Graph factorization and pseudofactorization with applications to hypercube embeddings

by

Kristin Sheridan

S.B., Electrical Engineering and Computer Science, Massachusetts Institute of Technology (2020)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2021

Author .....
Department of Electrical Engineering and Computer Science
May 26, 2021

Certified by .....
Virginia Vassilevska Williams
Steven and Renee Finn Career Development Associate Professor
Thesis Supervisor

Certified by .....
Mark Bathe
Professor of Biological Engineering
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Graph factorization and pseudofactorization with applications to hypercube embeddings

by

Kristin Sheridan

Submitted to the Department of Electrical Engineering and Computer Science
on May 26, 2021, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In many contexts, it is useful to determine if a particular distance metric can be broken down into other metrics about which more is known. In particular, if a metric can be embedded into a hypercube, the plethora of preexisting knowledge about the structure of a hypercube can provide knowledge about the structure in question. In this paper, we examine the concepts of graph factorization and pseudofactorization, in which a graph is broken up into smaller graphs whose Cartesian product it is isomorphic to or is an isometric subgraph of, respectively. We show that the same or slightly modified versions of the techniques used for this process in the context of unweighted graphs also work for weighted graphs. While it is NP-hard to decide if a general distance metric is hypercube embeddable, we also discuss how these results expand the number of known types of graphs and distance metrics for which this problem is polynomial time decidable. We also discuss why this kind of decomposition of graphs and distance metrics may be of interest in a variety of fields.

Thesis Supervisor: Virginia Vassilevska Williams
Title: Steven and Renee Finn Career Development Associate Professor

Thesis Supervisor: Mark Bathe
Title: Professor of Biological Engineering

Acknowledgments

This thesis would not have been possible without the help and support of many people.

I would like to give a huge thank you to Professor Virginia Williams and Professor Mark Bathe, my thesis advisors. They have both been super supportive and helpful throughout my time working on this project. I was working in Professor Bathe's lab during my junior year of undergrad at the time that I began switching my focus towards computer science, and he was very supportive of me finding projects that fit my interests and helped me figure out what it was that I wanted to study and research. He has also continued to be a supportive mentor over the last year as we began working remotely due to the pandemic.

I began working with Professor Williams last summer, and I have been so grateful to her for her advice on directions to take the project and the current state of various aspects of research in the field. She has been a wonderful mentor and has been especially helpful at all the points where I found myself stuck and was unsure of the direction to take the project.

I would also like to thank Joseph Berleant from Professor Bathe's lab, who originally acted as my undergraduate research mentor. He introduced me to a lot of fundamental papers in this area and has been a great collaborator over the last year and is responsible for an important part of the work tying graph pseudofactorization to hypercube embeddings. I wouldn't even know about this topic if it wasn't for Joseph and certainly wouldn't have all the results I do today.

I'd also like to thank Professor Anne Condon from the University of British Columbia for her support for and contributions to this project. Professor Condon provided great feedback and support throughout the process and I'm very grateful for her help.

Finally, I'd like to thank my family: Mom, Dad, Michael, and Ryan who were very supportive throughout the time I wrote this. I would also like to thank Liz Murray and David Mejorado for keeping me on track and organized throughout the

project and my amazing roommate Ally Geary who heard all my worries throughout the course of the year.

Contents

List of Figures

Chapter 1

Introduction

In fields like biology, models are commonly used to represent physical or theoretical concepts. In many cases those models include their own systems of measurement, and we often try to make this system of measurement such that it has properties of systems we already understand. For example, we can describe a model for measuring similarity between short DNA sequences. DNA is a molecular compound made up of smaller molecular components, known as bases. The four types of such components are A, T, C, and G, and two DNA strands may bind to each other, or stick together in a way that results in a lower energy state. When two DNA strands bind to each other, generally A pairs with T and G pairs with C. However, if two strands are closely matched, they may bind with each other even if it is not possible to make these perfect matches at every position, as represented in Figure 1-1. Under certain energy models, we may treat certain mismatches as having a similar "energy penalty" [21]. We can now re-frame our theoretical set of DNA strands by considering one strand to be the "starting" strand and characterizing all other strands based on the number of positions at which their sequences differ. At this point, we can abstract away further and view the DNA sequences as binary strings in which the starting strand is a string of 0s and each other strand has a zero at a given index if it matches the starting sequence's base on that index and a 1 otherwise. We can additionally simplify this model by taking the substrings that exclude indices where all strings have the same value. This kind of relationship is shown in Figure 1-2, subfigures (b)

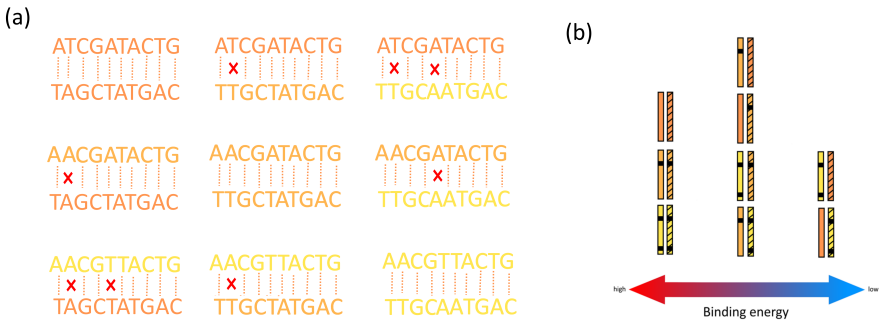

Figure 1-1: (a) An example of a set of three DNA sequences and their complements (the strand a sequence binds to perfectly) (b) A comparison of the binding energy of the pairs presented in (a)

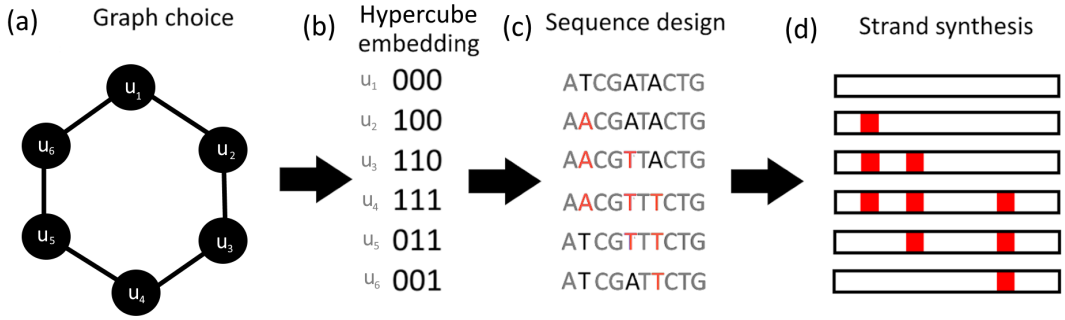

Figure 1-2: A schematic showing how hypercube embeddings can be used to construct a set of DNA sequences with a particular relationship

and (c).

Notably, in the previously described example, we have transformed a complex system of DNA molecules into a model in which to compare two molecules, we only have to compare the number of indices on which two strings differ. This method of measuring difference, known as *Hamming distance*, is very well-characterized, and we can use that to our advantage when working with these kinds of sets. In particular, as we will describe later, it is sometimes interesting to ask to design a set of DNA sequences with relative similarity defined by a given distance metric or graph. In this case, one can look for an assignment of binary strings to every point in that metric such that the number of indices on which two strings differ is equal to their points' distance in the metric or graph. In Figure 1-2, we see this through subfigure (a), which

is a cyclic graph for which we can find such an assignment of binary strings. From there, we can use the given energy model to construct DNA sequences with the desired similarity levels, as shown in the other subfigures. Finding a set of binary strings with the described properties for a given graph is known as a *hypercube embedding*, which as the name suggests is the same as directly embedding the graph into a hypercube.

The ability to find hypercube embeddings for graphs is useful in a variety of fields including linguistic theory [12] and molecular computing. Our DNA design example has potential applications in the field of molecular computation. For example, there has been recent interest in DNA-based neural networks for diagnostic tests. Current formulations rely on the use of non-interacting DNA strands and use different concentrations of particular strands to fill the role of weights [19, 3]. DNA strands with specific relative binding energies may be useful for improving such processes or reducing the number of required molecules, making the process of finding hypercube embeddings relevant to the design of certain molecular circuits. The particular model of DNA sequence design presented as part of the motivation here is due to the work of Joseph Berleant [2].

Throughout this paper, we will explore different methods of *isometrically embedding* one graph into another, that is, mapping the vertices of one graph to another such that distance is preserved. In particular, in Chapter 2, we will discuss the concepts of factorization and pseudofactorization, in which graphs are broken up into smaller graphs into whose Cartesian products they are isometrically embeddable. Then, in Chapter 3, we will discuss some of the implications of this work for hypercube and Hamming embeddings of graphs, as well as their relationship to decomposing general distance metrics. We finish this section by defining some of the terminology used throughout the paper.

1.1 Types of graphs

Throughout this paper, we will consider unweighted and weighted graphs. An unweighted graph $G = (V, E)$ consists of a set of vertices V and a set of edges E. We will

also use $V(G)$ and $E(G)$ to denote the vertices and edges of a graph G, respectively. We will generally denote vertices using a letter or a tuple. For the purposes of this paper, all graphs will be assumed to be undirected, meaning that the edges in the graph have no direction and the distance between any pair of nodes is the same in either direction. We will denote an edge between two vertices u and v as $\{u, v\}$, uv, or vu. In the case of weighted graphs, we also have a *weight function* on the edges which assigns a positive number to each edge in the graph. For these graphs, we will generally write $G = (V, E, w)$ where w is the weight function in question. For a general weighted graph G, we may also write w_G to denote the weight function of G.

We will denote the shortest path distance function associated with a graph as $d_G(\cdot, \cdot)$ (or just $d(\cdot, \cdot)$ if the graph in question is implicit). We will also work with graph embeddings, namely *isometric graph embeddings*. Informally, an isometric embedding from one graph into another is a map from the vertices of the first graph to the second such that the distance between vertices in the first graph is preserved under the mapping to the second graph. Formally, an isometric embedding of a graph G into a graph G' is a mapping $\pi : V(G) \to V(G')$ such that for each $u, v \in V(G)$, $d_G(u, v) = d_{G'}(\pi(u), \pi(v))$.

1.2 Cartesian products and factorization

In order to discuss graph embeddings, we will first discuss Cartesian products of graphs, which we define formally with Definitions 1.2.1 and 1.2.3.

Definition 1.2.1. *For two* ***unweighted*** *graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the* ***Cartesian product*** *of the graphs, $G_1 \times G_2$ is a graph $G = (V, E)$ with vertex and edge sets defined as:*

$$V = \{(u_1, u_2) \mid u_1 \in V_1, u_2 \in V_2\}$$
$$E = \{(u_1, u_2)(v_1, v_2) \mid \textit{either } u_1 = v_1 \textit{ and } u_2v_2 \in E_2 \textit{ or } u_2 = v_2 \textit{ and } u_1v_1 \in E_1\}.$$

We will denote the Cartesian product of m graphs as $G_1 \times G_2 \times \ldots \times G_m$ or $\Pi_{i=1}^{m} G_i$.

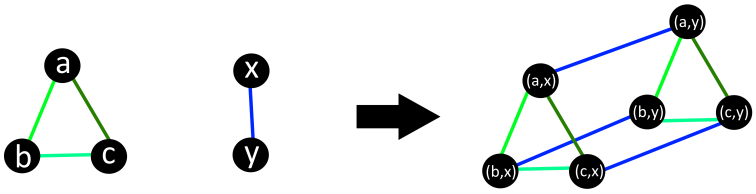

Figure 1-3: The graph on the right is the Cartesian product of the graphs on the left. The parent edge of a given edge in the product graph is the edge in a factor that has the same color as it. For example, edges $(a,x)(a,y)$, $(b,x)(b,y)$, and $(c,x)(c,y)$ in the product graph all have edge xy in the second factor as their parent edge.

Using Definition 1.2.1, we can see that for a graph $G = (V, E) = \Pi_{i=1}^{m} G_i$, the vertex and edge sets are defined as:

$$V = \{(u_1, u_2, ..., u_m) \mid u_1 \in V_1, u_2 \in V_2, \ldots, u_k \in V_m\} \tag{1.1}$$

$$= V_1 \times V_2 \times \ldots \times V_m \tag{1.2}$$

$$E = \{(u_1, u_2, ..., u_m)(v_1, v_2, ..., v_m) \mid \exists \text{ exactly one } \ell \text{ such that } u_\ell v_\ell \in E_\ell \text{ and for all } i \neq \ell, u_i = v_i\}. \tag{1.3}$$

Figure 1-3 shows an example of a Cartesian product of two graphs. We call $G_1, G_2, \ldots, G_m$ *factors* of G because for each G_i there exists a graph such that the Cartesian product of that graph with G_i produces a graph isomorphic to G. If a graph G is such that its only factors are itself and K_1, we call that graph *prime*. A set of prime graphs $\{G_1, G_2, ..., G_m\}$ such that $\Pi_{i=1}^{m} G_i$ is isomorphic to G is a *prime factorization* of G.

Looking at equation (1.3), we see that every edge in a product graph "corresponds" to an edge in exactly one factor graph. We formalize this intuition in Definition 1.2.2.

Definition 1.2.2. *Consider a graph $G = (V, E)$ with a factorization $\{G_1, G_2, ..., G_m\}$ and an edge $uv \in E$. We will take $u = (u_1, u_2, \ldots, u_m)$ and $v = (v_1, v_2, \ldots, v_m)$ where $u_i, v_i \in V_i$. By the definition of the edge set of a Cartesian product, there exists exactly one ℓ such that $u_\ell \neq v_\ell$, and we must have $u_\ell v_\ell \in E_\ell$. We consider $u_\ell v_\ell$ (an edge in G_ℓ) to be the **parent edge** of edge uv in G for the given factorization.*

Using the notion of a parent edge, we will extend our definition in 1.2.1 to apply to weighted graphs as well.

Definition 1.2.3. *For two graphs $G_1 = (V_1, E_1, w_1)$ and $G_2 = (V_2, E_2, w_2)$, we take the* ***Cartesian product*** *$G_1 \times G_2$ to be $G = (V, E, w)$ where V and E are exactly as defined in Definition 1.2.1. Additionally, for $(u_1, u_2)(v_1, v_2) = uv \in E$, $w(uv)$ equals $w_i(u_i v_i)$ where $u_i v_i$ is the parent edge of uv in the given factorization of G.*

The Cartesian product of graphs has an important property about its distance metric. For two nodes $(u_1, u_2), (v_1, v_2)$ in a Cartesian product $G_1 \times G_2$, we can write $d_{G_1 \times G_2}((u_1, u_2), (v_1, v_2)) = \sum_{i=1}^{2} d_{G_i}(u_i, v_i)$. To see that $d_{G_1 \times G_2}((u_1, u_2), (v_1, v_2)) \leq \sum_{i=1}^{2} d_{G_i}(u_i, v_i)$, we can consider a path composed of a shortest path from (u_1, u_2) to $(v_1, u_2) \in V(G_1 \times G_2)$ and a shortest path from (v_1, u_2) to (v_1, v_2). We see that to get this path we can follow a path of edges whose parents form a shortest path from u_1 to v_1 in G_1 and then a path of edges whose parents form a shortest path from u_2 to v_2 in G_2. To see $d_{G_1 \times G_2}((u_1, u_2), (v_1, v_2)) \geq \sum_{i=1}^{2} d_{G_i}(u_i, v_i)$, we take a path from (u_1, u_2) to (v_1, v_2) in $G_1 \times G_2$ and we see that the path formed in G_1 (respectively G_2) by the edge parents in G_1 (respectively G_2) is a path from u_1 to v_1 in G_1 (respectively u_2 to v_2 in G_2). This relationship between the distance metric of a Cartesian product and its factors extends to larger products, so that we have:

$$d_{\Pi_{i=1}^{m} G_i}((u_1, \ldots, u_m)(v_1, \ldots, v_m)) = \sum_{i=1}^{m} d_{G_i}(u_i, v_i).$$

These definitions will help us in defining factorization and pseudofactorization in Chapter 2 and in gaining a more complete understanding of hypercube embeddings in Chapter 3.

1.3 Hypercube and Hamming embeddings

Finally, we will also discuss hypercube and Hamming embeddings. For a given graph G, a Hamming embedding is a map $\eta : V(G) \to \Sigma^t$ for a finite alphabet Σ and integer t such that distance in the graph corresponds to Hamming distance between the strings. Formally, let $d_H(\cdot, \cdot)$ be the Hamming distance function, or the function that outputs the number of indices on which two strings differ. Then for any $u, v \in V(G)$,

$d_G(u, v) = d_H(\eta(u), \eta(v))$. Additionally, for such a mapping $\eta_i(u)$ will be the i^{th} value in the string $\eta(u)$. Finally, a *hypercube embedding* is a Hamming embedding with $|\Sigma| = 2$. We will deal mainly with hypercube embeddings and will generally use binary strings, or $\Sigma = \{0, 1\}$.

These definitions and concepts will be expanded upon further in the relevant sections, as we discuss their usefulness and calculations of certain properties.

Chapter 2

Factorization and pseudofactorization

In this chapter, we discuss factorization and pseudofactorization, which embody the idea of going "backwards" from a Cartesian product. At a high level, factoring a graph seeks to find smaller graphs that can be used to compose the original graph via Cartesian products. Pseudofactorization seeks to find smaller graphs such that the original graph makes up a part of the graph composed by the Cartesian product of those graphs. These concepts, particularly pseudofactorization, are useful for determining the existence of hypercube and Hamming embeddings, as we will discuss in Chapter 3, but they are also interesting in their own right, in that they tell us something about how one structure may be built up from smaller structures. In Chapter 3, we will also elaborate on some of the implications of this for trying to decompose distance metrics.

2.1 Background and definitions

In this section, we will discuss the work of previous authors on the topics of factorization and pseudofactorization. These concepts are very well-characterized for unweighted graphs and even for graphs of uniform weight. The algorithms and concepts used in this unweighted context are highly applicable to the weighted context and in many cases the same theorems and algorithms apply, though the correctness must be re-proven to show they work for weighted graphs.

Because certain concepts are essential to this section, we reiterate some of the definitions given in Chapter 1. In particular, the Cartesian product of two graphs, G_1, G_2, is the graph $G = G_1 \times G_2$ where:

$$V(G) = \{(u_1, u_2) \,|\, u_1 \in V(G_1), u_2 \in V(G_2)\}$$

$$E(G) = \{(u_1, u_2)(v_1, v_2) \,|\, (u_1 = v_1 \text{ and } u_2v_2 \in E(G_2)) \text{ or } (u_2 = v_2 \text{ and } u_1v_1 \in E(G_1))\}$$

For a product of m graphs $G = G_1 \times G_2 \times ... \times G_m = \Pi_{i=1}^{m} G_i$, we have:

$$V(G) = \{(u_1, u_2, ..., u_m) \,|\, \forall i, u_i \in V(G_i)\}$$

$$E(G) = \{(u_1, u_2, ..., u_m)(v_1, v_2, ..., v_m) \,|\, \exists i \text{ such that } u_iv_i \in E(G_i) \text{ and } \forall j \neq i, u_j = v_j\}$$

In Figure 2-1, we have that subfigure (a) is the Cartesian product of the graphs in subfigure (b). The Cartesian product of m graphs can be found in polynomial time in the size of the product graph (although the size of the product graph may be exponential in the size of the factor graphs). A *factorization* of a graph G is a set of factor graphs G_i such that $\Pi_i G_i$ is isomorphic to G. G is *prime* if its only factorizations consist of G itself and K_1. A *prime factorization* is a factorization of G such that all of the factors are prime. We will assume that a factorization never includes K_1 unless the graph being factored is K_1 (so a factorization of a prime graph would only include itself). Imrich and Klavžar [14] showed that deciding if a graph's prime factorization consists entirely of complete graphs can be done in as little as $O(|E|)$ time. More recently, Imrich and Peterin [15] showed that finding the prime factorization of an arbitrary unweighted, connected graphs can also be done in as little as $O(|E|)$ time. However, if the input graph is not connected, this problem is at least as hard as graph isomorphism, which is not known to be in P. We can see this by considering two connected graphs G, H, and noting that $G \cup H$ has the graph of two unconnected nodes as a factor if and only if G and H are isomorphic [9].

The concept of graph factorization can be extended by considering isometric embeddings of graphs into the Cartesian products of other graphs, forming a concept we

call *pseudofactorizations*. In [13], Graham and Winkler define this concept. (Notably, they refer to this process as *factorization*, but I have changed the name in order to distinguish from the concept of factorization used by other authors and addressed here.) Definition 2.1.1 defines pseudofactorization for unweighted graphs in the sense used by Graham and Winkler. In order to use the definition, we further extend our notation for isometric embeddings. Consider an isometric embedding of a graph G into a product of m graphs, $\pi : G \to \Pi_{i=1}^{m} G_i$. For each $u \in V(G)$, if $\pi(u) = (u_1, u_2, \ldots, u_m)$, we let $\pi_i(u) = u_i$, or the node in position i of the tuple that u maps to under π. We additionally let "$\hookrightarrow$" mean "can be isometrically embedded into."

Definition 2.1.1. *For a graph G, a **pseudofactorization** of G is a set of graphs $\{G_1, G_2, \ldots, G_m\}$ such that $G \hookrightarrow \Pi_{i=1}^{m} G_i$. Additionally, to avoid "extraneous" nodes in the product graph, we let π be an isometric embedding of G into $\Pi_{i=1}^{m} G_i$ and require that for each u_i in each G_i, there exists some $u \in V$ such that $\pi_i(u) = u_i$.*

We additionally consider an analog to prime graphs for pseudofactorization, which we call *irreducible* graphs, defined in Definition 2.1.2.

Definition 2.1.2. *A graph G is **irreducible** if all pseudofactorizations of G consist of K_1 and G itself.*

Analogously to a prime factorization, we let an irreducible pseudofactorization be a pseudofactorization in which all pseudofactors are irreducible. Graham and Winkler showed that for each connected unweighted G there is in fact a single irreducible pseudofactorization and thus we can call such a set of graphs the *canonical pseudofactor graphs* or the *canonical pseudofactorization* and the isometric embedding of G into their Cartesian product the *canonical embedding* of G [13, 23].

We note that in the case of weighted graphs, the previous definition of pseudofactorization implies the non-existence of irreducible pseudofactorizations for many graphs. (An example would be K_2, which is isometrically embeddable in the Cartesian product of x copies of K_2 scaled by $1/x$ for any integer x.) Because of this, we slightly alter the definition of pseudofactorization in a way that is consistent with

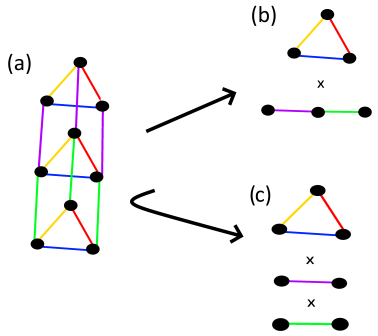

Figure 2-1: Subfigure (a) shows a non-prime, non-irreducible graph. Subfigure (b) shows the prime factorization of the graph in (a). Subfigure (c) shows the canonical pseudofactorization of the graph in (a).

the definition for unweighted graphs but admits irreducible pseudofactorizations for weighted graphs as well. This definition is presented in Definition 2.1.3.

Definition 2.1.3. *A **pseudofactorization** of a graph $G = (V, E, w)$ is a set of graphs $\{G_1, G_2, \ldots, G_m\}$ such that G is an isometric subgraph of $\Pi_{i=1}^{m} G_i$.*

Any H that is a member of some pseudofactorization of G is a *pseudofactor* of G. We note that if the original graph and the pseudofactor graphs are restricted to all have all edge weights 1, this definition is identical to Definition 2.1.1 and we also note that every factor of G is a pseudofactor but the converse is not necessarily true.

Two interesting problems emerge related to the concept of graph pseudofactorization: that of determining if a given graph G is irreducible and that of finding the canonical pseudofactorization of G. It turns out that in the case of connected unweighted graphs both of these questions can be answered in as little as $O(|V||E|)$ time [11] but that finding an irreducible pseudofactorization for an unconnected graph is hard for the same reasons that factorization is. Figure 2-1(c) shows the canonical pseudofactorization of Figure 2-1(a).

2.2 Pseudofactorization

In this section, we will discuss a method for pseudofactoring weighted grahs in polynomial time. To begin, we discuss the current state of the field in terms of pseudofactoring unweighted graphs, and we then show that one of the techniques used for this process can also be used to pseudofactor weighted graphs.

Graham and Winkler [13] showed that all unweighted graphs have a unique pseudofactorization under the definition given in 2.1.1. They additionally gave an $O(|E|^2)$ time algorithm to find such a pseudofactorization. To do so, they defined the θ relation on the edges of a graph as follows.

Definition 2.2.1. *For a graph $G = (V, E)$, two edges in the graph, $uv, ab \in E$ are related by θ if and only if:*

$$[d_G(u, a) - d_G(u, b)] - [d_G(v, a) - d_G(v, b)] \neq 0. \tag{2.1}$$

We note that this relation is symmetric and reflexive. We also let the equivalence relation $\hat{\theta}$ be the transitive closure of θ.

We call equation 2.1 the theta-difference for edges uv and ab, and we denote this difference by $\delta(uv, ab)$.

Algorithm 1 is a generalized version of the algorithm presented by Graham and Winkler. Its inputs are a graph and an equivalence relation on the edges of the graph, and it outputs a set of graphs. Graham and Winkler showed that when the input is $(G, \hat{\theta})$ for an unweighted graph G, the output is an irreducible pseudofactorization of G. Figure 2-2 shows an example of an application of this algorithm to a weighted graph when the input relation is $\hat{\theta}$.

Informally, the algorithm finds the equivalence classes of the graph edges. For each equivalence class E_k, it then looks at a version of G with all edges in E_k removed. The graph is now disconnected, and it is able to use this disconnected graph to construct the output graphs. In this paper, we will expand on this to show that the same algorithm works for weighted graphs.

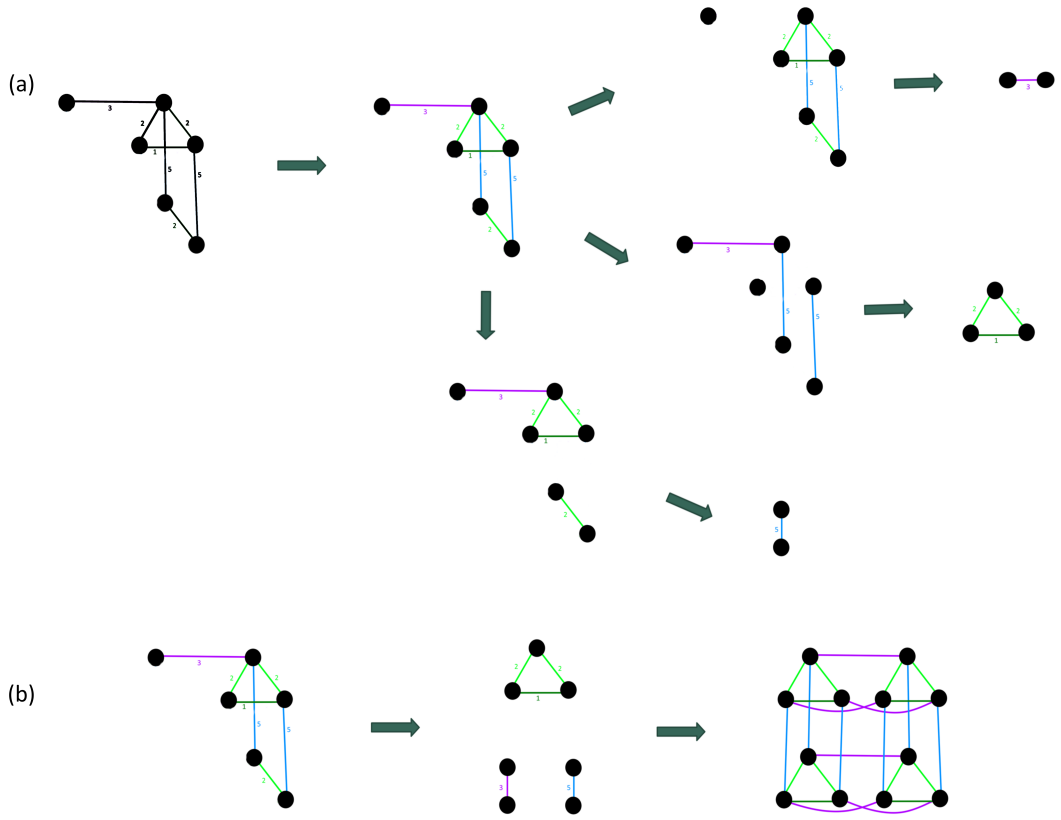

Figure 2-2: Subfigure (a) shows that, given an input graph, Algorithm 1 first finds the equivalence classes of $\hat{\theta}$ on the graph edges. The green, purple, and blue edges are each an equivalence class. The algorithm then removes each equivalence class individually and creates a graph from the connected components of the resulting graph. Subfigure (b) shows the irreducible pseudofactorization of an example graph, as well as the Cartesian product of the pseudofactors, of which the input graph is an isometric subgraph.

Algorithm 1 Algorithm for breaking up a graph over a relation

Input: A weighted graph $G = (V, E, w_G)$ and an equivalence relation R on the edges of G.

Output: A set $\mathbf{G}$=$\{G_1^*, G_2^*, ..., G_m^*\}$

```
Set G ← ∅
Find the set of equivalence classes of R, {E_1, E_2, ..., E_m}
Set E ← {E_1, E_2, ..., E_m}
for E_k ∈E do
    Let w_{G'_k} be w_G restricted to the edges in E(G)\E_k
    Find G'_k = (V(G), E(G)\E_k, w_{G'_k})
    Set C ← the set of connected components of G'_k
    Create a new graph G*_k
    Set V(G*_k) ← {a | C_a is a connected component of G'_k}
    Set E(G*_k) ← {ab | there is an edge in E_k between C_a and C_b}
    for C_a, C_b ∈C do
        if All edges between C_a and C_b the same weight, w_ab then
            Set w_{G*_k}(a, b) ← w_ab
        else
            Reject.
    Set G→ G∪{G*_k}
Return G.
```

Graham and Winkler further showed that an unweighted pseudofactorization is unique. Additionally, $\hat{\theta}$ is not the only relation that can be used with this algorithm to produce this pseudofactorization. In particular, Feder [11] expanded on this work by defining a new relation, $\hat{\theta}_T$ on the edges of a graph G, given a spanning tree T of G. In particular, he defined θ_T such that $uv\ \theta_T\ ab$ if and only if $\delta(uv, ab) \neq 0$ and at least one of uv and ab is in T. Letting $\hat{\theta}_T$ be the transitive closure of θ_T, he showed that Algorithm 1 on input $(G, \hat{\theta}_T)$ for an unweighted graph G also produces the irreducible pseudofactorization of G, noting that for this relation Algorithm 1 runs in $O(|V||E|)$ time due to the smaller time needed to find the equivalence classes. In a later section, we will discuss how Feder's algorithm may be modified to improve the runtime of pseudofactorization, but in this section we will focus on the application of Graham and Winkler's algorithm to weighted graphs.

In this section, we consider how to pseudofactor graphs. In order to do so, we actually restrict our attention to a certain subset of weighted graphs, which we deem

minimal graphs. This is formally defined in Definition 2.2.2, but informally a minimal graph is a graph without "extraneous edges," or edges that do not appear on shortest paths.

Definition 2.2.2. *A graph G is a **minimal graph** if and only if every edge in $E(G)$ forms a shortest path between its endpoints.*

We also note that for any graph, it is easy to find a minimal graph with the same distance metric by simply checking if the weight of an edge is equal to the distance between its endpoints. For this section, we assume that all graphs are minimal.

In this section, the overall goal is to prove that when the input to Algorithm 1 is a minimal weighted graph G and the relation $\hat{\theta}$, the output is an irreducible pseudofactorization of G. We note that our definition of pseudofactorization for weighted graphs actually came from looking at the kinds of outputs that came from the application of Graham and Winker's algorithm to weighted graphs. We found that the input graph was always an isometric subgraph of the Cartesian product of the output graphs and given the consistency of this fact with the original definition of pseudofactorization, we chose to use this as the definition for weighted graphs.

2.2.1 Testing irreducibility

We first show that if all edges in a graph are in the same equivalence class of $\hat{\theta}$, then the graph is irreducible. In the following section, we will show that Algorithm 1 with this relation as an input produces a pseudofactorization of the graph. Because this algorithm produces more than one graph (neither of which is K_1) if there is more than one equivalence class on the edges, we can use the results in this section and the next to conclude that checking the number of equivalence classes of $\hat{\theta}$ is a definitive check of irreducibility.

Lemma 2.2.1. *For $uv, xy \in E(G)$, if $uv\,\theta\,xy$, then for any pseudofactorization $\{G_1, \ldots, G_m\}$ of G with isometric embedding $\pi : V(G) \to V(\Pi_{i=1}^{m} G_i)$ of G into an isometric subgraph of the product of the pseudofactors, $\pi(u)\pi(v)$ and $\pi(x)\pi(y)$ must have parent edges in the same pseudofactor.*

Proof. Throughout this proof, we let $d := d_G$ and $d_i := d_{G_i}$. Say there exists a pseudofactorization $\{G_1, G_2, ..., G_m\}$ of G with such an embedding π such that $\pi(a) = (\pi_1(a), \pi_2(a), ..., \pi_m(a))$ for $a \in V(G)$. For simplicity, we let $\pi_i(a) = a_i$.

Assume for contradiction that this pseudofactorization is a pseudofactorization of G in which $\pi(x)\pi(y)$ and $\pi(u)\pi(v)$ have parent edges in different pseudofactors. Since xy and uv are edges, by the definition of the Cartesian product, we get that there is exactly one l such that $u_l \neq v_l$ and one j such that $x_j \neq y_j$. Since the two edges have parent edges in different pseudofactor graphs, we also get that $l \neq j$. (See Definition 1.2.2.)

Now we consider $[d(x,u) - d(x,v)] - [d(y,u) - d(y,v)]$. Since π is an isometric embedding with $\pi_i(x) = x_i$, we can rewrite this using the distance metric for $\Pi_i G_i^*$, which means it can be written as:

$$\sum_{i=1}^{m} [d_i(x_i,u_i) - d_i(x_i,v_i)] - [d_i(y_i,u_i) - d_i(y_i,v_i)].$$

Term i in this sum is 0 if $u_i = v_i$ or if $x_i = y_i$. However, since $l \neq j$, this means at least one of these equalities is true for every term in the sum, so xy and uv are not related by θ. From this, we get that if $xy\,\theta\,uv$, then $l = j$ and $\pi(x)\pi(y)$ and $\pi(u)\pi(v)$ must have parent edges in the same pseudofactor graph. □

We know that Algorithm 1 outputs one graph for each equivalence class of $\hat{\theta}$. Thus, from the preceding lemma, we get that if Algorithm 1 outputs a single graph, the input graph must be irreducible. In the following section, we will show that the output of this algorithm is necessarily a pseudofactorization of the input graph, which together with this proof implies that a graph is irreducible if and only if there is one equivalence class of $\hat{\theta}$ on its edges.

2.2.2 An algorithm for pseudofactorization

In this section, we will show that Algorithm 1 with $\hat{\theta}$ as the input relation can be used to pseudofactor a minimal weighted graph. In order to prove this fact, we divide

the proof into a series of lemmas, many of which have parallels to those we will use to prove factorization. Now, we have to show that this algorithm is well-defined for the inputs we are considering (a minimal graph and the relation $\hat{\theta}$), which we do with Lemma 2.2.2. Throughout this section, we assume the input graph is G and notation is used as it is in Algorithm 1.

Lemma 2.2.2. *If C_a, C_b are connected components in G'_k and there exists $x \in C_a, y \in C_b$ such that xy is an edge with weight w_{ab}, then for each $u \in C_a$, there exists at most one $v \in C_b$ such that uv forms an edge, and if it exists, the edge has weight w_{ab}.*

Proof. Throughout this proof, we take $d(\cdot, \cdot)$ to be the distance function on G. First, we show that if uv is an edge between C_a, C_b, then there cannot exist a distinct $v' \in C_b$ such that uv' is an edge. We will show this by contradiction. Assume that such a v' exists. Since v and v' are in the same connected component of G'_k, there is a path $Q = (v = q_0, q_1 \ldots, q_n = v')$ (represented in Figure 2-3(a)) consisting entirely of edges not in E_k (and thus not related to uv by θ), and we consider the sum

$$\sum_{i=1}^{n} [d(u, q_{i-1}) - d(u, q_i)] - [d(v, q_{i-1}) - d(v, q_i)] = 0.$$

By telescoping, this implies that:

$$[d(u, v) - d(u, v')] - [d(v, v) - d(v, v')] = d(u, v) + d(v, v') - d(u, v') = 0.$$

Since for $v \neq v'$, $d(v, v') > 0$, this gives us $d(u, v) < d(u, v')$. However, a symmetric analysis says $d(u, v') < d(u, v)$, so it's impossible that u has an edge to v and v'. This shows the first part of the lemma.

Now, we show that if uv is an edge from C_a to C_b, then it has weight w_{ab}, which will rely on the assumption that the graph in question is minimal. First, we define two paths. The first is $Q_a = (u = q_0^a, q_1^a, \ldots, q_n^a = x)$, which is a path of edges entirely in C_a. We will also have a path $Q_b = (v = q_0^b, q_1^b, \ldots, q_t^b = y)$, which will consist entirely of edges in C_b (represented in Figure 2-3(b)). We note that no pair of edges on either of these paths can be related to uv or to xy by θ, since the edges are not in

E_k. Using this fact, we get the following four sums.

$$\sum_{l=1}^{t}[d(u,q_{l-1}^b)-d(u,q_l^b)]-[d(v,q_{l-1}^b)-d(v,q_l^b)]=0$$

$$=[d(u,v)-d(u,y)]-[d(v,v)-d(v,y)]$$

$$=d(u,v)-d(u,y)+d(v,y)$$

$$=w_G(uv)+d(v,y)-d(u,y).$$

$$\sum_{l=1}^{n}[d(u,q_l^a)-d(u,q_{l-1}^a)]-[d(v,q_l^a),d(v,q_{l-1}^a)]=0$$

$$=[d(u,x)-d(u,u)]-[d(v,x)-d(v,u)]$$

$$=d(u,x)+d(v,u)-d(v,x)$$

$$=w_G(uv)+d(u,x)-d(v,x).$$

$$\sum_{l=1}^{n}[d(x,q_{l-1}^a)-d(x,q_l^a)]-[d(y,q_{l-1}^a),d(y,q_l^a)]=0$$

$$=[d(x,u)-d(x,x)]-[d(y,u)-d(y,x)]$$

$$=d(u,x)-d(y,u)+d(y,x)$$

$$=w_G(uv)+d(y,x)-d(y,u).$$

$$\sum_{l=1}^{t}[d(x,q_l^b)-d(x,q_{l-1}^b)]-[d(y,q_l^b)-d(y,q_{l-1}^b)]=0$$

$$=[d(x,y)-d(x,v)]-[d(y,y)-d(y,v)]$$

$$=d(x,y)-d(x,v)+d(v,y)$$

$$=w_G(uv)+d(v,y)-d(x,v).$$

From these equations, we get:

$$w_G(uv)=d(u,y)-d(v,y)$$

$$w_G(uv)=d(v,x)-d(u,x)$$

$$w_G(xy)=d(u,y)-d(u,x)$$

$$w_G(xy)=d(v,x)-d(v,y).$$

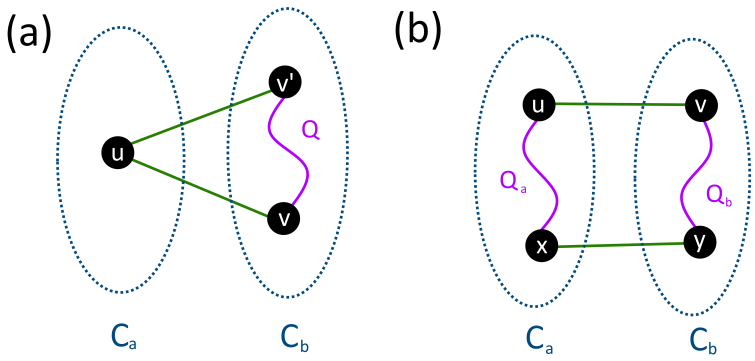

Figure 2-3: This figure gives illustrations of the paths described in the proof of Lemma 2.2.2. In particular, subfigure (a) shows a graph in which $u \in C_a$ has an edge to distinct $v, v' \in C_b$, which the proof of Lemma 2.2.2 shows is impossible. Subfigure (b) shows how Q_a and Q_b are defined in the proof when discussing $u \in C_a, v \in C_b$.

Subtracting the first and last of these equations gives us $w_G(uv) - w(xy) = d(u, y) - d(v, x)$ and subtracting the second and third equations gives $w_G(uv) - w_G(xy) = -[d(u, y) - d(v, x)]$. This gives us that $w_G(uv) - w_G(xy) = -[w_G(uv) - w_G(xy)]$, so the difference between these two weights is 0 and thus the weights are equal. This implies the lemma. □

With the next lemma, we introduce two general facts about the relationship between paths and equivalence classes of $\hat{\theta}$ that will be used in later claims.

Lemma 2.2.3. *The following hold:*

1. *If uv forms an edge and is in equivalence class E_k, then for any path $Q = (u = q_0, q_1, \ldots, q_t = v)$ between the two nodes, there is at least one edge from E_k.*

2. *Let $P = (u = p_0, p_1, \ldots, p_n = v)$ be a shortest path from u to v. If P contains an edge in the equivalence class E_k, then for any path $Q = (u = q_0, q_1, \ldots, q_t = v)$ there is at least one edge from E_k.*

Proof. First, we note that the second bullet implies the first in the case of unweighted and minimal graphs, since in those cases uv is a shortest path between u and v it is is an edge. However, this is not the case of general weighted graphs and we will use

this lemma when we discuss factorization of weighted graphs as well. Thus, we prove this lemma in two parts.

1. First, we consider the following sum.

$$\begin{aligned}\sum_{i=1}^{t}[d(u,q_{i-1})-d(u,q_i)]-[d(v,q_{i-1})-d(v,q_i)] &= [d(u,u)-d(u,v)]-[d(v,u)-d(v,v)]\\ &= -2d(u,v)\\ &\neq 0\end{aligned}$$

   The last inequality comes from the fact that $u \neq v$ and the assumption that the graph has only positive weight edges. However, we note that this sum is only non-zero if at least one term in the sum is non-zero. If term i is non-zero, then uv and $q_{i-1}q_i$ are related by θ and is thus in E_k. Thus, we prove the first part of the lemma.

2. Fix an edge $p_{\ell-1}p_\ell$ in P and consider the following sum.

$$\begin{aligned}&\sum_{i=1}^{t}[d(p_{\ell-1},q_{i-1})-d(p_{\ell-1},q_i)]-[d(p_\ell,p_{i-1})-d(p_\ell,q_i)]\\ &\qquad = [d(p_{\ell-1},u)-d(p_{\ell-1},v)]-[d(p_\ell,u)-d(p_\ell,v)]\end{aligned}$$

   We know that because P is a shortest path, $d(u,v) = d(u,p_{\ell-1}) + d(v,p_{\ell-1})$ and $d(u,v) = d(u,p_\ell) + d(v,p_\ell)$. Substituting, this makes the sum into:

$$d(u,v) - 2d(p_{\ell-1},v) - d(u,v) + 2d(p_\ell,v) = 2[d(p_\ell,v) - d(p_{\ell-1},v)].$$

   This value is only 0 if v is equidistant from p_ℓ and $p_{\ell-1}$, but since they form an edge on the shortest path between u and v, this is not possible and thus at least one edge on Q is related to $p_{\ell-1}p_\ell$ by θ and thus at least one edge in Q is in its equivalence class.

□

Now, we move on to the goal of showing that G is isomorphic to an isometric subgraph of $\Pi_i G_i^*$. To do so, we define a mapping π from G to $\Pi_i G_i^*$ with the goal to show that π is an injection and that shortest paths in G correspond to shortest paths under π. This will help us show our target property about isometric subgraphs.

First we will define π more formally. We will let $\pi : V(G) \to V(\Pi_i G_i^*)$ and will define π as: $\pi(u) = (\pi_1(u), \pi_2(u), \ldots, \pi_m(u))$. We then must define each π_i, so we let $\pi_i(u)$ be the node in G_i^* that corresponds to the connected component of G_i' that u is a member of. We begin by showing that π is an injection.

Claim 2.2.4. *As defined in this section, π is an injection.*

Proof. We show that $\pi(u) \neq \pi(v)$ for all $u, v \in V(G)$. To do so, let P be a shortest path between u and v. We know that if one of the edges is in E_k, then u and v are in different connected components of G_k' because Lemma 2.2.3 says that all paths between the two nodes have an edge in E_k. We know that if $u \neq v$ then there is at least one edge in P and thus there is at least one index k on which $\pi_k(u) \neq \pi_k(v)$. □

In many of our proofs, manipulation of the sum of theta-differences over a path (which we will call a theta-sum) is essential. We will now show an important property of that sum that we will use in our final proof. Informally, it says that for the theta-sum along a path, the contribution from the edges in each equivalence class does not depend on the path taken.

Lemma 2.2.5. *Let $P = (u = p_0, p_1, \ldots, p_n = v)$ and $Q = (u = q_0, q_1, \ldots, q_t = v)$ be two paths in G from u to v. Let P_k be the set of edges in P that are also in E_k and let Q_k be the same for Q. Define $T_k^P := \sum_{(p_i, p_{i+1}) \in P_k} [d(u, p_i) - d(u, p_{i+1})] - [d(v, p_i) - d(v, p_{i+1})]$. Define T_k^Q the same way for Q_k. Then, we claim that $T_k^P = T_k^Q$.*

Proof. We consider the following equations.

$$\begin{aligned}
T_k^P &= \sum_{p_ip_{i+1}\in P_k} [d(u,p_i) - d(v,p_i)] - [d(u,p_{i+1}) - d(v,p_{i+1})] \\
&= \sum_{p_ip_{i+1}\in P_k} \sum_{q_jq_{j+1}\in Q} [d(q_j,p_i) - d(q_{j+1},p_i)] - [d(q_j,p_{i+1}) - d(q_{j+1},p_{i+1})] \\
&= \sum_{p_ip_{i+1}\in P_k} \sum_{q_jq_{j+1}\in Q_k} [d(q_j,p_i) - d(q_{j+1},p_i)] - [d(q_j,p_{i+1}) - d(q_{j+1},p_{i+1})] \\
&= \sum_{q_jq_{j+1}\in Q_k} \sum_{p_ip_{i+1}\in P_k} [d(q_j,p_i) - d(q_{j+1},p_i)] - [d(q_j,p_{i+1}) - d(q_{j+1},p_{i+1})] \\
&= \sum_{q_jq_{j+1}\in Q_k} \sum_{p_ip_{i+1}\in P_k} [d(q_j,p_i) - d(q_j,p_{i+1})] - [d(q_{j+1},p_i) - d(q_{j+1},p_{i+1})] \\
&= \sum_{q_jq_{j+1}\in Q_k} \sum_{p_ip_{i+1}\in P} [d(q_j,p_i) - d(q_j,p_{i+1})] - [d(q_{j+1},p_i) - d(q_{j+1},p_{i+1})] \\
&= \sum_{q_jq_{j+1}\in Q_k} [d(q_j,u) - d(q_j,v)] - [d(q_{j+1},u) - d(q_{j+1},v)] \\
&= T_k^Q
\end{aligned}$$

The first equality comes from telescoping the inner sum, the second comes from the fact that only edges related by θ can contribute to the sum, so the only edges that might contribute are those in E_k. The third equality comes from switching the order of the sums, the fourth comes from switching the order of the terms in the summand, and the fifth again from the fact that only edges in E_k contribute. The sixth equality is by telescoping, and the last equality is by definition of T_k^Q. □

From here, we are able to prove our overall goal using Theorem 2.2.6

Theorem 2.2.6. *For any two nodes $u, v \in V(G)$, there is a shortest path between them in G that under π is a shortest path in $\Pi_i G_i^*$. Because G is minimal, this implies that π maps G to a subgraph of $\Pi_i G_i^*$ in which all edges in $E(G)$ are preserved under the map and the distance metric is preserved in the subgraph. Thus, for an input $(G, \hat{\theta})$, Algorithm 1 produces a pseudofactorization of G.*

Proof. First, we show that if P is a shortest path in G then the sequence of nodes in P under π does in fact form a path in $\Pi_i G_i^*$. We do so by showing that if there is

an edge $ab \in E(G)$, then there is an edge $\pi(a)\pi(b) \in E(\Pi_i G_i^*)$ and the edge has the same weight.

Assume $ab \in E_j$. First, let $C_{\pi_j(a)}$ and $C_{\pi_j(b)}$ be the connected components of G_j' corresponding to nodes $\pi_j(a)$ and $\pi_j(b)$ in G_j^*. We know that $E(G)$ has an edge between a node in $C_{\pi_j(a)}$ and a node in $C_{\pi_j(b)}$ of weight $w_G(ab)$ (by Lemma 2.2.2). By the same lemma, this means that all edges between nodes in $C_{\pi_j(a)}$ and nodes in $C_{\pi_j(b)}$ have the same weight and by the definition of G_j^*, there is an edge between $\pi_j(a)$ and $\pi_j(b)$ of that weight. This holds for all edges, which means that any path in G still exists in the image of π in $\Pi_i G_i^*$, so for any pair $u, v \in V(G)$, $d(u,v) \geq d^*(u,v)$, where $d^*(\cdot,\cdot)$ is the distance metric for $\Pi_i G_i^*$.

Now, assume the overall theorem does not hold. Let $u, v \in V(G)$ be the pair of nodes with the smallest value of $d^*(u,v)$ such that $d^*(\pi(u),\pi(v)) < d(u,v)$. (We know that such a pair must exist, by the fact that there must be some pair for which this distance differs and we know from the previous paragraph that we cannot have the distance be larger in the product than in the original graph.)

First, we show that any node on a shortest path from $\pi(u)$ to $\pi(v)$ in the product graph cannot be in the image of π. Consider a node $\pi(u')$ that is in the image of π and on such a shortest path. By the definition of shortest path, we get that $d^*(\pi(u),\pi(u'))+d^*(\pi(u'),\pi(v)) = d^*(\pi(u),\pi(v))$. Additionally, we have $d^*(\pi(u),\pi(u')) = d(u,u')$ because $d^*(\pi(u),\pi(u')) < d^*(\pi(u),\pi(v))$ and we get $d^*(\pi(u'),\pi(v)) = d(u',v)$ for parallel reasons. Then we get a contradiction:

$$\begin{aligned} d(u,v) &\leq d(u,u') + d(u',v) \\ &= d^*(\pi(u),\pi(u')) + d^*(\pi(u'),\pi(v)) \\ &= d^*(\pi(u),\pi(v)) \\ &< d(u,v). \end{aligned}$$

The first inequality is the triangle inequality and the last line is by the assumption. Since we have reached a contradiction, we know such a u' cannot exist and thus no node in the image of π can be on the shortest path between $\pi(u)$ and $\pi(v)$ in G^*.

We have shown that no node in the image of π can be on the shortest path between $\pi(u)$ and $\pi(v)$, but to get a contradiction, we will now show that such a node *must exist*.

Let uu' be the first edge on a shortest path from u to v and let the edge be in E_k. If P is the path in question, we show that P_k is a shortest path in G_k^*. As discussed in Section 1.2, this implies that uu' is an edge on a shortest path from u to v in $\Pi_i G_i^*$. Consider a path $Q^* = (\pi_k(u) = q_0, q_1, \ldots, q_n = \pi_k(v))$ between $\pi_k(u)$ and $\pi_k(v)$. Since the nodes in G_k^* are defined to be the labels of connected components in G_k' containing at least one node in G, for each j, there exists v_j such that $\pi_k(v_j) = q_j$ (i.e. there is a node in C_{q_j} in G_k'). We also have that because $q_j q_{j+1}$ forms an edge, there is a pair of nodes in G that has an edge between C_{q_j} and $C_{q_{j+1}}$. Let (x_j^j, x_{j+1}^j) be this pair. We have as a superscript the number of the edge we're considering and as a subscript the number of the connected component in G_k'.

We will construct the path Q from u to v in G shown in Figure 2-4(a). Let Q_j be a path from x_j^j to x_{j+1}^j that does not include any edges in E_k and let Q_0 be a path from u to x_0^1 and Q_n be a path from x_n^n to v. (Since each of the pairs is in the same connected component of G_k', these paths must exist.) Construct the path Q from u to v in G such that $Q := Q_0 + Q_1 + Q_2 + \cdots + Q_n = (u = f_0, f_1, \ldots, f_m = v)$. A visualization of this construction is given in Figure 2-4. This is a path from u to v and we consider the sum $T_k^Q = \sum_{f_j f_{j+1} \in Q_k} [d(v, f_j) - d(v, f_{j+1})] - [d(u, f_j) - d(u, f_{j+1})]$.

We know from Lemma 2.2.5 that $T_k^P = T_k^Q$. We use this to get

$$\begin{aligned} T_k^P &= T_k^Q \\ &= \sum_{q_i q_{i+1} \in Q_k} [d(v, q_i) - d(v, q_{i+1})] - [d(u, q_i) - d(u, q_{i+1})] \\ &\leq \sum_{q_i q_{i+1} \in Q_k} 2w(q_i q_{i+1}). \end{aligned}$$

Since P is a shortest path, every edge must contribute 2 times its weight to the overall theta-sum (or else we would not be able to get to the total) so we get $T_k^P = \sum_{p_i p_{i+1} \in P_k} 2w(p_i p_{i+1}) \leq \sum_{q_i q_{i+1} \in Q_k} 2w(q_i q_{i+1})$. Thus, the path P_k from u_k to v_k is a

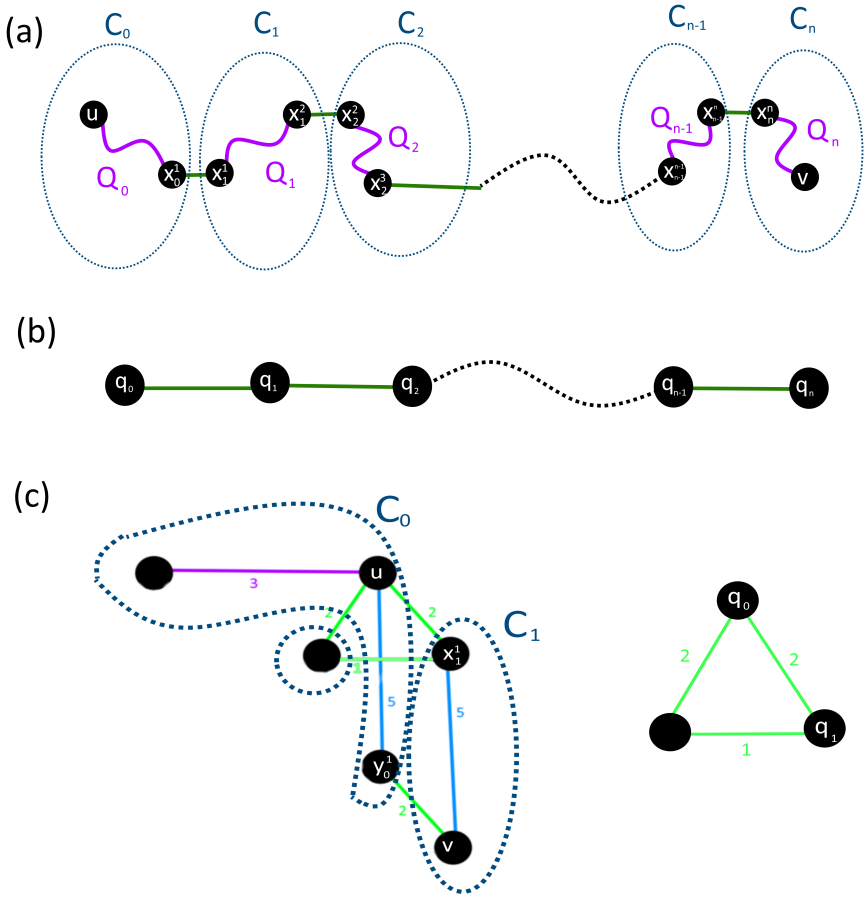

Figure 2-4: We have a visualization of the path constructed in the proof of Theorem 2.2.6.

In subfigure (a), each C_i represents a connected component in G'_k, where E_k is the equivalence class we're considering as in the theorem. Starting at u, we look for a path Q_0 through edges in C_0 to x_0^1, which is any node in C_0 with an edge to C_1. We then do the same process to find Q_1 from x_1^1 to x_1^2 and so on. The dotted line here represents an arbitrary length path that is not shown.

Subfigure (b) shows part of G_k^* for the graph in subfigure (a). We have $\pi_k(u) = q_0$ and $\pi_k(v) = q_n$, and the path $Q^* = (q_0, q_1, \ldots, q_n)$ is an arbitrary path from $\pi_k(u)$ to $\pi_k(v)$ through G_k^*. The theorem shows that because of how G_k^* is constructed, there must exist a path in G like the one shown in subfigure (a), where the only edges in E_k are the $x_{j-1}^j x_j^j$ and the weight of each $x_{j-1}^j x_j^j$ in G is the same weight as $q_{j-1}q_j$ in G_k^*.

Subfigure (c) shows an example of this process in the graph from Figure 2-2. In particular, we consider the pseudofactor shown on the right. When the graph on the left is isometrically embedded into the product of its pseudofactors (which is shown in Figure 2-2) with isometric embedding π, $\pi_k(u) = q_0$ and $\pi_k(v) = q_1$. We let $Q^* = (q_0, q_1)$. When constructing Q, we can select Q_0 to be (u) (so $x_0^1 = u$) and Q_1 to be (x_1^1, v). Alternatively, we could select Q_0 to be (u, y_0^1) and Q_1 to be (v) (so $v = y_1^1$). Thus, we see that while the path we construct in this theorem exists, it is not necessarily unique.

shortest path in G_k^*. This means $u_k u_k'$ is a first edge on a shortest path in G_k^*, which means $\pi(u)\pi(u')$ is a first edge on a shortest path in $\Pi_i G_i^*$ - a contradiction.

Thus, we can conclude $d(u,v) = d^*(\pi(u), \pi(v))$ and we already showed all edges are preserved, so G is an isometric subgraph of $\Pi_i G_i^*$. □

Using the previous theorem, we conclude that the proposed process does in fact produce graphs whose Cartesian product the original graph is an isometric subgraph of. Additionally, we can show that this is actually an irreducible pseudofactorization by showing that all of the produced pseudofactors are irreducible.

Lemma 2.2.7. *If a minimal graph G is the input graph to Algorithm 1 and $\hat{\theta}$ is the input relation, the output graphs are all irreducible.*

Proof. Using Lemma 2.2.1, we only have to show that the edges in each factor are all related by $\hat{\theta}$ (when evaluated on the factor itself). Assume we have two edges, $u_j v_j$ and $x_j y_j$ in G_j^* for some j that are not related by $\hat{\theta}$ in that graph. Then any edges in $\Pi_i G_i^*$ that have this edge as a parent edge are not related by $\hat{\theta}$. However, we know that there is an edge xy in G (and thus in $\Pi_i G_i^*$) from a node in C_{x_j} to a node in C_{y_j} and an edge uv from a node in C_{u_j} to one in C_{v_j} where $uv\ \hat{\theta}\ xy$. Since π is an isometric embedding, $\pi(x)\pi(y)\ \hat{\theta}\ \pi(u)\pi(v)$ and these edges have $x_j y_j$ and $u_j v_j$ as their respective parent edges in G_j^*. However, if d^* is the distance metric for $\Pi_i G_i^*$ and d_i is the metric for G_i^*, we know that:

$$\begin{aligned}
&[d^*(\pi(x),\pi(u)) - d^*(\pi(x),\pi(v))] - [d^*(\pi(y),\pi(u)) - d^*(\pi(u),\pi(v))] \\
&= \sum_{i=1}^{m} [d_i(\pi_i(x),\pi_i(u)) - d_i(\pi_i(x),\pi_i(v))] - [d_i(\pi_i(y),\pi_i(u)) - d_i(\pi_i(y),\pi_i(v))] \\
&= [d_j(\pi_j(x),\pi_j(u)) - d_j(\pi_j(x),\pi_j(v))] - [d_j(\pi_j(y),\pi_j(u)) - d_j(\pi_j(y),\pi_j(v))] \\
&= [d_j(x_j,u_j) - d_j(x_j,v_j)] - [d_j(y_j,u_j) - d_j(y_j,v_j)],
\end{aligned}$$

where the second to last equality is due to the fact that $\pi(x)\pi(y)$ is an edge with a parent in j and the last equality is due to how we defined x, y, u, v. Thus, we get that any pair of edges in G_j^* is related by $\hat{\theta}$. □

Thus, from Theorem 2.2.6 and Lemma 2.2.7, we conclude that Algorithm 1 with the input $(G, \hat{\theta})$ for a minimal weighted G produces an irreducible pseudofactorization of G.

2.3 Factorization

Feder [11] showed that Algorithm 1 could also be used to find the factorization of a graph. To do so, he defined a new relation, τ_1. For edges $uv, uv' \in E(G)$, $uv \ \tau_1 \ uv'$ if and only if there does not exist a 4-cycle containing edges uv and uv'. He further defined $(\theta \cup \tau_1)^*$ to be the transitive closure of $(\theta \cup \tau_1)$ and showed that for an unweighted graph G, Algorithm 1 on the input $(G, (\theta \cup \tau_1)^*)$ produces the prime factorization of G. We note that in this section, we are discussing a *general* weighted graph, not necessarily a minimal one.

2.3.1 A modified equivalence relation

For the purpose of graph factorization, we first define a property which we call the *square property* and which will be useful for factorization.

Definition 2.3.1. *Edges $uv, uv' \in E(G)$ satisfy the **square property** if there exists a vertex x such that $uvxv'$ forms a square (four-cycle) with $w_G(uv) = w_G(xv')$, $w_G(uv') = w_G(xv)$, $uv \,\theta\, xv'$, and $uv' \,\theta\, xv$. In other words, the two edges must make up two of the adjacent edges of at least one four-cycle in which the opposite edges have the same weight and are related by θ. This is depicted visually in Figure 2-5*

We now define two relations θ and τ as follows:

1. As before, two edges, $xy, uv \in E(G)$ are related by θ if and only if $[d_G(x,u) - d_G(x,v)] - [d_G(y,u) - d_G(y,v)] \neq 0$.

2. Two edges $uv, uv' \in E(G)$ are related by τ if and only if they do *not* satisfy the square property. If two edges do not share a common endpoint, they are not related by τ.

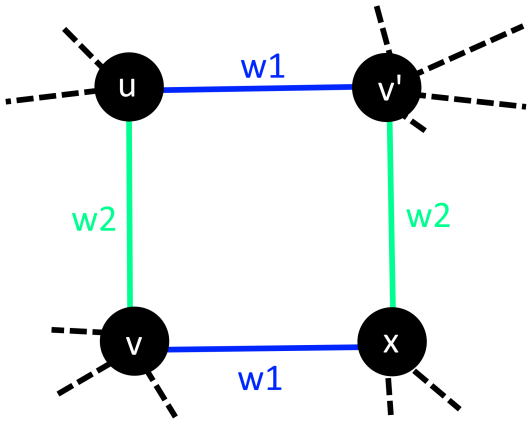

Figure 2-5: If $uv' \,\theta\, vx$ and $uv \,\theta\, v'x$, then existence of the subgraph depicted here implies that uv and uv' satisfy the square property.

We note that the θ relation used here is the same as that used by Graham and Winkler [13] and the τ relation is inspired by that used by Feder for factorization of unweighted graphs [11]. We also let $\hat{\theta}$ be the transitive closure of θ and $(\theta \cup \tau)^*$ be the transitive closure of $(\theta \cup \tau)$. Using these definitions, we prove that we can factor graphs using Algorithm 1.

2.3.2 Testing primality

Analogously to showing irreducibility with respect to pseudofactorization, in this section, we show that a graph is prime if its edges are in the same $(\theta \cup \tau)^*$ equivalence class. In particular, Lemma 2.3.1 proves this fact.

Lemma 2.3.1. *For $uv, xy \in E(G)$, if $uv \,\theta\, xy$ or $uv \,\tau\, xy$, then for any factorization of G, uv and xy must correspond to edges in the same factor. In other words, if α is an isomorphism from G to the product of the factors, $\alpha(u)\alpha(v)$ and $\alpha(x)\alpha(y)$ have parent edges in the same factor. From this, we get that if $uv \,(\theta \cup \tau)^*\, xy$ then uv and xy correspond to edges in the same factor.*

Proof. We will prove the lemma first for the θ relation and then for the τ relation. Throughout the proof of this lemma, we let $d := d_G$ and $d_i := d_{G_i}$. Say there exists a factorization $\{G_1, G_2, ..., G_m\}$ of G with isomorphism α such that $\alpha(a) =$

$(\alpha_1(a), \alpha_2(a), ..., \alpha_m(a))$ for $a \in V(G)$. For simplicity, we let $\alpha_i(a) = a_i$.

1. Consider $uv\,\theta\,xy$. Then by Lemma 2.2.1, we get that in any embedding into an isometric subgraph of the product graph, the images of these two edges have parents in the same factor. Since α is an isomorphism and thus is an isometric embedding into a subgraph of the product, this claim applies and we get that the parent edges of $\alpha(u)\alpha(v)$ and $\alpha(x)\alpha(y)$ must be in the same factor.

2. Without loss of generality, assume $j < l$ and consider $uv\,\tau\,uv'$, and we assume for contradiction that the above factorization of G is one in which $\alpha(u)\alpha(v)$ and $\alpha(u)\alpha(v')$ have parent edges in different factors. We get that there is exactly one l such that $u_l \neq v'_l$ and exactly one j such that $u_j \neq v_j$. Since they belong to different factor graphs, we have $l \neq j$.

   Now, consider the node $(u_1, \ldots, u_{j-1}, v_j, u_{j+1}, \ldots, u_{l-1}, v'_l, u_{l+1}, \ldots, u_m)$ in $\Pi_i G_i$. Since the vertex set of $\Pi_i G_i$ is the Cartesian product of the $V(G_i)$, this node must be in $\Pi_i G_i$ and since α is a bijection, there must exist $x \in V(G)$ such that $\alpha(x)$ equals this node. We also know that x must have an edge to v since $x_i = v_i$ for all $i \neq l$ and $v'_l v_l \in E(G_l)$. It also has an edge to v' since $x_i = v'_i$ for all $i \neq j$ and $v_j v'_j \in E(G_j)$.

   Thus, $uvxv'$ is a square. Additionally, we know that weights on opposite sides of the square are equal because they correspond to the same edge in the same factor graph. We can also show that opposite edges are related by θ. We will only show this for $uv\,\theta\,xv'$ and appeal to symmetry for the other argument. As before, we can rewrite $[d_G(u,x) - d_G(u,v')] - [d_G(v,x) - d_G(v,v')]$ as the sum:

$$\sum_{i=1}^{m} [d_{G_i}(u_i, x_i) - d_{G_i}(u_i, v'_i)] - [d_{G_i}(v_i, x_i) - d_{G_i}(v_i, v'_i)]$$

   Since u and v only differ on coordinate j, this becomes:

$$[d_j(u_j, x_j = v_j) - d_j(u_j, v'_j = u_j)] - [d_j(v_j, x_j = v_j) - d_j(v_j, v'_j = u_j)] = 2d_j(u_j, v_j) \neq 0.$$

The last inequality comes from the fact that $v_j \neq u_j$ and we assume that all distances are non-zero. Thus, $uv \,\theta\, xv'$ and by a symmetric argument $uv' \,\theta\, xv$. This means x is such that uv and uv' satisfy the square property, which means they cannot be related by τ.

Thus, if two edges are related by θ or by τ, then they correspond to edges in the same factor for any factorization of G. This property is preserved under the transitive closure, so if two edges are related by $(\theta \cup \tau)^*$ then they must correspond to edges in the same factor. □

In the following section, we will show that Algorithm 1 with $(\theta \cup \tau)^*$ as the input relation gives a prime factorization of the input graph. Since the algorithm outputs one graph for each equivalence class of the given relation, combined with Lemma 2.3.1, this tells us that a graph is prime if and only if all of its edges are in the same equivalence class of $(\theta \cup \tau)^*$.

2.3.3 An algorithm for factorization

In this section, we show that Algorithm 1 with inputs G and $(\theta \cup \tau)^*$ produces a prime factorization of G. To do so, we will prove a series of lemmas. Our first goal is to show that the algorithm in question is well-defined for general weighted graphs and with the input relation $(\theta \cup \tau)^*$. We use Lemma 2.3.2 to show this, and we note that it actually proves a stronger statement that we will continue to use later. Additionally, we note that throughout this section, we will refer to G'_k and G^*_k as they are used in the algorithm itself.

Lemma 2.3.2. *If C_a, C_b are connected components in G'_k and there exists $x \in C_a, y \in C_b$ such that xy is an edge with weight w_{ab}, then for each $u \in C_a$ there exists exactly one $v \in C_b$ such that uv forms an edge and that edge has weight w_{ab}.*

Proof. Throughout this proof, we take $d(\cdot, \cdot)$ to be the distance function on G. First, we have that if uv is an edge between C_a, C_b, then there cannot exist a distinct $v' \in C_b$

such that uv' is an edge. This proof is identical to that of Lemma 2.2.2 so we do not repeat it here.

Now we have that a node in C_a cannot have edges to more than one node in C_b, and we expand on that to show that each $u \in C_a$ has an edge to *at least one* node in C_b and that that edge has weight w_{ab}. Let $P(u)$ be a path from u to x that does not include any edges in E_k and has the smallest number of edges of all such paths. (Such a path must exist because u and x are in the same connected component of G'_k.)

In the base case, there are zero edges in P. In this case, we must have $u = x$. By the premise of the lemma, x has an edge of weight w_{ab} to $y \in C_b$, proving the inductive hypothesis.

Now, we assume that for all nodes in C_a with such a path P of n edges, the lemma holds. We let u be a node such that $P(u)$ has $n+1$ edges and let u' be the second node on this path. By definition, we know that $P(u')$ has n edges. By the inductive assumption, there exists $v' \in C_b$ such that $u'v' \in E(G)$ and with $w(u'v') = w_{ab}$. We know that uu' and $u'v'$ are not related by τ (or else uu' would be in E_k), so they must fulfill the square property. Let v be the node such that $uu'v'v$ is a square with opposite edges of equal weight and relatd by θ. Because $uu' \notin E_k$, we get $vv' \notin E_k$ and since $v' \in C_b$, we get that $v \in C_b$. This means uv is an edge between C_a and C_b with weight w_{ab}, proving the lemma. □

We now write Lemma 2.3.3, which is identical to Lemma 2.2.3, but now refers to the equivalence classes of our new relation. We note that the proof of this claim is identical to that of the original claim, so we do not repeat it here.

Lemma 2.3.3. *We have the following two facts about the equivalence classes of* $(\theta \cup \tau)^*$.

1. *If uv forms an edge and is in equivalence class E_k, then for any path $Q = (u = q_0, q_1, \ldots, q_t = v)$ between the two nodes, there is at least one edge from E_k.*

2. *Let $P = (u = p_0, p_1, \ldots, p_n = v)$ be a shortest path from u to v. If P contains an*

edge in the equivalence class E_k, then for any path $Q = (u = q_0, q_1, \ldots, q_t = v)$ there is at least one edge from E_k.

We now want to show that G is ismorphic to $\Pi_i G_i^*$, so we define an isomorphism $\alpha : V(G) \to V(\Pi_i G_i^*)$. For ease of notation, we let $\alpha(u) = (\alpha_1(u), \alpha_2(u), \ldots, \alpha_m(u)) = (u_1, u_2, \ldots, u_m)$. We define the function such that $\alpha_i(u)$ is the name of the connected component of G_i' that u is a member of. This is the same way we defined π in the previous section, but because we are using a new equivalence relation, we show that we now have an isomorphism. We now want to show that α is a bijection and that uv is an edge if and only if $\alpha(u)\alpha(v)$ is an edge (and that they have the same weight if so), which will show that α is an isomorphism. We use Lemma 2.3.4 to show the first part of this fact.

Lemma 2.3.4. *As defined in the preceding paragraph, α is a bijection.*

Proof. First, we show that α is an injection (i.e. $\alpha(u) \neq \alpha(v)$ for all $u, v \in V(G)$). This is identical to the proof that π is an injection in Lemma 2.2.4, but we reiterate it here using the terminology in this section. To show α is an injection, let P be a shortest path between u and v. We know that if one of the edges is in E_k, then u and v are in different connected components of G_k' because Lemma 2.3.3 says that all paths between the two nodes have an edge in E_k. We know that if $u \neq v$ then there is at least one edge in P and thus there is at least one index k on which $\alpha_k(u) \neq \alpha_k(v)$.

Now, we show that α is a surjection by showing that all nodes in $\Pi_i G_i$ are in the image of α. Assume for contradiction that there is at least one node in the product not in the image of α. Let $(u_1, u_2, \ldots, u_m)$ be one such node with an edge to a node $(u_1, \ldots, x_k, \ldots, u_m)$ in the image of α whose pre-image is x. Since the two nodes have an edge of some weight w_{xu} between them, we know that there is an edge between C_{u_k} and C_{x_k} of weight w_{xu} and Lemma 2.3.2 tells us that every node in C_{x_k} has an edge of that weight to exactly one node in C_{u_k}. Let u' be the node in C_{u_k} that x has an edge to in G. Because there is an edge between u' and x, we know that they appear in the same connected component for all G_i' with $i \neq k$. This tells us $\alpha_i(x) = \alpha_i(u')$

for all $i \neq k$ and thus $\alpha(u') = (u_1, u_2, \ldots, u_m)$, which means $(u_1, u_2, \ldots, u_m)$ is in the image of α. □

Finally, we use the next theorem to show that α is an isomorphism.

Theorem 2.3.5. *For $u, v \in V(G)$, $uv \in E(G) \iff \alpha(u)\alpha(v) \in E(\Pi_i G_i^*)$. If they do form edges, they have the same weight. Thus, Algorithm 1 on an input $(G, (\theta \cup \tau)^*)$ outputs a factorization of G.*

Proof. We divide this proof into two cases based on the number of indices i on which $\alpha_i(u) \neq \alpha_i(v)$. We note that there must be at least one such index, as α is a bijection.

1. Case 1: There is more than one index on which $\alpha_i(u) \neq \alpha_i(v)$. We know that there is no edge between $\alpha(u)$ and $\alpha(v)$ in this case, by definition of the Cartesian product. We also know that there are two G_i' in which u and v are in different connected components, which is impossible if there is an edge between them, as that edge is a path between them and it can only belong to one equivalence class. Thus, there is no edge between u and v either.

2. Case 2: There is exactly one k on which $\alpha_k(u) \neq \alpha_k(v)$. If there is no edge between $\alpha_k(u)$ and $\alpha_k(v)$ in G_k^*, then we know that there is no edge in G_k' between $C_{\alpha_k(u)}$ and $C_{\alpha_k(v)}$ so we can't have an edge between u and v (as that would create such an edge). If there is any edge between $\alpha_k(u)$ and $\alpha_k(v)$ in G_k^* of weight w_{uv} and by Lemma 2.3.2, we have that all nodes in $C_{\alpha_k(u)}$ have an edge to exactly one node in $C_{\alpha_k(v)}$. Let $v' \in C_{\alpha_k(v)}$ be the node that u has an edge to. Because they share an edge, $\alpha_i(v) = \alpha_i(u) = \alpha_i(v')$ for all $i \neq k$ so $\alpha(v) = \alpha(v')$. Since α is a bijection, this means $v = v'$ and thus uv is an edge of the same weight as $\alpha(u)\alpha(v)$.

The two parts of the proof together show the theorem. □

Theorem 2.3.5 shows that there is an isomorphism between the input graph to Algorithm 1 and the Cartesian product of the output graphs when the input relation is $(\theta \cup \tau)^*$, implying an $O(|E|^2)$ time factorization of G. We can also show that the output graphs themselves are prime. We do so using the following lemma.

Lemma 2.3.6. *If G is the input graph to Algorithm 1 with $(\theta \cup \tau)^*$, the output graphs are all prime.*

Proof. Using Lemma 2.3.1, we only have to show that the edges in each factor are all related by $(\theta \cup \tau)^*$ (when evaluated on the factor itself). We know that a factor G_i^* appears as an isometric subgraph of $\Pi_i G_i^*$ (and thus as an isometric subgraph of G) by definition of the Cartesian product. We know that all edges in this isometric subgraph are in the same equivalence class of $(\theta \cup \tau)^*$ because their pre-images under α are and α is an isomorphism. This implies their parent edges are all related by $(\theta \cup \tau)^*$ so G_i^* is prime. □

Using the above lemma and theorem, we see that the algorithm in question produces a prime factorization of the input graph.

2.3.4 Uniqueness of prime factorization

We additionally claim that for any graph G, there is at most one set $\{G_1, G_2, \ldots, G_m\}$ (up to graph isomorphisms) of graphs that form a prime factorization of G.

Theorem 2.3.7. *For a given weighted graph G, there is at most one set of weighted graphs $\{G_1, G_2, \ldots, G_m\}$ for which no graph is isomorphic to K_1 and the set forms a factorization of G.*

Proof. For contradiction, assume that there are two such sets, $\mathcal{G} = \{G_1, G_2, \ldots, G_m\}$ and $\mathcal{G}' = \{G'_1, G'_2, \ldots, G'_{m'}\}$. Let α and β be isomorphisms from G to the product of the graphs in $\mathcal{G}$ and $\mathcal{G}'$, respectively. We will define a mapping $f : \mathcal{G} \to \mathcal{G}'$ as follows. If there exists an edge $uv \in E(G)$ with $\alpha_i(u) \neq \alpha_i(v)$ and $\beta_j(u) \neq \beta_j(v)$, then $f(G_i) = G'_j$. We can now show that f is a well-defined bijection. First, we note that by Lemma 2.3.2, if two edges are related by $(\theta \cup \tau)^*$, then they must map to edges corresponding to the same factor in the factorization. Thus, if there exists $uv, xy \in E(G)$ such that uv and xy have their parent edges in factor i in $\mathcal{G}$, then those edges must also have their parent edges in the same factor in $\mathcal{G}'$, meaning G_i is

mapped to exactly one G'_j. The reverse reasoning also shows that each graph in $\mathcal{G}'$ is mapped to by exactly one graph in $\mathcal{G}$.

Now, we must show that for each $G_i \in \mathcal{G}$, $f(G_i)$ is isomorphic to G_i. We know that G_i appears as a subgraph of G, with all edges related by $(\theta \cup \tau)^*$. Because this subgraph must appear in any Cartesian product and all edges must be related by this relation, we know that it must appear as a subgraph of $f(G_i)$. The reverse reasoning implies that $f(G_i)$ and G_i are isomorphic. □

Thus, from this we get that Algorithm 1 with input $(G, (\theta \cup \tau)^*)$ for a general graph G produces a prime factorization of G.

2.4 Computing the transitive closure of a relation

In order to address runtime and prepare for the following section, we will discuss another view on how to compute the equivalence classes of the transitive closure of a relation R on the edges of a graph $G = (V, E)$ whose transitive closure is also symmetric and reflexive. To do this, we define a new graph $G_R = (V_R, E_R)$. The vertices of this graph are the edges of the original graph and there is an edge between two vertices if and only if the corresponding edges in the original graph are related by R. The connected components of the graph are then the equivalence classes of the original graph under the transitive closure of R. The time to compute the connected components can be found using BFS in $O(|V_R| + |E_R|)$ time. As an upper bound, we know that once this graph is computed, $|V_R| = |E|$ and $|E_R| = O(|V_R|^2) = O(|E|^2)$. Thus, computing the connected components with BFS takes at most $O(|E|^2)$ time. If further bounds can be placed on the number of edges in G_R, this time can be decreased further. Figure 2-6 shows an example of how G_θ is constructed for a given input graph.

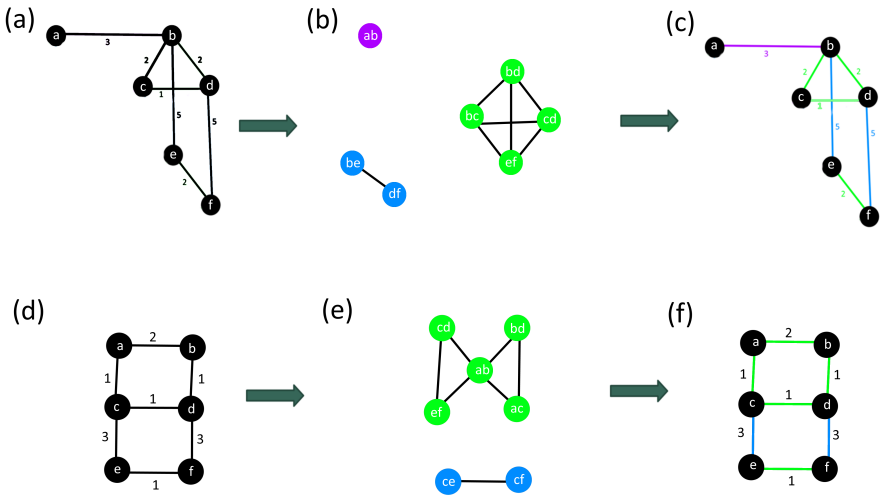

Figure 2-6: Subfigure (b) shows G_θ when when G is equal to the graph in subfigure (a). Subfigure (c) shows how G_θ determines the equivalence classes of the edges in G under $\hat{\theta}$, where edges of the same color are in the same equivalence class. Subfigure (e) shows G_θ when the input graph is that in subfigure (d) and subfigure (f) shows the equivalence classes of the edges in subfigure (d).

2.4.1 Runtime for pseudofactorization

To perform Graham and Winkler's algorithm, we compute the equivalence classes of $\hat{\theta}$, then for each equivalence class we perform linear time work by removing all edges in the equivalence class, computing the condensed graph, and checking which nodes in the new graph should have edges between them. In the worst case, each edge is in its own equivalence class, so we do $O(|V||E| + |E|^2) = O(|E|^2)$ work since the graph is connected. We additionally have to compute the equivalence classes of $\hat{\theta}$, which we can do by computing G_θ and finding the connected components. If we first find all pairs shortest path (APSP) distances, we can check if any pair of edges ab, xy is related by θ in $O(1)$ time. Thus, once we have found all of these distances, we can compute all edges from a given node in $O(|V_\theta|) = O(|E|)$ time, for a total of $O(|E|^2)$ time and then we take $O(|E|^2)$ time to compute the connected components. Thus, total runtime is $O(|E|^2)$ plus APSP computation time, which is currently known to be $O(|V|^2 \log\log|V| + |V||E|)$ [18] and thus gives us a runtime of $O(|V|^2 \log\log|V| + |E|^2)$ overall.

2.4.2 Runtime for factorization

For factorization using Algorithm 1, we can again bound the runtime of the main loop by $O(|E|^2)$ and thus just have to consider the time needed to compute the equivalence classes of $(\theta \cup \tau)^*$. We can first compute all distances using a known APSP algorithm. From here, we can determine all edges in $G_{\theta\cup\tau}$ that occur as a result of θ relations, and thus we only have to add in edges that occur as a result of τ relations. To do this, we can compare each pair of edges related by θ. If we have two edges of the format ab and cd that are related by θ and have the same edge weight, we can check if ac and bd exist as edges and have equal weights and an edge between them in G_θ. If so, we have that adjacent edges in this 4-cycle are not related by τ. From this, we can construct a graph on the edges in which there is an edge between every pair of edges that are not related by τ. To get the edges of $G_{\theta\cup\tau}$, then we simply have to add all edges from G_θ and edges between all pairs of vertices that do not have

an edge between them in this new graph. For a given pair of edges, this is constant time lookup in the graph G_θ, so it is $O(|E|^2)$ time to construct the new graph. Thus, it is $O(|E|^2)$ total time to construct $G_{\theta \cup \tau}$ and another $O(|E|^2)$ to get its connected components. This brings our total runtime for graph factorization up to $O(|E|^2)$ plus APSP time, which at the moment is $O(|V|^2 \log\log|V| + |E|^2)$ total.

2.5 Improved runtime for pseudofactorization

In the section on pseudofactorization, we showed that Graham and Winkler's algorithm on a weighted graph can be used to pseudofactor a minimal weighted graph. For a weighted graph, this algorithm takes $O(|E|^2)$ time plus the time to compute all pairs shortest paths, which at the moment is $O(|V|^2 \log\log|V| + |V||E|)$ [18]. Thus, this algorithm takes $O(|V|^2 \log\log|V| + |E|^2)$ time. For unweighted graphs, this time is just $O(|E|^2)$. Feder [11] showed that for unweighted graphs, rather than using the equivalence classes of $\hat{\theta}$, the same algorithm could use an alternative equivalence relation $\hat{\theta}_T$, which has the same equivalence classes but whose classes are faster to compute. In particular, θ_T is defined for a particular spanning tree T of the graph. Two edges ab, xy are related by θ_T if and only if $ab\ \theta\ xy$ and at least one of ab, xy is in the spanning tree T. As before, $\hat{\theta}_T$ is the transitive closure of θ_T. Feder showed that for an arbitrary tree T, this equivalence relation could be used in Graham and Winkler's algorithm to produce a psuedofactorization of an unweighted input graph. Because G_{θ_T} can be computed in $O(|V||E|)$ time and $\hat{\theta}_T$ can only have up to $|V|-1$ equivalence classes, this brings the total time for computing the pseudofactorization of an unweighted graph down to $O(|V||E|)$.

In the case of weighted graphs, we will show that we can find a tree T^* such that $\hat{\theta}_{T^*}$ has the same equivalence classes as $\hat{\theta}$. First, we reiterate proof that $\hat{\theta}_{T^*}$ is an equivalence relation by showing that $\hat{\theta}_T$ is an equivalence relation on the edges of G for any spanning tree T of G. (This is no different than for unweighted graphs, a proof of which can be found in [11], but we reiterate it here to emphasize that it still holds for weighted graphs.) First, since $\hat{\theta}_T$ is by definition transitive, we only

need to show reflexivity and symmetry. The relation is clearly symmetric because θ is symmetric. Take $ab \in E(G)$. Because T is a spanning tree, we know there is a path $P = (a = p_0, p_1, \ldots, p_n = b)$ from a to b that uses only edges in T. We get that $\sum_i [d(a,p_i) - d(a,p_{i-1})] - [d(b,p_i) - d(b,p_{i-1})] = [d(a,b) - d(a,a)] - [d(b,b) - d(b,a)] = 2d(a,b) \neq 0$, so there must exist $p_j p_{j-1}$ such that $ab\,\theta\,p_j p_{j-1}$, and since $p_j p_{j-1} \in T$, this means that $ab\ \theta_T\ p_j p_{j-1}\ \theta_T\ ab$ (where the last step is by symmetry). This means that $ab\ \hat{\theta}_T\ ab$ and thus the relation is reflexive as well. Thus, $\hat{\theta}_T$ is an equivalence relation for any T and we can discuss finding its equivalence classes. We also make the following claim about the equivalence classes:

Claim 2.5.1. *Let G be a graph with a spanning tree T and let $uv \in E(G)$ be such that $[uv]_{\hat{\theta}_T}$ is uv's equivalence class under $\hat{\theta}_T$ and $[uv]_{\hat{\theta}}$ is uv's equivalence class under $\hat{\theta}$. Then $[uv]_{\hat{\theta}_T} \subseteq [uv]_{\hat{\theta}}$.*

Proof. Take $xy \in [uv]_{\hat{\theta}_T}$. Since $uv\ \hat{\theta}_T\ xy$, there must exist a sequence of edges such that $uv = u_0v_0\,\theta_T\,u_1v_1\,\theta_T \cdots \theta_T\,u_nv_n = xy$. We have that $u_iv_i\,\theta_T\,u_{i+1}v_{i+1}$ if and only if $u_iv_i\,\theta\,u_{i+1}v_{i+1}$ and at least one edge is in the tree. Thus, we know that $uv = u_0v_0\,\theta\,u_1v_1\,\theta \cdots \theta\,u_nv_n = xy$, which means $uv\ \hat{\theta}\ xy$. Thus $xy \in [uv]_{\hat{\theta}}$. □

To find these equivalence classes, we will use Algorithm 2 to compute an appropriate tree T^*. Once we have such a tree and we have computed APSP, we can compute the equivalence classes of $\hat{\theta}_{T^*}$ in $O(|V||E|)$ time, as this only requires comparing each edge in T^* to each edge in the graph and computing if the two edges are related by θ, which can be done in constant time for each pair.

Informally, Algorithm 2 works by getting any spanning tree to start out and repeatedly modifying it to make the equivalence classes under the relation for the new spanning tree look more like the equivalence classes under $\hat{\theta}$. It does so by looking at a particular equivalence class for $\hat{\theta}_T$ and for each edge in that equivalence class, checking that every edge in the tree is in the current equivalence class or an already processed one, and swapping the edge into the tree if this does not hold. The idea behind this process is to try to grow the equivalence class we're working on as much as possible until it is the same as the equivalence class under $\hat{\theta}$.

Algorithm 2 Algorithm for breaking up a graph over a relation

Input: A weighted graph $G = (V, E, w_G)$ and all pairs of distances between nodes.
Output: A tree T^* such that the equivalence classes of $\hat{\theta}$ are the same as those of $\hat{\theta}_{T^*}$ on G.

```
Using BFS or DFS, find any spanning tree of G, which we will call T.
Compute G_{θ_T}
Set undiscovered ← V
Set current-class ← 0
while undiscovered is not empty do
    Pick xy ∈ undiscovered
    Run BFS to find all nodes reachable from xy in G_{θ_T}
    Mark all newly discovered edges with the number current-class in G
    Set reachable ← all nodes reachable from xy in G_{θ_T}
    while reachable is not empty do
        Pick ab ∈ reachable
        Find the path P from a to b in the tree T
        if there is an edge uv ∈ P that is not marked with a class number then
            Create a new spanning tree T' from T by adding ab and removing uv
            Set T ← T'
            Update G_{θ_T} based on the new T
            Update reachable to add any newly reachable nodes from xy in G_{θ_T}
            Mark any newly reachable edges with current-class
        Remove ab from reachable and from undiscovered
    current-class ← current-class +1
return T
```

We will first justify that this algorithm works correctly and then that it runs in $O(|V||E|)$ time. In particular, during the runtime section, we will go into more detail about how to update G_{θ_T} efficiently, but for now we take for granted that we update the graph correctly whenever needed.

2.5.1 Correctness of Algorithm 2

To prove that Algorithm 2 works correctly, we will show an invariant for the outer while loop.

Invariant: If at the top of the while loop, if an edge xy is not in the set *undiscovered*, then for the current tree T, $[xy]_{\hat{\theta}_T} = [xy]_{\hat{\theta}}$ where $[xy]_R$ is the equivalence class of xy over the equivalence relation R.

This invariant clearly holds in the base case, as in the first round of the while loop, all edges are in *undiscovered* and thus this is vacuously true.

Note that $[xy]_{\hat{\theta}_T} = [xy]_{\hat{\theta}}$ if and only if xy's connected component in G_{θ_T} includes the same nodes as that in G_θ. Thus, another way of phrasing the invariant is by saying that for any $xy \in V(G_{\theta_T})$ such that $xy \notin$ *undiscovered*, $C_{\theta_T}(xy)$ (xy's connected component in G_{θ_T}) has the same nodes as $C_\theta(xy)$ (xy's connected component in G_θ).

Consider a particular round of the while loop in which the edge we select from *undiscovered* is xy. Throughout the loop, we add and remove some edges from G_{θ_T} as we alter the tree. In particular, when we remove an edge uv, from T we may remove some edges that are adjacent to the node uv in G_{θ_T}. Removing edges from the tree T to create a new tree T' can potentially cause two edges, u_1v_1 and u_2v_2 that were related by θ_T not to be related by $\theta_{T'}$, but in the following lemma we restrict the kind of edges for which this may be true.

Lemma 2.5.2. *If an edge uv is removed from a spanning tree T and replaced with a new edge ab to create a new spanning tree T' and we have edges u_1v_1, u_2v_2 such that $u_1v_1 \ \theta_T \ u_2v_2$ but $u_1v_1 \ \not\theta_{T'} \ u_2v_2$, then one of u_1v_1 and u_2v_2 is equal to uv.*

Proof. We know that G_{θ_T} and $G_{\theta_{T'}}$ are identical to G_θ, but restricted to only keep edges adjacent to nodes that correspond to edges in T and T', respectively. Thus,

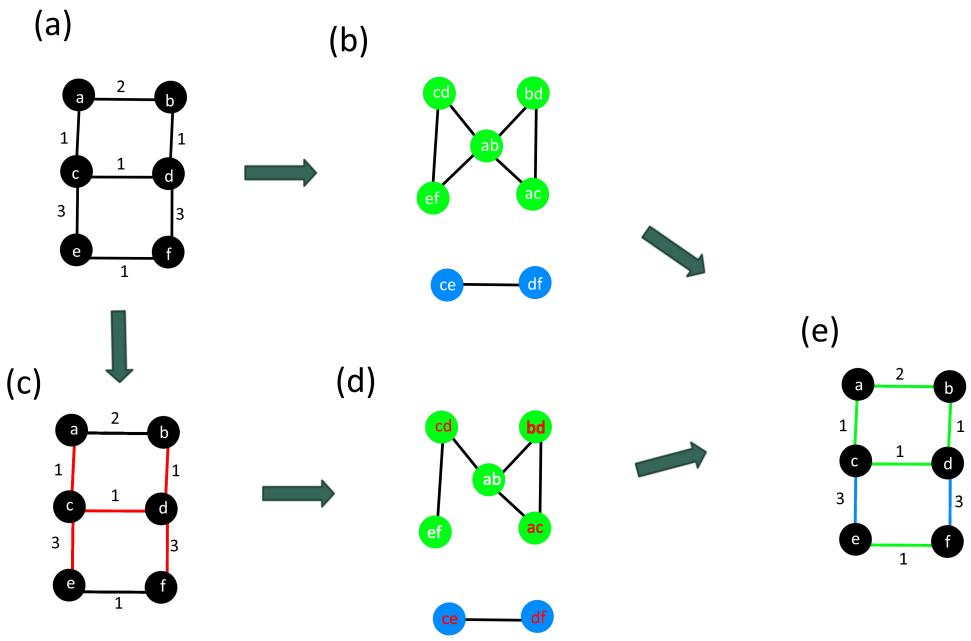

Figure 2-7: Subfigure (a) depicts an input graph G, and subfigure (b) shows G_θ. Subfigure (c) shows G with a spanning tree T (edges highlighted in red) and subfigure (d) shows G_{θ_T}. In particular, the nodes in subfigure (d) whose names are highlighted in red correspond to edges in G that exist in the spanning tree, and the graph is identical to that in G_θ but with edges that are not adjacent to a red node removed. Subfigure (e) shows that G_θ and G_{θ_T} produce the same equivalence classes for the edges of G.

by swapping out uv for ab, the only edges that may have been deleted from G_{θ_T} to form $G_{\theta_{T'}}$ are those adjacent to uv. Since edges between nodes in G_{θ_T} correspond to exactly the items that are related by θ_T, this means that the only way two edges can be related by θ_T but not by $\theta_{T'}$ is if one of them is uv. □

To follow up on the proof of Lemma 2.5.2, we can introduce a new picture of what G_{θ_T} looks like relative to G_θ, represented in Figure 2-7. In particular, if we take G_θ and color red all the nodes corresponding to edges in T, deleting the edges not adjacent to at least one red node produces G_{θ_T}. This helps provide a visual representation for the relationship between $\hat{\theta}$ and $\hat{\theta}_T$.

By the invariant, at the beginning of a particular iteration of the while loop, we assume that for all $uv \notin$ *undiscovered*, we have $C_{\theta_T}(uv) = C_\theta(uv)$. In the following lemma, we assert that at the end of the given iteration of the while loop, this will remain true for all edges that were discovered prior to this iteration.

Lemma 2.5.3. *If st is an node in G_{θ_T} that has been discovered when the spanning tree T is updated to a new tree T' by removing an edge uv and adding an edge ab, then $C_{\theta_T}(st) \subseteq C_{\theta_{T'}}(st)$*

Proof. By Lemma 2.5.2, when this update is done, the only edges that are removed from G_{θ_T} to create $G_{\theta_{T'}}$ are those that are adjacent to the node corresponding to uv. However, since st is discovered at this point, we know that all nodes reachable from st have also been discovered, as the algorithm updates knowledge about explored connected components every time it updates the graph. Since we never remove discovered edges from the tree, we know that uv is undiscovered, and thus the edges we removed in G_{θ_T} are not adjacent to anything in st's connected component, meaning that st's connected component in G_{θ_T} is a subset of that in $G_{\theta_{T'}}$ and we have $C_{\theta_T}(st) \subseteq C_{\theta_{T'}}(st)$. □

From the above lemma, we get the following.

Lemma 2.5.4. *If at any point in the algorithm we have a tree T such that $C_{\theta_T}(uv) = C_\theta(uv)$ for some edge uv, then we have $C_{\theta_{T^*}}(uv) = C_\theta(uv)$ where T^* is the final output tree.*

Proof. By contradiction. If $C_{\theta_{T^*}}(uv) \neq C_\theta(uv)$, then at some point after constructing T, the algorithm constructs a tree for which the equivalence class is not the same as under $\hat{\theta}$. Let the T' be the first tree the algorithm constructs after T for which $C_{\theta_{T'}}(uv) \neq C_\theta(uv)$. We know from Claim 2.5.1 $C_{\theta_{T'}}(uv) \subseteq C_\theta(uv)$. Let T'' be the tree constructed by the algorithm directly before T', so $C_{\theta_{T''}}(uv) = C_\theta(uv)$. By Lemma 2.5.3, we get that $C_{\theta_{T''}}(uv) \subseteq C_{\theta_{T'}}(uv)$. Thus, we have $C_\theta(uv) \subseteq C_{\theta_{T'}}(uv)$ and this gives us $C_{\theta_{T'}}(uv) = C_\theta(uv)$. □

Given Lemma 2.5.4, in order to show that every edge has the correct equivalence class under $\hat{\theta}_{T^*}$, we only need to show for each edge that at some point the algorithm constructs a tree T in which that edge has the correct equivalence class under $\hat{\theta}_T$. We will show that for a given uv, this occurs on an iteration of the while loop in which an edge $xy \in [uv]_{\hat{\theta}}$ is discovered. In particular, consider the first round of the while loop in which some $xy \in [uv]_{\hat{\theta}}$ is discovered. Because every node discovered in a given round of the while loop is related to the first discovered node by $\hat{\theta}$, We take xy to be the first edge discovered on this round of the while loop and show that at the end of the loop, if the tree held by the algorithm is T, then $[xy]_{\hat{\theta}_T} = [xy]_{\hat{\theta}}$. Because $uv \in [xy]_{\hat{\theta}}$, this implies $[uv]_{\hat{\theta}_T} = [uv]_{\hat{\theta}}$.

Thus, our current goal rests on showing that for each xy discovered at the beginning of an iteration of the outer while loop, if T is the tree at the end of that iteration of the while loop, $[xy]_{\hat{\theta}_T} = [xy]_{\hat{\theta}}$. We begin by considering an edge as "processed" after we have first discovered it and examined the path between its endpoints in the tree. We are able to make the following observation about the path between the endpoints of each processed edge.

Lemma 2.5.5. *If tree T is updated to tree T' by removing an edge uv from the graph and adding a new edge ab to the graph, then for any processed edge $a'b'$, the path from a' to b' through T' consists only of edges that we have already discovered/marked in the θ graph.*

Proof. We know that immediately after processing an edge $a'b'$, we updated the tree such that this lemma held. Since we never remove marked/discovered edges from the tree, this means that this path from a' to b' consisting of only marked edges still exists in the tree and since there is only one path from a' to b' through the tree, the lemma holds. □

Finally, using this lemma we are able to reach our final conclusion about xy's equivalence class, which by our earlier analysis tells us that at the end of the algorithm, the equivalence classes of $\hat{\theta}_{T^*}$ are those of $\hat{\theta}$, as desired.

Lemma 2.5.6. *If xy is an edge discovered at the beginning of a particular iteration of the outer while loop and T is the tree at the end of that iteration of the while loop, then $[xy]_{\hat{\theta}_T} = [xy]_{\hat{\theta}}$.*

Proof. From Lemma 2.5.5 and the fact that we process every edge in xy's connected component of G_{θ_T}, we know that at the end an iteration of the outer while loop, all edges in $C_{\theta_T}(xy)$ have paths through T that include only marked edges, which are edges that are in this connected component or some previously processed connected component in G_{θ_T}. Now, assume $[xy]_{\hat{\theta}_T} \neq [xy]_{\hat{\theta}}$. Since we know that $[xy]_{\hat{\theta}_T} \subseteq [xy]_{\hat{\theta}}$, this means that $[xy]_{\hat{\theta}_T} \subset [xy]_{\hat{\theta}}$. Pick $uv \in [xy]_{\hat{\theta}}, uv \notin [xy]_{\hat{\theta}_T}$. We know that because they're in the same $\hat{\theta}$ equivalence class, there is a sequence of edges such that $xy = u_1v_1\,\theta\,u_2v_2\,\theta\cdots\theta\,u_kv_k = uv$. If $u_iv_i\ \hat{\theta}_T\ u_{i+1}v_{i+1}$ for each i, then by transitivity we would have $xy\ \hat{\theta}_T\ uv$ and thus they'd be in the same equivalence class under $\hat{\theta}_T$. This means we can assume there exists j such that u_jv_j and $u_{j+1}v_{j+1}$ are not related by $\hat{\theta}_T$ but $u_jv_j \in [xy]_{\hat{\theta}_T}$. To simplify notation, we will call these two edges ab and st respectively. We have $ab\,\theta\,st$, but $ab \in [xy]_{\hat{\theta}_T}$ and $st \notin [xy]_{\hat{\theta}_T}$.

Consider the path $Q = (a = q_0, q_1, \ldots, q_n = b)$ from a to b through T. We get:

$$\sum_i [d(s,q_i) - d(s,q_{i+1})] - [d(t,q_i) - d(t,q_{i+1})] = [d(s,a) - d(s,b)] - [d(t,a) - d(t,b)]$$
$$\neq 0,$$

where the equality is by telescoping and the inequality is by the fact that $ab\,\theta\,st$. We know that this means $st\,\theta\,s^*t^*$ for some $s^*t^* \in Q$. Since $s^*t^* \in T$ and $st\,\theta\,s^*t^*$, we get $s^*t^* \in [st]_{\hat{\theta}_T}$. Thus, on this path there exists $s^*t^* \in [st]_{\hat{\theta}_T} \neq [xy]_{\hat{\theta}_T}$.

By our earlier lemma, we know that s^*t^* is marked and that every marked edge is either in xy's equivalence class or an equivalence class processed on a previous iteration of the while loop. Since we know s^*t^* is not in xy's equivalence class, it must be in an equivalence class we processed on a previous iteration of the while loop. However, our invariant tells us that this means $[s^*t^*]_{\hat{\theta}_T} = [s^*t^*]_{\hat{\theta}} = [xy]_{\hat{\theta}}$ (since $xy\,\theta\,s^*t^*$). However, since this equivalence class did not change over the course of this

iteration of the while loop, this means that xy was in an equivalence class we already processed before this iteration and thus it was marked, so it was not in *undiscovered*, a contradiction.

Thus, we know that at the end of the round, every edge not in *undiscovered* has $[xy]_{\hat{\theta}_T} = [xy]_{\hat{\theta}}$. Since we remove at least one edge from *undiscovered* in each iteration of the while loop, the loop terminates with everything popped and we get that all equivalence classes of $\hat{\theta}_T$ are the same as those of $\hat{\theta}$. Thus, we have shown that if the invariant holds at the beginning of an iteration of the outer while loop, it holds at the end. □

From here, we analyze the runtime of Algorithm 2 to determine if using it to find a new equivalence relation on the edges of the graph actually improves our runtime.

2.5.2 Runtime of Algorithm 2

For a graph $G = (V, E, w_G)$, finding some spanning tree T takes $O(|V| + |E|)$ time. When we compute G_{θ_T}, we know that every edge must be adjacent to some $uv \in T$, as if two edges in G are related by θ_T, then at least one of them must be in T. Thus, if we already have the APSP distances and can thus check if two edges are related by θ in constant time, it takes us $O(|V||E|)$ time to construct G_{θ_T}, as we compare each edge in T to every other edge in order to compute the edges of G_{θ_T}.

We notice that each time we look for the path from a to b in the tree, we pop ab from *unreachable*, and we never do this for a node not in *unreachable*, so we do this once for each edge. As it takes $O(|V|)$ time to find this path and we do it for $|E|$ edges, this is $O(|V||E|)$ total time for examining the paths in the tree. Because each edge is marked in the tree when it is discovered in G_{θ_T}, we can immediately check if any edge is marked since we mark it in the original graph when it is discovered in G_{θ_T} which we can do by adding pointers from one graph to the other. This leaves us just considering the total BFS time for all of the updates.

Consider a single iteration of the outer while loop, in which we pop edge xy. We begin by discovering all edges that xy can reach, each of which is popped from

undiscovered. At this point, if we add a new edge ab to the tree, we need to update *reachable*, as we have potentially added new edges adjacent to ab and thus potentially added new nodes to xy's connected component. At this point, we can resume BFS beginning at ab, visiting only nodes that we have not seen before. We note that this means we may consider the edges adjacent to ab a second time, but all other edges we traverse on this BFS run are new edges, as the origin of each edge must be a newly discovered node.

When we add a new edge ab into the tree, we must determine all edges out of ab in the new G_{θ_T}, which requires checking if ab is related by θ to all other edges in the graph. We also remove an edge from the tree, which means we must remove all of its edges to other edges not in the tree in G_{θ_T}. Each of these updates takes $O(|E|)$ time, so we must limit the number of times we do this. We note that no edge discovered in G_{θ_T} is ever removed from the graph, and at the point an edge ab is added to the tree, it has been discovered in G_{θ_T} and thus will never be removed. This means that we can add at most $|V| - 1$ edges to the tree. This means the updates to G_{θ_T} take at most $O(|V||E|)$ time total across all updates.

We know from our earlier analysis that when a new edge is added to the tree, we end up re-traversing at most $O(|E|)$ edges, and since this happens at most $|V| - 1$ times, re-traversing edges in G_{θ_T} happens at most $O(|V||E|)$ times. Aside from that, we do a full BFS search of the graph G_{θ_T}, which does have edges added and removed as we go along. However, we note that it has only $O(|V||E|)$ edges at the beginning and as we have just argued, only $O(|V||E|)$ new edges are added, so even if we manage to traverse every edge that appeared in any iteration of the graph (which actually doesn't happen since removed edges are connected to undiscovered nodes, which means they haven't been traversed), we take only $O(|V||E|)$ time to do so.

Thus, the total time for this algorithm is $O(|V||E|)$. We also note that because every equivalence class must contain an edge from T^*, there are most $|V| - 1$ equivalence classes for this relation. Thus, using this relation with Algorithm 1, the primary *for* loop is only traversed $O(|V|)$ times. Since Algorithm 2 both provides a tree to T^* to use for $\hat{\theta}_{T^*}$ and computes the relation's equivalence classes in $O(|V||E|)$ time given

the APSP distances, total time for pseudofactoring a graph using the equivalence relation $\hat{\theta}_T$ for T produced by Algorithm 2 takes only $O(|V||E|)$ plus APSP time, as opposed to $O(|E|^2)$ plus APSP time for the same algorithm with $\hat{\theta}$ as the input relation. Using current APSP algorithms, this is a speedup from $O(|V|^2 \log\log|V| + |E|^2)$ to $O(|V|^2 \log\log|V| + |V||E|)$ for overall runtime.

Chapter 3

Hypercube embeddings and general distance metrics

In this chapter, we discuss some applications and extensions of graph pseudofactorization. In particular, we will discuss when pseudofactoring a graph may inform us about its hypercube or Hamming embeddability and ways in which the concepts may be extended to certain types of decompositions of general distance metrics.

3.1 Background and notation for hypercube embeddings

Given a graph G, a Hamming embedding $\eta : V(G) \to \Sigma^t$ for some positive integer t and alphabet Σ is a mapping of the vertices of the graph to strings of fixed length such that the Hamming distance between two strings is the graph distance between the corresponding nodes. In other words, $d_H(\eta(u), \eta(v)) = d_G(u, v)$, where $\eta(u) = \eta_1(u)\eta_2(u)...\eta_t(u)$ is a string over the alphabet Σ and d_H is Hamming distance. A hypercube embedding is exactly a Hamming embedding with $\Sigma = \{0, 1\}$. Djoković showed that deciding whether or not a hypercube embedding exists for an unweighted graph can be done in polynomial time [10] and Winkler showed that deciding if an unweighted graph has a Hamming embedding can also be done in polynomial time

[24]. In fact, a Hamming embedding exists for an unweighted graph G if and only if its canonical pseudofactors are all complete graphs, and it is hypercube embeddable if and only if they are all isomorphic to K_2.

In some cases, it is useful to generalize the notion of hypercube embeddings to the more general concepts of scaled hypercube embeddings and ℓ_1 embeddings. To do so, we first define scaled hypercube embeddings. A *scale-k* hypercube embedding of an unweighted graph G is a hypercube embedding of kG, where kG is a version of G in which all edges have weight k. From there, we can consider the idea of an ℓ_1 metric. For a given dimension t, define $d_{\ell_1} : \mathbb{R}^t \times \mathbb{R}^t \to \mathbb{R}_{\geq 0}$ such that for $x, y \in \mathbb{R}^t$, $d_{\ell_1}(x, y) = \sum_{i=1}^{t} |x_i - y_i|$. A graph G has an ℓ_1 metric if and only if there is a map $\alpha : V(G) \to \mathbb{R}^t$ such that $d_G(u, v) = d_{\ell_1}(\alpha(u), \alpha(v))$. Such an embedding is an ℓ_1 embedding, and a graph G has one if and only if there is a k such that kG is hypercube embeddable [7]. Thus, deciding if a graph G has an ℓ_1 metric is equivalent to deciding if there exists an integer k such that kG is hypercube embeddable. In the case of bipartite graphs, a graph has ℓ_1 embedding if and only if it is hypercube embeddable at scale 1 [10, 20]. Firsov was the first to show that there exists a graph G for which kG is never hypercube embeddable [12]. This is equivalent to saying that some unweighted graphs do not have ℓ_1 metrics.

Shpectorov showed that for a graph with uniform weights, it can be decided in polynomial time if the graph is hypercube embeddable [22]. With Deza, he showed that this determination could be made in as little as $O(|V||E|)$ time [6]. Shpectorov additionally showed that if an unweighted graph G is an ℓ_1 metric, there is a scaled hypercube embedding whose image has size polynomial in the size of G [22]. However, while it takes only polynomial time to decide if an unweighted (or uniformly weighted graph) is an ℓ_1 metric, the same decision for a weighted graph is NP-hard, even when the edge weights are restricted to positive integers [16, 1]. It is additionally NP-hard to decide if an undirected graph with integer weights is hypercube embeddable [4, 7], although under particular restrictions on distances, it can be decided in polynomial time [17]. We mention these concepts of ℓ_1 embeddability and scaled hypercube embeddability in order to provide context as we discuss the relationship between

hypercube embeddings and pseudofactorization.

3.2 Hypercube embeddings and pseudofactorization

In this section, we will show that a graph has a hypercube embedding if and only if every one of its pseudofactors has a hypercube embedding. Much of this is the direct result of Lemma 3.2.1, and in this section we seek to elaborate on why this is interesting and how we come to this conclusion. The following lemma is the result of a personal communication by Joseph Berleant [2].

Lemma 3.2.1. *Given a graph $G = (V, E, w_G)$ and a hypercube embedding $\eta : V \to \{0,1\}^s$, define D_η on $V \times V$ such that $D_\eta(u,v) = \{i \mid \eta_i(u) \neq \eta_i(v)\}$. Define a relation γ on the integers from 0 to $t-1$ such that $i \ \gamma \ j$ if there exist $u, v \in V$ such that $i, j \in D_\eta(u,v)$. Let $\hat{\gamma}$ be the transitive closure of γ.*

Then for $uv, u'v' \in E$ and $j \in D_\eta(u,v), j' \in D_\eta(u',v')$, we have $j \ \hat{\gamma} \ j'$ if and only if $uv \ \hat{\theta} \ u'v'$.

From the given lemma, we can see that for a given hypercube embedding $\eta : V \to \{0,1\}^s$ of a graph $G = (V, E, w_G)$, the equivalence classes of $\hat{\theta}$ are in one-to-one correspondence with the equivalence classes of $\hat{\gamma}$. In particular, if $[j]_{\hat{\gamma}}$ is equivalence class of index j under $\hat{\gamma}$ and $[uv]_{\hat{\theta}}$ is the equivalence class of uv under $\hat{\theta}$, we can define a bijection β such $\beta : [j]_{\hat{\gamma}} \mapsto [uv]_{\hat{\theta}}$ where $j \in D_\eta(u,v)$. First we show that this is a bijection.

Lemma 3.2.2. *As defined in this section, β is a well-defined bijection.*

Proof. This follows directly form Lemma 3.2.1. To see that it is well-defined, we see that if $j' \in [j]_{\hat{\gamma}}$ and $j \in D_\eta(u,v)$, then for any $xy \in E$ such that $j' \in D_\eta(x,y)$, we have $xy \ \hat{\theta} \ uv$ and thus $[xy]_{\hat{\theta}} = [uv]_{\hat{\theta}}$.

To see that β is surjective, we see that for any $uv \in E$, there must be at least one index j on which $\alpha_j(u) \neq \alpha_j(v)$, so $[uv]_{\hat{\theta}}$ is mapped to by some equivalence class of $\hat{\gamma}$. To see that it is injective, we consider j, j' such that $j \not\hat{\gamma} \ j'$. If $\beta([j]_{\hat{\gamma}}) = [uv]_{\hat{\theta}} = \beta([j']_{\hat{\gamma}})$, then we must have $uv, xy \in [uv]_{\hat{\theta}}$ such that $j \in D_\eta(u,v)$ and $j' \in D_\eta(x,y)$,

but by Lemma 3.2.1, this implies $j \;\hat{\gamma}\; j'$ so this is impossible and the function is bijective. □

To prepare for our overall claim about the relationship between pseudofactorization and hypercube embeddings, we first prove the following.

Claim 3.2.3. *Let $P = (p^0, p^1, \ldots, p^n)$ be a shortest path in a product graph $\Pi_{i=1}^m G_i$, where $p^j = (p_1^j, p_2^j, \ldots, p_m^n)$. Then let the "projection" of P onto G_i be $(p_i^0, p_i^1, \ldots, p_i^n)$. The projection of P onto G_i is a shortest path from p_1 in G_i.*

Proof. First, we note that the projection $(p_i^0, p_i^1, \ldots, p_i^n)$ of P onto G_i may have adjacent vertices that are the same, in which case the path is the same if we remove these adjacent repeats (as the total weight of the path is the sum of the edge weights and there is no edge traversed to get from a node to itself). Clearly, this path is no shorter than a shortest path in the factor graph. Say this is not a shortest path. We use it to construct a shorter path in $\Pi_{l=1}^m G_l$. In particular, we know that by how the Cartesian product is constructed, there is a path from p^0 to p^n whose weight is equal to $\sum_{l=1}^m d_{G_l}(u_l, v_l)$. We know that this means $w(P) = \sum_i w(P_i) = \sum_{l=1}^m d_{G_l}(u_l, v_l)$ where $w(P_i)$ is the weight of the projection onto G_i. Since $w(P_i) \geq d_{G_i}(u_i, v_i)$, in order to get equality here we must have $w(P_i) = d_{G_i}(u_i, v_i)$ for all i. □

Using this fact, we prove the following theorem.

Theorem 3.2.4. *A graph $G = (V, E, w_G)$ is hypercube embeddable if and only if all of its pseudofactors are hypercube embeddable.*

Proof. First, we show that if all pseudofactors $\{G_1, \ldots, G_m\}$ of a graph G are hypercube embeddable, then G is hypercube embeddable. We know that there must exist an isometric embedding $\pi : V(G) \to V(\Pi_{i=1}^m G_i)$ such that $\pi(u) = (u_1, \ldots, u_m)$ for $u \in V(G), u_i \in V(G_i)$ for $1 \leq i \leq m$. Let $\eta^i : V(G_i) \to \{0,1\}^{s_i}$ be a hypercube embedding for G_i. Define a hypercube embedding η for G such that $\eta(u)$ is the concatenation of the embeddings for each of the u_i, or $\eta(u) = \eta^1(u_1)\eta^2(u_2)\ldots\eta^m(u_m)$.

Then for $u, v \in V(G)$, we have

$$\begin{aligned} d_H(\eta(u), \eta(v)) &= \sum_{i=1}^{m} d_H(\eta^i(u_i), \eta^i(v_i)) \\ &= \sum_{i=1}^{m} d_{G_i}(u_i, v_i) \\ &= d_G(u, v), \end{aligned}$$

where the second equality is from the fact that η^i is a hypercube embedding of G_i and the third equality is from the fact that distance between $\pi(u)$ and $\pi(v)$ in $\Pi_{i=1}^m G_i$ is equal to the sum of the distances between each u_i and v_i in G_i and the fact that α is an isometric embedding, so $d_G = d_{\Pi_{i=1}^m G_i}$. Thus, the η defined here is a hypercube embedding for G and we have that if a graph's pseudofactors are all hypercube embeddable, then so is the graph as whole.

Now we consider a graph $G = (V, E, w_G)$ with hypercube embedding η and pseudofactors $\{G_1, G_2, \ldots, G_m\}$. First, by the definition of pseudofactorization, we know there must exist an isometric embedding $\pi : V(G) \to V(\Pi_{i=1}^m G_i)$ such that $\pi(u) = (u_1, u_2, \ldots, u_k)$ and for each $x \in V(G_i)$, there exists u such that $u_i = x$. For a given pseudofactor G_i, we define a hypercube embedding η^i. For $u_i \in V(G_i)$, select a node u such that $\alpha_i(u) = u_i$, and we define $\eta^i(u_i)$ to be equal to $\eta(u)$ restricted to the indices in $\beta^{-1}([xy]_{\hat{\theta}})$ where xy is an edge with a parent in G_i.

We must show that η^i is in fact a hypercube embedding for G_i and that it is well-defined. In order to show this, we consider $u_i, v_i \in V(G_i)$ such that we define $\eta^i(u_i)$ and $\eta^i(v_i)$ based on $\eta(u)$ and $\eta(v)$ for $u, v \in V(G)$, respectively. Let P_i a shortest path from u_i to v_i in G_i. We show that $d_H(\eta^i(u_i), \eta^i(v_i)) = d_{G_i}(u_i, v_i)$.

Pick a shortest path from u to v in G, $P = (u = p_0, p_1, \ldots, p_n = v)$ and let P_i be the set of edges in P_i that are in E_i. We know that $\eta(p_i)$ and $\eta(p_{i+1})$ differ on exactly $w(p_i p_{i+1})$ indices because this edge is a shortest path between them. Take an index j such that $j \notin D_\eta(p_i, p_{i+1})$ for each i. Then we claim that $j \notin D_\eta(u, v)$. In particular, this means that for each i, $\eta_j(p_i) = \eta_j(p_{i+1})$, which transitively means $\eta(u) = \eta(v)$. Thus, we know that $\eta(u)$ and $\eta(v)$ can only differ on indices in $\cup_i D_\eta(p_i, p_{i+1})$, but we

also see that $\sum_{i=0}^{n-1} |D_\eta(p_i, p_{i+1})| = \sum_{i=0}^{n-1} w(p_i p_{i+1}) = w(P) = d_G(u, v)$, so $\eta(u), \eta(v)$ must differ exactly the indices in $\cup_i D_\eta(p_i, p_{i+1})$ (and we get that they must all be disjoint). When we define $\eta^i(u_i)$ and $\eta^i(v_i)$, we restrict $\eta(u)$ and $\eta(v)$ to the indices j on which some edge xy in $\hat{\theta}$ equivalence class i has $\eta_j(x) \neq \eta_j(y)$. This means that $\eta^i(u_i)$ and $\eta^i(v_i)$ differ on exactly the indices in $\cup_{p_l p_{l+1} \in [xy]_{\hat{\theta}}} D_\eta(p_l, p_{l+1})$, which implies that they differ on $w(P_i)$ indices and we have $d_H(\eta^i(u_i), \eta^i(v_i)) = w(P_i)$.

We now show that $w(P_i)$ equals $d_{G_i}(u_i, v_i)$. We note that because of how π is defined, $\pi(P) = (\pi(u) = \pi(p_0), \pi(p_1), \ldots, \pi(p_n) = \pi(v))$ is a shortest path in $\Pi_{i=1}^m G_i$, each edge with the same weight as in P. From Claim 3.2.3, we know that if $\pi(P)_i$ is the set of edges in equivalence class i of $\hat{\theta}$ of $\pi(P)$, then the weight of this path is equal to the weight of a shortest path from $\pi_i(p_0) = u_i$ to $\pi_i(p_n) = v_i$ in G_i. Thus, we get that $w(P_i) = w(\pi(P)_i) = d_{G_i}(u_i, v_i)$ and tying it into the previous paragraph, we get that $d_H(\eta^i(u_i), \eta^i(v_i)) = d_{G_i}(u_i, v_i)$. This was independent of the choice of u, v such that $\pi_i(u) = u_i$ and $\pi_i(v) = v_i$. Thus, we have found that if G is hypercube embeddable, its pseudofactors are and this completes the proof. □

I additionally note, that in the personal communication [2], I was also informed that Lemma 3.2.1 can be extended to replace hypercube embedding with Hamming embedding at every point. The proofs given here can also be extended to replace hypercube embedding with Hamming embedding at every point, so we also get the following lemma:

Lemma 3.2.5. *A graph $G = (V, E, w_G)$ is Hamming embeddable if and only if all of its pseudofactors are Hamming embeddable.*

While it is NP-hard to decide if general weighted graphs are hypercube embeddable, some authors have shown that this can be decided in polynomial time for certain classes of weighted graphs. In particular, if a graph is a line graph or a cycle graph [5] or if a graph's distances are all in $\{x, y, x + y\}$ for integers x, y, at least one of which is odd [17], it is polynomial time decidable. Shpectorov's results also tell us that for graphs with uniform weights, we can decide this in polynomial time [22]. This result extends these results by showing that it also takes only polynomial time to

decide if isometric subgraphs of the Cartesian products of such graphs are hypercube embeddable (i.e. it is polynomial time to decide for graphs whose pseudofactors fit into one of these categories). The number of graphs that fit into these categories is very restricted, but if future categories of graphs are found to have polynomial time decidability for this property, this result will apply to extend such a result to the isometric subgraphs of Cartesian product of such graphs with each other and with other graphs in this category.

3.3 Minimum graphs and general distance metrics

In the previous sections, we have discussed weighted graphs and their relationships to each other. In this section, we shift focus somewhat to consider general distance metrics and their relationship to the concepts we have already seen.

To introduce this section, we begin by giving some definitions for distance metrics and distance semimetrics. A distance metric on a set of points S is a function $d : S \times S \to \mathbb{N}$ such that the following criteria hold:

1. For $x, y \in S$, $d(x, y) = d(y, x)$
2. For $x \neq y \in S$:
   (a) $d(x, x) = 0$
   (b) $d(x, y) \neq 0$
3. For $x, y, z \in S$: $d(x, y) + d(y, z) \leq d(x, z)$ (triangle inequality)

A distance *semimetric* relaxes the definition of a distance metric by dropping requirement 2(b) [8]. In some cases, a distance metric can be written as the sum of distance semimetrics on the same set of points. In particular, consider the ℓ_1 metric on $\mathbb{R}^2$, which is defined such that $d_{\ell_1}((x_1, x_2), (y_1, y_2)) = |x_1 - y_1| + |x_2 - y_2|$. This can be written as the sum of two distance semimetrics, d_1, d_2 on $\mathbb{R}^2$ such that $d_1((x_1, x_2), (y_1, y_2)) = |x_1 - y_1|$ and $d_2((x_1, x_2), (y_1, y_2)) = |x_2 - y_2|$. As we defined them here, d_1 and d_2 essentially project the two input points down onto $\mathbb{R}$ and then

apply an ℓ_1 distance metric. However, distinct points may be projected onto the same point in $\mathbb{R}$, so distance between two distinct points may be non-zero and thus these are semimetrics but not metrics.

Questions about distance metrics are highly related to questions about hypercube embeddings. In particular, if S' is a subset of S, then define $\delta(S')(\cdot,\cdot)$ to be a distance semimetric such that for $x, y \in S$:

$$\delta(S')(x,y) = \begin{cases} 1 & x \in S', y \notin S' \text{ or } x \notin S', y \in S' \\ 0 & x, y \in S \text{ or } x, y \notin S' \end{cases}$$

If $d(\cdot,\cdot)$ is the distance metric that corresponds to graph distance on a particular graph G, then that graph has a hypercube embedding if and only if $d(\cdot,\cdot)$ can be written as the sum $\sum_{i=1}^{s} \delta(S_i)(\cdot,\cdot)$ for some integer s and subsets S_i of $V(G)$ [7]. In this section, we will discuss the relationship between pseudofactoring a graph and writing its corresponding distance metric as the sum of distance semimetrics.

In the previous sections, we discussed factorization, pseudofactorization, and hypercube embeddings for weighted graphs. However, in many cases we actually have questions about how distance metrics can be embedded into each other rather than graphs. In particular, given a particular distance metric, we may want to know if it can be embedded into a hypercube. Going back to the example from the introduction, if we want to design DNA sequences with a given relationship, it is more likely that we are given a list of points and desired distances than a graph representation. The overall problem is NP-hard, but we can use the relationship between distance metrics and graphs to make conclusions about hypercube embeddability in some cases. In particular, we will discuss the concept of a *minimum* graph, which will be analogous to a minimal graph, except that we require each edge be the *unique* shortest path between its end points. While there is up to an exponential number of minimal graphs representing a given distance metric, there is only one such minimum graph, and this graph has the handy property that if it is irreducible, then any graph with the same distance metric is also irreducible, as expressed in Lemma 3.3.1.

Lemma 3.3.1. *If a minimum graph G is irreducible, then any graph G' with the same distance metric as G is also irreducible.*

Proof. We have that G is an isometric subgraph of G'. Because they have the same distance metric, clearly there is an isometric embedding $\pi : V(G) \to V(G')$, so we only have to show that if $uv \in E(G)$, then $\pi(u)\pi(v) \in E(G')$. However, we know that uv is the unique shortest path from u to v in G, which means that there is no path from u to v that goes through another intermediate node and thus in order to preserve the distance metric, G' must have edge $\pi(u)\pi(v)$ with the same weight as uv.

Say G' has pseudofactors $\{G_1, \dots G_m\}$ for $m > 1$. Then G' is an isometric subgraph of $\Pi_{i=1}^m G_i$. Because G is an isometric subgraph of G', it is also an isometric subgraph of $\Pi_{i=1}^m G_i$ and thus is not irreducible. □

Given a distance metric, we can create a minimum graph that has the same metric by first creating an edge between every pair of nodes whose weight is their distance and then removing all edges that do not form unique shortest paths. If we then pseudofactor this graph G, we find that it is an isometric subgraph of $\Pi_{i=1}^m G_i$ for some integer m and graphs G_i. This means $d_{\Pi_i G_i}(\cdot, \cdot)$ restricted to the nodes in G is the same distance metric as $d_G(\cdot, \cdot)$. We can further see that for $u = (u_1, \dots, u_m)$ and $v = (v_1, \dots, v_m), u_i, v_i \in G_i$, $d_{\Pi_i G_i}(u, v) = \sum_{i=1}^m d_{G_i}(u_i, v_i)$, which has a structure very reminiscent of decomposing a distance metric into the sum of distance semimetrics. In fact, similarly to the concept of "projecting" a point in $\mathbb{R}^2$ down onto a point in $\mathbb{R}$ in the opening example for this section, we can consider the idea of projecting a node u in $\Pi_i G_i$ down onto a node in G_j for some j and then taking the distance in that graph.

Thus, if we have a graph $\Pi_i G_i$, we can write the distance metric $d_{\Pi_i G_i}(\cdot, \cdot)$ as the sum of distance semimetrics $d_i(\cdot, \cdot)$ such that $d_i(u, v) = d_{G_i}(u_i, v_i)$. Because we have an isometric embedding π of G into $\Pi_i G_i$, we can write its own distance metric $d_G(\cdot, \cdot)$ as $\sum_{i=1}^m d_{\pi_i}(\cdot, \cdot)$ such that $d_{\pi_i}(u, v) = d_i(\pi(u), \pi(v))$. If a graph G is hypercube embeddable, then there is an isometric embedding into a hypercube, which

is a product of K_2. If $G_i = K_2$, then $d_{\alpha_i}(u, v)$ is 1 if u and v are projected down onto different nodes in K_1 and 0 otherwise, showing again that the definition given by Deza and Laurent holds [7].

Because the pseudofactorization of a graph leads to a decomposition of its distance metric into the sum of distance semimetrics, a natural question to ask is what kind of decomposition we get from pseudofactoring the minimal graph for a given metric. Because it is NP-hard to decide if a given distance metric is hypercube embeddable, we know this will not necessarily tell us if we can decompose it into the sum of $\delta(S')(\cdot, \cdot)$ semimetrics. In fact, going off of the definition of pseudofactorization, we see that if G decomposes into pseudofactors $\{G_1, \ldots, G_m\}$, then there must be an isometric embedding $\pi : V(G) \to V(\Pi_i G_i)$ such that for each $uv \in E(G)$, $\pi(u)\pi(v) \in E(\Pi_i G_i)$. Because $\pi(u)\pi(v) \in E(\Pi_i G_i)$ implies that there exists exactly one j on which $\pi_j(u) \neq \pi_j(v)$, we get that there is only one factor graph on which $\pi(u)$ and $\pi(v)$ are projected onto different nodes and because all edges have positive weight (and the graph metric is a distance metric), this means that there is only one semimetric in this decomposition on which u, v have non-zero distance. The u, v that have edges between them in a minimum graph G are the points in the distance metric for which there does *not* exist a third point w with $d(u, w) + d(w, v) = d(u, v)$. We will call such points *adjacent*. Thus, decomposing a distance metric into semimetrics in this way comes with the restriction that any adjacent points have non-zero distance in exactly one of these semimetrics. Such a decomposition seems more intuitive in a graph setting where we want to preserve the existence of certain edges, so it's unclear if this is useful in the context of distance metrics, but it is interesting to apply the definition to this context.

As pointed out to me by Joseph Berleant, [2], the following lemma holds:

Lemma 3.3.2. *A minimum graph may have pseudofactors that are not minimum.*

Proof. Consider the minimum graph shown in Figure 3-1(a). Its pseudofactorization is in Figure 3-1(b), but the leftmost graph in this pseudofactorization is not minimum, as the edge with weight 2 is extraneous, since there is already an alternative shortest

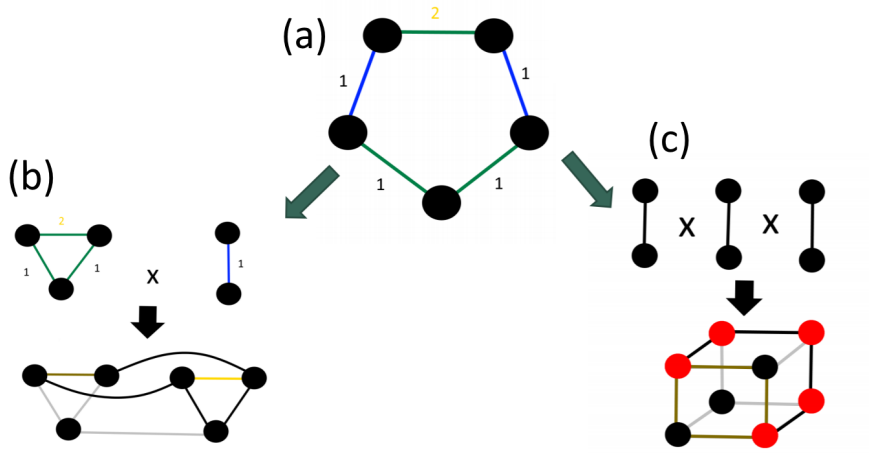

Figure 3-1: The graph in subfigure (a) is minimum. Green and blue edges correspond to edges in different equivalence classes. The irreducible pseudofactorization of the graph in subfigure (a) is the two upper graphs in subfigure (b), whose Cartesian product is the lower graph in subfigure (b). Subfigure (c) shows how the graph in subfigure (a) can be isometrically embedded into the 3D hypercube, where the red nodes are the nodes that would be in the image of such an embedding.

path between its endpoints that does not go through this edge. □

Because minimum graphs may not produce minimum pseudofactors, it is natural to continue to take the minimum graph corresponding to the distance metric for each pseudofactor produced and continue to find its pseudofactorization until the graph can no longer be pseudofactored. This essentially just decomposes the semimetrics further into sums of other semimetrics, which means that the original metric is still equal to the sum of these new semimetrics. In the case of the graph in Figure 3-1(a), doing this actually produces copies of K_2, which tells us that the overall distance metric is hypercube embeddable. In particular, we see that if we take the minimum graphs with the same distance metrics as those given by the pseudofactors in Figure 3-1(b), we end up with two line graphs, whose pseudofactors are all K_2. Thus Figure 3-1(c) gives us the pseudofactorization of these new graphs and tells us that the original graph was hypercube embeddable. However, following this procedure of taking the minimum graph for each distance metric output by a pseudofactorization will not always tell us if a graph is hypercube embeddable, as shown by the existence of a minimum irreducible graph that is hypercube embeddable but is not K_2 in Figure 3-2.

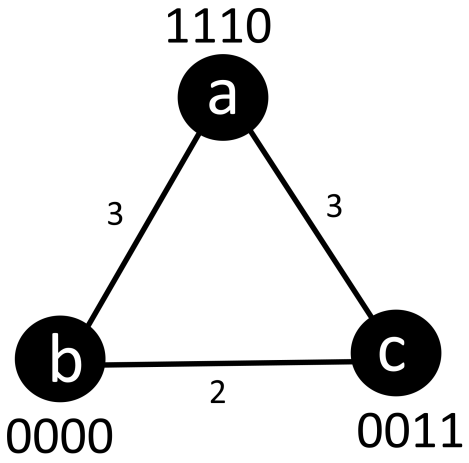

Figure 3-2: An example of a minimum irreducible graph that is hypercube embeddable and is not K_2

An interesting future direction for this work would be a further characterization of the kind of metric decompositions that can be gotten by pseudofactoring the minimum graph for a given metric.

Chapter 4

Conclusion and open questions

In this thesis, we discussed the extension of current methods for factoring and pseudofactoring unweighted graphs to use for weighted graphs. In fact, we saw that for any minimal weighted graph, a pseudofactorization can be found in as little as $O(|V|^2 \log\log|V| + |V||E|)$ time and a factorization can be found in as little as $O(|V|^2 \log\log|V| + |E|^2)$ time. In the context of unweighted graphs, determining the factors and pseudofactors of a graph determines hypercube and Hamming embeddability of the graph as a whole. While this is not the case in weighted graphs, and in fact a polynomial time algorithm for deciding hypercube embeddability is unlikely to exist, we do find that these processes help extend the number of graphs for which we can decide hypercube embeddability in polynomial time. From this work though several open questions remain. In particular, we ask each of the following as open questions to explore in the future:

- Can we use a similar method to that described in chapter 2.5 for speeding up graph pseudofactorization to speed up graph factorization as well?
- Can we place lower bounds on the time needed to find a graph's factorization or pseudofactorization? In particular, can we lower bound these processes by $O(|V||E|)$ time?
- In what other contexts might decomposing distance metrics by the method described in chapter 3 find applications?

- Are there further characterizations of the kind of graphs we can decide hypercube embeddability for in polynomial time?

In partial response to the first question, we can see by the proofs given in chapter 2.5 that a modification of Algorithm 2 can be used to find a tree T such that $(\theta_T \cup \tau)^*$ has the same equivalence classes as $(\theta \cup \tau)^*$, but at the moment it is unclear if the time determining all pairs of edges related by τ can be bounded or if τ can be modified in a way that makes these relations faster to compute. An interesting direction to take this project in the future would be to examine if this is the case.

Bibliography

[1] David Avis and Michel Deza. The cut cone, l^1 embeddability, complexity, and multicommodity flows. *Networks*, 21:6:183–198, 10 1991.

[2] John Berleant. personal communication, 2021.

[3] Kevin M. Cherry and Lulu Qian. Scaling up molecular pattern recognition with DNA-based winner-take-all neural networks. *Nature*, 559:370–376, 2018.

[4] V. Chvatal. Recognizing intersection patterns. *Annals of Discrete Mathematics*, 8:249–252, 1980.

[5] M. Deza and M. Laurent. ℓ_1-rigid graphs. *Journal of Algebraic Combinatorics*, 3:153–175, 1994.

[6] M. Deza and S. Shpectorov. Recognition of the ℓ_1-graphs with complexity O(nm), or football in a hypercube. *European Journal of Combinatorics*, 17:2-3:279–289, 1996.

[7] Michel Deza and Monique Laurent. The cut cone and ℓ_1 metrics. In R. L. Graham, B. Korte, Bonn L. Lovasz, A.Wigderson, and G.M. Ziegler, editors, *Geometry of Cuts and Metrics*, chapter 4, pages 37–52. Springer, 1997.

[8] Michel Deza and Monique Laurent. Hypercube embeddings of the equidistant metric. In R. L. Graham, B. Korte, Bonn L. Lovasz, A.Wigderson, and G.M. Ziegler, editors, *Geometry of Cuts and Metrics*, chapter 23, pages 341–352. Springer, 1997.

[9] Michel Deza and Monique Laurent. The prime factorization of a graph. In R. L. Graham, B. Korte, Bonn L. Lovasz, A.Wigderson, and G.M. Ziegler, editors, *Geometry of Cuts and Metrics*, chapter 20.2, pages 305–306. Springer, 1997.

[10] D Ž Djoković. Distance-preserving subgraphs of hypercubes. *Journal of Combinatorial Theory, Series B*, 14(3):263–267, 1973.

[11] Tomàs Feder. Product graph representations. *Journal of Graph Theory*, 16:5:467–488, 1992.

[12] VV Firsov. Isometric embedding of a graph in a boolean cube. *Cybernetics*, 1(6):112–113, 1965.

[13] Ronald L Graham and Peter M Winkler. On isometric embeddings of graphs. *Transactions of the American Mathematical Society*, 288(2):527–536, 1985.

[14] Wilfried Imrich and Sandi Klavžar. On the complexity of recognizing Hamming graphs and related classes of graphs. *European Journal of Combinatorics*, 17:209–221, 1996.

[15] Wilfried Imrich and Iztok Peterin. Recognizing Cartesian products in linear time. *Discrete Mathematics*, 307:472–483, 02 2007.

[16] Alexander Karzanov. Metrics and undirected cuts. *Mathematical Programming*, 32:183–198, 02 1985.

[17] Monique Laurent. Hypercube embedding of distances with few values. In Helene Barcelo and Gil Kalai, editors, *Contemporary Mathematics*, volume 178, pages 179–207, Providence, Rhode Island, 7 1993. Jerusalem Combinatorics '93, American Mathematical Society.

[18] Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theoretical Computer Science*, 312(1):47–74, 2004. Automata, Languages and Programming.

[19] Lulu Qian, Erik Winfree, and Jehoshua Bruck. Neural network computation with DNA strand displacement cascades. *Nature*, 475:368–72, 07 2011.

[20] R. L. Roth and P.M. Winkler. Collapse of the metric hierarchy for bipartite graphs. *European Journal of Combinatorics*, 7:371–375, 1986.

[21] John Santalucia and Donald Hicks. The thermodynamics of dna structural motifs. *Annual Review of Biophysics and Biomolecular Structure*, 33(1):415–440.

[22] S. V. Shpectorov. On scale embeddings of graphs into hypercubes. *Eur. J. Comb.*, 14(2):117–130, March 1993.

[23] Peter Winkler. The metric structure of graphs: theory and applications. *Surveys in Combinatorics*, 123:197–221, 1987.

[24] Peter M Winkler. Isometric embedding in products of complete graphs. *Discrete Applied Mathematics*, 7(2):221–225, 1984.