mathcal{Q}^i$ be any collection of pairwise $\Delta_2$-independent templates with vertex set in $\{v_1, \ldots, v_i\}$ and such that $\{Q \in Q^i : v_i \in Q\} = \mathcal{C}^i$. From $\mathcal{C}^i$, let us define the collection $\mathcal{C}^{i-1} = \big\{(X \backslash \{v_i\}, [a, b]) : (X, [a, b]) \in \mathcal{C}^i,\ |X| > 2\big\}$. Proposition 2.21 shows that $\mathcal{C}^{i-1}$ is feasible for $v_{i-1} \in V(P_n)$, and $\mathcal{C}^i$ extends $\mathcal{C}^{i-1}$.

We now calculate the minimum number of modifications required to transform $\mathcal{P}_i$ into a collection of $\Delta_1$-temporal cliques which realises $\mathcal{Q}^i$, going via a collection which realises $\mathcal{Q}^{i-1} = (\mathcal{Q}^i \setminus \mathcal{C}^i) \cup \mathcal{C}^{i-1}$. We do this in two steps. First, we count the minimum number of edits $\alpha$ required to transform $\mathcal{P}_{i-1}$ into a $(\Delta_1, \Delta_2)$-cluster temporal graph that realises $\mathcal{Q}^{i-1}$. By inductive hypothesis we have $\alpha \geq \vartheta_{i-1}[\mathcal{C}^{i-1}]$. After this first set of modifications, if the resulting temporal graph restricted to the vertex set $\{v_1, \ldots, v_{i-1}\}$ then it agrees with both $\mathcal{Q}^{i-1}$ and $\mathcal{Q}^i$ (where we neglect edge of the form $(v_j v_i, t)$ in the latter case). So all that remains is to consider the number $\beta$ of additions and deletions of time-edges of the form $(v_j v_i, t)$, where $j < i$. However by Claim 2.23 we have $\beta = f_i(\mathcal{C}^i) + g_i(\mathcal{C}^i) + h_i(\mathcal{C}^i)$. Since $\mathcal{Q}^i$ was arbitrary this establishes this case.

Now, we prove that $\vartheta_i[\mathcal{C}^i] \geq \mathrm{Opt}(\mathcal{C}^i)$**:** Let $\mathcal{C}^{i-1} \in \Gamma^i$ be an arbitrary collection such that $\mathcal{C}^{i-1} \prec \mathcal{C}^i$ and let $\mathcal{K}^{i-1}$ be any collection such that

$$\{K \in \mathcal{K}^{i-1} : v_{i-1} \in K\} = \mathcal{C}^{i-1} \quad \text{and} \quad \min_{\mathcal{H} \text{ realises } \mathcal{K}^{i-1}} |\mathcal{P}_{i-1} \triangle \mathcal{H}| = \vartheta_{i-1}[\mathcal{C}^{i-1}].$$

We can then adapt $\mathcal{K}^{i-1}$ into a collection $\mathcal{K}^i$ such that $\{K \in \mathcal{K}^i : v_i \in K\} = \mathcal{C}^i$. Assuming we have modified $\mathcal{P}_{i-1}$ into a $(\Delta_1, \Delta_2)$-cluster temporal graph realising $\mathcal{K}^{i-1}$, any additional modification must be to add or remove time-edges of the form $(v_x v_i, t)$ where $x < i$ to give a temporal graph such that each template in $\mathcal{C}^i$ induces a $\Delta_1$-temporal clique and each such time-edge is contained within a template of $\mathcal{C}^i$. By Claim 2.23 the number of these changes is counted by $f_i(\mathcal{C}^i) + g_i(\mathcal{C}^i) + h_i(\mathcal{C}^i)$. It follows from inductive hypothesis that $\mathcal{P}_i$ can be transformed into a $(\Delta_1, \Delta_2)$-cluster temporal graph realising $\mathcal{K}^i$ by making

$$\vartheta_{i-1}[\mathcal{C}^{i-1}] + f_i(\mathcal{C}^i) + h_i(\mathcal{C}^i) + g_i(\mathcal{C}^i) \tag{2.8}$$

edge alterations. Recall that $\mathcal{C}^{i-1}$ was arbitrary and let $\tilde{\mathcal{C}}^{i-1}$ be a collection minimising (2.8) such that $\tilde{\mathcal{C}}^{i-1} \prec \mathcal{C}^i$. By (2.2), if we set $\mathcal{C}^{i-1} = \tilde{\mathcal{C}}^{i-1}$ then (2.8) evaluates to $\vartheta_i[\mathcal{C}^i]$. Since the final collection of templates found in this way is also considered by the function Opt we obtain $\vartheta_i[\mathcal{C}^i] \geq \mathrm{Opt}(\mathcal{C}^i)$. □

We are now ready to prove Theorem 2.19.

***Proof of Theorem 2.19.*** We will implement the function $\vartheta_i$, defined in (2.2), as a dynamic program to solve the TCE problem on $(\mathcal{P}_n, \Delta_1, \Delta_2, k)$. We start from $\vartheta_1[\mathcal{C}^1] = 0$, where $\mathcal{C}^1 = \emptyset$ by definition of feasibility, and tabulate bottom up until the value of $\vartheta_n[\mathcal{C}^n]$ is calculated for any possible choice of $\mathcal{C}^n \in \Gamma^n$. The result then follows by Lemma 2.20 since we minimise $\vartheta_n[\mathcal{C}^n]$ over all choices of $\mathcal{C}^n \in \Gamma^n$.

We assume that for any edge $e \in E(\mathcal{P}_i)$, the set $\mathcal{T}(e)$ of its time appearances is ordered. Since each edge appears at most $\sigma$ times this prepossessing step takes time at most $\mathcal{O}(n\sigma \log \sigma)$ and so is negligible. We also assume that arithmetic operations on $\mathcal{O}(\log n)$-bit numbers can be performed in constant time.

At each step $i$ the algorithm must scan over all possible choices for the collection $\mathcal{C}^i$ and, for each of them, calculate $\vartheta_i[\mathcal{C}^i]$. The following Claim bounds the number of possibilities for the choice of $\mathcal{C}^i$ from above.

**Claim 2.24.** *For any $1 \leq i \leq n$ we have $|\Gamma^i| \leq T^{2\sigma} n^\sigma$*

*Proof of claim.* [2.24] Let $\mathcal{C}^i = \{C_1, \ldots, C_r\}$ be any collection of templates contained in $\Gamma^i$. By hypothesis, $|\mathcal{T}(v_{i-1}v_i)| \leq \sigma$, so, because $\mathcal{G}[C_j]$ with $j \in [r]$ is connected by Lemma 2.10, then the vertex $v_i$ can appear in at most $\sigma$ templates in $\mathcal{C}^i$, thus there are at most $\sigma$ templates in $\mathcal{C}^i$. From Lemma 2.20, the vertex set of each template $C^i_j \in \mathcal{C}^i$ which contains at least two vertices, is of the form $X_j = \{v_l, \ldots, v_i\}$ with $l \leq i$. Thus, each $X_j$ is completely specified by its left-hand end-vertex $v_\ell$. There are at most $\binom{n}{\sigma} \leq n^\sigma$ ways to choose the $\sigma$ left-hand endpoints. Finally, there are $\binom{T}{2} \leq T^2$ ways to choose the start and finish points of the lifetime interval $[a_j, b_j]$ of a given template $C_j$, thus there are at most $T^{2\sigma}$ ways to choose the time intervals. So, in total there are at most $T^{2\sigma} n^\sigma$ ways to choose the collection $\mathcal{C}^i$. ◊

Now, given a collection $\mathcal{C}^i$ of feasible templates for the vertex $v_i$ of the path, let us analyse the time needed to compute the functions $f_i$, $g_i$ and $h_i$.

From the definition (2.3) of $g_i(\mathcal{C}^i)$ we can see $g_i$ is a sum of $|\mathcal{C}^i|$ items. Each of these items is of the form $(|X^i_j| - 2) \cdot \max\left(\lfloor \frac{|b_j - a_j|}{\Delta_1} \rfloor, 1\right)$. Recall that $|\mathcal{C}^i| \leq \sigma$, so $g_i$ can be calculated in $\mathcal{O}(\sigma)$ time.

From the definition (2.4), $h_i(\mathcal{C}^i)$ checks if each time appearance of $e_i$ lies within one of the intervals $[a_1, b_1] \ldots, [a_{|\mathcal{C}^i|}, b_{|\mathcal{C}^i|}]$. This takes time at most $\mathcal{O}(\sigma^2)$ since each edge appears at most $\sigma$ times and $|\mathcal{C}^i| \leq \sigma$.

Finally, we see from (2.5) that $f_i(\mathcal{C}_i)$ is a double sum. The inner sum needs to compute the set $\mathcal{T}_k^i$ and order it increasingly. Since $\mathcal{T}(v_{i-1}v_i)$ is already sorted this takes time $\mathcal{O}(\sigma)$. Having computed this it performs $|\mathcal{T}_k^i| + 2 \leq \sigma + 2$ many operations, each costing $\mathcal{O}(1)$ time. The outer sum has $|\mathcal{C}^i| \leq \sigma$ many items and so we conclude that computing $f_i(\mathcal{C}_i)$ requires $\mathcal{O}(\sigma^2)$ time.

Since we are tabulating from the bottom up, the values $\vartheta_{i-1}[\mathcal{C}^{i-1}]$ for any $\mathcal{C}^{i-1} \in \Gamma^{i-1}$ are already known. So the computational cost required for looking up each such value is constant. For any two fixed collections $\mathcal{C}^{i-1}$ and $\mathcal{C}^i$, it takes time at most $\mathcal{O}(\sigma^2)$ to check whether $\mathcal{C}^{i-1} \prec \mathcal{C}^i$. This is since we can compare each template in $\mathcal{C}^{i-1}$ with each template in $\mathcal{C}^i$, there $\mathcal{O}(\sigma^2)$ such pairs, and each of these comparison can be done in constant time. Furthermore, the min operator in the recursive definition (2.2) of $\vartheta_i[C^i]$ is taken over all the possible collections $\mathcal{C}^{i-1}$ such that $\mathcal{C}^{i-1} \prec \mathcal{C}^i$, so contributes time $\mathcal{O}(T^{2\sigma} n^\sigma \cdot \sigma^2)$ to the computational cost of (2.2).

To conclude, computing $\vartheta_i[\mathcal{C}^i]$ for any fixed collection $\mathcal{C}^i$ at each step $i$ requires $\mathcal{O}(T^{2\sigma} n^\sigma \cdot \sigma^2)$ time. So, doing this calculation for all feasible collections $\mathcal{C}^i \in \Gamma^i$ at a fixed step $i$ can be done in $\mathcal{O}(T^{4\sigma} n^{2\sigma} \cdot \sigma^2)$ time. Furthermore, this calculation should be repeated $n$ times, one for each step of the algorithm. Thus this dynamic programming algorithm runs in $\mathcal{O}(T^{4\sigma} n^{2\sigma+1} \cdot \sigma^2)$ time. □

## 2.5 Conclusions and Open Problems

In this Chapter we introduced a new temporal interpretation of Cluster Editing which is meant to be a natural extension of the static case, in the sense that similarly to the non-temporal case, we intend a cluster temporal graph as a disjoint union of independent clusters. The technical meanings that we give in this context to the words "independent" and "cluster" is specified in Section 2.2. We called this new version Editing to Temporal Cliques problem and we showed that it turns out to be NP-Complete even if we restrict the class of possible input graphs to that of the temporal graphs with a path as underlying graph. In addition to that we also present a result that shows how this problem becomes instead tractable in polynomial time if the underlying graph is a path and the maximum number of appearances allowed for any of its single edges is bounded by a constant. From our hardness reduction (in particular from Claim 2.17), it follows that the variant of the problem in which we are only allowed to delete, but not add, edge appearances, is also hard in the same setting. For what concern other related questions about the temporal variant of Cluster Editing we propose here, in [7] we also prove the fact that the slightly different version of Editing to Temporal Cliques problem, where only time-edges additions are allowed is solvable in $\mathcal{O}\big(|\mathcal{E}|^3|V|\big)$ time. One natural open question

arising from the first of these positive results is whether bounding the number of appearances of each edge can lead to tractability in a wider range of settings: we conjecture that the techniques used here can be generalised to obtain a polynomial-time algorithm when the underlying graph has bounded pathwidth, and it may be that they can be extended even further.

# Chapter 3

# A Local Characterisation of Cluster Temporal Graphs

In this chapter we are going to present a characterisation that has the aim of allowing to recognise if a given temporal graph $\mathcal{G}$ is or not a $(\Delta_1, \Delta_2)$-cluster temporal graph. Before entering in the details of the characterisation of cluster temporal graphs, we think it is worthwhile to spend a few words about a well known characterisation of cluster graphs in the static case. To this end, we will use the same notation and terminology as in [10]. Let $u, v, w \in V$ be three vertices of a given graph $G$. These three vertices are said to form a *conflict triple* if two edges $uv, vw \in E(G)$ but the third edge $uw$, that one which would complete the triangle, does not belong to the edge set of the graph $G$. So, for any unweighted static graph, the following is considered a standard characterisation.

**Characterisation of Cluster Graphs.** A graph $G$ is a *cluster graph* if and only if it does not contains any *conflict triple* as induced subgraph.

In other words, $G$ is a cluster graph if and only if it does not contain any $P_3$ as induced subgraph. Thinking about some of the complications that arise even in the attempt of defining the analogous of a static clique into the temporal setting, that we discussed in Chapter 2, it will be no surprise the fact that finding a characterisation of cluster temporal graph it is not likewise straightforward as in the static case. In the rest of this Chapter we will focus on the proof of a local characterisation of $(\Delta_1, \Delta_2)$-cluster temporal graph, where the adjective "local" is meant in the sense that it is given in terms of five-vertex subgraphs which are required to satisfy a specific property. In the following we will refer to this property as *five vertex condition*. Furthermore, we will show that this characterisation is the best possible in terms of the number of vertices in the induced subgraphs.

First of all, let us recall that $(\Delta_1, \Delta_2)$-cluster temporal graphs are only defined for $\Delta_2 > \Delta_1 \geq 1$, since otherwise the decomposition of such a graph into $\Delta_1$-temporal cliques is not unique. In particular the characterisation we want to show has the following formulation.

**Theorem 3.1.** *Let $\mathcal{G}$ be any temporal graph, and $\Delta_2 > \Delta_1 \geq 1$. Then, the following holds.*

- *If $\Delta_1 = 1$ then $\mathcal{G}$ is a $(1, \Delta_2)$-cluster temporal graph if and only if $\mathcal{G}[S]$ is a $(1, \Delta_2)$-cluster temporal graph for every set $S \subseteq V(\mathcal{G})$ of at most three vertices.*
- *If $\Delta_1 > 1$ then $\mathcal{G}$ is a $(\Delta_1, \Delta_2)$-cluster temporal graph if and only if $\mathcal{G}[S]$ is a $(\Delta_1, \Delta_2)$-cluster temporal graph for every set $S \subseteq V(\mathcal{G})$ of at most five vertices.*

Before proving in detail Theorem 3.1, in the following lemma, we shows through a counter-example that for $\Delta_1 > 1$ a characterisation as above cannot be obtained by considering only subsets of at most four vertices. As an immediate consequence, they are needed at least five vertices to find a proper characterisation of this king of $(\Delta_1, \Delta_2)$-temporal graphs and this validate Theorem 3.1.

**Lemma 3.2.** *For any $\Delta_2 > \Delta_1 \geq 2$, there exists a temporal graph on 5 vertices which is not a $(\Delta_1, \Delta_2)$-cluster temporal graph but every induced temporal subgraph of size at most four is a $(\Delta_1, \Delta_2)$-cluster temporal graph.*

*Proof.* Let us consider the temporal graph $\mathcal{G} = (G, \mathcal{T})$ shown in Figure 3.1 and fix any $\Delta_2 > \Delta_1 \geq 2$. We claim that $\mathcal{G}$ is not a $(\Delta_1, \Delta_2)$-cluster temporal graph. First note that $\mathcal{G}$ cannot be a $\Delta_1$-temporal clique on five vertices because at least the two static edges $ad$ and $be$ are missing in the underlying graph. Secondly, $\mathcal{G}$ is not a union of disjoint temporal cliques. By $\Delta_1$-density, the three time-edges $(ab, 2), (bc, 1)$ and $(ca, 1)$ and the three $(cd, \Delta_2 + 2), (de, \Delta_2 + 1)$ and $(ce, \Delta_2 + 2)$ must necessarily belong to two distinct maximal $\Delta_1$-temporal cliques, let us say $\mathcal{Q}_1$ and $\mathcal{Q}_2$ respectively. However, these two cliques are not $\Delta_2$-independent because they are adjacent in the underlying graph, sharing the vertex $c$, and the two edges $(ab, 2) \in \mathcal{Q}_1$ and $(de, \Delta_2 + 1) \in \mathcal{Q}_2$ are not $\Delta_2$-independent, i.e. $L(\mathcal{Q}_2) - L(\mathcal{Q}_1) < \Delta_2$.

Let us consider now any subset $S$ of $V(G)$ with 4 vertices.

*Case (i):* $c \in S$. In this case, for any choice of the other vertices in $S$ the underlying graph of $\mathcal{G}[S]$ is a triangle which includes $c$ plus an edge $cx$ where $x \in \{a, b, d, e\}$ is not in the triangle. If $x \in \{a, b\}$ then $(cx, 1)$ is $\Delta_2$-independent with respect to the triangle $cde$. On the other hand, if $x \in \{d, e\}$ then $(cx, \Delta_2 + 2)$ is $\Delta_2$-independent with respect to the triangle $abc$. Thus, in any case, we would obtain two $\Delta_2$-independent $\Delta_1$-temporal cliques.

*Case (ii):* $c \notin S$. In this case $\mathcal{G}[S]$ consists only of two disjoint time-edges; so clearly it is a $(\Delta_1, \Delta_2)$-cluster temporal graph. □

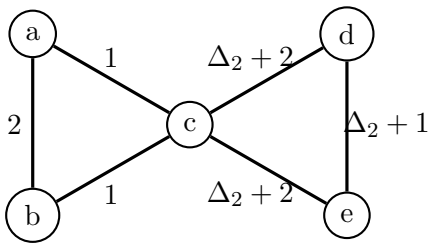



Figure 3.1: The counter-example from Lemma 3.2. This temporal graph shows that 4-vertex subgraphs are not sufficient to characterise $(\Delta_1, \Delta_2)$-cluster temporal graphs.

By Lemma 3.2 it is clear that for a proper characterisation of temporal cluster graphs, it is necessary to search for a condition which involves more than four vertices. In the following we are going to show in detail the proof of Theorem 3.1, whose statement instead imposes a condition on subsets of vertices of at most five vertices.

One direction of Theorem 3.1 follows easily from the following Lemma 3.3. The other direction is far more challenging to prove and is shown in Lemmas 3.5 and 3.10 below. In particular, in Lemma 3.10, we will analyse the special case of $\Delta_1 = 1$, where it is clear that one must consider sets of at least three vertices, as this is the case for static cluster graphs. In the following we will say that a temporal graph $\mathcal{G}$ satisfies the *five vertex condition* if every temporal subgraph of $\mathcal{G}$ induced by (at most) five vertices is a $(\Delta_1, \Delta_2)$-cluster temporal graph.

As before mentioned, we will start proving the following result which essentially concerns induced structures in cluster temporal graphs and from which directly follows the first direction of Theorem 3.1.

**Lemma 3.3.** *Let $\mathcal{G}$ be a $(\Delta_1, \Delta_2)$-cluster temporal graph and $S \subseteq V(\mathcal{G})$. Then, $\mathcal{G}[S]$ is also a $(\Delta_1, \Delta_2)$-cluster temporal graph.*

*Proof.* Let $\mathcal{K}$ be an arbitrary $\Delta_1$-temporal clique of $\mathcal{G}$. We shall show that $\mathcal{K}[S]$ is a $\Delta_1$-temporal clique. The result will then follow since a $\mathcal{G}$ consists of a union of $\Delta_2$-independent $\Delta_1$-temporal cliques and two such $\Delta_1$-temporal cliques will still be $\Delta_2$-independent after restricting to $S$ since $\mathcal{K}[S] \subseteq \mathcal{K}$.

Recall that $\mathcal{K}[S] = \{(xy, t) \in \mathcal{K} : x, y \in S\}$ and let $C = (X, [a, b])$ be the template realised by the set of time-edges $\mathcal{K}[S]$. It remains to show that for all $x, y \in X$, the edge $xy$ is $\Delta_1$-dense in $[a, b]$. Assume otherwise, then we can find some $x, y \in X = V(\mathcal{K}) \cap S$ and an interval $[c, d] \subseteq [a, b]$ of length at least $\Delta_1$ with no appearance of $xy$. However since $x, y \in V(\mathcal{K})$ and $[a, b] \subseteq L(\mathcal{K})$ this contradicts the fact that $\mathcal{K}$ is a $\Delta_1$-temporal clique. □

From Lemma 3.3 we can deduce immediately that the first direction of Theorem 3.1, precisely that one which says that any $(\Delta_1, \Delta_2)$-cluster temporal graph respects the five vertex condition,

holds no matter what the value of $\Delta_1$ is. The other direction will be established in Lemma 3.5 and requires considerably more work. The following lemma is quite simple but illustrates a key idea in the proof of Theorem 3.1: the five vertex condition allows us to 'grow' certain sets of time-edges.

**Lemma 3.4.** *Let $\mathcal{G}$ be any temporal graph satisfying the property that $\mathcal{G}[S]$ is a $(\Delta_1, \Delta_2)$-cluster temporal graph for every set $S \subseteq V(\mathcal{G})$ of at most five vertices. Let $\mathcal{H}$ be a $\Delta_1$-temporal clique realising the template $(H, [c, d])$, and $x, y \in H$. Suppose that $xy$ is $\Delta_1$-dense in the set $[a, b] \supseteq [c, d]$ and let $r_1 = \min(\mathcal{T}(xy) \cap [a, b])$ and $r_2 = \max(\mathcal{T}(xy) \cap [a, b])$. Then there exists a $\Delta_1$-temporal clique $\mathcal{H}' \supseteq \mathcal{H}$ which realises the template $(H, [r_1, r_2])$ where $[r_1, r_2] \supseteq [a + \Delta_1 - 1, b - \Delta_1 + 1]$.*

*Proof.* Fix an arbitrary pair $w, z \in H$. Because $\mathcal{H}$ is a $\Delta_1$-temporal clique, the edge $wz$ is $\Delta_1$-dense in $[c, d]$. We now consider the set $A = \{x, y, w, z\} \subseteq H$. Observe that $\mathcal{K}[A]$ is a $\Delta_1$-temporal clique and thus an indivisible set. By hypothesis $xy$ is $\Delta_1$-dense in $[a, b]$ and so if $A_{xy}$ is the set of time appearances of $xy$ in $[a, b]$ then this set is $\Delta_2$-indivisible. Since $x, y \in A$ and $[a, b] \supseteq [c, d]$ we have $\mathcal{K}[A] \cap A_{xy} \neq \emptyset$, thus $\mathcal{K}[A] \cup A_{xy}$ is $\Delta_2$-indivisible by Lemma 2.3. By five vertex condition we know that $\mathcal{G}[A]$ is a $(\Delta_1, \Delta_2)$-cluster temporal graph, therefore by Lemma 2.7 the set $\mathcal{K}[A] \cup A_{xy}$ must be contained within one $\Delta_1$-temporal clique $\mathcal{H}'$ with lifetime $[r_1, r_2]$. So, since the choice of $w, z \in H$ was arbitrary the result holds. Since $A_{xy} \subseteq \mathcal{H}'$, where $A_{xy}$ is a set of edges which are $\Delta_1$-dense on $[a, b]$, it follows that $r_1 \leq a + \Delta_1 - 1$ and $r_2 \geq b - \Delta_1 + 1$. □

Now, we have all the elements for proving the converse direction of Theorem 3.1, which is separately formulated in the following Lemma 3.5. As we mentioned before, this direction results to be a lot more involved than the direct one which instead immediately follows from Lemma 3.3.

**Lemma 3.5.** *Let $\mathcal{G}$ be a temporal graph and suppose that, for every set $S$ of at most five vertices in $\mathcal{G}$, the induced temporal subgraph $\mathcal{G}[S]$ is a $(\Delta_1, \Delta_2)$-cluster temporal graph. Then $\mathcal{G}$ is a $(\Delta_1, \Delta_2)$-cluster temporal graph.*

*Proof.* Let $\mathcal{P}_\mathcal{G}$ be the partition of $\mathcal{E}(\mathcal{G})$ into $\Delta_2$-saturated subsets; we know that this partition exists and is unique by Lemma 2.4. Let us select any subset $S \in \mathcal{P}_\mathcal{G}$ and suppose that $L(S) = [s, t]$. We want to show that $S$ forms a $\Delta_1$-temporal clique in $\mathcal{G}$.

To prove this, we introduce a collection $\varkappa_S = \{S_1, \ldots, S_m\}$ of subsets of $S$, such that each $S_i$ is a $\Delta_1$-temporal clique for any $i \in \{1, \ldots, m\}$, $S = \bigcup_{i=1}^{m} S_i$ and for every $S_i \in S$ there does not exist any other $\Delta_1$-temporal clique $K \subseteq S$ such that $S_i \subsetneq K$. First of all, we note that this collection exists: since a singleton set consisting of any time-edge is a $\Delta_1$-temporal clique, it follows that every time-edge in $S$ belongs to at least one $\Delta_1$-temporal clique. We also observe

that for the way we define this collection the subsets $S_i$ with $i \in \{1, \ldots, m\}$ are not required to be pairwise disjoint. Then, we will say that each $S_i$ is a *maximal $\Delta_1$-temporal clique within $S$*.

We will assume for a contradiction that $m \geq 2$. Because $S$ is $\Delta_2$-saturated, it is not possible that all the $\Delta_1$-temporal cliques contained in $\varkappa_S$ are pairwise $\Delta_2$-independent, since this would imply that $S$ is not $\Delta_2$-indivisible. Thus, as we assume $m \geq 2$, let us consider any distinct $S_i, S_j \in \varkappa_S$ that are not $\Delta_2$-independent. We shall then show that they must both be contained within a larger $\Delta_1$-temporal clique, which itself is contained in $S$, contradicting maximality. It will then follow that $m = 1$ and thus $S$ is itself a $\Delta_1$-temporal clique, which establishes the lemma.

The next claim shows that if two maximal $\Delta_1$-temporal cliques in $S$ are not $\Delta_2$-independent, then there is a small subgraph witnessing this non-independence.

**Claim 3.6.** *Let $S_i, S_j \in \varkappa_S$ be any pair of $\Delta_1$-temporal cliques which are not $\Delta_2$-independent. Then, there exists a set $W \subseteq V(S_i) \cup V(S_j)$ with $|W| \leq 5$ such that $W$ contains a vertex $c \in V(S_i) \cap V(S_j)$, and $(S_i \cup S_j)[W]$ is $\Delta_2$-indivisible and contains the time-edges $(xc, r_i) \in S_i$ and $(zc, r_j) \in S_j$.*

*Proof of claim.* [3.6] As $S_i$ and $S_j$ are $\Delta_1$-temporal cliques, it follows that $S_i[W]$ and $S_j[W]$ are $\Delta_1$-temporal cliques for any $W \subseteq V$ by Lemma 3.3. Since $S_i$ and $S_j$ are chosen so that they are not $\Delta_2$-independent, they must share at least one vertex $c$ and the following cases can occur:

*Case 1:* there exist respectively two time-edges $(xc, r_i) \in S_i$ and $(zc, r_j) \in S_j$ such that $|r_i - r_j| < \Delta_2$. For this case set $W = \{c, x, z\}$. The set $I = \{(cx, r_i), (cz, r_j)\}$ is $\Delta_2$-indivisible and has intersection with both $S_i[W]$ and $S_j[W]$. Thus by Lemma 2.3, $I \cup S_i[W]$ and $I \cup S_j[W]$ are both $\Delta_2$-indivisible. Applying Lemma 2.3 again gives that $S_i[W] \cup S_j[W] = (S_i \cup S_j)[W]$ is $\Delta_2$-indivisible.

*Case 2:* there exist respectively two time-edges $(xy, t) \in S_i$ and $(zw, t') \in S_j$ such that $|t - t'| < \Delta_2$, (incidentally, Figure 3.1 is an example of this kind of structure). Let us note that, because $c \in V(S_i) \cap V(S_j)$, then both $x, y, z$ and $w$ must be adjacent to the vertex $c$ in the underlying graph. Let us set $W = \{c, x, y, z, w\} \subset V(S_i) \cup V(S_j)$. Let $A_c$ be the set of appearances of the edges $cx, cy, cz$ and $cw$ in $L(S_i) \cup L(S_j)$, thus $A_c \cup \{(xy, t), (zw, t')\} \subseteq (S_i \cup S_j)[W]$ is $\Delta_2$-indivisible and intersects $S_i[W]$ and $S_j[W]$. So, as before, applying Lemma 2.3 gives that $S_i[W] \cup S_j[W] = (S_i \cup S_j)[W]$ is $\Delta_2$-indivisible. Furthermore, since $x, y, c \in V(S_i)$ and $S_i$ is a $\Delta_1$-temporal clique, there must exist at least one time edge $(xc, r_i) \in S_i$ that is at distance at most $\Delta_1 < \Delta_2$ from $(xy, t)$, thus $(xc, r_i) \in (S_i \cup S_j)[W]$ by Lemma 2.3. The same applies for some $(cz, r_j) \in S_j$. $\Diamond$

Recall that $S_i$ and $S_j$ are both $\Delta_2$-indivisible as they are $\Delta_1$-temporal cliques, and that by Claim 3.6 there exists a set $W \subseteq V$ such that $(S_i \cup S_j)[W]$ is $\Delta_2$-indivisible and has non-empty

intersection with $S_i$ and $S_j$. It follows from Lemma 2.3 that both $S_i[W] \cup S_j$ and $S_j[W] \cup S_i$ are $\Delta_2$-indivisible. As their intersection is $(S_i \cup S_j)[W] \neq \emptyset$, invoking Lemma 2.3 once again gives that $S_i \cup S_j$ is $\Delta_2$-indivisible. Recall that $L(S) = [s, t]$.

**Claim 3.7.** *Let $S_i$ and $S_j$ be as above, where $L(S_i) = [s_i, t_i]$, and $L(S_j) = [s_j, t_j]$. Then, there exists some $K \subseteq V$ and $s', t' \in \mathbb{Z}^+$ such that $\mathcal{G}$ contains a $\Delta_1$-temporal clique $\mathcal{K}$ which generates the template $(K, [s', t'])$ where $s' \in [s, \min\{s_i, s_j\} + \Delta_1 - 1]$, $t' \in [\max\{t_i, t_j\} - \Delta_1 + 1, t]$, $(xc, r_i) \in S_i \cap \mathcal{K}$, and $(cz, r_j) \in S_j \cap \mathcal{K}$.*

*Proof of claim.* [3.7] By Claim 3.6, there exists a set $W \subseteq V$ of at most five vertices and two time-edges $(xc, r_i) \in S_i$ and $(cz, r_j) \in S_j$ such that $(S_i \cup S_j)[W]$ is $\Delta_2$-indivisible and $(xc, r_i), (cz, r_j) \in (S_i \cup S_j)[W]$. As $\mathcal{G}[W]$ is $(\Delta_1, \Delta_2)$-cluster temporal graph, by the five vertex condition we can apply Lemma 2.7 to see that $(S_i \cup S_j)[W]$ is contained within a single $\Delta_1$-temporal clique $\mathcal{K}'$. Since $(xc, r_i), (cz, r_j) \in (S_i \cup S_j)[W]$, we see that the lifetime $L(\mathcal{K}') = [k_1, k_2]$ of this $\Delta_1$-temporal clique intersects both $L(S_i)$ and $L(S_j)$.

The edge $xc$ is $\Delta_1$-dense in both $S_i$ and $\mathcal{K}'$. Since $(xc, r_i) \in S_i \cap \mathcal{K}'$, we can further deduce that the edge $xc$ is $\Delta_1$-dense in $L(S_i) \cup L(\mathcal{K}') = [\min\{s_i, k_1\}, \max\{t_i, k_2\}]$. Applying Lemma 3.4 to $\mathcal{K}'$ and $xc$, where we take $[a, b] = L(S_i) \cup L(\mathcal{K}')$ and $[c, d] = L(\mathcal{K}')$, we can extend $L(\mathcal{K}')$ to an interval $[r_1^{xc}, r_2^{xc}]$ satisfying

$$[r_1^{xc}, r_2^{xc}] \supseteq [\min\{s_i, k_1\} + \Delta_1 - 1, \max\{t_i, k_2\} - \Delta_1 + 1]\,. \tag{3.1}$$

Similarly, $cz$ is $\Delta_1$-dense in $L(S_j) \cup L(\mathcal{K}') = [\min\{s_j, k_1\}, \max\{t_j, k_2\}]$. Applying Lemma 3.4 to $\mathcal{K}'$ and $cz$ with $[a, b] = L(S_j) \cup L(\mathcal{K}')$ and $[c, d] = L(\mathcal{K}')$ extends the lifetime of $\mathcal{K}'$ to an interval $[r_1^{cz}, r_2^{cz}]$ satisfying

$$[r_1^{cz}, r_2^{cz}] \supseteq [\min\{s_j, k_1\} + \Delta_1 - 1, \max\{t_j, k_2\} - \Delta_1 + 1]\,. \tag{3.2}$$

It follows that there exists some $\Delta_1$-temporal clique $\mathcal{K}$ in $\mathcal{G}$ with vertex set $K$ and lifetime $L(\mathcal{K})$ satisfying $L(\mathcal{K}) \supseteq [\min\{r_1^{xc}, r_1^{cz}\}, \max\{r_2^{xc}, r_2^{cz}\}]$. Observe that from (3.1) and (3.2) we have

$$\begin{aligned}
\min\{r_1^{xc}, r_1^{cz}\} &= \min\{\min\{s_i, k_1\} + \Delta_1 - 1, \min\{s_j, k_1\} + \Delta_1 - 1\} \\
&= \min\{\min\{s_i, k_1\}, \min\{s_j, k_1\}\} + \Delta_1 - 1 \\
&\leq \min\{s_i, s_j\} + \Delta_1 - 1,
\end{aligned}$$

and similarly $\max\{r_2^{xc}, r_2^{cz}\} \geq \max\{t_i, t_j\} - \Delta_1 + 1$. Finally, since $S$ is $\Delta_2$-saturated, it follows by Lemma 2.6 that $L(\mathcal{K}) \subseteq L(S) = [s, t]$. $\Diamond$

Let us now consider $\mathcal{K}$, the $\Delta_1$-temporal clique of Claim 3.7. From this we want to extend $S_i$ and $S_j$ to a $\Delta_1$-temporal clique with vertex set $V(S_i) \cup V(S_j)$ and lifetime at least $L(\mathcal{K}) \cup L(S_i) \cup L(S_j) = [\min\{s_i, s_j, s'\}, \max\{t_i, t_j, t'\}]$. We achieve this over the following two claims.

**Claim 3.8.** *Let $\mathcal{K} = (K, [s', t'])$ be as specified in Claim 3.7. Then there exist $h_1 \leq h_2$ satisfying*

$$[h_1, h_2] \supseteq [\min\{s_i, s_j, s'\} + \Delta_1 - 1, \max\{t_i, t_j, t'\} - \Delta_1 + 1],$$

*and a $\Delta_1$-temporal clique in $\mathcal{G}$ generating the template $(V(S_i) \cup V(S_j), [h_1, h_2])$. Furthermore, this $\Delta_1$-temporal clique contains two edges $(xc, r_i) \in S_i \cap \mathcal{K}$ and $(cz, r_j) \in S_j \cap \mathcal{K}$.*

*Proof of claim.* [3.8] By Claim 3.7 the $\Delta_1$-temporal clique $\mathcal{K}$ contains the time-edges $(xc, r_i) \in S_i$ and $(cz, r_j) \in S_j$. Let $A_{xc}$ and $A_{cz}$ be the set of appearances of $xc$ in $L(S_i) \cup L(\mathcal{K})$ and $L(S_j) \cup L(\mathcal{K})$ respectively. The edge $xc$ is $\Delta_1$-dense in both $L(S_i)$ and $L(\mathcal{K})$, thus since $(xc, r_i) \in S_i \cap \mathcal{K}$, it follows that $xc$ is $\Delta_1$-dense in $L(S_i) \cup L(\mathcal{K})$. Similarly, $cz$ is $\Delta_1$-dense in $L(S_j) \cup L(\mathcal{K})$. Since both $A_{xc}$ and $A_{cz}$ contain a set of edges that is dense in $L(\mathcal{K})$, we see that $A_{xc} \cup A_{cz}$ must be $\Delta_2$-indivisible. Let $r_1$ and $r_2$ respectively be the earliest and latest times in $L(S_i) \cup L(S_j) \cup L(\mathcal{K})$ at which either $xc$ or $cz$ appear. Note that as $xc$ is $\Delta_1$-dense in $L(S_i) \cup L(\mathcal{K})$ we must have $r_1 \leq \min\{s_i, s'\} + \Delta_1 - 1$. Applying the same reasoning to $cz$ we have $r_1 \leq \min\{s_j, s'\} + \Delta_1 - 1$, thus $r_1 \leq \min\{s_i, s_j, s'\} + \Delta_1 - 1$. By a symmetric argument we also see that $r_2 \geq \max\{t_i, t_j, t'\} - \Delta_1 + 1$.

Let us fix an arbitrary pair of vertices $u, v \in V(S_i) \cup V(S_j)$. We want to show that $uv$ is $\Delta_1$-dense in an interval containing $[r_1, r_2]$. Suppose first that $u \in V(S_i)$ and $v \in V(S_j)$. Since $S_i$ and $S_j$ are $\Delta_1$-temporal cliques, we know that the edge $ux$ exists in the underlying graph of $S_i$ and is $\Delta_1$-dense in $L(S_i)$; similarly $zv$ is present in the underlying graph of $S_j$ and is $\Delta_1$-dense in $L(S_j)$. Let $A_{ux}$ be the set of appearances of $ux$ in $L(S_i)$ and $A_{zv}$ the set of appearances of $zv$ in $L(S_j)$. Note that, because both the $\Delta_2$-indivisible sets $A_{ux}$ and $A_{xc}$ are $\Delta_1$-dense in $L(S_i)$, then $A_{ux} \cup A_{xc}$ is $\Delta_2$-indivisible; similarly $A_{zv} \cup A_{cz}$ is $\Delta_2$-indivisible too. Recall that $A_{xc} \cup A_{cz}$ is also $\Delta_2$-indivisible. It follows that $A_{ux} \cup A_{xc} \cup A_{cz} \cup A_{zv}$ is an $\Delta_2$-indivisible set by two applications of Lemma 2.3. Let us consider the temporal graph $\mathcal{G}[N]$ induced by the set $N = \{u, x, c, z, v\}$. By the five vertex condition, we know that this must form a $(\Delta_1, \Delta_2)$-cluster temporal graph and hence, since the set $A_{ux} \cup A_{xc} \cup A_{cz} \cup A_{zv}$ is $\Delta_2$-indivisible, the restriction of $\mathcal{G}[N]$ to the interval $L(S_i) \cup L(S_j) \cup L(\mathcal{K})$ must form a single $\Delta_1$-temporal clique by Lemma 2.7. The lifetime $L_{u,v}$ of this $\Delta_1$-temporal clique contains the interval $[r_1, r_2]$, and every edge appears at least once within $\Delta_1$ timesteps of $r_1$ and once within $\Delta_1$ timesteps of $r_2$. For the remaining cases where $u, v \in V(S_i)$ or $u, v \in V(S_j)$ we can use a similar argument. However if $u, v \in V(S_i)$ it is enough to consider indivisibility of the set $A_{xc} \cup A_{ux} \cup A_{xv}$, and if $u, v \in V(S_i)$ one considers $A_{cz} \cup A_{uz} \cup A_{zv}$.

Up to this point we have shown that for any $u, v \in V(S_i) \cup V(S_j)$ the edge $uv$ is $\Delta_1$-dense in an interval $L_{u,v}$ containing $[r_1, r_2]$. We now argue that there is an interval $[h_1, h_2]$ such that, for every pair $u, v \in V(S_i) \cup V(S_j)$, the edge $uv$ is $\Delta_1$-dense in $[h_1, h_2]$. If every such edge has at least one appearance in the interval $[r_1, r_2]$, then we can set $[h_1, h_2] = [r_1, r_2]$ and we are done. In particular we can note that this condition is certainly respected if $|r_2 - r_1| \geq \Delta_1 - 1$. Otherwise, it could happen that there exists another pair of vertices $u', v' \in V(S_i) \cup V(S_j)$ such that $|L_{u,v} \cap L_{u',v'}| < \Delta_1 - 1$, in which case there might be no interval in which both the edges $uv$ and $u'v'$ are $\Delta_1$-dense.

Let us assume for a contradiction that there is no interval containing $[r_1, r_2]$ in which every edge with both endpoints in $V(S_i) \cup V(S_j)$ is $\Delta_1$-dense. Recall that this can only happen if $r_2 < r_1 + \Delta_1 - 1$. Moreover we can assume there does not exist an interval $[r_1', r_2'] \supseteq [r_1, r_2]$ with $r_2' \leq r_1' + \Delta_1 - 1$ such that every edge $u'v'$, where $u', v' \in V(S_i) \cup V(S_j)$, has at least one appearance in $[r_1', r_2']$. We now fix two edges $uv$ and $u'v'$, with $u, v, u', v' \in V(S_i) \cup V(S_j)$, such that there is no interval containing $[r_1, r_2]$ in which both $uv$ and $u'v'$ are $\Delta_1$-dense. Recall from earlier reasoning that any edge $e$ with both endpoints in $V(S_i) \cup V(S_j)$ is $\Delta_1$-dense in the interval $L_e$ which contains $[r_1, r_2]$ and moreover, that there are appearances of $e$ within $L_e$ which are also within $\Delta_1$ time steps of $r_1$ and $r_2$ respectively. We claim that each such $e$ appears at least once in the interval $I = [r_2 - \Delta_1 + 1, r_1 + \Delta_1 - 1]$. Observe that this interval contains at least $\Delta_1 + 1$ and at most $2\Delta_1 - 1$ time steps by the condition $r_2 < r_1 + \Delta_1 - 1$. Thus the interval must contain at least one time appearance of $e$ since there are appearances of $e$ in $L_e$ at or before time $r_1 + \Delta_1 - 1$ and at or after time $r_2 - \Delta_1 + 1$. For some edge $e$ let $I_e$ be the set of appearances of $e$ in $I = [r_2 - \Delta_1 + 1, r_1 + \Delta_1 - 1]$. To find a contradiction we prove that $uv$ and $u'v'$ belong to a $\Delta_1$-temporal clique whose lifetime contains $[r_1, r_2]$.

We will begin by showing that $I_{uv} \cup I_{uc} \cup I_{vc}$ and $I_{u'v'} \cup I_{u'c} \cup I_{v'c}$ are $\Delta_2$-indivisible. Recall that $u, v, c$ belong to a $\Delta_1$-temporal clique whose lifetime contains the interval $[r_1, r_2]$, and that each edge in this clique must appear at least once in the time interval $I$. Recall also that $cx$ appears at some time $r_i \in [r_1, r_2]$ and thus $cx$ is $\Delta_1$-dense in $I$. Thus, no appearance of $uc$ and $vc$ in $I$ can be $\Delta_2$-independent of $I_{cx}$ and so $I_{uc} \cup I_{vc} \cup I_{cx}$ is $\Delta_2$-indivisible. Hence, the set $I_{uc} \cup I_{vc} \cup I_{cx}$ is contained in a single $\Delta_1$-temporal clique $\mathcal{C}$ of the $(\Delta_1, \Delta_2)$-cluster temporal graph $\mathcal{G}[\{u, v, c, x\}]$ by Lemma 2.7 together with the five vertex condition. Any appearance of $uv$ in $I$ must also be contained in $\mathcal{C}$, since it cannot be $\Delta_2$-independent of $\mathcal{C}$, and $\mathcal{G}[\{u, v, c, x\}]$ is a $(\Delta_1, \Delta_2)$-cluster temporal graph. It follows that $I_{uv} \cup I_{uc} \cup I_{vc}$ is $\Delta_2$-indivisible since all these edges are contained within the same clique $\mathcal{C}$ in $\mathcal{G}$. A symmetric argument shows that $I_{u'v'} \cup I_{u'c} \cup I_{v'c}$ is $\Delta_2$-indivisible.

We now show and $I_{cu} \cup I_{cu'}$ is $\Delta_2$-indivisible. As before, $u, u', c$ belong to a $\Delta_1$-temporal clique whose lifetime contains the interval $[r_1, r_2]$, and each edge in this clique appears at least

once in the time interval $I$. As $cx$ appears at some time $r_i \in [r_1, r_2]$, no appearance of $uc$ and $u'c$ in $I$ can be $\Delta_2$-independent of $I_{cx}$, thus $I_{uc} \cup I_{u'c} \cup I_{cx}$ is $\Delta_2$-indivisible. Hence $I_{uc} \cup I_{u'c} \cup I_{cx}$ is contained within a single $\Delta_1$-temporal clique in $\mathcal{G}[\{c, u, u'\}]$ by Lemma 2.7 and the five vertex condition. It follows that $I_{cu} \cup I_{cu'}$ is $\Delta_2$-indivisible.

We have shown that $I_{uv} \cup I_{uc} \cup I_{vc}$, $I_{u'v'} \cup I_{u'c} \cup I_{v'c}$ and $I_{cu} \cup I_{cu'}$ are $\Delta_2$-indivisible, so two applications of Lemma 2.3 give us that $M = I_{uv} \cup I_{uc} \cup I_{vc} \cup I_{u'v'} \cup I_{u'c} \cup I_{v'c}$ is $\Delta_2$-indivisible. Then it follows from Lemma 2.7 that all edges of $M$ are contained in the same $\Delta_1$-temporal clique in $\mathcal{G}[\{u, v, u', v', c\}]$. Since this clique contains the non-empty sets $I_{uv}$ and $I_{u'v'}$ it follows that there exists an interval contained in $I$ in which both $uv$ and $u'v'$ are $\Delta_1$-dense. Since both endpoints of $I$ are at distance at most $\Delta_1$ from $r_1$ and $r_2$ there must be an interval containing $[r_1, r_2]$ in which both $uv$ and $u'v'$ are $\Delta_1$-dense, a contradiction.

We can therefore conclude that there is some interval

$$[h_1, h_2] \supseteq [r_1, r_2] \supseteq [\min\{s_i, s_j, s'\} + \Delta_1 - 1, \max\{t_i, t_j, t'\} - \Delta_1 + 1],$$

such that every edge with both endpoints in $V(S_i) \cup V(S_j)$ is $\Delta_1$-dense in $[h_1, h_2]$, as required. $\Diamond$

**Claim 3.9.** *There is a $\Delta_1$-temporal clique in $\mathcal{G}$ generating*

$$(V(S_i) \cup V(S_j), [\min\{s_i, s_j, h_1\}, \max\{t_i, t_j, h_2\}]).$$

*Proof of claim.* [3.9] Let $u, v \in V(S_i) \cup V(S_j)$; it suffices to demonstrate that $uv$ is $\Delta_1$-dense in the interval $[\min\{s_i, s_j\}, \max\{t_i, t_j\}]$, as each edge $uv$ is $\Delta_1$-dense in $[h_1, h_2]$ by Claim 3.8. Indeed, by Claim 3.8, if $h_1 \leq \min\{s_i, s_j\}$ and $h_2 \geq \max\{t_i, t_j\}$ then the result holds. We thus assume that at least one of these conditions does not hold and begin by considering the lower end of the interval $[\min\{s_i, s_j, h_1\}, \max\{t_i, t_j, h_2\}]$.

Assume that $h_1 > \min\{s_i, s_j\}$, then by the definition of $s_i, s_j$ there exists at least one edge $ab$ with $a, b \in V(S_i) \cup V(S_j)$ that appears at time $\min\{s_i, s_j\}$. We can assume that without loss of generality that $s_i \leq s_j$, thus $s_i = \min\{s_i, s_j\}$, and also that $(ab, s_i) \in L(S_i)$.

For the first step we recall that the edge $cx$ is $\Delta_1$-dense in both $L(S_i)$ and $[h_1, h_2]$ and, by Claim 3.8, $(cx, r_i) \in S_i$ and $r_i \in [h_1, h_2]$. Thus $cx$ is $\Delta_1$-dense in $L(S_i) \cup [h_1, h_2]$.

For the second step, recall that $ab$ is $\Delta_1$-dense in $L(S_i)$ and $[h_1, h_2]$, and appears at time $s_i$. Observe that $ax, bx, ac$ and $bc$ must all have time appearances in $[h_1, h_2]$ by Claim 3.8. Thus the set of time edges in $L(S_i) \cup [h_1, h_2]$ with endpoints in $\{a, b, c, x\}$ forms a $\Delta_2$-indivisible set and hence, by Lemma 2.7 and the five vertex condition, all these edges must be part of the same $\Delta_1$-temporal clique in $\mathcal{G}[\{a, b, c, x\}]$. It follows that $ab$ is $\Delta_1$-dense in $L(S_i) \cup [h_1, h_2] = [s_i, h_2]$.

For the third step we consider the set $A = \{a, b, u, v\}$. By Claim 3.8 the set of time-edges with both endpoints in $A$ which occur in $[h_1, h_2]$ forms a $\Delta_1$-temporal clique, and thus is $\Delta_2$-indivisible. Since $ab$ is $\Delta_1$-dense in $[s_i, h_2]$, it follows that the set of time-edges appearing in

$[s_i, h_2]$ with both endpoints in $A$ is also $\Delta_2$-indivisible. By the five vertex condition and Lemma 2.7, all these edges must belong to the same clique in $\mathcal{G}[A]$. Thus, since $(ab, s_i) \in S_i$ and $uw$ appears in $[h_1, h_2]$, it follows that $uw$ is $\Delta_1$-dense in $[s_i, h_2]$.

Similarly, assuming $h_2 < \max\{t_i, t_j\}$, there exists at least one edge $a'b'$ with $a', b' \in V(S_i) \cup V(S_j)$ that appears at time $\max\{t_i, t_j\}$. We now follow the same steps as before. For the first step, if $t_i \geq t_j$, the we take $cx$ as this is $\Delta_1$-dense in $L(S_i) \cup [h_1, h_2]$ and appears in $L(S_i)$. Otherwise, if $t_i < t_j$, we take $cz$ which is $\Delta_1$-dense in $L(S_j) \cup [h_1, h_2]$ and appears in $L(S_j)$. Once we have completed the first step to find some edge $fg$ that is dense in a suitable interval, we complete the second step by using $\mathcal{G}[\{a', b', f, g\}]$ to show that $a'b'$ is $\Delta_1$-dense in $[h_1, \max\{t_i, t_j\}]$. We can then apply the third step to the set $A' = \{a', b', u, v\}$ to show that $uv$ is $\Delta_1$-dense in $[h_1, \max\{t_i, t_j\}]$.

Finally, since $uv$ appears in $[h_1, h_2]$ it follows that $uv$ is $\Delta_1$-dense in

$$[\min\{s_i, s_j, h_1\}, \max\{t_i, t_j, h_2\}].$$

$\Diamond$

Observe that Claim 3.9 contradicts the initial assumption that $S_i$ and $S_j$ were maximal in $S$. Hence the assumption that $m \geq 2$ must be incorrect and thus $S$ consists of a single $\Delta_1$-temporal clique. Because $S$ was a generic set of the partition $\mathcal{P}_\mathcal{G}$ of the given temporal graph $\mathcal{G}$ into $\Delta_2$-saturated subsets, then $\mathcal{G}$ must be a $(\Delta_1, \Delta_2)$-cluster temporal graph by Lemma 2.8, so the statement of the lemma holds.

$\square$

This concludes the proof of the general statement of Theorem 3.1. It remains to prove that, in the special case where $\Delta_1 = 1$, three vertices are enough to detect whether $\mathcal{G}$ is a $(1, \Delta_2)$-cluster temporal graph. The proof will follow the same high level idea as the proof of Lemma 3.5, however it will be significantly simpler.

**Lemma 3.10.** *Let $\mathcal{G}$ be a temporal graph and suppose that, for every set $S$ of at most three vertices in $\mathcal{G}$, the induced temporal subgraph $\mathcal{G}[S]$ is a $(1, \Delta_2)$-cluster temporal graph. Then $\mathcal{G}$ is a $(1, \Delta_2)$-cluster temporal graph.*

*Proof.* We shall follow the same structure as the proof of Lemma 3.5. In particular, we shall assume that within the unique decomposition of $\mathcal{E}(\mathcal{G})$ into saturated subsets, there exists a saturated set $S$ that is not a $\Delta_1$-temporal clique. We then have a decomposition of $S$ into maximal $\Delta_1$-temporal cliques $\{S_1, \ldots, S_m\}$. It suffices to demonstrate that, if there exist two 1-temporal cliques $S_i$ and $S_j$ in $\mathcal{G}$ that are not $\Delta_2$-independent, then there must be some clique $S'$ with $S_i, S_j \subseteq S'$, since this contradicts the maximality of $S_i$ and $S_j$. As in the proof of

Lemma 3.5, we suppose that $L(S_i) = [s_i, t_i]$ and $L(S_j) = [s_j, t_j]$, and note that, by definition of 1-density, every edge in $S_i$ (respectively $S_j$) appears at every time in $[s_i, t_i]$ (respectively $[s_j, t_j]$).

Since $S_i$ and $S_j$ are not $\Delta_2$-independent, they must share a common vertex $c$. We first argue that every edge of the form $cz$, where $z \in (V(S_i) \cup V(S_j)) \backslash \{c\}$, must be 1-dense in $L := [\min\{s_i, s_j\}, \max\{s_j, t_j\}]$. For any edge $e$ let $A_e$ be the sets of appearances of $e$ in $L$. We now claim that for any vertices $x \in V(S_i)$ and $y \in V(S_j)$, the set $A_{cx} \cup A_{cy}$ is $\Delta_2$-indivisible. This follows since $cx$ and $cy$ appear at all times in $[s_i, t_i]$ and $[s_j, t_j]$ respectively, and there exists $t \in [s_i, t_i]$ and $t' \in [s_j, t_j]$ such that $|t - t'| < \Delta_2$. By hypothesis $\mathcal{G}[\{c, x, y\}]$ is a $(1, \Delta_2)$-cluster temporal graph and thus, by Lemma 2.7, the set $A_{cx} \cup A_{cy}$ of time-edges must contained within a single 1-temporal clique, with lifetime which contains the interval

$$[\min\{t : (cx, t) \in A_{cx} \text{ or } (cy, t) \in A_{cy}\}, \max\{t : (cx, t) \in A_{cx} \text{ or } (cy, t) \in A_{cy}\}]$$

$$= [\min\{s_i, s_j\}, \max\{s_j, t_j\}] = L.$$

Since $x \in V(S_i)$ and $y \in V(S_j)$ were arbitrary, the edge $cz$ is 1-dense in $L$ for any $z \in V(S_i) \cup V(S_j)$.

Now fix two arbitrary vertices $u, v \in V(S_i) \cup V(S_j) \setminus \{c\}$; we wish to demonstrate that $uv$ is also 1-dense in $L$, which will complete the proof. Recall that we have just shown that $uc$ and $vc$ are 1-dense in $L$. It follows immediately that $A_{uv} \cup A_{vc}$ is $\Delta_2$-indivisible. Thus, by the three-vertex condition, there must be a single 1-temporal clique containing $u$, $v$ and $c$, whose lifetime contains the non-empty interval $L$.

It follows that every edge $uv$, where $u, v \in V(S_i) \cup V(S_j)$ is 1-dense in an interval containing $L$ and, since $uv$ therefore appears at every time during this interval. Thus there exists a $\Delta_1$-temporal clique $S'$ in $\mathcal{G}$ generating the template $(V(S_i) \cup V(S_j), L)$. This contradicts the assumption that $S_i$ and $S_j$ were maximal. □

## 3.1 A Search-Tree Algorithm

From the observation that a certain graph is a cluster, in the static setting, if and only if it does not contains any induced $P_3$s, it is possible to deduce the fixed-parameter tractability of Cluster Editing via a search tree algorithm. Many algorithms of this kind have been studied so far. In particular, in [13], it is shown an algorithm for the problem running in time $\mathcal{O}(1.76^k + m + n)$ for graphs with $n$ vertices and $m$ edges. Using a similar intuition, from the characterisation for cluster temporal graphs shown in Theorem 3.1, we prove through a search tree algorithm that ETC is FPT if parameterised simultaneously by the number of modifications and the lifetime $T$ of the input temporal graph.

**Theorem 3.11.** ETC *can be solved in time* $(10T)^k \cdot T^3|V|^5$.

*Proof.* In the following we will show through a search tree algorithm that ETC is FPT if parameterised by the maximum number $k$ of allowed time-edges modifications and the lifetime $T$ of the input graph. First of all, let us recall that, by Theorem 3.1, we know that a temporal graph is a $(\Delta_1, \Delta_2)$-cluster temporal graph if and only if every subset of at most five vertices induces itself a $(\Delta_1, \Delta_2)$-cluster temporal graph. By Lemma 3.3 we can deduce that it is enough to verify that this property holds for every subset of exactly five vertices.

Let us assume now that $(\mathcal{G}, \Delta_1, \Delta_2, k)$ is an input instance of ETC.

**Claim 3.12.** *If $\mathcal{G}$ is a temporal graph, then it is possible to verify in time $\mathcal{O}(T^3|V|^5)$ either if $\mathcal{G}$ is already a $(\Delta_1, \Delta_2)$-cluster temporal graph, or identify a set $W$ of five vertices such that $\mathcal{G}[W]$ is not a $(\Delta_1, \Delta_2)$-cluster temporal graph.*

*Proof of claim.* By Lemma 2.9, it is possible to verify in time $\mathcal{O}(|\mathcal{E}|^3|V|)$ whether a temporal graph $\mathcal{G}$ is a $(\Delta_1, \Delta_2)$-cluster temporal graph. For any subset $W \subseteq V(G)$ of five vertices, the cardinality of the vertex set of the induced subgraph $\mathcal{G}[W]$ is clearly 5 by construction. The maximum number of time-edges in $\mathcal{G}$ with both end-points in $W$, instead, is bounded by $10T$, since there are at most 10 possibilities for the choice of static edges and at most $T$ possibilities for the time at which that edge can occur (Lemma 2.11), since $T$ is the lifetime of $\mathcal{G}$. So, since we are working with an asymptotic notation which, by definition, neglects constants in the account of the running time, it is possible to verify in time $\mathcal{O}((10T)^3 5) = \mathcal{O}(T^3)$ if $\mathcal{G}[W]$ is or not a $(\Delta_1, \Delta_2)$-cluster temporal graph. Furthermore, the number of possible vertex subsets of five vertices of $V$ is bounded by $\mathcal{O}(|V|^5)$. From these facts and from Theorem 3.1, Claim 3.12 follows. ◊

So, our search tree algorithm works as follows. If $\mathcal{G}$ is already a cluster temporal graph with respect to the parameters $\Delta_1$ and $\Delta_2$, then the algorithm halts and returns that $(\mathcal{G}, \Delta_1, \Delta_2, k)$ is a *yes*-instance. Otherwise, it identifies a set $W \subseteq V$ with $|W| = 5$ which does not induce a $(\Delta_1, \Delta_2)$-cluster temporal graph, which must exist by Theorem 3.1. By Claim 3.12 this process requires $\mathcal{O}(T^3|V|^5)$ time. Thus, in order to modify $\mathcal{G}$ to be a $(\Delta_1, \Delta_2)$-cluster temporal graph, at least one time-edge with both endpoints in $W$ must be added or deleted. As mentioned before, we have at most $10T$ possibilities for choosing that edge. This would branch our problem in at most $10T$ sub-problems, for each of which one of these time-edges is modified. This bounds the degree of our search tree to $10T$. For each of these possibilities, we repeat the same reasoning as before. So, we check if after the current modification has been applied we have obtained or not a cluster temporal graph. If that is the case, then the algorithm stops returning that $(\mathcal{G}, \Delta_1, \Delta_2, k)$ was a *yes*-instance. Otherwise, we select another subset $W'$ of five vertices that does not induces

a $(\Delta_1, \Delta_2)$-cluster temporal graph in the graph modified up to the current point and we branch again over the $10T$ possible choices for modifying a time-edge with both endpoints in $W'$. In this setting, at each branching step of this algorithm the value of $k$ decreases by 1, because at each step one time-edge is modified. This bounds the depth of the search tree to $k$.

If at least for one of these branches, through these modifications we obtain a $(\Delta_1, \Delta_2)$-cluster temporal graph, then the algorithm returns that $(\mathcal{G}, \Delta_1, \Delta_2, k)$ was a *yes*-instance. Otherwise, the algorithm ends when every branch reaches $k = 0$, in which case, it returns that the input instance was a "*no*"-instance. Since, as we mentioned, the degree and the depth of the search tree are respectively bounded by $10T$ and $k$, then the number of its nodes is bounded by $(10T)^k$. To sum up, because by Claim 3.12 at each step we need to perform a check that requires time $\mathcal{O}(T^3|V|^5)$, then this search tree algorithm runs in time $(10T)^k \cdot T^3|V|^5$. □

## 3.2 Conclusions and Open Problems

In this Chapter we presented the main technical contribution of this thesis which is a Characterisation of $(\Delta_1, \Delta_2)$-cluster temporal graphs given in terms of five vertex subsets; which is formulated in Theorem 3.1. As we reported also in [7], from this theorem follows a simple search tree algorithm, which is an FPT algorithm parameterised simultaneously by the number $k$ of permitted modifications and the lifetime of the input temporal graph. An interesting direction for further research would be to investigate whether this result can be strengthened. For example, two natural questions that immediately rise are on the one hand if there exists a polynomial kernel with respect to this dual parameterisation and on the other, if ETC is in FPT also when it is only parameterised by the number of permitted modifications.

# Chapter 4

# Realisation of distance matrices by unicyclic graphs

During my Ph.D I also worked on the problem of realisation of distance matrices by unicyclic graphs. This kind of problem results to be quite interesting especially in Phylogenetics. In this field, graphs are often used to represent genetic similarities or the direct or indirect progeny between species.

In particular, let $D$ be a matrix whose rows and columns are indexed by a set $X$. We assume that $D$ is symmetric and has zero entries on the main diagonal. In Phylogenetics, this kind of matrices are called *dissimilarity matrices*. Hence, a dissimilarity matrix $D$ can also be seen as a map $D : [n]^2 \to \mathbb{R}$, with $D(i,j) = D(j,i)$ and $D(i,i) = 0$ for each $i, j \in [n]$. A non-negative dissimilarity matrix which satisfies the triangle inequality $D(i,j) \leq D(i,k) + D(k,j)$ for all $i, j, k \in X$ is called a *distance matrix* (sometimes also called *metric*).

We say that $D$ has a *realisation* if there exists a weighted graph (so one where a non-negative weight is assigned to each edge) whose vertex set contains the set $\{v_i : i \in X\}$ and such that the distance (i.e. the length of the shortest path) between vertices $v_i, v_j \in X$ is exactly $D(i,j)$. Determining the distance matrix which realises a given weighted graph is a straightforward process. However, the converse problem of realising a distance matrix by a graph of minimal total weight is much more complicated in general. One exception of this is the special case where the graph is a tree, in which case the distance matrix is called a *tree metric* or also *additive metric*. This particular case, has been studied intensively and is now well understood. More specifically, the main result about realisation of distance matrices by trees is the so-called Tree Metric Theorem, (see [11]), which is based on the so called *four-point condition*, which consists in a necessary and sufficient condition for a matrix to be realised by a tree.

Efficient algorithms constructing such trees were published in [15] or [45]. However, real data can merely describe a tree metric because they arise from a similarity measure that includes

errors. Such a measure appears in various fields such as the study of evolution ( [38]), the synthesis of certain electrical circuits ( [23]) or the traffic modelling ( [12]). Thus, many authors address to the problem of realisation by a graph which is not a tree. For example in [2] can be found results of realisation for some particular classes of graphs: paths, caterpillar, cycles, bipartite graphs, complete graphs and planar graphs. The particular class of cactus metric is studied deeply in [26].

As mentioned before, in this chapter we consider the problem of realising a distance matrix non only by trees but particularly by unicyclic graphs, namely graphs containing only one cycle. This kind of graphs represents a first non-trivial case in the study of phylogenetic networks. See, for example, Figures 2 and 5 in [28] and Figures 1 and 4 in [46]. More precisely, we will present an algorithm that in a first part, taken as input a distance matrix $D$, returns if or not this matrix can be realised by a unicyclic graph (or by a tree) and, in a second part, in case of positive answer it reconstruct this graph.

Before entering in the details of this topic, we need to specify that in the following, with a slight abuse of notation, for a given weighted graph $G$, we will indicate with $(uv, w)$ one of its weighted edges. In this notation, in particular, $uv$ is an edge of the graph and $w = W(uv)$ is the weight assigned to that edge by the weight function $W : E(G) \to \mathbb{R}^+$.

## 4.1 Preliminaries

As a first step, we need to mention a few fundamental concepts that will turn out to be crucial in the rest of this Chapter. More specifically, we recall from [22] and [47], the definitions of compaction and reduction processes. Let $D$ be a distance matrix of order $n \times n$ and consider the $n \times n-$matrices $E_i$ where

$$(E_i)(j,k) = \begin{cases} 1 & \text{if } j = i \neq k \\ 1 & \text{if } j \neq i = k \\ 0 & \text{elsewhere} \end{cases}$$

Let $D_i(\alpha) = D - \alpha \cdot E_i$. The matrix $D_i(\alpha)$, obtained from $D$, is called the *$i-$th compaction matrix of $D$ with respect to $\alpha$*. It is worth noting the fact that this matrix is not necessarily anymore a distance matrix. However, we are interested in compaction matrices that are still distance matrices. The following result establishes for which values of $\alpha$, $D_i(\alpha)$ is still a distance matrix. The complete proof of this theorem can be found in [22] (Lemma 1).

**Theorem 4.1.** *$D_i(\alpha)$ is a distance matrix if and only if $\alpha \leq \frac{1}{2} \cdot (D(p,i) + D(i,r) - D(p,r))$, for all $p, r \neq i$.*

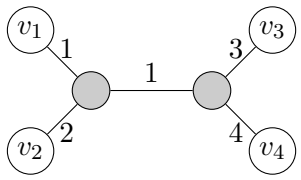



Figure 4.1: The tree realising the distance matrix 4.1.

This compaction process applied to the $i$-th index of $D$, leads to a new matrix with possible equal rows and, by symmetry, equal columns (see [47]). By deleting all, but one the repeated rows and columns, we obtain a new matrix which is called *$i-$th reduction matrix of $D$ with respect to $\alpha$.*

In this work, we are interested in considering the maximal value $\alpha$ for which $D_i(\alpha)$ is still a distance matrix. Moreover, we want to work for each index $i$ of compaction, with $i = 1, \ldots, n$, and collect these data into a vector. This leads to the following definition.

**Definition 4.2.** *Given a distance matrix $D$ of order $n \times n$, the compaction vector of $D$, $\mathbf{a}_D = (a_1, \ldots, a_n)$, is defined as*

$$a_i = \frac{1}{2} \cdot \min_{p \neq i, r \neq i} \{D(p,i) + D(i,r) - D(p,r)\}.$$

To give an example, let us consider the tree in Figure 4.1. Remembering that to every vertex $v_i$ of a given graph corresponds the $i-$th row and column of its distance matrix, the distance matrix of this graph is

$$D = \begin{pmatrix} 0 & 3 & 5 & 6 \\ 3 & 0 & 6 & 7 \\ 5 & 6 & 0 & 7 \\ 6 & 7 & 7 & 0 \end{pmatrix}. \tag{4.1}$$

and the compaction vector of $D$ is $\mathbf{a}_D = (1, 2, 3, 4)$.

It is a well known fact [47] that, in case $D$ is obtained from a weighted graph and if $v_i$ is a leaf, then there is a realisation of $D$ in which the $i$-th entry $a_i$ of the compaction vector is precisely the weight of the edge connecting the leaf $v_i$ to its internal vertex. As a matter of fact, this can not to be actually the only realisation of $D$. Let us consider for instance the graph in Figure 4.2(a). Its distance matrix is

$$D = \begin{pmatrix} 0 & 3 & 5 & 4 \\ 3 & 0 & 5 & 5 \\ 5 & 5 & 0 & 5 \\ 4 & 5 & 5 & 0 \end{pmatrix} \tag{4.2}$$

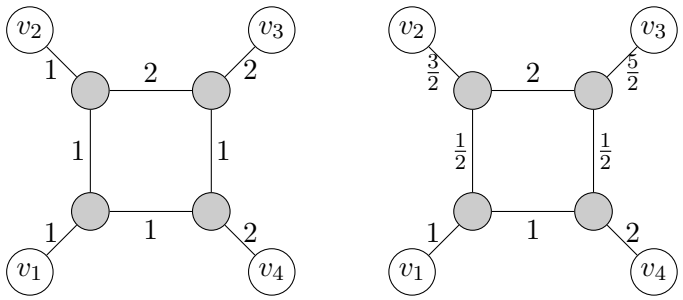



Figure 4.2: Two graphs realising matrix in (4.2).

and its compaction vector is $\mathbf{a}_D = (1, \frac{3}{2}, \frac{5}{2}, 2)$. In this case, the entries of $\mathbf{a}_D$ do not correspond to the weights of the edges incident to the leaves. However, we can notice that the graph in Figure 4.2(b) has the same distance matrix, and hence the same compaction vector, of the graph in Figure 4.2(a). Both these graphs are realisations of the matrix in (4.2). For the one in Figure 4.2(b), by the way, the weights of the edges connecting the leaves to the internal vertices actually correspond to the entries of the compaction vector.

Once the compaction vector $\mathbf{a}_D$ of $D$ is computed, we can consider the following definition.

**Definition 4.3.** *The matrix $D(\mathbf{a}_D) = D - a_1 \cdot E_1 - \cdots - a_n \cdot E_n$ is the compaction matrix of the distance matrix $D$.*

**Definition 4.4.** *Let $D(\mathbf{a}_D)$ be the compaction matrix of $D$. The matrix $\hat{D}_{\mathbf{a}_D}$ obtained removing from $D(\mathbf{a}_D)$ all but one repeated rows and columns is called reduction matrix of $D$. If there are not equal rows (and columns) in $D(\mathbf{a}_D)$, then $\hat{D}_{\mathbf{a}_D} = D(\mathbf{a}_D)$.*

Notice that now we omit the expression "with respect to $\mathbf{a}_D$" and we do not refer to a specific index $i$, since we are considering the compaction process along all the indices and only with respect to the corresponding values of the compaction vector; while in the standard definition of compaction matrix the computation is taken only with respect to an index and a given value $\alpha$. Thus the process of reduction is different with respect to the one in [47], where only one of the equal rows (and columns) is removed. Let us consider the following matrix $D$ as an example.

$$D = \begin{pmatrix} 0 & 4 & 6 & 6 & 3 \\ 4 & 0 & 5 & 5 & 5 \\ 6 & 5 & 0 & 2 & 5 \\ 6 & 5 & 2 & 0 & 5 \\ 3 & 5 & 5 & 5 & 0 \end{pmatrix}$$

By Definition 4.2, it is possible to calculate its compaction vector

$$\mathbf{a}_D = \left(1, \frac{3}{2}, 1, 1, 1\right)$$

Taking as reference Definition 4.3, from $\mathbf{a}_D$ we get the compaction matrix

$$D(\mathbf{a}_D) = D - E_1 - \frac{3}{2}E_2 - E_3 - E_4 - E_5 = \begin{pmatrix} 0 & \frac{3}{2} & 4 & 4 & 1 \\ \frac{3}{2} & 0 & \frac{5}{2} & \frac{5}{2} & \frac{5}{2} \\ 4 & \frac{5}{2} & 0 & 0 & 3 \\ 4 & \frac{5}{2} & 0 & 0 & 3 \\ 1 & \frac{5}{2} & 3 & 3 & 0 \end{pmatrix}.$$

Notice that the third and fourth rows of $D(\mathbf{a}_D)$ are equal, so the reduction matrix is

$$\hat{D}_{\mathbf{a}_D} = \begin{pmatrix} 0 & \frac{3}{2} & 4 & 1 \\ \frac{3}{2} & 0 & \frac{5}{2} & \frac{5}{2} \\ 4 & \frac{5}{2} & 0 & 3 \\ 1 & \frac{5}{2} & 3 & 0 \end{pmatrix}.$$

We now prove two results concerning particular behavior of compaction and reduction matrices.

**Proposition 4.5.** *Let $D$ be a distance matrix of order $n \times n$, if its reduction matrix $\hat{D}_{\mathbf{a}_D}$ has order 2, then $D$ is realisable by a tree.*

*Proof.* If $\hat{D}_{\mathbf{a}_D}$ has order 2, then

$$\hat{D}_{\mathbf{a}_D} = \begin{pmatrix} 0 & \delta \\ \delta & 0 \end{pmatrix} \tag{4.3}$$

for some $\delta \in \mathbb{R}_{>0}$.

Moreover, for the reduction process, there exist positive integers $n_1$ and $n_2$, with $n_1 + n_2 = n$, such that $D(\mathbf{a}_D)$ has a set of $n_1$ equal rows (and correspondent columns) and a set of $n_2$ equal rows (and correspondent columns). For simplicity we assume that the first $n_1$ rows are equal, and hence the remaining $n_2$ rows are equal.

Then by (4.3) we get that $D(\mathbf{a}_D)$ has the form

$$\left(\begin{array}{c|c} \mathbf{0}_{n_1 \times n_1} & \delta \mathbf{1}_{n_1 \times n_2} \\ \hline \delta \mathbf{1}_{n_2 \times n_1} & \mathbf{0}_{n_2 \times n_2} \end{array}\right) \tag{4.4}$$

where $\mathbf{0}_{k \times l}$ denotes the null matrix of order $k \times l$ and $\delta \mathbf{1}_{k \times l}$ denote the matrix of order $k \times l$ whose entries are all equals to $\delta$.

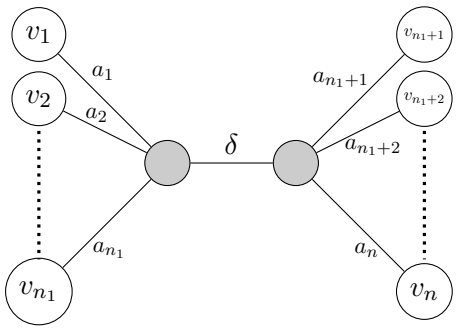



Figure 4.3: Realisation tree for Proposition 4.5.

Let $\mathbf{a}_D = (a_1, \ldots, a_n)$ be the compaction vector of $D$, by definition of compaction matrix we know that

$$D(\mathbf{a}_D) = \begin{pmatrix} 0 & D(1,2) - a_1 - a_2 & \cdots & D(1,n) - a_1 - a_n \\ D(2,1) - a_1 - a_2 & 0 & \cdots & D(2,n) - a_2 - a_n \\ \vdots & \vdots & \ddots & \vdots \\ D(n,1) - a_1 - a_n & D(n,2) - a_2 - a_n & \cdots & 0 \end{pmatrix} \tag{4.5}$$

Comparing the entries of the matrices in (4.4) and (4.5) one has

$$D(i,j) = \begin{cases} 0 & \text{if } i = j \\ a_i + a_j & \text{if } 1 \leq i < j \leq n_1 \text{ or } n_1 + 1 \leq i < j \leq n_1 + n_2 \\ \delta + a_i + a_j & \text{otherwise} \end{cases}$$

from which follows that $D$ has a realisation by the tree in Figure 4.3. □

Now, we give a second result which concerns the compaction matrix. Both these results will be fundamental concepts for our algorithm of realisation, since they will determine its stopping conditions.

**Proposition 4.6.** *Let $D$ be a distance matrix of order $n \times n$, if its compaction matrix $D(\mathbf{a}_D)$ is the null matrix, then $D$ is realisable by a star tree with $n$ leaves (and a unique internal node of degree $n$).*

*Proof.* Let $\mathbf{a}_D = (a_1, \ldots, a_n)$ be the compaction vector of $D$. By definition of compaction matrix we know that

$$D(\mathbf{a}_D) = \begin{pmatrix} 0 & D(1,2) - a_1 - a_2 & \cdots & D(1,n) - a_1 - a_n \\ D(2,1) - a_1 - a_2 & 0 & \cdots & D(2,n) - a_2 - a_n \\ \vdots & \vdots & \ddots & \vdots \\ D(n,1) - a_1 - a_n & D(n,2) - a_2 - a_n & \cdots & 0 \end{pmatrix}$$

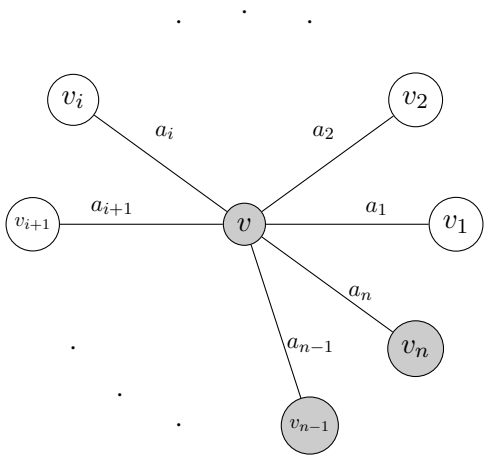



Figure 4.4: Realisation for Proposition 4.6.

Since by hypothesis, $D(\mathbf{a}_D)$ is the null matrix, one has

$$D(i,j) = \begin{cases} a_i + a_j & \text{if } i \neq j \\ 0 & \text{otherwise} \end{cases}$$

from which easily follows that $D$ has a realisation by the star tree in Figure 4.4. □

### 4.1.1 Clarifying examples

Before entering in the details of our results, we want to give a few examples about the preliminary concepts of compaction and reduction that will turn out to be crucial in the explanation of our algorithm that will be discussed later.

In the first example, in reference to Proposition 4.5, we will study the compaction process on a distance matrix $D$, whose reduction matrix has order 2.

Let $D$ be the distance matrix

$$D = \begin{pmatrix} 0 & 2 & 3 & 3 \\ 2 & 0 & 3 & 3 \\ 3 & 3 & 0 & 2 \\ 3 & 3 & 2 & 0 \end{pmatrix} \tag{4.6}$$

From its compaction vector

$$\mathbf{a}_D = (1, 1, 1, 1)$$

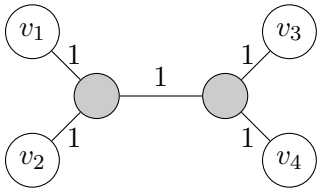



Figure 4.5: Tree which realises the matrix in (4.6).

we get

$$D(\mathbf{a}_D) = D - E_1 - E_2 - 3E_3 - E_4 = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}.$$

In this case $D(\mathbf{a}_D)$ has two pairs of equal rows and, after the reduction process, we get

$$\hat{D}_{\mathbf{a}_D} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Since $\hat{D}_{\mathbf{a}_D}$ has order 2, by Proposition 4.5, $D$ has a realisation by the tree in Figure 4.5.

Now, we want to show an example that refers to Proposition 4.6. So, in this case, we will consider a distance matrix whose compaction matrix is the null matrix and which consequently is realisable by a star.

Let $D$ be the distance matrix

$$D = \begin{pmatrix} 0 & 3 & 4 & 5 \\ 3 & 0 & 5 & 6 \\ 4 & 5 & 0 & 7 \\ 5 & 6 & 6 & 0 \end{pmatrix} \tag{4.7}$$

From its compaction vector

$$\mathbf{a}_D = (1, 2, 3, 4)$$

we get

$$D(\mathbf{a}_D) = D - E_1 - 2E_2 - 3E_3 - 4E_4 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Then, by Proposition 4.6, $D$ has a realisation by the tree in Figure 4.6.

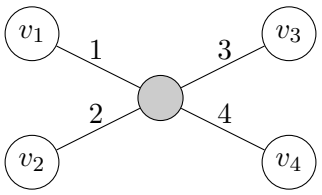



Figure 4.6: Star tree which realises the matrix in (4.7).

## 4.2 Realisations by cycles

In this section we give a useful result for matrices which are realisable by cycles. While in other research works the order of the vertices is given a priori, or the vertices are eventually re-labelled after some algorithm (see, for example, Definition 4.2 in [2]), the following theorem gives a more general characterization of such matrices also if the order of vertices is not known.

Let $\pi = (i_1, i_2, \ldots, i_n)$ be a permutation on $[n]$. We denote by $\pi^s(i)$ the image of $i$ after applying $s$ time the permutation $\pi$.

We recall that a permutation $\pi$ of $[n]$ is *real* if it has only one term in the cycle-notation. For example $\pi_1 = (2, 3, 4, 5, 1)$ is real, while $\pi_2 = (2, 1, 4, 5, 3)$ is not since it can be written as $(2, 1)(4, 5, 3)$.

**Theorem 4.7.** *A distance matrix $D$, of order $n$, has a realisation by an $n-$cycle $C$ if and only if there exists a real permutation $\pi$ of $[n]$, such that*

$$D(i, \pi^s(i)) = \min\left\{\sum_{t=0}^{s-1} D(\pi^t(i), \pi^{t+1}(i)), \sum_{t=s}^{n-1} D(\pi^t(i), \pi^{t+1}(i))\right\} \tag{4.8}$$

*for all $i = 1, ..., n$ and $s = 1, ..., n-1$.*

*Proof.* If $D$ has a realisation by an $n-$cycle $C$, let $i_1, i_2, \ldots, i_n$, with $i_1 = 1$, be the labels of the vertices taken in clockwise order. Then $\pi = (i_2, \ldots, i_n, 1)$ is a real permutation and since, on an $n-$cycle, the distance between vertices $v_i$ and $v_{\pi^s(i)}$ is the minimum of the two paths from $v_i$ to $v_{\pi^s(i)}$, equations (4.8) are satisfied, for all $i = 1, ..., n$ and $s = 1, ..., n-1$.

Vice versa, suppose there exists a real permutation $\pi$ for which $D$ satisfies equation (4.8), for all $i = 1, ..., n$ and $s = 1, ..., n-1$. Then let $C$ be the graph of vertices $V = \{v_1, v_2, \ldots, v_n\}$ and edges

$$E = \{(v_{\pi^i(1)} v_{\pi^{i+1}(1)}),\ i = 0, \ldots, n-1\}.$$

It is easy to check that $C$ is an $n-$cycle. Then, putting the weight

$$D(\pi^i(1), \pi^{i+1}(1))$$

on the edge $(v_{\pi^i(1)} v_{\pi^{i+1}(1)})$, for all $i = 0, \ldots, n-1$, we get a weighted $n-$cycle that, by equation (4.8), is a realisation of $D$. □

According to the fact that, on an $n-$cycle, the distance between vertices $v_i$ and $v_j$ is the minimum of the two paths from $v_i$ to $v_j$, the reader could be surprised of formula (4.8) instead of the following formula

$$\begin{aligned} D(i,j) = \min\{&D(i,i+1) + D(i+1,i+2) + \cdots + D(j-1,j), \\ &D(i,i-1) + D(i-1,i-2) + \cdots + D(j+1,j)\} \end{aligned} \tag{4.9}$$

where the indices are taken modulo $n$. Notice that (4.9) expresses the condition of a path for an $n-$cycle where the indices are labeled in clockwise order.

However in general, it is not true that if a distance matrix $D$, of order $n \times n$, has a realisation by an $n-$cycle $C$, then the adjacent vertices of $C$ correspond to adjacent rows (or columns) of $D$.

For example, if we consider the distance matrix

$$D = \begin{pmatrix} 0 & 3 & 4 & 1 & 3 & 2 \\ 3 & 0 & 2 & 4 & 3 & 1 \\ 4 & 2 & 0 & 3 & 1 & 3 \\ 1 & 4 & 3 & 0 & 2 & 3 \\ 3 & 3 & 1 & 2 & 0 & 4 \\ 2 & 1 & 3 & 3 & 4 & 0 \end{pmatrix} \tag{4.10}$$

we can notice that (4.9) does not work for some choice of $i$ and $j$. For example

$$\begin{aligned} 3 = D(1,5) &\neq \min\{D(1,2) + D(2,3) + D(3,4) + D(4,5), D(1,6) + D(6,5)\} \\ &= \min\{3+2+3+2, 2+4\} \\ &= \min\{10, 6\} = 6 \end{aligned} \tag{4.11}$$

However, $D$ has a realisation by the $6-$cycle in Figure 4.7. Here, we can notice that the vertices are ordered following the permutation $\pi = (4,5,3,2,6,1)$. Since $\pi^2(1) = 5$, we can compute again (4.11) using (4.8) and so obtain

$$\begin{aligned} 3 = D(1,5) &= D(1,\pi^2(1)) = \\ &= \min\left\{\sum_{t=0}^{1} D(\pi^t(i), \pi^{t+1}(i)), \sum_{t=2}^{5} D(\pi^t(i), \pi^{t+1}(i))\right\} \\ &= \min\{D(1,4) + D(4,5), D(5,3) + D(3,2) + D(2,6) + D(6,1)\} \\ &= \min\{1+2, 1+2+1+2\} \\ &= \min\{3, 6\} = 3. \end{aligned}$$

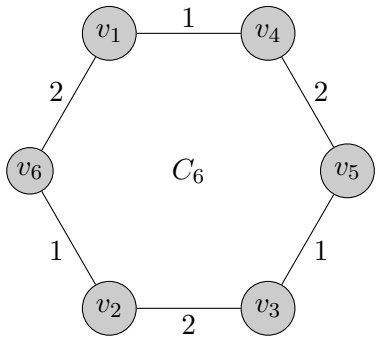



Figure 4.7: Realisation of the matrix in 4.10.

It is worth to note that, if $s = 1$, then formula (4.8) become

$$D(i, \pi(i)) = \min \left\{ D(i, \pi(i)), \sum_{t=1}^{n-1} D(\pi^t(i), \pi^{t+1}(i)) \right\}$$

which is trivially satisfied. The same happens when $s = n - 1$. Hence these two cases can be omitted in formula (4.8).

A realisation of a metric $D$ is minimal if the removal of an arbitrary edge yields a graph that does not realise $D$. Given a metric $D$ on $X$ with $|X| \geq 4$, we say that $D$ is *cyclelike* if there is a minimal realisation for $D$ that is a cycle. This type of metric was discussed in, e.g., [2], [29] and [44]. In [2], the authors characterize cyclelike metrics according to equation (4.9), after performing an algorithm that re-enumerate rows and columns of the metric. We want to mention a useful result from [29] (Theorem 4.4).

**Theorem 4.8.** *Suppose $D$ is a cyclelike metric on a finite set $X$ and a cycle $C$ is a minimal realisation of $D$ with $V(C) = \{v_1, v_2, \ldots, v_m\}$, $m \geq 4$ and $E(C) = \{\{v_i v_{i+1}\} : 1 \leq i \leq m\}$, where the indices are taken modulo $m$. Then, $C$ is an optimal realisation of $D$ if and only if*

$$D(i-1, i) + D(i, i+1) = D(i-1, i+1) \tag{4.12}$$

*holds for all $i$. In this case, $C$ is the unique optimal realisation of $D$.*

It is not true, in general, that a metric $D$ satisfying the conditions of Theorem 4.7, must satisfy also equations (4.12), where the order of the indices is given by the permutation $\pi$. However, this becomes true when we add an extra condition on $D$, thus we have the following result.

**Proposition 4.9.** *Let $D$ be a distance matrix such that $\mathbf{a}_D$ is the null vector. If $D$ satisfies the conditions of Theorem 4.7, then the realisation of $D$ by a cycle $C$ is optimal.*

*Proof.* For simplicity of notation, we assume that $D$ satisfies equations (4.8) with respect to the permutation $\pi = (2, 3, \ldots, n-1, n, 1)$ of the indices of the vertices.

Suppose that the realisation $C$ is not optimal, then, according to Theorem 4.8 there are vertices $v_{i-1}, v_i, v_{i+1}$ such that

$$D(i-1, i+1) < D(i-1, i) + D(i, i+1). \tag{4.13}$$

Since $D$ satisfies the conditions of Theorem 4.7 we must have

$$D(i-1, i+1) = D(i+1, i+2) + D(i+2, i+3) + \cdots + D(i-2, i-1),$$

where the indices are taken modulo $n$. Let us add an edge between vertices $v_{i-1}$ and $v_{i+1}$ of weight $w = D(i+1, i+2) + D(i+2, i+3) + \cdots + D(i-2, i-1)$. We can then apply an elementary cycle reduction (see [22]) obtaining a new vertex $w$ and the following set of weighted edges

- the edge $(v_{i-1}w)$ of weight $\frac{1}{2}\left(D(i-1, i) + W - D(i, i+1)\right)$
- the edge $(v_i w)$ of weight $\frac{1}{2}\left(D(i-1, i) + D(i, i+1) - W\right)$
- the edge $(v_{i+1}w)$ of weight $\frac{1}{2}\left(D(i+1, i) + W - D(i-1, i)\right)$

By (4.13), the edge $(v_i w)$ has strictly positive weight and this number is exactly the $i-$th entry of the compaction vector $\mathbf{a}_D$, which is a contradiction. □

The application of Theorem 4.7 requires, in principle, to check all possible $n!$ permutations, but this is intractable also for $n$ of moderate size. However, the following greedy algorithm permits to find an expected permutation $\pi$ in $O(n^2)$ time.

INPUT: a distance matrix $D$ of order $n$

(1) set $\pi_n = 1$ and $L = \{1\}$

(2) choose the two minimal entries $D(1, i)$ and $D(1, j)$ in the first row (if more, choose two of them);

(3) set $\pi_2 = i$, $\pi_{n-1} = j$ and $L = \{1, i, j\}$

(4) for $s$ from 2 to $n-2$ do

  (4.1) choose minimal entry $D(\pi_s, k)$ in $\pi_s$-th row with $k \notin L$ (if more, choose one of them);

  (4.2) set $\pi_{s+1} = k$ and $L := L \cup \{k\}$

(5) if $D(\pi_{n-1}, j)$ is one of the two minimal entries in $\pi_{n-1}$-th row, then $\pi$ is the expected permutation

## 4.3 Algorithm of realisation of a matrix by a unicyclic graph

The idea of the algorithm is to iterate the processes of compaction and reduction, obtaining at every step $i$ a new matrix $D(i)$. The algorithm will stop when, at a certain step $t$, we will be in one of the following cases

(1) the matrix $D(t)$ has order 2;

(2) the compaction matrix $D(\mathbf{a}_{D(t)})$ is the null matrix;

(3) the compaction vector $\mathbf{a}_{D(t)}$ is the null vector and neither cases (1) or (2) are verified.

If we have kept track of all compaction vectors $\mathbf{a}_{D(i)}$ and of all the rows and columns eliminated in the matrices $D(\mathbf{a}_{D(i)})$, by going backwards, we can construct the graph realising the starting matrix $D$, from the graph $G(t)$ which realises the last matrix $D(t)$, adding at every step $t-i$, with $1 \leq i \leq t$, new distinct vertices with edges adjacent to the graph $G_{t-i+1}$.

From Propositions 4.5 and 4.6 we know that in cases (1) and (2), the matrix $D_t$ is realisable by a tree and hence, as we have just said, we will get that $D$ is realisable by a tree. Finally, if we are in case (3) and we verify, by Theorem 4.7, that $D(t)$ is realisable by an $m-$cycle, with $m \leq n$ then the matrix $D$ can be realised by a unicyclic graph.

We divide the algorithm into two parts: a first part of analysis, which determines if or not a distance matrix can be realised by a unicyclic graph and then, a part of reconstruction which, using data from the analysis part, constructs the desired unicyclic graph.

### Algorithm

**INPUT:** a distance matrix $D$ of order $n$.

**Analysis**:

**Step 0:** Let $t = 0$, $D(t) = D$, $\Theta = \{1, 2, \ldots, n\}$ and $\rho = n$.

**Step 1:** Compute the compaction vector $\mathbf{a}_{D(t)}$ of $D(t)$.

If $\mathbf{a}_{D(t)}$ is the null vector go to **Step 4**.

**Step 2:** Compute the compaction matrix $D(\mathbf{a}_{D(t)})$ of $D(t)$. If $D(\mathbf{a}_D(t))$ is the zero matrix, then, by Proposition 4.6, $D(t)$ is realisable by a tree $G(V, E)$ and go to **Step 5**.

**Step 3:** Let

$$S(t) = \Big\{\big\{i_{1,1}(t), i_{1,2}(t), \ldots, i_{1,s_1(t)}(t)\big\}, \ldots, \big\{i_{\theta(t),1}(t), i_{\theta(t),2}(t), \ldots, i_{\theta(t),s_\theta(t)}(t)\big\}\Big\}$$

be the collection of all distinct (ordered) subset of $\Theta$ of indices of equal rows of $D(\mathbf{a}_{D(t)})$, with $|S(t)| = \theta(t)$, and let

$$S'(t) = \{j_1(t), j_2(t), \ldots, j_{\sigma(t)}(t)\}$$

be the set of (ordered) indices of $\Theta$ not in any subset of $S(t)$, with $|S'(t)| = \sigma(t)$.

For $k$ from 1 to $\theta(t)$ do

**Step 3.1:** Remove from $D(t)$ all rows and columns indexed by

$$i_{k,2}(t), i_{k,3}(t), \ldots i_{k,s_k(t)}(t)$$

**Step 3.2:** Relabel $i_{k,1}(t)$-th row and column of $D(t)$ by $\rho + k$

**Step 3.3:** Set

$$\Theta := \Big(\Theta \cup \{\rho + k\}\Big) \setminus \{i_{k,1}(t), i_{k,2}(t), \ldots, i_{k,s_k}(t)\}.$$

For $k$ from 1 to $\sigma(t)$ do

**Step 3.4:** Relabel $j_k(t)$-th row and column of $D(t)$ by $\rho + \theta(t) + k$

**Step 3.5:** Set

$$\Theta := \Big(\Theta \cup \{\rho + \theta(t) + k\}\Big) \setminus \{j_k(t)\}.$$

We get the reduction matrix $\hat{D}(\mathbf{a}_{D(t)})$ with a new labelling of rows and columns.

If $\hat{D}(\mathbf{a}_{D(t)})$ has order 2 then, by Proposition 4.5, $D(t)$ is realisable by a tree $G(V, E)$ and go to **Step 5**.

Otherwise set

$$\rho := \rho + \theta(t) + \sigma(t),$$

$$t := t + 1,$$

$$D(t) := \hat{D}(\mathbf{a}_{D(t-1)})$$

and return to **Step 1**.

**Step 4:** Check if $D(t)$ is realisable by an $m-$cycle $G(V, E)$ with $m = |\Theta|$. If so, then $D$ can be realised by a unicyclic graph and go to **Step 5**.

Otherwise, $D$ has not a realisation by a tree or a unicyclic graph then EXIT.

**Reconstruction**:

**Step 5** For $\tau$ from $t-1$ to 0 do

Set $\rho := \rho - \theta(\tau) - \sigma(\tau)$

**Step 5.1:** for $\kappa$ from 1 to $\sigma(\tau)$ do

If $(\mathbf{a}_{D(\tau)})_{j_\kappa(\tau)} \neq 0$ then

add vertex: $V := V \cup \left\{v_{j_\kappa(\tau)}\right\}$

add weighted edge: $E := E \cup \left\{(v_{\rho+\theta(\tau)+\kappa} v_{j_\kappa(\tau)}, (\mathbf{a}_{D(\tau)})_{j_\kappa(\tau)})\right\}$

Else

replace vertex: $V := \left(V \setminus \{v_{\rho+\theta(\tau)+k}\}\right) \cup \left\{v_{j_\kappa(\tau)}\right\}$

**Step 5.2:** for $\kappa$ from 1 to $\theta(\tau)$ do

Add vertices: $V := V \cup \left\{v_{i_{\kappa,1}(\tau)}, v_{i_{\kappa,2}(\tau)}, \ldots, v_{i_{\kappa,s_\kappa(\tau)}(\tau)}\right\}$

Add weighted edges:

$$E := E \cup \left\{(v_{\rho+\kappa} v_{i_{\kappa,1}(\tau)}, (\mathbf{a}_{D(\tau)})_{i_{\kappa,1}(\tau)}), \ldots, (v_{\rho+\kappa} v_{i_{\kappa,s_\kappa}(\tau)}, (\mathbf{a}_{D(\tau)})_{i_{\kappa,s_\kappa(\tau)}(\tau)})\right\}$$

**OUTPUT:** the graph $G$ realising $D$.

It is easy to observe that, in the algorithm, the equal rows in a compaction matrix will correspond to vertices which are adjacent to the same interior vertex.

The study of recursive compaction vectors is equivalent to removing all pendant trees in the optimal realisation. In such a procedure, the number of operations grows in a time complexity of $O(n^4)$, where $n$ is the order of the distance matrix $D$. As a matter of fact, computing an entry of the compaction vector is done in a time complexity of $O(n^2)$. This must be done for the $n$ entries of the compaction vector. Moreover, we need to compute the compaction vector for the successive steps of the algorithm. The number of steps is at most $n$.

We conclude this section with a result about optimality of this realisation which is enunciated in Proposition 4.11. Before that, we need to recall a preliminary Theorem from [22] (Theorem 5).

**Theorem 4.10.** *If $0 \leq a \leq a_i$ and if $G_i(a)$ is an optimal realisation of $D_i(a)$, then $G$ obtained from $G_i(a)$ adding the vertex $v_i'$ to $G_i(a)$ and the edge $(v_i' v_i)$ of weight $a$, is an optimal realisation of $D$.*

**Proposition 4.11.** *If the algorithm outputs a unicyclic graph realising $D$, then this realisation is optimal.*

*Proof.* In the analysis part, the algorithm checks if the matrix $D(t)$ satisfies equations (4.8) at Step 4, which is reached if the compaction vector of $D(t)$ is the null vector. Hence, by Proposition 4.9, if $D(t)$ has a realisation by a cycle $C$, this realisation is optimal.

Successively, the reconstruction part of the algorithm reconstructs the graph $G$ realising $D$, starting from $C$ and adding, recursively, pendant edges. Hence, by Theorem 4.10, the realisation of $D$ is optimal. □

## 4.4 Two computational examples

The following examples shows how the algorithm works.

**Example 1.** Consider the following matrix

$$D = \begin{array}{c} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \\ \mathbf{6} \end{array} \begin{array}{c} \begin{array}{cccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} \end{array} \\ \begin{pmatrix} 0 & 2 & 5 & 6 & 7 & 4 \\ 2 & 0 & 5 & 6 & 7 & 4 \\ 5 & 5 & 0 & 4 & 5 & 3 \\ 6 & 6 & 4 & 0 & 3 & 2 \\ 7 & 7 & 5 & 3 & 0 & 3 \\ 4 & 4 & 3 & 2 & 3 & 0 \end{pmatrix} \end{array}$$

We set $t = 0$, $D(0) = D$, $\Theta = \{1, 2, 3, 4, 5, 6\}$ and $\rho = 6$. The compaction vector of $D(0)$ is $\mathbf{a}_{D(0)} = (1, 1, \frac{3}{2}, 1, 2, 0)$. Since $\mathbf{a}_{D(0)}$ is not the zero vector we pass to **Step 2** computing the compaction matrix

$$D(\mathbf{a}_{D(0)}) = \begin{array}{c} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \\ \mathbf{6} \end{array} \begin{array}{c} \begin{array}{cccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} \end{array} \\ \begin{pmatrix} 0 & 0 & \frac{5}{2} & 4 & 4 & 3 \\ 0 & 0 & \frac{5}{2} & 4 & 4 & 3 \\ \frac{5}{2} & \frac{5}{2} & 0 & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} \\ 4 & 4 & \frac{3}{2} & 0 & 0 & 1 \\ 4 & 4 & \frac{3}{2} & 0 & 0 & 1 \\ 3 & 3 & \frac{3}{2} & 1 & 1 & 0 \end{pmatrix} \end{array}$$

Since $D(\mathbf{a}_{D(0)})$ is not the zero matrix we pass to **Step 3**. Notice that rows 1 and 4 are respectively equal to rows 2 and 5. Hence $\theta(0) = 2$, $\sigma(0) = 2$ and

$$S(0) = \Big\{\{1, 2\}, \{4, 5\}\Big\} \quad S'(0) = \{3, 6\}$$

Hence, we remove rows (and columns) labeled by 2 and 5 and relabel

- rows (and columns) 1 and 4 respectively by $7(= \rho + 1)$ and $8(= \rho + 2 = \rho + \theta(0))$,

- rows (and columns) 3 and 6 respectively by $9(=\rho+\theta(0)+1)$ and $10(=\rho+\theta(0)+2=\rho+\theta(0)+\sigma(0))$.

We get

$$\Theta=\{1,2,3,4,5,6\}\cup\{7,8,9,10\}\setminus\{1,2,3,4,5,6\}=\{7,8,9,10\}$$

and

$$\hat{D}(\mathbf{a}_{D(0)})=\begin{array}{c}\\ \mathbf{7}\\ \mathbf{9}\\ \mathbf{8}\\ \mathbf{10}\end{array}\begin{array}{c}\begin{array}{cccc}\mathbf{7} & \mathbf{9} & \mathbf{8} & \mathbf{10}\end{array}\\ \begin{pmatrix}0 & \frac{5}{2} & 4 & 3\\ \frac{5}{2} & 0 & \frac{3}{2} & \frac{3}{2}\\ 4 & \frac{3}{2} & 0 & 1\\ 3 & \frac{3}{2} & 1 & 0\end{pmatrix}\end{array}$$

Since $\hat{D}(\mathbf{a}_{D(0)})$ has not order 2, we come back to **Step 1** after setting $t=1$, $\rho=10$ and $D(1)=\hat{D}(\mathbf{a}_{D(0)})$.

The compaction vector of $D(1)$ is $\mathbf{a}_{D(1)}=(2,0,\frac{1}{2},0)$ and again we pass to **Step 2** computing the compaction matrix

$$D(\mathbf{a}_{D(1)})=\begin{array}{c}\\ \mathbf{7}\\ \mathbf{9}\\ \mathbf{8}\\ \mathbf{10}\end{array}\begin{array}{c}\begin{array}{cccc}\mathbf{7} & \mathbf{9} & \mathbf{8} & \mathbf{10}\end{array}\\ \begin{pmatrix}0 & \frac{1}{2} & \frac{3}{2} & 1\\ \frac{1}{2} & 0 & 1 & \frac{3}{2}\\ \frac{3}{2} & 1 & 0 & \frac{1}{2}\\ 1 & \frac{3}{2} & \frac{1}{2} & 0\end{pmatrix}\end{array}$$

This matrix has not equal rows so $\theta(1)=0$ and $\sigma(1)=4$ with $S'(1)=\{7,9,8,10\}$. After **Step 3.4** and **Step 3.5** we get $\Theta=\{11,12,13,14\}$ and

$$\hat{D}(\mathbf{a}_{D(1)})=\begin{array}{c}\\ \mathbf{11}\\ \mathbf{12}\\ \mathbf{13}\\ \mathbf{14}\end{array}\begin{array}{c}\begin{array}{cccc}\mathbf{11} & \mathbf{12} & \mathbf{13} & \mathbf{14}\end{array}\\ \begin{pmatrix}0 & \frac{1}{2} & \frac{3}{2} & 1\\ \frac{1}{2} & 0 & 1 & \frac{3}{2}\\ \frac{3}{2} & 1 & 0 & \frac{1}{2}\\ 1 & \frac{3}{2} & \frac{1}{2} & 0\end{pmatrix}\end{array}$$

Since it has not order 2, we set $t=2$, $\rho=14$ and $D(2)=\hat{D}(\mathbf{a}_{D(1)})$.

Computing the compaction vector of $D(2)$ we get the null vector. Hence, in the analysis part of the algorithm we move to **Step 4**.

Notice that $D(2)$ satisfies Formula (4.8) of Theorem 4.7 for $\pi=(2,3,4,1)$ hence it is realisable by the $n-$cycle $G$ in Figure 4.8(a) and, moreover, $D$ will be realisable by a unicyclic graph.

We start now the reconstruction part of the algorithm to determine such a graph. Since for $\tau=1(=t-1)$ one has $\theta(1)=0$ and $\sigma(1)=4$ we set $\rho=10$ and we perform only **Step 5.1**. More in details, one has that

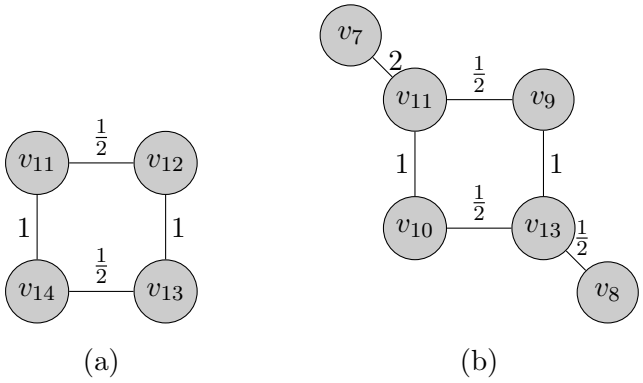



Figure 4.8: Intermediate steps for the realisation of $D$ of Example 1.

- since $(\mathbf{a}_{D(1)})_{\mathbf{7}} = 2$ we add vertex $\{v_7\}$ and edge $(v_{11}v_7, 2)$.
- since $(\mathbf{a}_{D(1)})_{\mathbf{9}} = 0$ we replace vertex $\{v_{12}\}$ by vertex $\{v_9\}$.
- since $(\mathbf{a}_{D(1)})_{\mathbf{8}} = \frac{1}{2}$ we add vertex $\{v_8\}$ and edge $(v_{13}v_8, \frac{1}{2})$.
- since $(\mathbf{a}_{D(1)})_{\mathbf{10}} = 0$ we replace vertex $\{v_{14}\}$ by vertex $\{v_{10}\}$.

Thus, we get the graph in Figure 4.8(b).

Now we perform one more time **Step 5** for $\tau = 0$ (hence $\rho = 10 - \theta(0) - \sigma(0) = 6$). Performing **Step 5.1** one has that

- since $(\mathbf{a}_{D(0)})_{\mathbf{3}} = \frac{3}{2}$ we add vertex $\{v_3\}$ and edge $(v_9v_3, \frac{3}{2})$.
- since $(\mathbf{a}_{D(0)})_{\mathbf{6}} = 0$ we replace vertex $\{v_{10}\}$ by vertex $\{v_6\}$.

Performing **Step 5.2** we add

- vertices $\{v_1\}$ and $\{v_2\}$ and edges $(v_7v_1, 1)$ and $(v_7v_2, 1)$
- vertices $\{v_4\}$ and $\{v_5\}$ and edges $(v_8v_4, 1)$ and $(v_8v_5, 2)$

This concludes the algorithm giving the unicyclic graph in Figure 4.9, which realises $D$.

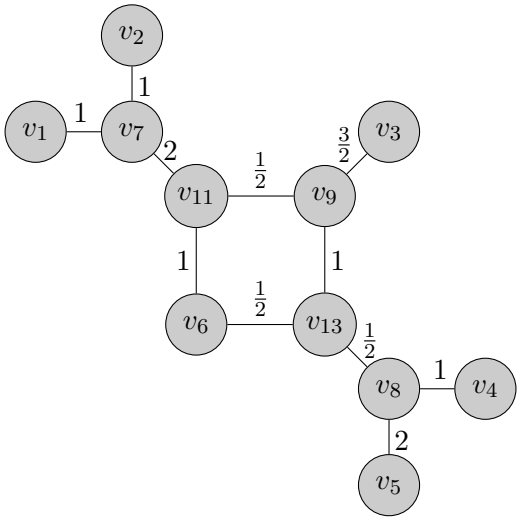



Figure 4.9: Unicyclic graph realising the matrix $D$ of Example 1.

**Example 2.** Consider the following matrix

$$D = \begin{array}{c} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \\ \mathbf{6} \\ \mathbf{7} \\ \mathbf{8} \\ \mathbf{9} \end{array} \begin{array}{c} \begin{array}{ccccccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} & \mathbf{9} \end{array} \\ \begin{pmatrix} 0 & 4 & 2 & 6 & 4 & 6 & 2 & 2 & 5 \\ 4 & 0 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 2 & 4 & 0 & 6 & 4 & 6 & 2 & 2 & 5 \\ 6 & 4 & 6 & 0 & 5 & 2 & 6 & 6 & 4 \\ 4 & 4 & 4 & 5 & 0 & 5 & 4 & 4 & 3 \\ 6 & 4 & 6 & 2 & 5 & 0 & 6 & 6 & 4 \\ 2 & 4 & 2 & 6 & 4 & 6 & 0 & 2 & 5 \\ 2 & 4 & 2 & 6 & 4 & 6 & 2 & 0 & 5 \\ 5 & 4 & 5 & 4 & 3 & 4 & 5 & 5 & 0 \end{pmatrix} \end{array} \tag{4.14}$$

We set $t = 0$, $D(0) = D$, $\Theta = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and $\rho = 9$. The compaction vector of $D(0)$ is $\mathbf{a}_{D(0)} = (1, 1, 1, 1, 1, 1, 1, 1, 1)$. Since $\mathbf{a}_{D(0)}$ is not the zero vector we pass to **Step 2** computing

the compaction matrix

$$D(\mathbf{a}_{D(0)}) = \begin{array}{c} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \\ \mathbf{6} \\ \mathbf{7} \\ \mathbf{8} \\ \mathbf{9} \end{array} \begin{array}{c} \begin{array}{ccccccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} & \mathbf{9} \end{array} \\ \begin{pmatrix} 0 & 2 & 0 & 4 & 2 & 4 & 0 & 0 & 3 \\ 2 & 0 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 0 & 2 & 0 & 4 & 2 & 4 & 0 & 0 & 3 \\ 4 & 2 & 4 & 0 & 3 & 0 & 4 & 4 & 2 \\ 2 & 2 & 2 & 3 & 0 & 3 & 2 & 2 & 1 \\ 4 & 2 & 4 & 0 & 3 & 0 & 4 & 4 & 2 \\ 0 & 2 & 0 & 4 & 2 & 4 & 0 & 0 & 3 \\ 0 & 2 & 0 & 4 & 2 & 4 & 0 & 0 & 3 \\ 3 & 2 & 3 & 2 & 1 & 2 & 3 & 3 & 0 \end{pmatrix} \end{array}$$

Since $D(\mathbf{a}_{D(0)})$ is not the zero matrix we pass to **Step 3**. Here we have

$$S(0) = \Big\{\{1,3,7,8\},\{4,6\}\Big\} \quad S'(0) = \{2,5,9\},$$

$$\theta(0) = 2 \qquad \sigma(0) = 3.$$

Hence, we remove rows (and columns) labeled by 3, 7, 8 and 6 and relabel

- rows (and columns) 1 and 4 respectively by $10(= \rho + 1)$ and $11(= \rho + 2 = \rho + \theta(0))$,
- rows (and columns) 2,5 and 9 respectively by $12(= \rho + \theta(0) + 1)$, 13 and $14(= \rho + \theta(0) + 3 = \rho + \theta(0) + \sigma(0))$.

We get

$$\Theta(t) = \{10, 11, 12, 13, 14\}$$

and

$$\hat{D}(\mathbf{a}_{D(0)}) = \begin{array}{c} \\ \mathbf{10} \\ \mathbf{12} \\ \mathbf{11} \\ \mathbf{13} \\ \mathbf{14} \end{array} \begin{array}{c} \begin{array}{ccccc} \mathbf{10} & \mathbf{12} & \mathbf{11} & \mathbf{13} & \mathbf{14} \end{array} \\ \begin{pmatrix} 0 & 2 & 4 & 2 & 3 \\ 2 & 0 & 2 & 2 & 2 \\ 4 & 2 & 0 & 3 & 2 \\ 2 & 2 & 3 & 0 & 1 \\ 3 & 2 & 2 & 1 & 0 \end{pmatrix} \end{array}$$

Since $\hat{D}(\mathbf{a}_{D(0)})$ has not order 2, we come back to **Step 1** after setting $t = 1$, $\rho = 14$ and $D(1) = \hat{D}(\mathbf{a}_{D(0)})$.

The compaction vector of $D(1)$ is $\mathbf{a}_{D(1)} = (1, 0, 1, 0, 0)$ and again we pass to **Step 2** computing the compaction matrix

$$D(\mathbf{a}_{D(1)}) = \begin{array}{c} \\ \mathbf{10} \\ \mathbf{12} \\ \mathbf{11} \\ \mathbf{13} \\ \mathbf{14} \end{array} \begin{array}{c} \begin{array}{ccccc} \mathbf{10} & \mathbf{12} & \mathbf{11} & \mathbf{13} & \mathbf{14} \end{array} \\ \begin{pmatrix} 0 & 1 & 2 & 1 & 2 \\ 1 & 0 & 1 & 2 & 2 \\ 2 & 1 & 0 & 2 & 1 \\ 1 & 2 & 2 & 0 & 1 \\ 2 & 2 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

This matrix has not equal rows so $\theta(1) = 0$ and $\sigma(1) = 5$ with

$$S'(1) = \{10, 12, 11, 13, 14\}.$$

After **Step 3.4** and **Step 3.5** we get $\Theta = \{15, 16, 17, 18, 19\}$ and

$$\hat{D}(\mathbf{a}_{D(1)}) = \begin{array}{c} \\ \mathbf{15} \\ \mathbf{16} \\ \mathbf{17} \\ \mathbf{18} \\ \mathbf{19} \end{array} \begin{array}{c} \begin{array}{ccccc} \mathbf{15} & \mathbf{16} & \mathbf{17} & \mathbf{18} & \mathbf{19} \end{array} \\ \begin{pmatrix} 0 & 1 & 2 & 1 & 2 \\ 1 & 0 & 1 & 2 & 2 \\ 2 & 1 & 0 & 2 & 1 \\ 1 & 2 & 2 & 0 & 1 \\ 2 & 2 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

Since it has not order 2, we set $t = 2$, $\rho = 19$ and $D(2) = \hat{D}(\mathbf{a}_{D(1)})$.

Computing the compaction vector of $D(2)$ we get the null vector. Hence, in the analysis part of the algorithm we move to **Step 4**.

Notice that $D(2)$ satisfies Formula (4.8) of Theorem 4.7 for $\pi = (4, 5, 3, 2, 1)$ hence it is realisable by the 5-cycle $G$ in Figure 4.10(a) and, moreover, $D$ will be realisable by a unicyclic graph.

We start now the reconstruction part of the algorithm to determine such a graph. Since for $\tau = 1 (= t - 1)$ one has $\theta(1) = 0$ and $\sigma(1) = 5$ we set $\rho = 14$ and we perform only **Step 5.1**. More in details, one has that

- since $(\mathbf{a}_{D(1)})_{\mathbf{10}} = 1$ we add vertex $\{v_{10}\}$ and edge $(v_{15}v_{10}, 1)$.
- since $(\mathbf{a}_{D(1)})_{\mathbf{12}} = 0$ we replace vertex $\{v_{16}\}$ by vertex $\{v_{12}\}$.
- since $(\mathbf{a}_{D(1)})_{\mathbf{11}} = 1$ we add vertex $\{v_{11}\}$ and edge $(v_{17}v_{11}, 1)$.
- since $(\mathbf{a}_{D(1)})_{\mathbf{13}} = 0$ we replace vertex $\{v_{18}\}$ by vertex $\{v_{13}\}$.
- since $(\mathbf{a}_{D(1)})_{\mathbf{14}} = 0$ we replace vertex $\{v_{19}\}$ by vertex $\{v_{14}\}$.

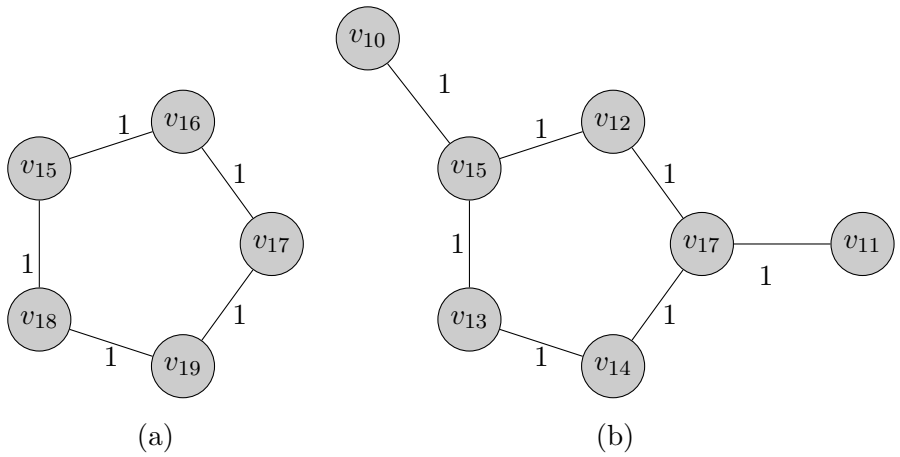



Figure 4.10: Intermediate steps for the realisation of $D$ of Example 2.

Thus, we get the graph in Figure 4.10(b).

Now, we perform one more time **Step 5** for $\tau = 0$ (hence $\rho = 14 - \theta(0) - \sigma(0) = 9$). Performing **Step 5.1** one has that

- since $(\mathbf{a}_{D(0)})_{\mathbf{2}} = 1$ we add vertex $\{v_2\}$ and edge $(v_{12}v_2, 1)$.
- since $(\mathbf{a}_{D(0)})_{\mathbf{5}} = 1$ we add vertex $\{v_5\}$ and edge $(v_{13}v_5, 1)$.
- since $(\mathbf{a}_{D(0)})_{\mathbf{9}} = 1$ we add vertex $\{v_9\}$ and edge $(v_{14}v_9, 1)$.

Performing **Step 5.2** we add

- vertices $\{v_1\}$, $\{v_3\}$,$\{v_7\}$ and $\{v_8\}$ and edges $(v_{10}v_1, 1)$, $(v_{10}v_3, 1)$ $(v_{10}v_7, 1)$ and $(v_{10}v_8, 1)$
- vertices $\{v_4\}$ and $\{v_6\}$ and edges $(v_{11}v_4, 1)$ and $(v_{11}v_6, 1)$

This concludes the algorithm giving the unicyclic graph in Figure 4.11, which realises $D$.

## 4.5 Conclusions and Open Problems

In this chapter we presented an algorithm, running in time $O(n^4)$, such that, taken as input a distance matrix $D$, firstly returns if $D$ is or not realisable by a tree or a unicyclic graph and, in case of affirmative answer, it reconstruct the graph itself. In Proposition 4.11, we also show that if the algorithm outputs a unicyclic graph realising $D$, then this realisation is optimal. In [6] we also show some examples performed by our algorithm in the software `Maple`$^{\text{TM}}$. Here, starting from some weighted unicyclic graphs, we computed their distance matrices and we used them, as input, for the algorithm. The Figure 4.12 show the original graphs (on the left) and the outputs of the algorithm in `Maple`$^{\text{TM}}$ (on the right).

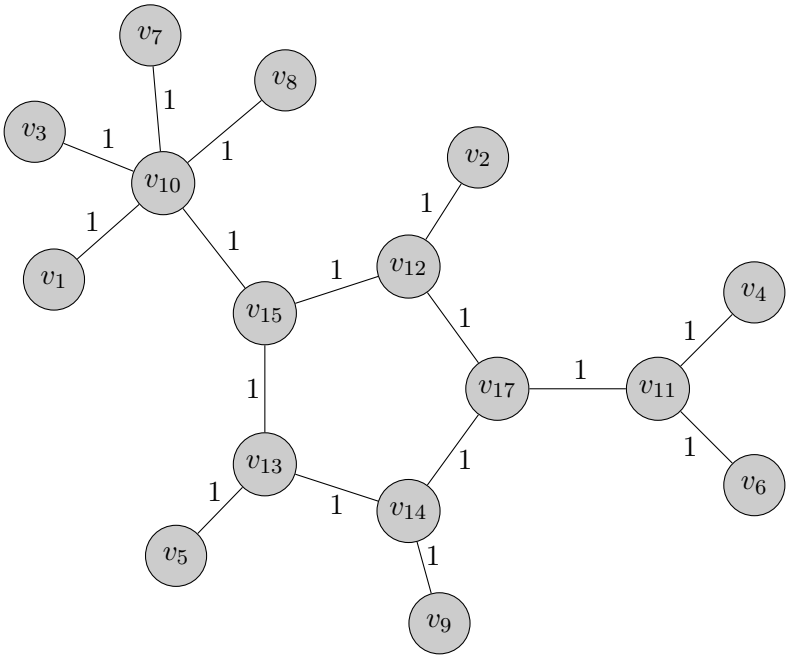



Figure 4.11: Unicyclic graph realising the matrix $D$ of Example 2.

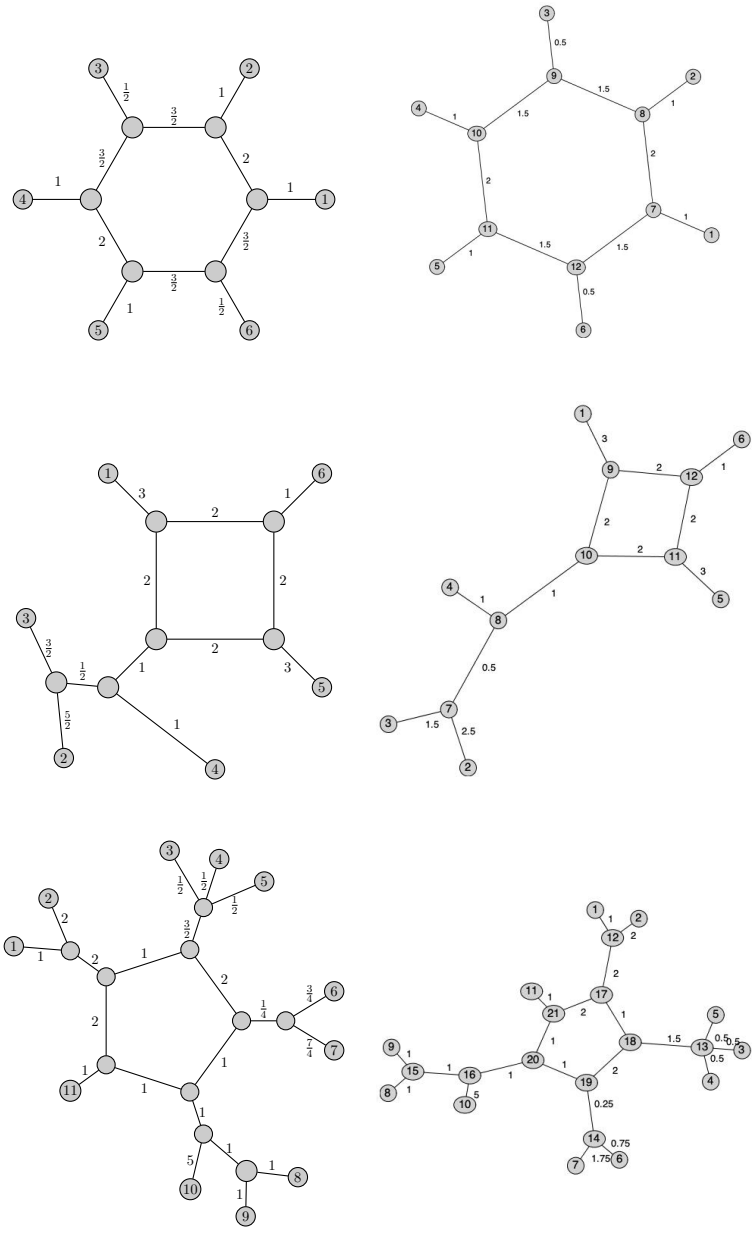
The conditions given by Theorem 4.7, to check if a distance matrix has a realisation by a $n-$cycle, suggests us a possible direction of research. As a matter of fact, formula (4.8 of Theorem 4.7 means that $D(i,\pi^s(i))$ is equal to the minimum between $D(i,\pi(i)) + D(\pi(i),\pi^2(i)) + \cdots + D(\pi^{s-1}(i),\pi^s(i))$ and $D(\pi^s(i),\pi^{s+1}(i)) + \cdots + D(\pi^{n-1}(i),i)$. Then the minimum is attained at least twice among the terms

$$\begin{gathered} D(i,\pi^s(i)), \\ D(i,\pi(i)) + D(\pi(i),\pi^2(i)) + \cdots + D(\pi^{s-1}(i),\pi^s(i)) \\ D(\pi^s(i),\pi^{s+1}(i)) + \cdots + D(\pi^{n-1}(i),i) \end{gathered}$$

In terms of Tropical Mathematics ( [36]) this means that the entries of the distance matrix are a zero of the tropical polynomial

$$\begin{aligned} p_{is} := {} & D(i,\pi^s(i)) \oplus \\ & \oplus \left( D(i,\pi(i)) \otimes \cdots \otimes D(pi^{s-1}(i),\pi^s(i)) \right) \oplus \\ & \oplus \left( D(\pi^s(i),\pi^{s+1}(i)) \otimes \cdots \otimes D(\pi^{n-1}(i),i) \right). \end{aligned} \tag{4.15}$$

Hence, the $p_{is}$'s are exactly the tropical equations for the space of weighted $n-$cycles. The next step is that of understanding how to modify these equations to be satisfied by a distance matrix which has a realisation by a unicyclic graph. Reaching this goal, will give us, not only a complete characterisation, tropically speaking, of such distance matrices, but also the tropical equations for the moduli space $\mathcal{M}_1$ of elliptic tropical curves.



(a) (b)

Figure 4.12

Some preliminary results, in this direction, show, for example that if $D$ has a realisation by an $n-$cycle $C$ with only one pendant edge for each node of $C$, then its entries are a zero of the following kind of polynomials

$$\begin{aligned}\tilde{p}_{is} := &\left[\left(\bigotimes_{j\neq i,\pi^s(i)} 2a_j\right)\otimes D(i,\pi^s(i))\right]\oplus\\ &\oplus\left[\left(\bigotimes_{j=\pi^{s+1}(i)}^{\pi^{n-1}(i)} 2a_j\right)\otimes D(i,\pi(i))\otimes\cdots\otimes D(\pi^{s-1}(i),\pi^s(i))\right]\oplus\\ &\oplus\left[\left(\bigotimes_{j=\pi^1(i)}^{\pi^{s-1}(i)} 2a_j\right)\otimes D(\pi^s(i),\pi^{s+1}(i))\otimes\cdots\otimes D(\pi^{n-1}(i),i)\right]\end{aligned} \tag{4.16}$$

where the $a_i$ are the entries of the compaction vector of $D$. Since these entries can be expressed in terms of the entries of $D$, it is easy to check that the polynomials $\tilde{p}_{is}$ are homogeneous, while the ones in (4.15) are not. Moreover, if $D$ has a realisation by an $n-$cycle, then its entries are a zero also of this new set of polynomials, giving the hope to use them for a kind of "four point condition" for unicyclic graphs and then characterize $\mathcal{M}_1$.

# Appendix A

# Symbols and Notation

| | |
|---|---|
| $\mathbb{N}$ | Natural numbers (with zero) |
| $\mathbb{Z}^+$ | Set of positive integers |
| $[i, j]$ | Interval of length $j - i + 1$. |
| $[h]$ | Set of indexes $\{1, \ldots, h\}$. |
| $\binom{n}{k}$ | Binomial coefficient. |
| $\Sigma$ | Finite alphabet. |
| $\Sigma^*$ | Set of strings on $\Sigma$. |
| $\leq_p$ | Karp reducibility. |
| $G$ | Generic static undirected graph |
| $V(G)$ | Vertex set of a static graph $G$ |
| $E(G)$ | Edge set of a static graph $G$ |
| $V$ | Vertex set of a graph clearly identif. from the context. |
| $E$ | Edge set of a graph clearly identif. from the context. |
| $e$ | Generic edge $e \in E(G)$ of a static graph. |
| $uv$ | Generic edge of a given static graph where $u, v \in V(G)$. |
| $G[U]$ | subgraph *induced* by the vertex set $U \in V(G)$. |
| $P_n$ | Static path on $n$ vertices. |
| $\mathcal{G}$ | Generic temporal graph. |
| $G$ | Static underlying graph of a given temporal graph $\mathcal{G}$. |
| $e = uv$ | Edge with end-vertices $u$ and $v$. |
| $\mathcal{T} : E \to 2^{\mathbb{Z}^+} \setminus \{\emptyset\}$ | Labeling function of a temporal graph. |
| $\mathcal{T}(e)$ | Set of time appearances of an edge $e$ in a temporal graph. |
| $(e, t)$ | Time-edge of a given temporal graph $\mathcal{G}$. |
| $(uv, t)$ | Time-edge of $\mathcal{G}$ with end-vertices $u, v \in V(G)$. |

| | |
|---|---|
| $\mathcal{E}$ | Set of time-edges of $\mathcal{G}$. |
| $T$ | Lifetime of a temporal graph $\mathcal{G}$ clearly identified from the context. |
| $(X, [a, b])$ | Template with vertex set $X$ and interval $[a, b]$. |
| $S$ | Set of time-edges. |
| $L(S)$ | Lifetime of a set $S$ of time edges. |
| $(V(S), L(S))$ | Template generated by $S$. |
| $S[A]$ | Set of time-edges induced by a fixed vertex set of vertices $A$. |
| $\Delta_1 \in \mathbb{Z}^+$ | Temporal parameter establishing the *density* of an edge in an interval. |
| $\Delta_2 \in \mathbb{Z}^+$ | Temporal parameter determining *independence* between time-edges. |
| $\mathfrak{T}(\mathcal{G}, \Delta_2)$ | Class of all collections of pairwise $\Delta_2$-independent templates on $V(\mathcal{G})$. |
| $\mathcal{G}[A]$ | Temporal graph induced by a set $A$ of time-edges. |
| $\mathcal{G}\|_{[a,b]}$ | Temporal graph restricted to the interval $[a, b]$. |
| $\mathcal{G}[A]\|_{[a,b]}$ | Temporal graph restricted to vertex set $A \subseteq V$ and interval $[a, b]$. |
| $\mathcal{G}_1 \triangle \mathcal{G}_2$ | The set of time-edges appearing in exactly one of $\mathcal{G}_1$ or $\mathcal{G}_2$. |
| $S_i$ | Set of time-edges. |
| $\mathcal{G}_\mathcal{C}$ | Temporal cluster graph which realises the collection of templates $\mathcal{C}$. |
| $\mathcal{M}$ | Temporal matching. |
| $\mathfrak{F}_n$ | Set of all temporal graphs $\mathcal{P}_n = (P_n, \mathcal{T})$ with $P_n$ as underlying graph. |
| $(\mathcal{P}_n, \Delta_1, \Delta_2, k)$ | Instance of ETC. |
| $(\mathcal{P}_n, \Delta_k)$ | Instance of TM. |
| $\sigma$ | Maximum number of appearances allowed for each edge in Section 2.4.1. |
| $D(i, j)$ | Entry of the matrix corresponding to the $i$-th row and $j$-th column. |
| $w : E(G) \to \mathbb{R}$ | Weight function of a weighted graph. |
| $D_i(\alpha)$ | $i-$th compaction matrix of $D$ with respect to $\alpha$. |
| $\mathbf{a}_D$ | Compaction vector of the matrix $D$. |
| $D(\mathbf{a}_D)$ | Compaction matrix, |
| $\hat{D}_{\mathbf{a}_D}$ | Reduction matrix. |

# Acknowledgements

The past three years have been challenging but also extremely exciting for me. Due to the unexpected arrival of the pandemic, Covid 19, unfortunately my plans had to change and I could not live the experience of my Ph.D in the exact way I initially expected. For example, I had to give up the original plan of spending a research period in Glasgow, an experience that I truly wish to have done, as I believe that having the opportunity of working abroad in different places and the experience of collaborating with other professionals in a new environment can broaden professionally and personally your horizons and equip you further for the future. Despite this, I can finally conclude this chapter of my professional life and reflect on my personal growth through the experience of completing this PhD.

During these three years I had the opportunity of meeting and collaborating with highly professionals and dedicated people, who "see the magic" in mathematics as much as I do and inspired me to pursue my passion in research. Particularly, I would like to thank my supervisor prof. Cristiano Bocci from the University of Siena for having supported me in the choice of starting a career in this field. For having had the opportunity of living as much as it had been possible all the experiences that a Ph.D can lead to, I would also like to offer my special thanks to my co-supervisor Prof. Kitty Meeks from the University of Glasgow, through whose mentoring I have learnt and developed significantly. Furthermore, I would like to extend my sincere gratitude to Dr. John Sylvester for the brilliant collaboration in work and thank him for all of his technical advice and suggestions. Moreover, I would like to express my gratitude to my family; my parents Rosaria and Rutilio for the consistent support and a special mention to my brother, Alessandro, for always being there for me. During the uncertain moments of these three years, I knew I could always rely on the unlimited support from my best friend Chiara and all my other friends who stayed by my side. I would also like to mention a particular thanks to my friends from university, who shared with me the passion of 'numbers'. Specifically, I would like to mention Anna and Marco, who always supported me discreetly, but also Antonella, Andrea M., Andrea G., Michela and Serenella together with my PhD colleagues and friends Ilaria, Luca and Valentino. Their friendship and closeness have been fundamental for me also during the period of the lockdown. For the same reasons I wish to show my gratitude to my very close friends

Valentina and Irene, who were always able to support and encourage me, even though they were far away, proving that the geographic distance cannot affect sincere friendships. Finally, I would also like to highlight among these acknowledgements all my friends from dancing for their continuous cheerfulness and express my appreciation for lifting my spirit, showing their kindness and care throughout. Unfortunately, due to the large number of the 'big family' I am unable to name everyone, but I would like to mention among the others Claudio, Helena, Marco, Laura, Matteo, Teresa, Francesco, Nadia, Serena, Alice, Leonardo, Salvo and many more who made this period special. Finally, my PhD has come to its conclusion and this chapter of my life is ready to close, leaving behind wonderful memories with some difficulties on the way but full of enthusiasm and exciting moments, which led me to an enormous amount of professional and personal growth and development. This journey has taught me that research cannot provide any security, neither for the future and sometimes nor even for the present. It happens that it leave you "with bated breath" until the very last and it encourages you to continue almost blindfolded through unmarked paths exactly like life does every day. For this reason, I love this occupation and I truly hope that this is not a final goodbye, but a temporary pause until the next adventure.

# Bibliography


[1] Faisal N. Abu-Khzam. The multi-parameterized cluster editing problem. In Peter Widmayer, Yinfeng Xu, and Binhai Zhu, editors, *Combinatorial Optimization and Applications*, pages 284–294, Cham, 2013. Springer International Publishing.

[2] Baldisserri Agnese and Rubei Elena. Distance matrices of some positive-weighted graphs. *Australasian Journal of Combinatorics*, 70(2):185–201, 2018.

[3] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[4] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine learning*, 56(1):89–113, 2004.

[5] Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *J. Comput. Biol.*, 6:281–297, 1999.

[6] Cristiano Bocci and Chiara Capresi. Realization of distance matrices by unicylic graphs. *Rendiconti Sem. Mat. Univ. Pol. Torino*, vol. 79:pp. 55–76, 2021.

[7] Cristiano Bocci, Chiara Capresi, Kitty Meeks, and John Sylvester. A new temporal interpretation of cluster editing. *arXiv preprint arXiv:2202.01103*, 2022.

[8] Sebastian Böcker. A golden ratio parameterized algorithm for cluster editing. *Journal of Discrete Algorithms*, 16:79–89, 2012.

[9] Sebastian Böcker and Jan Baumbach. Cluster editing. In Paola Bonizzoni, Vasco Brattka, and Benedikt Löwe, editors, *The Nature of Computation. Logic, Algorithms, Applications*, pages 33–44, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[10] Sebastian Böcker and Jan Baumbach. Cluster editing. In *Conference on Computability in Europe*, pages 33–44. Springer, 2013.

[11] Peter Buneman. A note on the metric properties of trees. *Journal of combinatorial theory, series B*, 17(1):48–50, 1974.

[12] Didier Burton and Ph L Toint. On the use of an inverse shortest paths algorithm for recovering linearly correlated costs. *Mathematical Programming*, 63(1):1–22, 1994.

[13] Sebastian Böcker. A golden ratio parameterized algorithm for cluster editing. *Journal of Discrete Algorithms*, 16:79–89, 2012. Selected papers from the 22nd International Workshop on Combinatorial Algorithms (IWOCA 2011).

[14] Jiehua Chen, Hendrik Molter, Manuel Sorge, and Ondrej Suchý. Cluster Editing in Multi-Layer and Temporal Graphs. In Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao, editors, *29th International Symposium on Algorithms and Computation (ISAAC 2018)*, volume 123 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[15] Joseph C Culberson and Piotr Rudnicki. A fast algorithm for constructing trees from distance matrices. *Information Processing Letters*, 30(4):215–220, 1989.

[16] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

[17] Steven Delvaux and Leon Horsten. On best transitive approximations to simple graphs. *Acta informatica*, 40(9):637–655, 2004.

[18] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, August 2005.

[19] Jessica Enright, Kitty Meeks, George B Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. In *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[20] Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Yngve Villanger. Tight bounds for parameterized complexity of cluster editing with a small number of clusters. *Journal of Computer and System Sciences*, 80(7):1430–1447, 2014.

[21] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.

[22] S Louis Hakimi and Stephen S Yau. Distance matrix of a graph and its realizability. *Quarterly of applied mathematics*, 22(4):305–317, 1965.

[23] SL Hakimi and SS Yau. Distance matrix and the synthesis of n-port resistance network. Technical report, Technical Report 5, Network Theory Group, 1963.

[24] John M Harris, JL Hirst, and MJ Mossinghoff. Combinatorics and graph theory (2000), 2001.

[25] Erez Hartuv, Armin Schmitt, J Lange, Sebastian Meier-Ewert, Hans Lehrach, and Ron Shamir. An algorithm for clustering cdna fingerprints. *Genomics*, 66 3:249–56, 2000.

[26] Momoko Hayamizu, Katharina T Huber, Vincent Moulton, and Yukihiro Murakami. Recognizing and realizing cactus metrics. *Information Processing Letters*, 157:105916, 2020.

[27] Anne-Sophie Himmel, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. Adapting the bron-kerbosch algorithm for enumerating maximal cliques in temporal graphs. *Soc. Netw. Anal. Min.*, 7(1):35:1–35:16, 2017.

[28] Katharina T Huber, Vincent Moulton, Charles Semple, and Taoyang Wu. Quarnet inference rules for level-1 networks. *Bulletin of mathematical biology*, 80(8):2137–2153, 2018.

[29] Wilfried Imrich, JMS Simoes-Pereira, and Christina M Zamfirescu. On optimal embeddings of metrics in graphs. *Journal of Combinatorial Theory, Series B*, 36(1):1–15, 1984.

[30] David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002.

[31] Christian Komusiewicz and Johannes Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012.

[32] Bernhard H Korte, Jens Vygen, B Korte, and J Vygen. *Combinatorial optimization*, volume 1. Springer, 2011.

[33] Mirko Křivánek and Jaroslav Morávek. Np-hard problems in hierarchical-tree clustering. *Acta informatica*, 23(3):311–323, 1986.

[34] Shaohua Li, Marcin Pilipczuk, and Manuel Sorge. Cluster Editing Parameterized Above Modification-Disjoint $P_3$-Packings. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*, volume 187 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[35] Junjie Luo, Hendrik Molter, André Nichterlein, and Rolf Niedermeier. Parameterized dynamic cluster editing. *Algorithmica*, 83(1):1–44, 2021.

[36] Diane Maclagan and Bernd Sturmfels. *Introduction to tropical geometry*, volume 161. American Mathematical Society, 2021.

[37] Bassel Mannaa. Cluster editing problem for points on the real line: A polynomial time algorithm. *Information processing letters*, 110(21):961–965, 2010.

[38] Bob Mau and Michael A Newton. Phylogenetic inference for binary data on dendograms using markov chain monte carlo. *Journal of Computational and Graphical Statistics*, 6(1):122–131, 1997.

[39] George B. Mertzios, Hendrik Molter, Rolf Niedermeier, Viktor Zamaraev, and Philipp Zschoche. Computing maximum matchings in temporal graphs. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[40] Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016.

[41] Assaf Natanzon. *Complexity and approximation of some graph modification problems*. University of Tel-Aviv, 1999.

[42] Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.

[43] Roded Sharan, Adi Maron-Katz, and Ron Shamir. Click and expander: a system for clustering and visualizing gene expression data. *Bioinformatics*, 19 14:1787–99, 2003.

[44] JMS Simões-Pereira. A note on distance matrices with unicyclic graph realizations. *Discrete mathematics*, 65(3):277–287, 1987.

[45] JMS Simoes-Pereira. An algorithm and its role in the study of optimal graph realizations of distance matrices. *Discrete mathematics*, 79(3):299–312, 1990.

[46] Leo Van Iersel and Vincent Moulton. Trinets encode tree-child and level-2 phylogenetic networks. *Journal of mathematical biology*, 68(7):1707–1729, 2014.

[47] Sacha C Varone. A constructive algorithm for realizing a distance matrix. *European journal of operational research*, 174(1):102–111, 2006.

[48] Tiphaine Viard, Matthieu Latapy, and Clémence Magnien. Computing maximal cliques in link streams. *Theor. Comput. Sci.*, 609:245–252, 2016.

[49] Tiphaine Viard, Matthieu Latapy, and Clémence Magnien. Computing maximal cliques in link streams. *Theoretical Computer Science*, 609:245–252, 2016.