Factorization and pseudofactorization of weighted graphs

Kristin Sheridan [*†1], Joseph Berleant[2], Mark Bathe[2], Anne Condon[3], and Virginia Vassilevska Williams[4]

[1]Department of of Computer Science, University of Texas, Austin, TX
[2]Department of Biological Engineering, Massachusetts Institute of Technology, Cambridge, MA
[3]Department of Computer Science, University of British Columbia, Vancouver, Canada
[4]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA

Abstract

For unweighted graphs, finding isometric embeddings of a graph G is closely related to decompositions of G into Cartesian products of smaller graphs. When G is isomorphic to a Cartesian graph product, we call the factors of this product a factorization of G. When G is isomorphic to an isometric subgraph of a Cartesian graph product, we call those factors a pseudofactorization of G. Prior work has shown that an unweighted graph's pseudofactorization can be used to generate a canonical isometric embedding into a product of the smallest possible pseudofactors. However, for arbitrary weighted graphs, which represent a richer variety of metric spaces, methods for finding isometric embeddings or determining their existence remain elusive, and indeed pseudofactorization and factorization have not previously been extended to this context. In this work, we address the problem of finding the factorization and pseudofactorization of a weighted graph G, where G satisfies the property that every edge constitutes a shortest path between its endpoints. We term such graphs minimal graphs, noting that every graph can be made minimal by removing edges not affecting its path metric. We generalize pseudofactorization and factorization to minimal graphs and develop new proof techniques that extend the previously proposed algorithms due to Graham and Winkler [Graham and Winkler, '85] and Feder [Feder, '92] for pseudofactorization and factorization of unweighted graphs. We show that any n-vertex, m-edge graph with positive integer edge weights can be factored in $O(m^2)$ time, plus the time to find all pairs shortest paths (APSP) distances in a weighted graph, resulting in an overall running time of $O(m^2+n^2 \log\log n)$ time. We also show that a pseudofactorization for such a graph can be computed in $O(mn)$ time, plus the time to solve APSP, resulting in an $O(mn + n^2 \log\log n)$ running time.

[*]To whom correspondence should be addressed: kristin.sheridan3@gmail.com

[†]Research performed while at the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA

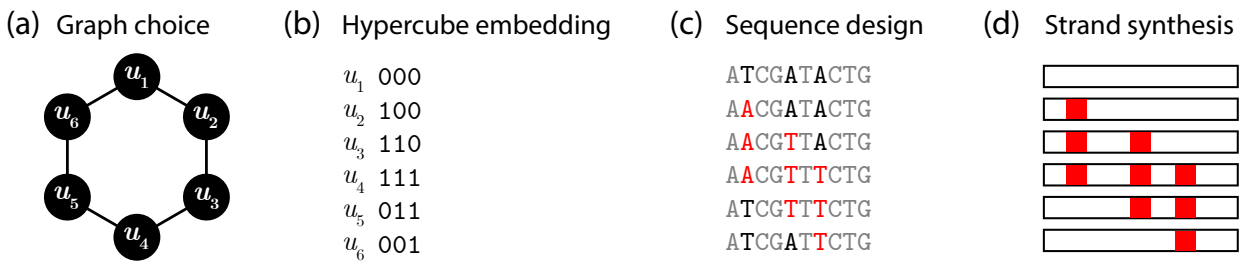

Figure 1: A schematic showing how hypercube embeddings can be used to construct a set of DNA sequences with a particular relationship. a) An unweighted graph representing the desired binding strength relationships between DNA strands. b) A hypercube embedding (equivalent to an assignment of binary strings) of this graph. c-d) Each binary string in the hypercube embedding corresponds to a DNA sequence, by associating 0 and 1 with point mutations within a sequence. When mutations are not adjacent and the pre- and post-mutation bases are chosen carefully, the binding strength between DNA strands is approximately determined by the number of mutations between one strand and the complement of the other. This relationship can be captured in a hypercube or Hamming graph.

1 Introduction

The task of finding *isometric embeddings*, or mappings of the vertex set of one graph to another while preserving pairwise distances between vertices, is widely applicable but unsolved for general weighted graphs. One of the most important applications is in molecular engineering. Attempting to design biomolecules with the same control as seen in natural biological systems, molecular engineers may focus on designing sets of DNA strands with pre-specified binding strengths, as these binding strengths can be essential to the emergent behavior of a network of interacting molecules [23, 29, 1, 16]. Under certain conditions, the binding strength between pairs of DNA strands can be approximated by the distance between pairs of vertices in a hypercube graph, and so the DNA strand-design problem reduces to the task of finding a mapping between a graph whose pairwise vertex distances correspond to desired binding strengths and the hypercube graph whose distances correspond to the actual binding strengths between pairs of DNA strands (Figure 1). This strand design problem could be applied for DNA data storage [2, 17], DNA logic circuits [20, 28], and DNA neural networks [21, 4].

Isometric embeddings may also be applied in communications networks by embedding the connectivity graph of a network into a Hamming graph, or product of complete graphs, which allows shortest paths between nodes to be computed using only local connectivity information [11]; in linguistics as a method of representing the similarities between various linguistic objects [10]; and in coding theory for the design of certain error-checking codes [15].

In general, many graphs will not have isometric embeddings into a particular destination graph, and the problem of efficiently finding isometric embeddings has been solved for only certain classes of graphs. Prior work has addressed this task for unweighted graphs into hypercubes [7], Hamming graphs [27, 24], and arbitrary Cartesian graph products [12, 8]. These works on unweighted graphs related the isometric embedding problem to representations of a graph either as isomorphic to a Cartesian product of graphs, which we call a *factorization*, or as isomorphic to an isometric subgraph of a Cartesian product of graphs, which we call a *pseudofactorization*. In this work, we extend the concepts of factorization and pseudofactorization

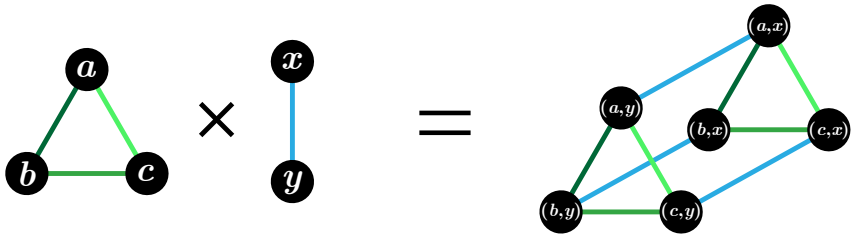

Figure 2: The graph on the right is the Cartesian product of the graphs on the left. The parent edge of a given edge in the product graph is the edge in a factor that has the same color as it. For example, edges $(a, x)(a, y)$, $(b, x)(b, y)$, and $(c, x)(c, y)$ in the product graph all have edge xy in the second factor as their parent edge. Here, edges of the same color have the same weight.

to weighted graphs, a task that to our knowledge has not been addressed before. Due to the work of Berleant *et al.* [3], this has implications for finding hypercube and Hamming embeddings of certain weighted graphs.

Except for Section 4, which applies to all weighted graphs, this paper focuses on weighted graphs for which every edge is a shortest path between its endpoints. We call such graphs *minimal graphs*. Minimal graphs form a natural subset of the class of weighted graphs,consisting of those graphs for which every edge is part of at least one shortest path.

Many aspects of our results also apply to arbitrary weighted graphs, because any weighted graph may be made minimal simply by removing any edges that do not affect its path metric. Our results also apply in some cases to arbitrary finite metric spaces, because any finite metric space has a corresponding minimal graph generated by taking a weighted complete graph and removing all extra edges. Other weighted graphs may also be constructed for a given finite metric space, and the isometric embeddings described by our methods will in general depend on which weighted graph representation is used [5].

1.1 Other work

1.1.1 Factorization and pseudofactorization of unweighted graphs

The *Cartesian graph product* (defined formally in Section 2) combines $k \geq 1$ graphs called *factors*, so that every vertex in the product graph is a k-tuple of vertices, one from each of the factors, and each edge of the product graph corresponds to a unique edge in one of the factors (see Figure 2).

The problem of representing a given graph as a Cartesian product of factor graphs is of central importance to isometric embeddings because of the property that every distance in the product graph may be decomposed as a sum of distances in the factor graphs. We consider two Cartesian product representations, factorization and pseudofactorization, both described in the following paragraph and depicted in Figure 3. For unweighted graphs, the problems of finding both representations for a given graph G are well studied.

Given an unweighted graph, a factorization is a set of factor graphs whose Cartesian product is isomorphic to the given graph. A *prime* graph is one whose factorizations always include itself. Sabidussi [22] showed that every connected, unweighted graph has a unique factorization into prime graphs, and Feigenbaum *et al.* [9] and Winkler [25] first showed that the prime factorization of such graphs can can be found in polynomial

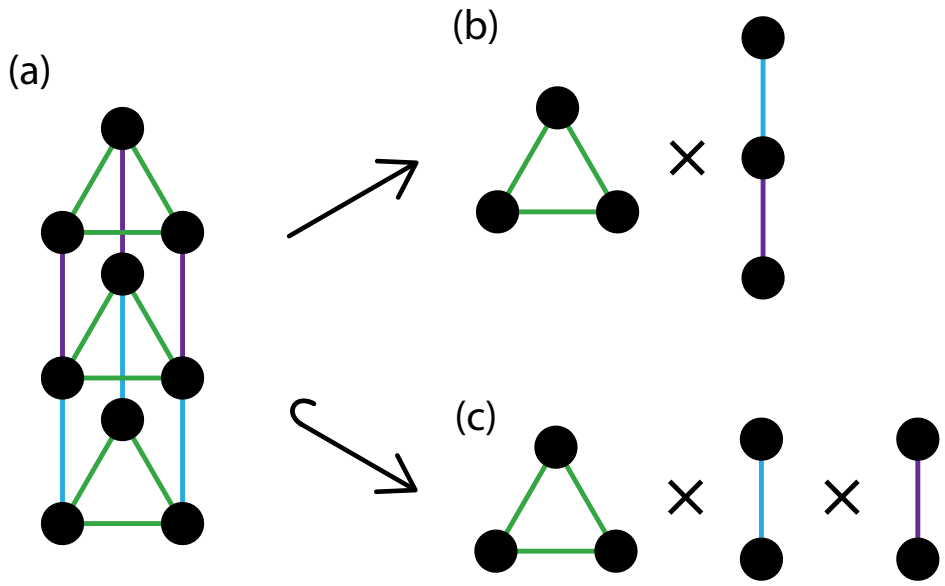

Figure 3: Subfigure (a) shows a non-prime, non-irreducible graph. Subfigure (b) shows the prime factorization of the graph in (a). Subfigure (c) shows the canonical pseudofactorization of the graph in (a).

time. Feder [8] showed that the prime factorization of any unweighted m-edge, n-vertex graph can be found in $O(mn)$ time. Imrich and Klavžar [13] showed that deciding if the prime factorization of an unweighted m-edge graph consists entirely of complete graphs can be done in $O(m)$ time. More recently, Imrich and Peterin [14] showed that finding the prime factorization of an arbitrary unweighted m-edge graph can also be done in $O(m)$ time. However, for a disconnected graph, this problem is at least as hard as the graph isomorphism problem, which is not known to be solvable in polynomial time. In particular, two graphs G and H are isomorphic if and only if the graph of two isolated nodes is a factor of their disjoint union (see [6]). In fact, the prime factorization of a disconnected graph is no longer unique [26].

A pseudofactorization of an unweighted graph relaxes the condition of a factorization, only requiring that the input graph is isometrically embeddable into the Cartesian product of a set of graphs. Clearly, any factorization is also a pseudofactorization; however, the converse is not true (e.g., see Figure 3). The analog of a prime graph in the context of pseudofactorization is an *irreducible* graph. Each connected, unweighted graph has a unique pseudofactorization into irreducible graphs, its *canonical pseudofactorization* [12][1]. Importantly, defining pseudofactorization for weighted graphs must be done with care, since with the above definition no weighted graph would have an irreducible pseudofactorization (Figure 4).

1.1.2 Isometric embeddings of unweighted graphs

Despite the connection between pseudofactorization and isometric embedding, studies of isometric embeddings of unweighted graphs preceded Graham and Winkler's original description of pseudofactorization. In 1973, Djoković [7] was the first to characterize the unweighted graphs with isometric embeddings into a hypercube, and showed that constructing such an embedding for an unweighted graph can be done in poly-

[1]Notably, Graham and Winkler [12, 26] use the term "factor" both for graphs in a factorization and graphs in a pseudofactorization. Here we specifically use the term "pseudofactors" to refer to the graphs forming some pseudofactorization.

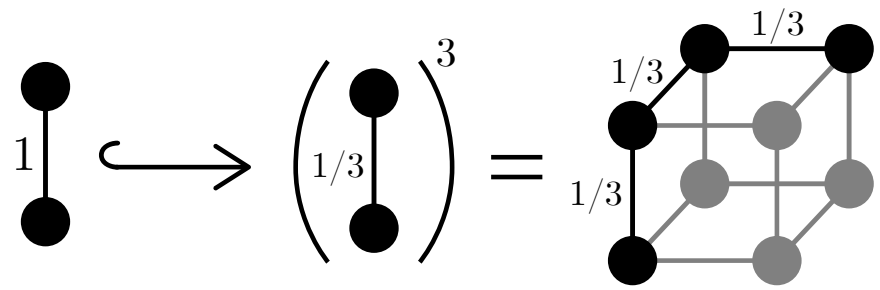

Figure 4: A generalization of pseudofactorization to weighted graphs must be done with care. For example, if pseudofactorization requires only that the input graph is isometrically embeddable into a Cartesian graph product, then even graphs such as K_2 have no irreducible pseudofactorization. We require that a graph also be isomorphic to a subgraph of the Cartesian graph product, which precludes this example.

nomial time. In 1984, Winkler [27] extended these results to finding isometric embeddings of unweighted graphs into Hamming graphs, designing an $O(n^5)$ algorithm to do so (for n-vertex, m-edge graphs). Soon after, Graham and Winkler [12] generalized the earlier results to arbitrary Cartesian graph products, showing that each unweighted graph has a unique isometric embedding into the Cartesian product of its irreducible pseudofactorization. Graham and Winkler [12] proposed an $O(m^2)$ algorithm for finding this unique isometric embedding, and later results, such as the $O(mn)$ pseudofactorization algorithm designed by Feder, may be used to construct this isometric embedding more efficiently [8].

1.2 Our results

In this paper, we start by generalizing the notions of factorization and pseudofactorization to weighted graphs. While defining factorization for weighted graphs is straightforward, pseudofactorization of weighted graphs is more subtle. As an example, if, in analogy to unweighted graphs, a pseudofactorization of graph $G = (V(G), E(G), w_G)$ is defined as a set of graphs for which G is isometrically embeddable into their product, then the graph K_2 will have no irreducible pseudofactorization into weighted graphs because it can be isometrically embedded into the Cartesian product of k copies of K_2 with edge weight $\frac{1}{k}$ for any positive integer k (Figure 4). Instead, we require G to be isomorphic to an isometric subgraph of the product of the pseudofactors. This constraint implies both preservation of distances and of edges between G and the pseudofactor product, which are characteristics of unweighted graph embeddings previously noted by Winkler . When all graphs are unweighted, this is equivalent to previous work.

With pseudofactorization defined as such, we first prove that the $O(m^2)$ algorithm proposed by Graham and Winkler [12] can be adapted slightly to work on weighted m-edge graphs. While the algorithm itself is largely unchanged, additional proof is required to show that the output is a correct pseudofactorization of the input graph into irreducible graphs, and that the output does not depend on the order in which edges and vertices are traversed.

Theorem 1.1. *Given a minimal weighted graph $G = (V(G), E(G), w_G)$ and the distances between all pairs of vertices, a pseudofactorization into irreducible weighted graphs can be achieved in $O(m^2)$ time, where $m = |E(G)|$. If the distances are not pre-computed, the time required to compute all-pairs shortest paths (APSP) must be included, and pseudofactorization may be achieved in $O(n^2 \log\log n + m^2)$ time,*

where $n = |V(G)|$.

The APSP running time of $O(n^2 \log\log n + mn)$ is by Pettie [18] for n-vertex, m-edge graphs, and the pseudofactorization runtime is dominated by the $O(m^2)$ term for dense enough graphs.

Our proof uses the Djoković-Winkler relation θ and its transitive closure θ^*, which are relations on the edges of a graph and are frequently used to pseudofactor unweighted graphs [26]. For unweighted graphs, each equivalence class E_i of θ^* is used to generate one pseudofactor by removing the edges in E_i from the input graph and taking each connected component of the resulting graph to be a vertex of the pseudofactor. Vertices are adjacent in that pseudofactor if there is an edge in E_i connecting the corresponding connected components [12]. To apply this process to minimal weighted graphs, we must prove that all edges connecting any two connected components have the same edge weight (Lemma 3.4), and that this edge weight may be used as the edge weight in the corresponding pseudofactor. The proofs that the input graph is isometrically embeddable into the resulting set of pseudofactors, and that each pseudofactor is irreducible, are given in Theorems 3.9 and 3.10. The proof of the runtime of this algorithm is given in Section 5.1.

In addition, we adapt the reasoning of Graham and Winkler on unweighted graphs [12, 26] to prove that the irreducible pseudofactorization of a minimal weighted graph is unique. As a result, we call the irreducible pseudofactorization output by this algorithm the *canonical pseudofactorization* and each pseudofactor a *canonical pseudofactor*.

Theorem 1.2. *For a minimal weighted graph, any two pseudofactorizations into irreducible weighted graphs are equivalent in the following sense: there exists a bijection between the two sets of pseudofactors such that corresponding pairs of pseudofactors are isomorphic to each other.*

Theorem 1.2 is proven in Section 3.3.

Finally, we modify a pseudofactorization algorithm on unweighted graphs due to Feder [8] to speed up the pseudofactorization of minimal weighted graphs. Feder improved upon Graham and Winkler's runtime by finding a spanning tree T for the graph and defining a new relation θ_T on the edges of a graph such that two edges in the graph are related by θ_T if they are related by θ and at least one of them is in T. Feder showed that applying Graham and Winkler's pseudofactorization algorithm using the transitive closure of this relation, θ_T^*, produces an irreducible pseudofactorization of a weighted graph, no matter the choice of T. Because these equivalence classes are quicker to compute than those of θ^* and the number of equivalence classes is necessarily limited to $n-1$, the runtime is improved to $O(mn)$ for an n-vertex, m-edge graph. Motivated by Feder's work [8], we propose Algorithm 2, which finds in $O(mn)$ time a spanning tree T of an n-vertex, m-edge weighted graph for which θ_T^* has the same equivalence classes as θ^*. This allows us to improve the time complexity of pseudofactorization given precomputed APSP distances to $O(mn)$.

Theorem 1.3. *Given a minimal weighted graph* $G = (V(G), E(G), w_G)$ *and the distances between all pairs of vertices, a pseudofactorization into irreducible weighted graphs is achievable in* $O(mn)$ *time, where* $m = |E(G)|$ *and* $n = |V(G)|$. *If distances are not pre-computed, this is achievable in* $O(n^2 \log\log n + mn)$ *time.*

Our results on factorization parallel those for pseudofactorization, and many of our proofs for factorization rely on those for pseudofactorization. Feder showed that an unweighted graph can be factored by replacing the Djoković-Winkler relation θ with a different relation. We use the replacement relation $\theta \cup \tau$, where τ relates edges based on a so-called *square property* [8]. This is similar to the relation of the same name proposed by Feder, but with additional restrictions needed for use with weighted graphs. This allows us to

make the following statement, whose proof follows similar steps to that for pseudofactorization.

Theorem 1.4. *Given a weighted graph $G = (V(G), E(G), w_G)$ and the distances between all pairs of vertices, a factorization into prime weighted graphs is achievable in $O(m^2)$ time, where $m = |E(G)|$. If distances are not pre-computed, this is achievable in $O(n^2 \log\log n + m^2)$ time, where $n = |V(G)|$.*

The prime factorization algorithm and proof of correctness are presented in Section 4.2. The runtime of the algorithm is analyzed in Section 5.2.

As with pseudofactorization, we are also able to show that the prime factorization of a minimal graph G is unique. The proof in this case is much simpler because of the restriction that G be isomorphic to the Cartesian product of its prime factors.

Theorem 1.5. *For a minimal weighted graph, any two factorizations into prime graphs are equivalent in the following sense: there is a bijection between both sets of factors such that the corresponding pairs of factor graphs are isomorphic.*

The proof of this theorem is given in Section 4.3.

1.3 Overview

Section 2 describes notation and definitions necessary for the remaining sections. In Section 3, we show the correctness of the algorithm for pseudofactorization of minimal weighted graphs. We also show that each graph has a unique irreducible pseudofactorization up to graph isomorphism. In Section 4, we show that weighted graphs may be factored using a modified version of Feder's algorithm [8] and prove uniqueness of prime factorization of weighted graphs. Section 5 introduces our method for analyzing runtimes of the algorithms presented here, and shows that for an m-edge graph, the given algorithms run in $O(m^2)$ time plus the time to find all pairs shortest paths (APSP) distances. In Section 6, we use a modified version of Feder's algorithm [8] for pseudofactorization to show that the irreducible pseudofactorization of a minimal weighted n-vertex, m-edge graph is computable in $O(nm)$ time plus the time to compute APSP. Section 7 concludes with our final thoughts on this work and proposes some remaining open questions.

2 Preliminaries

We consider finite, connected, undirected graphs, written $G = (V(G), E(G), w_G)$ with vertex set V, edge set E, and edge weight function $w_G : E \to \mathbb{R}_{>0}$. For unweighted G, we let $w_G(e) = 1$ for all $e \in E$. Edges of G are written uv or vu for vertices $u, v \in V$; since all edges are undirected, $uv \in E \iff vu \in E$. The shortest path metric for G, written $d_G : V \times V \to \mathbb{R}_{\geq 0}$, maps pairs of vertices to the minimum edge weight sum along a path between them.

Definition 2.1. *A graph G is a* minimal graph *if and only if every edge in $E(G)$ forms a shortest path between its endpoints. That is, $w_G(uv) = d_G(u, v)$ for all $uv \in E(G)$.*

Clearly, all unweighted graphs are minimal. We note that for any non-minimal weighted graph, a minimal graph can be generated with the same path metric by simply removing edges not satisfying the minimality condition.

A graph embedding $\pi : V(G) \to V(G^*)$ maps vertices of a graph G onto those of a graph G^*. If π satisfies $d_G(u, v) = d_{G^*}(\pi(u), \pi(v))$ all $u, v \in V(G)$, then π is an isometric embedding. Note that π must be an

injection, as both d_G and d_{G^*} are distance metrics and thus the distance between a pair of vertices is 0 if and only if the vertices are the same. When such a π exists, we write $G \hookrightarrow G^*$.

The Cartesian graph product of one or more graphs $G_1, \ldots, G_m$ is written $G = G_1 \times \cdots \times G_m$ or $G = \prod_{i=1}^{m} G_i$. For $G_i = (V(G_i), V(E_i), w_{G_i})$, G is defined as $V(G) = V(G_1) \times \cdots \times V(G_m)$, $E(G)$ is the set of all $(u_1, \ldots, u_m)(v_1, \ldots, v_m)$ with exactly one ℓ such that $u_\ell v_\ell \in E(G_\ell)$ and $u_i = v_i$ for all $i \neq \ell$, and $w_G(uv) = w_{G_\ell}(u_\ell v_\ell)$ for ℓ chosen as above (Figure 2). Note that Cartesian products are typically defined with respect to unweighted graphs, and for our generalization to weighted graphs, we let each edge inherit its weight from one of the factors. This allows us to retain an important distance-preservation property associated with Cartesian products of unweighted graphs. In particular, for any two vertices $u, v \in V(G)$, $u = (u_1, \ldots, u_m)$ and $v = (v_1, \ldots, v_m)$, we have that

$$d_G(u, v) = \sum_{i=1}^{m} d_{G_i}(u_i, v_i) \tag{1}$$

We prove Equation (1) after defining a *parent edge* in Definition 2.2

We are particularly concerned with cases where G is isometrically embeddable into a Cartesian graph product. The following definition relates edges in G to edges in a Cartesian product into which it has an isometric embedding, and for which edges are preserved between G and the product.

Definition 2.2. *For graphs $G_1, \ldots, G_m$, consider the Cartesian product $G = \Pi_{i=1}^{m} G_i$. For every edge $uv \in E(G)$, $u = (u_1, \ldots, u_m)$, $v = (v_1, \ldots, v_m)$, there exists exactly one ℓ such that $u_\ell \neq v_\ell$, and we must have $u_\ell v_\ell \in E(G_\ell)$. We call $u_\ell v_\ell$ the* parent edge *of uv. Further, if another graph G' has an isometric embedding $\pi : V(G') \to V(G)$ and there is an edge $u'v' \in E(G')$ such that $\pi(u')\pi(v') = uv$, then we say that $u_\ell v_\ell$ is the* parent edge under π *of $u'v'$.*

Note that from our definition of Cartesian product, edge $uv \in E(G)$ with parent edge $u_\ell v_\ell \in E(G_\ell)$ must have $w_G(uv) = w_{G_\ell}(u_\ell v_\ell)$.

Proof of Equation 1. We show that Equation 1 holds by considering both directions of the inequality.

$d_G(u, v) \leq \sum_{i=1}^{m} d_{G_i}(u_i, v_i)$: Consider a shortest path $P_i = (u_i = p_1^i, p_2^i, \ldots, p_{k_i}^i = v_i)$ from u_i to v_i in G_i. Then there is a path from $u = (u_1, \ldots, u_m)$ to $(v_1, u_2, \ldots, u_m)$ of weight equal to that of P_1, or $d_{G_1}(u_1, v_1)$. In particular the path through the product graph that holds all elements but the first one constant and follows P_1 along the first index meets this criteria, which can be expressed as the path $((u_1 = p_1^1, u_2, \ldots, u_m), (p_2^1, u_2, \ldots, u_m), \ldots, (v_1 = p_{k_1}^1, u_2, \ldots, u_m))$. We can apply this iteratively to construct a path from $(v_1, v_2, \ldots, v_{t-1}, u_t, \ldots, u_m)$ to $(v_1, v_2, \ldots, v_{t-1}, v_t, u_{t+1} \ldots, u_m)$ of weight $d_{G_t}(u_t, v_t)$, by following a shortest path in factor t and holding all other elements constant. Pasting these paths together, we get a path from u to v of weight $\sum_{i=1}^{n} d_{G_i}(u_i, v_i)$.

$d_G(u, v) \geq \sum_{i=1}^{m} d_{G_i}(u_i, v_i)$: We prove by induction on the number of edges on a shortest path P between u and v that $d_G(u, v) \geq \sum_{i=1}^{m} d_{G_i}(u_i, v_i)$. As a base case, observe that if P crosses a single edge then $uv \in E(G)$. By the definition of the Cartesian product there is exactly one factor G_ℓ for which $u_\ell \neq v_\ell$ and $u_\ell v_\ell \in E(G_\ell)$, $w_{G_\ell}(u_\ell, v_\ell) = w_G(u, v)$. Since P is a shortest path, $d_G(u, v) = w_G(u, v) = w_{G_\ell}(u_\ell, v_\ell) \geq d_{G_\ell}(u_\ell, v_\ell)$. For the inductive step let P cross k edges, $k > 1$, and assume $d_G(u, v) \geq \sum_{i=1}^{m} d_{G_i}(u_i, v_i)$ for all $u, v \in V(G)$ having a shortest path of less than k edges. Subdivide P into two subpaths of length less than k, $(u, \ldots, x)$ and $(x, \ldots, v)$, where $x = (x_1, \ldots, x_m)$. Then we have

$d_G(u,v) = d_G(u,x) + d_G(x,v) \geq \sum_{i=1}^{m}[d_{G_i}(u_i,x_i) + d_{G_i}(x_i,v_i)] \geq \sum_{i=1}^{m} d_{G_i}(u_i,v_i)$, where the final step is due to the triangle inequality.

□

The term factorization is used when a graph G is isomorphic to a Cartesian graph product (e.g., Figure 3b). Here we generalize the concept of unweighted graph isomorphisms to require preservation of weights under isomorphism as well. In particular, we consider two weighted graphs $G = (V(G), E(G), w_G)$ and $G' = (V(G'), E(G'), w_{G'})$ to be isomorphic when there is a bijective map $f : V(G) \to V(G')$ such that for all $u, v \in V(G)$, $f(u)f(v) \in E(G') \iff uv \in E(G)$ and $uv \in E(G) \implies w_G(uv) = w_{G'}(f(u)f(v))$.

Definition 2.3. *Whenever G is isomorphic to the Cartesian graph product $\prod_{i=1}^{m} G_i$, we say that the set $\{G_1, \ldots, G_m\}$ forms a* factorization *of G and refer to each G_i as a* factor.

If all factorizations of G include G as a factor, we say that G is *prime*. A prime factorization is one with only prime factors. For convenience, we assume that a factorization does not include K_1, except in the case where $G = K_1$, since a factor of K_1 does not affect the final product up to isomorphism.

Pseudofactorization generalizes factorization to situations where G is not isomorphic to the graph product. Instead, we require that G is only isomorphic to an isometric subgraph of the graph product.

Definition 2.4. *Consider graphs G and $G^* = \prod_{i=1}^{m} G_i^*$. For an embedding $\pi : V(G) \to V(G^*)$, we define for each i, $1 \leq i \leq m$, a function $\pi_i : V(G) \to V(G_i)$ such that $\pi(u) = (\pi_1(u), \ldots, \pi_m(u))$ for all $u \in V(G)$. If an isometric embedding $\pi : V(G) \to V(G^*)$ with the associated π_i exists satisfying the following criteria:*

1. $\forall u, v \in V(G) :\ d_G(u,v) = d_{G^*}(\pi(u), \pi(v))$,
2. $\forall u, v \in V(G) :\ uv \in E(G)$ *implies* $\pi(u)\pi(v) \in E(G^*)$ *and* $w_G(uv) = w_{G^*}(\pi(u)\pi(v))$,
3. *every vertex in G_i^* is in the image of π_i, $1 \leq i \leq m$, and*
4. *every edge in G_i^* is the parent of an edge in G, $1 \leq i \leq m$*

then we say the set $\{G_1^*, \ldots, G_m^*\}$ is a pseudofactorization *of G and refer to each G_i^* as a* pseudofactor.

If all pseudofactorizations of G include G as a pseudofactor, we say that G is *irreducible*. An irreducible pseudofactorization is one with only irreducible pseudofactors. As with factorization, we assume that a pseudofactorization does not include K_1, except in the case where $G = K_1$.

Clearly, any pseudofactorization is also a factorization; however, the converse is not true (see Figure 3c). Informally, the definition of pseudofactorization requires both that G can be isometrically embedded into G^* and that edges are preserved within this embedding. In other words, the embedding must be a homomorphism. This second condition is a natural one for manipulating graph structures, but may be less applicable to other situations (e.g., finite metric spaces). The final two conditions ensure that there are no unnecessary vertices and edges in the pseudofactors (or any graph would be a pseudofactor of the graph in question, as G is an isometric subgraph of $G \times H$ for any graph H).

2.1 Binary relations on the edge set of a graph

In this manuscript, several binary relations on edge sets of a graph will become relevant. Given a binary relation R on a set X we will denote its transitive closure as R^*, defined such that for each $x, y \in X$, $x\ R^*\ y$

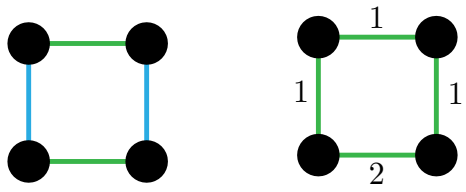

Figure 5: An unweighted graph (left) and a minimal weighted graph with the same vertex adjacency (right). Edges that are the same color are members of the same equivalence class under θ^*. The graph on the left has two equivalence classes and it is not irreducible, while the graph on the right has only one equivalence class and is irreducible.

if and only if there exists a sequence $(x = x_1, \ldots, x_t = y)$ of elements in X such that $x_i \; R \; x_{i+1}$ for each i, $1 \leq i < t$. When R^* is an equivalence relation on a set E, we denote by $[e]_{R^*}$, $e \in E$, the equivalence class under R^* with e.

Our first binary relation, θ is due to Graham and Winkler [12].

Definition 2.5 (θ relation)**.** *The binary relation θ on the edges of a graph $G = (V(G), E(G), w_G)$ is defined such that $uv \, \theta \, xy$ if and only if*

$$[d_G(x,u) - d_G(x,v)] - [d_G(y,u) - d_G(y,v)] \neq 0.$$

This is a symmetric and reflexive relation, and its transitive closure θ^* is an equivalence relation. We call the left-hand-side of this equation the *theta-difference* between edges uv and ab, so that if the theta-difference between two edges is nonzero, those edges are related by θ.

Although θ is the same as in [12], we note that the addition of edge weights may change the equivalence classes of θ^*, as shown in Figure 5.

In Section 6, we will use an additional relation we call θ_T, based on a relation presented by Feder [8], as defined in Definition 2.6.

Definition 2.6 (θ_T relation)**.** *Given a weighted graph $G = (V(G), E(G), w_G)$ and a spanning tree T of G, two edges $ab, xy \in E(G)$ satisfy $ab \, \theta_T \, xy$ if and only if $ab \, \theta \, xy$ and at least one of ab, xy is in T.*

The transitive closure θ_T^* of θ_T is an equivalence relation on the edges of G for any spanning tree T of G. (This is not different than for unweighted graphs, a proof of which can be found in [8], but for completeness we reiterate it here for weighted graphs.) First, since θ_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 $ab \; \theta_T^* \; ab$ and the relation is reflexive. Thus, θ_T^* is an equivalence relation for any T.

Secondly, we define a modified version of a relation presented by Feder [8] for use in graph factorization. To do this, we first define the *square property* for two edges in a weighted graph.

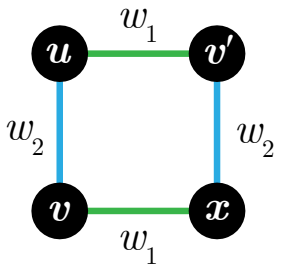

Figure 6: If $uv' \,\theta\, vx$ and $uv \,\theta\, v'x$, then existence of the subgraph depicted here implies that uv and uv' satisfy the square property.

Definition 2.7. *Edges $uv, uv' \in E(G)$ satisfy the* square property *if there exists a vertex x such that $uvxv'$ forms a chordless 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. If no such square exists, the edges do not satisfy the square property.*

The square property is depicted visually in Figure 6.

We define binary relation τ such that edges $uv, uv' \in E(G)$ are related by τ if and only if they do *not* satisfy the square property. This is a modified version of the τ relation proposed by Feder.

Definition 2.8 (τ relation)**.** *The binary relation τ on the edges of a graph $G = (V(G), E(G), w_G)$ is defined as follows: 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.*

We make a small note regarding Definitions 2.7 and 2.8. For the square property, we require that the square be chordless, based on the intuition that a square subgraph of this form would be created by the Cartesian product of two edges in separate factor graphs. However, we note that the requirement that the square be chordless does not affect the equivalence classes generated in Section 4 using the relation $(\tau \cup \theta)^*$, because all edges in a square that is not chordless will be in the same equivalence class of θ^* (and thus of $(\tau \cup \theta)^*$), regardless of τ.

2.2 Edge-relation graphs

In our algorithmic time analysis, as well as the improved pseudofactorization algorithm discussed in Section 6, we will make use of what we call *edge-relation graphs*.

Definition 2.9 (edge-relation graph G_R)**.** *Given a graph $G = (V(G), E(G), w_G)$ and a symmetric binary relation R on $E(G)$ whose transitive closure is also reflexive, the edge-relation graph of G with respect to R is the unweighted graph $G_R = (V(G_R), E(G_R))$ defined as follows.*

For each $xy \in E(G)$, there is a vertex u_{xy} (or u_{yx}, exactly one vertex for each edge) in V_R. Two nodes u_{xy} and u_{ab} in G_R are adjacent if and only if xy R ab in G.

The connected components of the graph G_R are then the equivalence classes of the edges of the original graph under the transitive closure of R.

3 Pseudofactorization of weighted graphs

In this section, we will discuss a method for pseudofactoring weighted graphs in polynomial time. To begin, we discuss current techniques for pseudofactoring unweighted graphs, and we then show that one of these techniques can also be applied to weighted graphs.

Algorithm 1 is a generalized version of the $O(m^2)$-time algorithm presented by Graham and Winkler [12]. Its inputs are a graph and an equivalence relation θ^*, $(\theta \cup \tau)^*$, or θ_T^* on the edges of the graph (the latter relations to be discussed in subsequent sections), and it outputs a set of graphs. Graham and Winkler [12] showed that when the input is (G, θ^*) for an unweighted graph G, the output is an irreducible pseudofactorization of G. Figure 7 shows an example of an application of this algorithm to a weighted graph when the input relation is θ^*.

Informally, the algorithm finds the equivalence classes of the graph edges under the given relation. Then, for each equivalence class E_k, it constructs the subgraph of G with all edges in E_k removed. The connected components of this subgraph are subsequently used to form the corresponding output graph. In this section, we show that this algorithm works for weighted graphs.

Algorithm 1 Algorithm for breaking up a graph over a relation θ^*, $(\theta \cup \tau)^*$, or θ_T^*

Input: A weighted graph G and an equivalence relation $R \in \{\theta^*, (\theta \cup \tau)^*, \theta_T^*\}$ defined on G (where T is the output of Algorithm 2 on G and some spanning tree if $R = \theta_T^*$).
Output: A set $\mathbf{G}=\{G_1^*, G_2^*, ..., G_m^*\}$

1: Set $\mathbf{G} \leftarrow \emptyset$
2: Find the set of equivalence classes of R, $\{E_1, E_2, ..., E_m\}$
3: Set $\mathbf{E} \leftarrow \{E_1, E_2, ..., E_m\}$
4: **for** $E_k \in \mathbf{E}$ **do**
5: Let $w_{G'_k}$ be w_G restricted to the edges in $E(G) \setminus E_k$
6: Set $G'_k \leftarrow (V(G), E(G) \setminus E_k, w_{G'_k})$
7: Set $\mathbf{C} \leftarrow \{C_1, C_2, \ldots, C_\ell\}$ (the set of connected components of G'_k)
8: Create a new graph G_k^*
9: Set $V(G_k^*) \leftarrow \{a \,|\, C_a \in \mathbf{C}\}$
10: Set $E(G_k^*) \leftarrow \{ab \,|$ there is an edge in E_k between C_a and $C_b\}$
11: **for** $ab \in E(G_k^*)$ **do**
12: Set w_{ab} to be the weight of an edge in G between C_a and C_b
13: Set $w_{G_k^*}(ab) \leftarrow w_{ab}$
14: Set $\mathbf{G} \leftarrow \mathbf{G} \cup \{G_k^*\}$
15: Return $\mathbf{G}$.

Algorithm 1 implies a set of natural embeddings between the input graph G and each of the output graphs, which is formalized here.

Definition 3.1. *Given a graph G and relation $R \in \{\theta^*, (\theta \cup \tau)^*, \theta_T^*\}$, let $\{G_1^*, G_2^*, \ldots, G_m^*\}$ be the output of Algorithm 1 on (G, R), and let $G^* = \Pi_{i=1}^m G_i^*$. Then the* natural embedding of G under R *is the function that maps each $u \in V(G)$ to $(u_1^*, \ldots, u_m^*) \in V(G^*)$ such that u_i^* is the vertex in $V(G_i^*)$ corresponding to the connected component of G'_i containing u (i.e. the vertex created in line 9 of the algorithm to represent the connected component of G'_i that u is a member of). If σ is the natural embedding of G under R, then we*

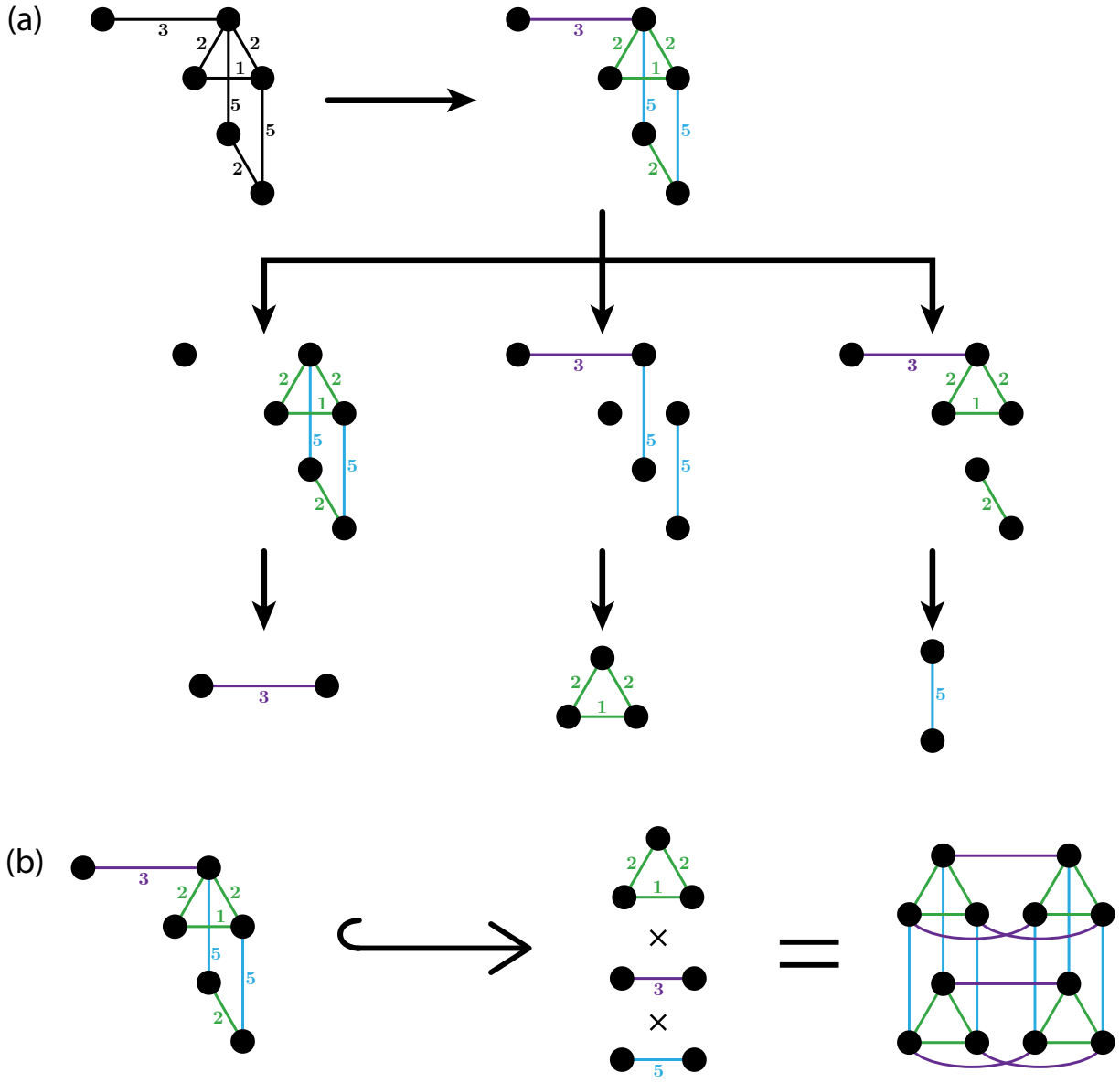

Figure 7: Illustration of Algorithm 1 on a weighted graph of six nodes.
(a) The algorithm first finds the equivalence classes of θ^*, shown here with distinct colors. The algorithm then considers the remaining graph when each equivalence class is removed and uses that to construct the final output graphs. In this case, there are three θ^* equivalence classes and thus three pseudofactors.
(b) This subfigure shows in the first panel the same graph as in (a), in the second panel it shows its irreducible pseudofactorization, and in the third panel it shows Cartesian product of those pseudofactors, of which the original graph is an isometric subgraph.

use σ_i to refer to the projection of σ onto its i-th element, so that $\sigma_i(u) = u_i^*$.

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 θ^*, the output is an irreducible pseudofactorization of G , and that this irreducible pseudofactorization is unique for any G. Note that the relation θ_T^* can be used with Algorithm 1 for more efficient pseudofactorization, and we consider this in Section 6.

3.1 Testing irreducibility

We first show that if all edges in a graph are in the same equivalence class of θ^*, then the graph is irreducible.

Lemma 3.2. *For any graph $G = (V(G), E(G), w_G)$ and any $uv, xy \in E(G)$, if $uv\,\theta^*\,xy$, then for any pseudofactorization $\{G_1, \ldots, G_m\}$ of G and isometric embedding $\pi : V(G) \to V(\Pi_{i=1}^m G_i)$, uv and xy must have parent edges under π 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 isometric embedding π , and let $\pi(a) = (a_1, a_2, \ldots, a_m)$ for $a \in V(G)$.

We first prove the lemma for $uv\,\theta\,xy$. Assume for contradiction that xy and uv have parent edges under π in different pseudofactors. Since xy and uv are edges, by Definition 2.4, $\pi(x)\pi(y)$ and $\pi(u)\pi(v)$ are also edges, and by the definition of the Cartesian product, 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 have $l \neq j$ (see Definition 2.2).

Now we consider $[d(x,u) - d(x,v)] - [d(y,u) - d(y,v)]$. Since π is an isometric embedding, we can rewrite this using the distance metric for $\Pi_i G_i$, as follows:

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

Term i in this sum is 0 if $u_i = v_i$ or if $x_i = y_i$, as $u_i = v_i$ would cause $d_i(x_i, u_i) - d_i(x_i, v_i) = d_i(y_i, u_i) - d_i(y_i, v_i) = 0$ and $x_i = y_i$ would cause $d_i(x_i, u_i) - d_i(x_i, v_i) = d_i(y_i, u_i) - d_i(y_i, v_i)$. 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 conclude that if $xy\,\theta\,uv$, then xy and uv must have parent edges under π in the same pseudofactor graph.

To prove the lemma when $uv\,\theta^*\,xy$, observe that $uv\,\theta^*\,xy$ implies that there is a sequence of edges, $e_1 = uv, e_2, \ldots, e_l = xy$ for which $e_k\,\theta\,e_{k+1}$ for all $1 \leq k < l$. By the above reasoning, parent edges under π for adjacent pairs of edges belong to the same pseudofactor, so the same is true for uv and xy. □

We know that Algorithm 1 outputs a set whose size is the number of equivalence classes of θ^*. Thus, from the preceding lemma, if Algorithm 1 outputs a set containing a single graph, then the input graph must be irreducible. We summarize this finding in Corollary 3.3.

Corollary 3.3. *For any graph $G = (V(G), E(G), w_G)$, if all edges in G are in the same equivalence class of θ^*, the graph is irreducible. Thus, if Algorithm 1 outputs a single graph on input (G, θ^*), G is irreducible.*

Corollary 3.3 is extended in the following subsection with Corollary 3.11, which proves that in fact a graph is irreducible if and only if it has a single equivalence class of θ^*.

3.2 An algorithm for pseudofactorization

In this section, we will show that Algorithm 1 with θ^* as the input relation can be used to pseudofactor a minimal weighted graph. Many of the lemmas used in this section have parallels to those that we will use to prove factorization. First, we show in Lemma 3.4 that this algorithm is well-defined for the inputs we are considering (a minimal graph and the relation θ^*). Throughout this section, notation is used as in Algorithm 1.

We first prove that each step of Algorithm 1 is deterministic. Specifically, the following lemma proves that the choice of edge between C_a and C_b in lines 12-13 of Algorithm 1 does not affect the output.

Lemma 3.4. *Let $G = (V(G), E(G), w_G)$ be a minimal graph serving as input to Algorithm 1 with input relation θ^*. 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 to be the distance function on G. First, we show by contradiction 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. 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 8(a)) consisting entirely of edges not in E_k (and thus not related to uv by θ). 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 u cannot have edges to two distinct $v, v' \in V(C_b)$ 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, and thus that for any $uv \in E(G)$, $w_G(uv) = d_G(u, v)$. 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. The second is $Q_b = (v = q_0^b, q_1^b, \ldots, q_t^b = y)$, which will consist entirely of edges in C_b (represented in Figure 8(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. Construct the cycle of L edges , $Q = (u, v, Q_b, Q_a^{-1}) = (u = q_0, v = q_1, q_2, \ldots, q_L = u)$ (i.e. the cycle going through uv, following Q_b, through yx, and then following Qa backwards back to u).

Now consider the sum

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

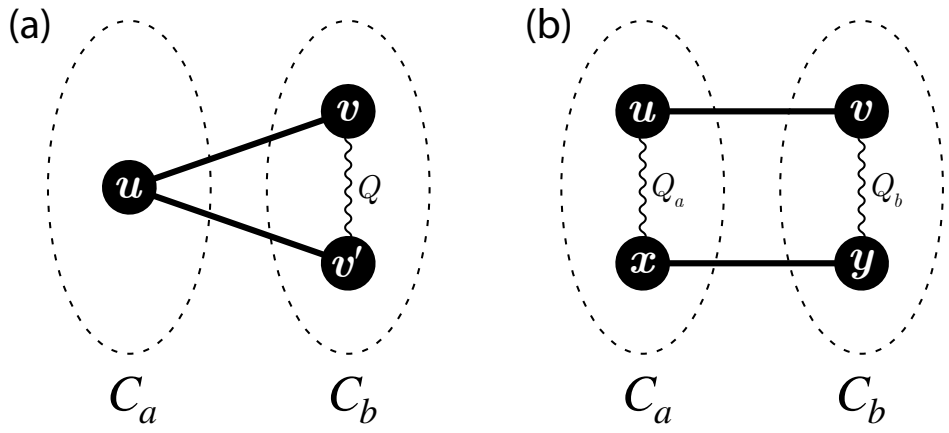

Figure 8: Paths described in the proof of Lemma 3.4. (a) A graph in which $u \in C_a$ has an edge to distinct $v, v' \in C_b$, which the proof of Lemma 3.4 shows is impossible. (b) Definitions of Q_a and Q_b from the proof when considering $u \in C_a, v \in C_b$.

where the equality comes from telescoping. Because uv is not related by θ to any edge on this cycle except itself and possibly xy, all terms cancel except the ones for the ones for edges uv and xy. Thus we have

$$[d(u,y) - d(u,x)] - [d(v,u) - d(v,x)] + [d(u,u) - d(u,v)] - [d(v,u) - d(v,v)] = 0,$$

which using the facts that $w_G(uv) = d(u,v)$ and $d(u,u) = d(v,v) = 0$, simplifies to

$$d(u,y) - d(u,x) + d(v,x) - d(v,y) = 2w_G(uv).$$

Then taking the absolute values of both sides, we get

$$\begin{aligned} 2w_G(uv) &= |d(u,y) - d(u,x) + d(v,x) - d(v,y)| \\ &\leq |d(u,y) - d(u,x)| + |d(v,x) - d(v,y)| \\ &\leq d(x,y) + d(x,y) \\ &= 2w_G(xy). \end{aligned}$$

Thus $w_G(uv) \leq w_G(xy)$. Through symmetric analysis, $w_G(xy) \leq w_G(uv)$, so we must have $w_G(xy) = w_G(uv)$. This implies the lemma. □

With the next lemma, we introduce two general facts about the relationship between paths and equivalence classes of θ^* that will be used in later claims. These are similar to claims presented by Graham and Winkler [12] but we generalize them to weighted graphs.

Lemma 3.5. *Given a weighted graph $G = (V(G), E(G), w_G)$ and a set $\{E_1, E_2, \ldots, E_m\}$ that is the set of equivalence classes on $E(G)$ under equivalence relation θ^*, 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 statement implies the first one in the case of minimal graphs, since in those cases any edge uv is a shortest path between u and v. However, this is not the case for general weighted graphs and we will use this lemma when we discuss factorization of general 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 $q_{i-1}q_i$ 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.

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

   We know that because P is a shortest path, $d(u,v) = d(u,p_{\ell-1}) + d(v,p_{\ell-1})$ and $d(u,v) = d(u,p_\ell) + d(v,p_\ell)$. Substitution yields:

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

   This value is $-2w_G(p_{\ell-1}p_\ell) \neq 0$ because $p_{\ell-1}p_\ell$ is an edge on a shortest path from u to v, and thus some edge on Q is related to $p_{\ell-1}p_\ell$ by θ.

□

Now, we show that G is isomorphic to an isometric subgraph of $\Pi_i G_i^*$ (the Cartesian product of the algorithm's output graphs). To do so, we define a mapping π from G to $\Pi_i G_i^*$ with the goal of showing that π is an injection and that shortest paths in G correspond to shortest paths under π.

We notice that for each vertex in G_i^* created by Algorithm 1 in line 9, there is at least one vertex in G_i' and thus in G corresponding to that vertex. Additionally, for each edge of G_i^* created in line 10, there is a corresponding edge in G between connected components of G_i'. Because of this, we see that criteria 3 and 4 of Definition 2.4 are met by this mapping. We now show that π is an injection.

Claim 3.6. *For minimal graph $G = (V(G), E(G), w_G)$, let $\pi : V(G) \to V(\Pi_{i=1}^m G_i^*)$ be the natural embedding of G under θ^* (see Definition 3.1), where $\{G_1^*, G_2^*, \ldots, G_m^*\}$ is the output of Algorithm 1 on input (G, θ^*). Then π is an injection.*

Proof. We show that $\pi(u) \neq \pi(v)$ for all $u, v \in V(G)$, $u \neq v$. 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 3.5 says that all paths between the two vertices have an edge in E_k. Since $u \neq v$ there is at least one edge in P, and thus there is at least one index k for which $\pi_k(u) \neq \pi_k(v)$. □

In many of our proofs, manipulation of the sum of theta-differences over a path 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 sum of theta-differences between uv and the sequences of edges along a path, the contribution from the edges in each equivalence class does not depend on the path taken.

We define the following notation for paths in a graph.

Definition 3.7. *For path $P = (u = p_0, p_1, \ldots, p_n = v)$ in $G = (V(G), E(G), w_G)$, define P^k to be the sequence of edges in P that are also in E_k, where E_k is one of the equivalence classes of E under θ^*. Define T_k^P as $T_k^P := \sum_{p_i p_{i+1} \in P^k} [d_G(u, p_i) - d_G(u, p_{i+1})] - [d_G(v, p_i) - d_G(v, p_{i+1})]$.*

Lemma 3.8. *Let $G = (V(G), E(G), w_G)$ be a weighted graph and $P = (u = p_0, p_1, \ldots, p_n = v)$ and $Q = (u = q_0, q_1, \ldots, q_t = v)$ be two paths in G from u to v. Then $T_k^P = T_k^Q$.*

Proof. We consider the following equations. For brevity, in this proof we define $d := d_G$.

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

The second equality comes from telescoping the inner sum, the third 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 fourth equality comes from switching the order of the sums, the fifth comes from switching the order of the terms in the summand, and the sixth again from the fact that only edges in E_k contribute. The seventh equality is by telescoping, and the last equality is by definition of T_k^Q. □

Theorem 3.9. *Let $G = (V(G), E(G), w_G)$ be a minimal graph and $\pi : V(G) \to V(\Pi_{i=1}^m G_i^*)$ be the natural embedding of G under θ^*, where $\{G_1^*, G_2^*, \ldots, G_m^*\}$ is the output of Algorithm 1 on input (G, θ^*). Then for*

any edge $ab \in E(G)$ it must be that $\pi(a)\pi(b) \in E(\Pi_i G_i^*)$, and for any pair of vertices $u, v \in V(G)$ it must be that $d_G(u,v) = d_{\Pi_i G_i^*}(\pi(u), \pi(v))$ and $w_G(uv) = w_{\Pi_i G_i^*}(\pi(u)\pi(v))$. It follows that for an input (G, θ^*) with minimal graph G, Algorithm 1 produces a pseudofactorization of G.

Proof. First, we show that if $ab \in E(G)$, then $\pi(a)\pi(b) \in E(\Pi_i G_i^*)$ and $w_G(ab) = w_{\Pi_i G_i^*}(\pi(a)\pi(b))$. Assume $ab \in E_j$ and let G_j' be as defined in line 6 of the algorithm. Now, 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^*, respectively, as defined in line 7. We know that $E(G)$ has an edge between a vertex in $C_{\pi_j(a)}$ and a vertex in $C_{\pi_j(b)}$ of weight $w_G(ab)$. By Lemma 3.4, 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 edges in G_j^* in lines 10-13, there is an edge between $\pi_j(a)$ and $\pi_j(b)$ of that weight.

Let $d := d_G$ and $d^* := d_{\Pi_i G_i^*}$.

Assume $d(u,v) > d^*(\pi(u), \pi(v))$ for at least one pair of vertices $u, v \in V(G)$. Choose $u, v \in V(G)$ to be one such pair of vertices having the smallest value of $d^*(\pi(u), \pi(v))$.

First, we show that any vertex on a shortest path from $\pi(u)$ to $\pi(v)$ in the product graph cannot be in the image of π. Consider a vertex $\pi(u')$ that is in the image of π and on such a shortest path , which implies $d^*(\pi(u), \pi(u')) + d^*(\pi(u'), \pi(v)) = d^*(\pi(u), \pi(v))$. Additionally, we have $d^*(\pi(u), \pi(u')) = d(u, u')$ because $d^*(\pi(u), \pi(u')) < d^*(\pi(u), \pi(v))$ and we similarly get $d^*(\pi(u'), \pi(v)) = d(u', v)$. This yields 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. This contradiction implies that such a u' cannot exist and thus no vertex in the image of π can be on a shortest path between $\pi(u)$ and $\pi(v)$ in $\Pi_i G_i^*$.

We have shown that no vertex 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 vertex must exist.

Let uu' be the first edge on a shortest path P from u to v and assume without loss of generality that $uu' \in E_k$. We show that the image under π_k of P_k is a shortest path in G_k^*. This will imply that $\pi(u)\pi(u')$ is an edge on a shortest path from $\pi(u)$ to $\pi(v)$ in $\Pi_i G_i^*$, as shortest paths in $\Pi_i G_i^*$ can be formed by concatenating paths in $\Pi_i G_i^*$ whose edges' parent edges form shortest paths in the G_i^*. Consider a path $Q^* = (\pi_k(u) = q_0, q_1, \ldots, q_n = \pi_k(v))$ in G_k^* 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', each of which consists of one or more nodes in G, for each j there exists v_j such that $\pi_k(v_j) = q_j$ (i.e. there is a vertex 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 $(x_j^{j+1}, x_{j+1}^{j+1})$ be this pair. We have as a superscript the index of the edge we're considering and as a subscript the index of the connected component in G_k'.

We will construct the path Q from u to v in G shown in Figure 9(a). Let Q_j be a path from x_j^j to x_j^{j+1} 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 without any edges from E_k. 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 $Q := Q_0Q_1Q_2 \cdots Q_n = (u = f_0, f_1, \ldots, f_m = v)$. A visualization of this construction is given in Figure 9. We consider the sums T_k^Q and T_k^P as in Definition 3.7.

We know from Lemma 3.8 that $T_k^P = T_k^Q$. We use this to get

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

Since P is a shortest path, the theta-difference for every edge on this path must contribute 2 times its weight to the overall sum (or else we would not be able to get to the total) so we get $T_k^P = \sum_{p_ip_{i+1} \in P^k} 2w(p_ip_{i+1}) \leq \sum_{f_if_{i+1} \in Q^k} 2w(f_if_{i+1})$. Thus, the image under π_k of P^k from u_k to v_k is a shortest path in G_k^*. This means u_ku_k' is a first edge on a shortest path from $\pi_k(u)$ to $\pi_k(v)$ in G_k^*, which means $\pi(u)\pi(u')$ is a first edge on a shortest path from $\pi(u)$ to $\pi(v)$ in $\Pi_iG_i^*$ - 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 isomorphic to an isometric subgraph of $\Pi_iG_i^*$, where the G_i^* are the graphs in the set output by Algorithm 1 on input (G, θ^*). This shows that criteria 1 and 2 of the pseudofactorization definition are satisfied. Additionally, we note that criteria 3 and 4 are satisfied by the fact that Algorithm 1 only creates vertices and edges in each G_i^* that are in the image of π or a parent edge of an edge in G under π, respectively. □

The following lemma proves that the pseudofactorization given by Algorithm 1 under the conditions of Theorem 3.9 is, in fact, irreducible.

Lemma 3.10. *If (G, θ^*) is the input to Algorithm 1 for minimal graph G, the output graphs $\{G_1^*, \ldots, G_m^*\}$ are all irreducible.*

Proof. Using Corollary 3.3, we need only show that the edges of each pseudofactor G_i^* are all related by θ^*. Consider two edges $ab, a'b' \in E(G)$ with $ab\,\theta^*\,a'b'$. We know that there must exist a sequence of edges $ab = a^1b^1, a^2b^2, \ldots, a^tb^t = a'b'$ such that $a^ib^i\,\theta\,a^{i+1}b^{i+1}$ for all $i, 1 \leq i < t$. From Lemma 3.2 we know that all edges in this sequence must have parents in the same pseudofactor, which we will call pseudofactor j. Let $u_j := \pi_j(u)$ for all $u \in V(G)$. We only need to show that for each i, $a_j^ib_j^i\,\theta\,a_j^{i+1}b_j^{i+1}$ in order to show that $a_jb_j\,\theta^*\,a_j'b_j'$.

For simplicity, we will consider any two edges $xy, uv \in E(G)$ such that $xy\,\theta\,uv$ and show that if their parent edges under π are in pseudofactor j, then $x_jy_j\,\theta\,u_jv_j$. For brevity, define $d := d_G$, $d_i := d_{G_i^*}$, and $d^* := d_{\Pi_iG_i^*}$. We get the following:

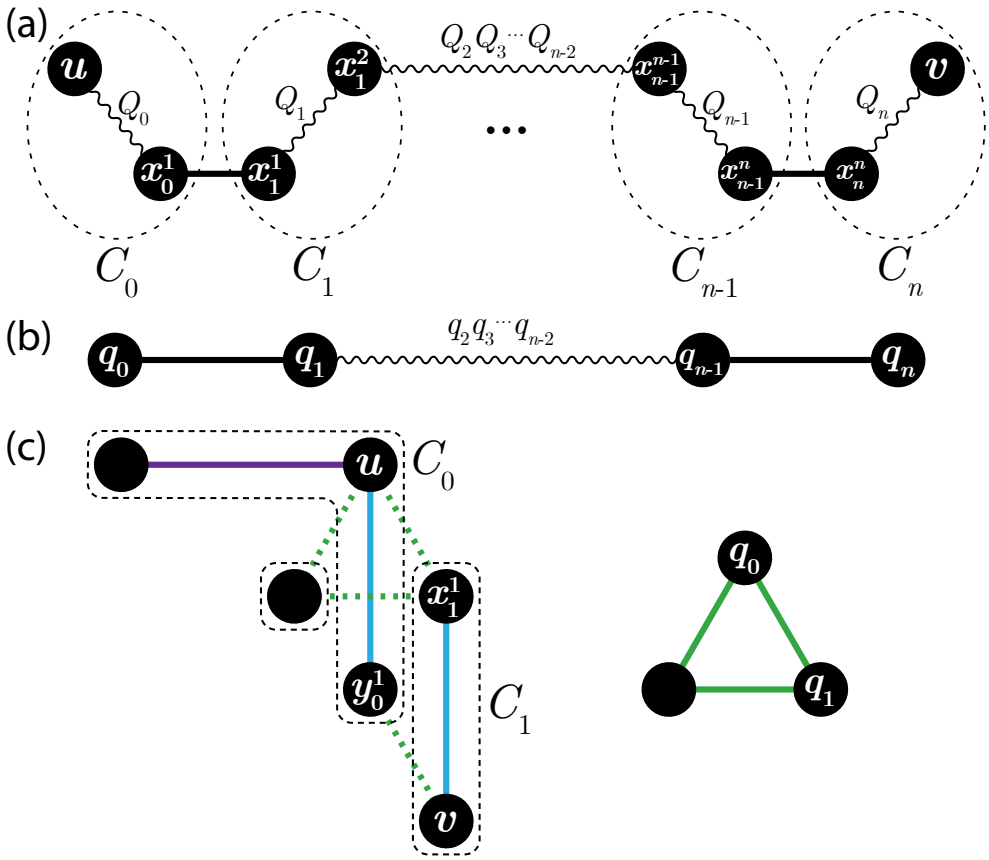

Figure 9: A visualization of the path constructed in the proof of Theorem 3.9.
(a) Each C_i is a connected component in G'_k, where E_k is the equivalence class we consider in the theorem. Starting at u, we construct a path Q_0 through edges in C_0 to x_0^1, which is some vertex in C_0 with an edge to C_1. We similarly construct Q_1 from x_1^1 to x_1^2 and so on. The path $Q_2 \cdots Q_{n-2}$ is the concatenation of the paths through connected components C_2 through C_{n-2}.
(b) Part of G_k^* for the graph in (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 (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^*.
(c) An example of this process applied to the graph from Figure 7, when considering the green pseudofactor shown on the right. Edge weights are omitted for simplicity. When the graph on the left is isometrically embedded into the product of its pseudofactors (see Figure 7b) 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, although existence of the path Q is guaranteed, Q is not necessarily unique.

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

where the second to last equality is due to the fact that xy is an edge with a parent under π in pseudofactor j. Since every pair edges in G_j^* is the parent of an edge in E_j, we get that any pair of edges in G_j^* is related by θ^*. □

Thus, from Theorem 3.9 and Lemma 3.10, we conclude that Algorithm 1 with the input (G, θ^*) for a minimal weighted G produces an irreducible pseudofactorization of G. We additionally note that because the number of graphs produced by Algorithm 1 is exactly the number of equivalence classes of θ^* on the input graph, from Theorem 3.9 and Corollary 3.3, we get that a graph is irreducible if and only if there is exactly one θ^* equivalence class on its edges.

Corollary 3.11. *A weighted graph G is irreducible if and only if there is exactly one θ^* equivalence class on its edges.*

3.3 Uniqueness of pseudofactorization

Here, we prove that the irreducible pseudofactorization of any minimal graph is unique in the following sense: for any two irreducible pseudofactorizations of a minimal graph G, the two sets of pseudofactors are equal up to graph isomorphism. We call the irreducible pseudofactorization generated by Algorithm 1 with input (G, θ^*) the *canonical pseudofactorization* or *canonical pseudofactors* of G.

We note that uniqueness is guaranteed in part because conditions 3 and 4 of Definition 2.4 require that no unnecessary vertices or edges are included in the pseudofactors. Of course, if these conditions are removed, an arbitrary number of vertices and edges may be added without affecting isometric embeddability into the product and the uniqueness property no longer holds. In fact, removing conditions 3 and 4 make it so that no graph is irreducible, as for any graph G, we can construct a graph G' that is identical to G but contains an extra vertex x with an edge to a single vertex $u \in V(G)$, with weight 1. Without conditions 3 and 4, then G' is a pseudofactor of G.

Theorem 3.12. *Let $G = (V(G), E(G), w_G)$ be a minimal weighted graph and let $\{G_1, \ldots, G_p\}$ and $\{H_1, \ldots, H_r\}$ be two irreducible pseudofactorizations of G. Then $p = r$ and the pseudofactors may be reordered so that G_i is isomorphic to H_i for all $i, 1 \leq i \leq p$.*

Proof. This proof follows the reasoning of Graham and Winkler [12], with modifications for weighted graphs.

Number the θ^* equivalence classes of $E(G)$ as $E_1, \ldots, E_m$. Let $\pi : V(G) \to V\left(\prod_{i=1}^{p} G_i\right)$ and $\rho : V(G) \to V\left(\prod_{i=1}^{r} H_i\right)$ be isometric embeddings of G into $\prod_{i=1}^{p} G_i$ and $\prod_{i=1}^{r} H_i$, respectively, with $\pi = (\pi_1, \ldots, \pi_p)$ and $\rho = (\rho_1, \ldots, \rho_r)$. Note that by the definition of pseudofactorization, if $u, v \in V(G)$ are adjacent then

$\pi(u), \pi(v)$ are adjacent, and there is exactly one i such that $\pi_i(u) \neq \pi_i(v)$; edge $\pi_i(u)\pi_i(v)$ is the parent edge under π of $uv \in E(G)$. The same reasoning applies to $\rho(u)$ and $\rho(v)$.

First, we show that $uv\,\theta^*\,u'v'$ if and only if they have parent edges belonging to the same pseudofactor. That is, $uv\,\theta^*\,u'v'$ if and only if $\pi_i(u) \neq \pi_i(v) \iff \pi_i(u') \neq \pi_i(v')$ for all $i, 1 \leq i \leq p$. For the forward case, when $uv\,\theta^*\,u'v'$ then Lemma 3.2 applies and we are done. For the reverse case, let uv and $u'v'$ have parent edges in G_j and note that because G_j is irreducible, it has exactly one equivalence class of θ^* on its edges, since Algorithm 1 outputs a pseudofactorization with one pseudofactor per equivalence class of θ^*. Further, we have that

$$\begin{aligned}
&\big[d_G(u,u') - d_G(u,v')\big] - \big[d_G(v,u') - d_G(v,v')\big] && (2)\\
&= \big[d_G(\pi(u),\pi(u')) - d_G(\pi(u),\pi(v'))\big] - \big[d_G(\pi(v),\pi(u')) - d_G(\pi(v),\pi(v'))\big] && (3)\\
&= \sum_{i=1}^{p} \big[d_{G_i}(\pi_i(u),\pi_i(u')) - d_{G_i}(\pi_i(u),\pi_i(v'))\big] - \big[d_{G_i}(\pi_i(v),\pi_i(u')) - d_{G_i}(\pi_i(v),\pi_i(v'))\big] && (4)\\
&= \big[d_{G_j}(\pi_j(u),\pi_j(u')) - d_{G_j}(\pi_j(u),\pi_j(v'))\big] - \big[d_{G_j}(\pi_j(v),\pi_j(u')) - d_{G_j}(\pi_j(v),\pi_j(v'))\big] && (5)
\end{aligned}$$

where the first equality holds because π is an isometric embedding, the second equality holds because of the path decomposition property of the Cartesian graph product, and the third equality holds because each summand is nonzero only if $\pi_i(u') \neq \pi_i(v')$. Thus, by the definitions of θ and θ^*, $uv\,\theta^*\,u'v'$. Since all edges in G_j are related by θ^*, then any edges in G to which they are a parent are also related by θ^*.

This shows there is a bijection from the θ^* equivalence classes to the set of G_i, as well as to the set of H_i. So $p = r = m$.

Renumber both pseudofactorizations so that G_i and H_i contain only parent edges under π and ρ, respectively, of the edges in E_i, $1 \leq i \leq m$. Fix j, $1 \leq j \leq m$, and $u \in V(G)$. Take any u' for which there is a path P to u not using any edge in E_j. Clearly, $\pi_j(u) = \pi_j(u')$ because no edges in this path have parent edges in G_j and so π_j is constant along this path. Alternatively, take any u' for which every path to u has at least one edge in E_j and consider a shortest path P from u to u'. Let c_i be the sum of the edge weights of the edges along P in E_i. Observe:

$$d_G(u,u') = \sum_{i=1}^{n} d_{G_i}(\pi_i(u),\pi_i(u')) = \sum_{i=1}^{n} c_i \tag{6}$$

Each $d_{G_i}(\pi_i(u),\pi_i(u')) \leq c_i$ because the edges along p in E_i trace out a path in G_i. Thus, for the sums to be equal $d_{G_i}(\pi_i(u),\pi_i(u')) = c_i$. So in this case $d_{G_i}(\pi_i(u),\pi_i(u')) = c_j > 0$ and $\pi_j(u) \neq \pi_j(u')$. Thus, u and u' are connected by a path without edges in E_j if and only if $\pi_j(u) = \pi_j(u')$. Now let V_u be the set of all $u' \in V(G)$ for which there is a path from u to u' without an edge in E_j. Then by this reasoning $V_u = \{u' \in V(G) : \pi_j(u) = \pi_j(u')\} = \{u' \in V(G) : \rho_j(u) = \rho_j(u')\}$. Thus, there is a single $\rho_j(u) \in V(H_j)$ for each $\pi_j(u) \in V(G_j)$. Let $f : V(G_j) \to V(H_j)$ map each vertex of G_j to the corresponding vertex in H_j given by V_u.

Each edge $\pi_j(u)\pi_j(v)$ in G_j is the parent of an edge in G (see condition 4 of Definition 2.4), which we assume without loss of generality to be $uv \in E_j$. As observed above, $uv \in E_j$ must have a parent edge under ρ, $\rho_j(u)\rho_j(v) \in E(H_j)$. From the definition of pseudofactorization, the edge weights of $\pi_j(u)\pi_j(v)$, uv, and $\rho_j(u)\rho_j(v)$ must be equal. Thus, for every edge $\pi_j(u)\pi_j(v) \in E(G_j)$ there is an edge $f(\pi_j(u))f(\pi_j(v)) = \rho_j(u)\rho_j(u') \in E(H_j)$, and, by symmetry, the converse is true. So G_j is isomorphic to H_j. □

4 Factorization of weighted graphs

The purpose of this section is to prove that we can factor graphs using Algorithm 1. We note that this section focuses on weighted graphs, which need not be minimal.

Recall from Section 2 that two edges $uv, u'v'$ of a weighted graph G satisfy the square property if the two edges are two adjacent edges of a 4-cycle in which the opposite edges have the same weight and are related by θ. Recall also that two edges are related by τ if and only if they share an endpoint and do *not* satisfy the square property. We will see below that the relation $\theta \cup \tau$ has a similar connection to factorization as the θ relation has to pseudofactorization.

4.1 Testing primality

Analogously to showing irreducibility with respect to pseudofactorization, in this section, we show in Lemma 4.1 that a graph is prime if it has one equivalence class under $(\theta \cup \tau)^*$ on its edges. Thus, Lemma 4.1 is analogous to Lemma 3.2.

Lemma 4.1. *Let $G = (V(G), E(G), w_G)$ be a weighted graph. For $uv, xy \in E(G)$, if $uv\,\theta\,xy$ or $uv\,\tau\,xy$, then for any factorization of G, uv and xy must have parent edges in the same factor under α, where α is an isomorphism from G to the product of the factors. From this, we conclude that if $uv\,(\theta \cup \tau)^*\,xy$ then uv and xy have parent edges under α 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 $uv\,\theta\,xy$. Since α is an isomorphism, it must be an isometric embedding into an isometric subgraph of the product graph. So by Lemma 3.2, $uv\,\theta\,xy$ implies that their parent edges under α must be in the same pseudofactor.

2. Consider $uv\,\tau\,uv'$ and assume for contradiction that uv and uv' have parent edges under α in different factors. We know 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 $l \neq j$. Without loss of generality, assume $j < l$.

   Now, consider the vertex $(u_1, \ldots, u_{j-1}, v_j, u_{j+1}, \ldots, u_{l-1}, v'_l, u_{l+1}, \ldots, u_m)$ in $\Pi_i G_i$ (ie the vertex that matches $\alpha(u)$ on all indices except j and l, where it matches v_j and v'_l respectively). Since the vertex set of $\Pi_i G_i$ is the Cartesian product of the $V(G_i)$, this vertex 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 vertex. We also know that x must have an edge to v since $x_i = v_i$ for all $i \neq l$ and $v_l v'_l = u_l v'_l \in E(G_l)$. It also has an edge to v' since $x_i = v'_i$ for all $i \neq j$ and $v_j v'_j = v_j u_j \in E(G_j)$.

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

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

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 that all edge weights are positive. 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 have parent 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 have parent edges in the same factor. □

From this, we get the following corollary, which states that if all edges are in the same equivalence class of $(\theta \cup \tau)^*$, the corresponding graph is prime.

Corollary 4.2. *For any weighted graph $G = (V(G), E(G), w_G)$, if all edges in G are in the same equivalence class of $(\theta \cup \tau)^*$, the graph is prime. Thus, if Algorithm 1 outputs a single graph on input $(G, (\theta \cup \tau)^*)$, G is prime.*

Corollary 4.2 is extended in the following subsection with Corollary 4.8, which proves that in fact a graph is irreducible if and only if it has a single equivalence class of $(\theta \cup \tau)^*$.

4.2 An algorithm for factorization

In this section, we show that Algorithm 1 with inputs weighted graph $G = (V(G), E(G), w_G)$ (not necessarily minimal) and $(\theta \cup \tau)^*$ produces a prime factorization of G. We first show that the algorithm in question is well-defined for general weighted graphs and with the input relation $(\theta \cup \tau)^*$. We use Lemma 4.3 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 defined in Algorithm 1.

Lemma 4.3. *Let $G = (V(G), E(G), w_G)$ be a weighted graph serving as input to Algorithm 1 with input relation $(\theta \cup \tau)^*$. If C_a, C_b are distinct 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. 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 the first part of the proof of Lemma 3.4 so we do not repeat it here.

Now we expand on that to show that each $u \in C_a$ has an edge to exactly one vertex 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.

We proceed by induction on the number of edges in the path $P(u)$. In the base case, there are zero edges in $P(u)$. In this case, we must have $u = x$. By assumption, 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 a path of n edges to x, the lemma holds. We let u be a vertex such that $P(u)$ has $n+1$ edges and let u' be the second vertex on this path. By definition, we know that $P(u')$ has n edges. By the inductive hypothesis, there exists $v' \in C_b$ such that $u'v' \in E(G)$ and $w(u'v') = w_{ab}$. 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 vertex such that $uu'v'v$ is a square with opposite edges of equal weight and related by θ. Because $uu' \notin E_k$, we have $vv' \notin E_k$ and since $v' \in C_b$, we must have $v \in C_b$. This means uv is an edge between C_a and C_b with weight w_{ab}, proving the lemma. □

We now write Lemma 4.4, which is identical to Lemma 3.5, but now refers to the equivalence classes of our new relation. The proof of this lemma is identical to that of Lemma 3.5, so we do not repeat it here.

Lemma 4.4. *Given a weighted graph $G = (V(G), E(G), w_G)$ and the set $\{E_1, E_2, \ldots, E_m\}$ of equivalence classes on $E(G)$ under equivalence relation $(\theta \cup \tau)^*$, 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.*

We now want to show that G is isomorphic to $\Pi_i G_i^*$, so we define an isomorphism $\alpha : V(G) \to V(\Pi_i G_i^*)$.

In analogy to Section 3, we will show that α is an isomorphism. To do this, we show that α is a bijection, 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. We use Lemma 4.5 to show first that α is a bijection.

Lemma 4.5. *For weighted graph $G = (V(G), E(G), w_G)$, let $\alpha : G \to \Pi_{i=1}^m G_i^*$ be the natural embedding of G under $(\theta \cup \tau)^*$, where $\{G_1^*, G_2^*, \ldots, G_m^*\}$ is the output of Algorithm 1 on input $(G, (\theta \cup \tau)^*)$. Then α is a bijection.*

Proof. First, we show that α is an injection (i.e. $\alpha(u) \neq \alpha(v)$ for all $u, v \in V(G)$, $u \neq v$). This is identical to the proof that π is an injection in Claim 3.6, 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. If $u \neq v$, then there is at least one edge e in P. Assume $e \in E_k$. By Lemma 4.4, all paths from u to v have an edge in E_k, so u and v are in different connected components of G'_k and $\alpha_k(u) \neq \alpha_k(v)$.

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 vertex in the product not in the image of α. Let $(u_1, u_2, \ldots, u_m)$ be one such vertex with an edge to a vertex $(u_1, \ldots, x_k, \ldots, u_m)$ in the image of α whose pre-image is $x \in V(G)$. 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} (connected components of G'_k that were responsible for the creation of nodes u_k and v_k, respectively, in G_k^* in line 9 of Algorithm 1) of weight w_{xu} and Lemma 4.3 tells us that every vertex in C_{x_k} has an edge of that weight to exactly one vertex in C_{u_k}. Let u' be the vertex 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. □

The next theorem shows that α is an isomorphism.

Theorem 4.6. *Let $G = (V(G), E(G), w_G)$ be a weighted graph and $\alpha : V(G) \to V(\Pi_{i=1}^{m} G_i^*)$ be the natural embedding of G under $(\theta \cup \tau)^*$, where $\{G_1^*, G_2^*, \ldots, G_m^*\}$ is the output of Algorithm 1 on input $(G, (\theta \cup \tau)^*)$. For all $u, v \in V(G)$, $u \neq v$, $uv \in E(G) \iff \alpha(u)\alpha(v) \in E(\Pi_i G_i^*)$. If $uv \in E(G)$ and $\alpha(u)\alpha(v) \in E(\Pi_i G_i^*)$, then $w_G(uv) = w_{\Pi_i G_i^*}(\alpha(u)\alpha(v))$. 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 in line 7 of the algorithm, which is impossible if there is an edge between them, as that edge would be a path between them in all but one G_i'. Thus, there is no edge between u and v either.
2. Case 2: There is exactly one k on which $\alpha_k(u) \neq \alpha_k(v)$. Let $C_{\alpha_k(u)}$ and $C_{\alpha_k(v)}$ be the connected components of G_k' responsible for the creation of nodes $\alpha_k(u)$ and $\alpha_k(v)$, respectively, in G_k^* (i.e. the connected components containing u and v in G_k'). 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. If there is an edge between $\alpha_k(u)$ and $\alpha_k(v)$ in G_k^* of weight w_{uv} then by Lemma 4.3, we have that all nodes in $C_{\alpha_k(u)}$ have an edge of weight w_{uv} to exactly one vertex in $C_{\alpha_k(v)}$. Let $v' \in C_{\alpha_k(v)}$ be the vertex 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 $\alpha(u)\alpha(v)$.

□

Lemma 4.7. *If $G = (V(G), E(G), w_G)$ is the weighted input graph for Algorithm 1 with $(\theta \cup \tau)^*$, the output graphs $\{G_1^*, \ldots, G_m^*\}$ are all prime.*

Proof. Using Corollary 4.2, we only have to show that the edges of each factor G_i^* are all related by $(\theta \cup \tau)^*$. We know that a factor G_i^* appears as an isometric subgraph of $\Pi_i G_i^*$ 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. □

Theorem 4.6 and Lemma 4.7 prove that Algorithm 1 with input $(G, (\theta \cup \tau)^*)$ produces a prime factorization of weighted graph G. Additionally, because Algorithm 1 on input $(G, (\theta \cup \tau)^*)$ outputs ~~thea~~ number of graphs equal to the number of equivalence classes of $(\theta \cup \tau)^*$ on $E(G)$, Theorem 4.6 and Corollary 4.2 give us the following corollary.

Corollary 4.8. *A weighted graph G is prime if and only if there is exactly one $(\theta \cup \tau)^*$ equivalence class on its edges.*

4.3 Uniqueness of prime factorization

We additionally claim that for any weighted graph G, the prime factorization of G is unique up to graph isomorphism.

Theorem 4.9. *Let $G = (V(G), E(G), w_G)$ be a minimal weighted graph and let $\mathcal{G} = \{G_1, \ldots, G_p\}$ and $\mathcal{H} = \{H_1, \ldots, H_r\}$ be two prime factorizations of G. Then $p = r$ and the pseudofactors may be reordered so that G_i is isomorphic to H_i for all $i, 1 \leq i \leq p$.*

Proof. Let α and β be isomorphisms from G to the product of the graphs in $\mathcal{G}$ and $\mathcal{H}$, respectively. We will define a mapping $f : \mathcal{G} \to \mathcal{H}$ 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 $f(G_i) = H_j$. We show that f is a well-defined bijection. First, we note that by Lemma 4.3, two edges are related by $(\theta \cup \tau)^*$ if and only if they have parent edges in the same factor of any factorization. Thus, if there exists $uv, xy \in E(G)$ such that uv and xy have parent edges under α in factor G_i of $\mathcal{G}$, then those edges must also have parent edges under β in the same factor of $\mathcal{H}$, meaning G_i is mapped to exactly one H_j. The reverse reasoning also shows that each graph in $\mathcal{H}$ 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 $(\theta \cup \tau)^*$. We have that G is isomorphic to $\Pi_i G_i$ and $\Pi_i H_i$, so G_i is isomorphic to an isometric subgraph of $\Pi_i H_i$. However, because G_i is prime, all edges in this subgraph are related by $(\theta \cup \tau)^*$ and by Lemma 4.1 must have parent edges in the same factor of $\Pi_i H_i$. By the definition of f, this factor is $f(G_i)$. Thus, G_i must be a subgraph of $f(G_i)$. □

5 Computing the runtime of factorization and pseudofactorization

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 $G = (V(G), E(G), w_G)$ whose transitive closure is also symmetric and reflexive. To do this, we use the edge-relation graph for G, as defined in Definition 2.9. For convenience, we discuss the structure of such a graph again here. For a graph $G = (V(G), E(G), w_G)$ with edge relation R, we define an unweighted graph $G_R = (V(G_R), E(G_R))$. The vertices of this graph correspond to the edges of G where the vertex corresponding to edge xy is denoted u_{xy} or u_{yx} 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 $O(|V(G_R)| + |E(G_R)|)$ time. As an upper bound, we know that once this graph is computed, $|V(G_R)| = |E(G)|$ and $|E(G_R)| = O(|V(G_R)|^2) = O(|E(G)|^2)$. Thus, computing the connected components with BFS takes $O(|E(G)|^2)$ time. If further bounds can be placed on the number of edges in G_R, this time can be decreased further. Figure 10 shows an example of how G_θ is constructed for a given input graph.

5.1 Runtime for pseudofactorization

To perform Graham and Winkler's algorithm, we compute the equivalence classes of θ^*, 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, which requires $O(mn + m^2) = O(m^2)$ time for an n-vertex, m-edge graph since the graph is connected. To compute the equivalence classes of θ^*, we compute G_θ and find 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 vertex of G_θ in $O(|V(G_\theta)|) = O(m)$ time, for a total of

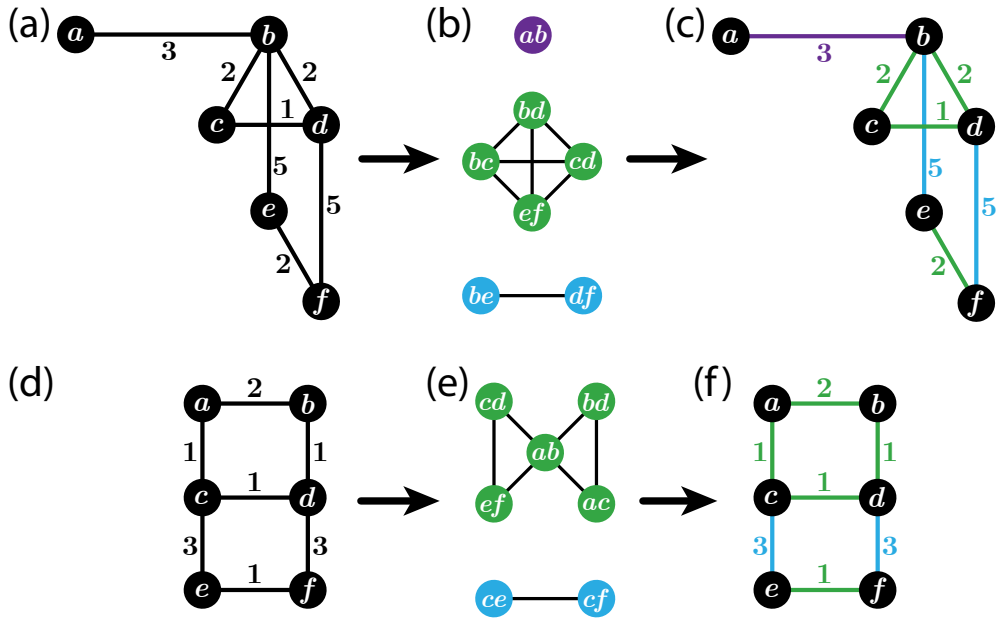

Figure 10: a) A weighted graph G. b) G_θ. c) G with edges colored according to their equivalence class under θ^*. Each equivalence class corresponds to a connected component of G_θ. d-f) Analogous illustration for the graph shown in (d).

$O(m^2)$ time and then we take $O(m^2)$ time to compute the connected components. Thus, total runtime is $O(m^2)$ plus APSP computation time, which is currently known to be $O(n^2 \log\log n + mn)$ [19] and thus gives us a runtime of $O(n^2 \log\log n + m^2)$ overall.

5.2 Runtime for factorization

For factorization using Algorithm 1, we can again bound the runtime of the main loop by $O(m^2)$ for an n-vertex, m-edge graph, and thus we need only 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 form ab and cd that are related by θ and have the same edge weight, we can check if ac and bd are 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. Compute the set $\mathcal{E}$ of all adjacent edges of G not related by τ, and then we can compute the set $\mathcal{E}'$ of all pairs of edges that are adjacent but not in $\mathcal{E}$. Add these edge pairs as edges in G_θ to compute $G_{\theta\cup\tau}$. For a given pair of edges in G their contribution to $\mathcal{E}$ is found in constant time for $O(m^2)$ time to compute $\mathcal{E}$, and $\mathcal{E}'$ may also be found in $O(m^2)$ time given $\mathcal{E}$. Thus, it takes $O(m^2)$ total time to construct $G_{\theta\cup\tau}$ and get its connected components. This brings our total runtime for graph factorization up to $O(m^2)$ plus APSP time, for $O(n^2 \log\log n + m^2)$ time overall.

6 An algorithm for improved pseudofactorization runtime

In Section 5, we proved that Algorithm 1 could be used with relation θ^* to pseudofactor an n-vertex, m-edge minimal graph in $O(n^2 \log\log n + m^2)$ time. Feder [8] showed that, for unweighted graphs, θ^* could be replaced by an alternative equivalence relation θ^*_T, which has the same equivalence classes but whose classes are faster to compute and for which there are at most $n-1$ equivalence classes Recall from Definition 2.6 that, given a spanning tree T of a graph G, 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, and that the transitive closure θ^*_T is an equivalence relation. Feder showed that for unweighted graphs θ^*_T with any spanning tree T could be used to produce a pseudofactorization for any G; we will show the slightly weaker property that at least one T exists for every minimal weighted graph and that such a tree can be found efficiently.

Recall from Section 2 that $[uv]_{\theta^*_T}$ is the equivalence class under θ^*_T containing uv. We make the following claim about the equivalence classes of θ^*_T:

Claim 6.1. *Let $G = (V(G), E(G), w_G)$ be a graph with a spanning tree T and $uv \in E(G)$ Then $[uv]_{\theta^*_T} \subseteq [uv]_{\theta^*}$.*

Proof. Take $xy \in [uv]_{\theta^*_T}$. Since $uv\ \theta^*_T\ xy$, 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 $uv\,\theta^*\,xy$. Thus $xy \in [uv]_{\theta^*}$. □

For an n-vertex, m-edge minimal graph G, we have not proven that $[uv]_{\theta^*_T} = [uv]_{\theta^*}$ for arbitrary T. However, we can prove that there is some tree T^* such that $[uv]_{\theta^*_{T^*}} = [uv]_{\theta^*}$ for all $uv \in E(G)$. Algorithm 2 finds such a T^*. Computing the equivalence classes of $\theta^*_{T^*}$ for a given T^* takes only $O(mn)$ time, as they can be computed by comparing each edge to only the edges in T^*, of which there are exactly $n-1$.

Informally, Algorithm 2 starts with any spanning tree and repeatedly modifies it until the equivalence classes under θ^*_T equal those under θ^*. It does so by looking at a particular equivalence class for θ^*_T and for each edge in that equivalence class, checking that every edge on the simple path between its endpoints through the tree is in the current equivalence class or an already processed one, and swapping the edge in question into the tree if this does not hold. The idea behind this process is to try to grow the equivalence class we are working on as much as possible until it is the same as the equivalence class under θ^*.

We will first justify that this algorithm works correctly and then that it runs in $O(mn)$ time . In particular, during the runtime subsection, we will go into more detail about how to update G_{θ_T} efficiently as we modify T, but for now we take for granted that we update the edge-relation graph correctly whenever T is updated.

6.1 Correctness of Algorithm 2

Let $C_R(u_{xy})$ be the vertices of the connected component of u_{xy} in G_R. Note that $[xy]_{\theta^*_T} = [xy]_{\theta^*}$ if and only if u_{xy}'s connected component in G_{θ_T} includes the same vertices as its connected component in G_θ, or in other words if and only if $C_\theta(u_{xy}) = C_{\theta_T}(u_{xy})$. Consider a particular iteration of the while loop beginning at line 5 in which the edge we select from *undiscovered* in line 6 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 incident to the vertex u_{uv} in G_{θ_T}. Removing edges from the tree

Algorithm 2 Algorithm for breaking up a graph over a relation

Input: A weighted graph 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* $\leftarrow V(G_{\theta_T})$
Set *current-class* $\leftarrow 0$
while *undiscovered* is not empty **do**
    Pick $xy \in$ *undiscovered*
    Run BFS to find all nodes *reachable* from xy in G_{θ_T}
    Mark all newly discovered edges of G with the number *current-class*
    Set *reachable* $\leftarrow$ all edges whose corresponding nodes are *reachable* from xy in G_{θ_T}
    while *reachable* is not empty **do**
        Pick $ab \in$ *reachable*
        Find the path P from a to b in the tree T
        if there is an edge $uv \in P$ that is not marked with a class number **then**
            Create a new spanning tree T' from T by adding ab and removing uv
            Set $T \leftarrow T'$
            Update G_{θ_T} based on the new T
            Mark any newly *reachable* edges with *current-class*
            Update *reachable* to add any newly *reachable* nodes from xy in G_{θ_T} *current-class*
        Remove ab from *reachable* and from *undiscovered*
    current-class $\leftarrow$ *current-class* $+1$
return T

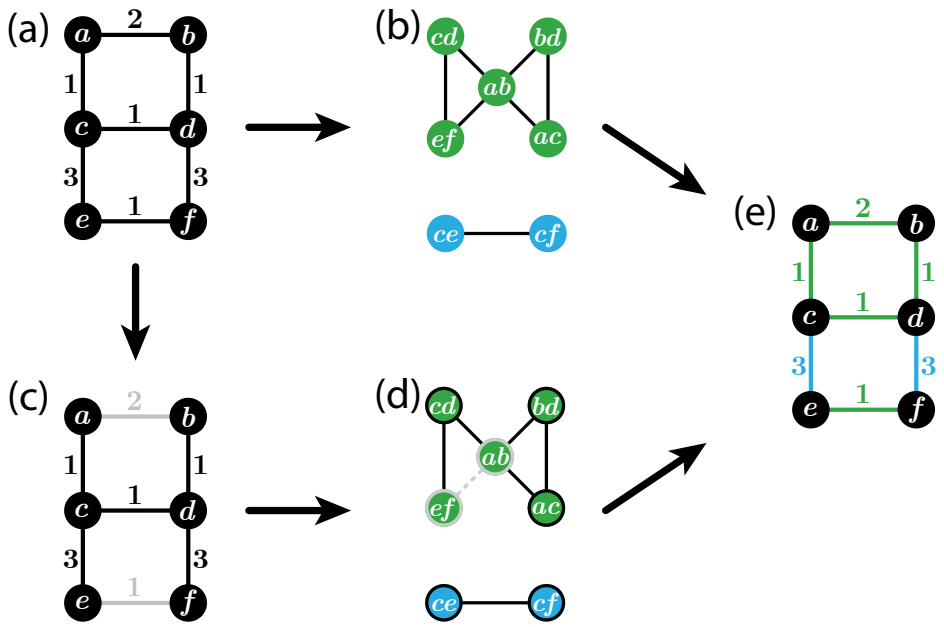

Figure 11: (a) An input graph G. (b) G_θ. (c) G with a spanning tree T (edges not in T are in gray). (d) G_{θ_T}. The black and gray outlines indicate if each edge is in spanning tree T or not, respectively. The graph is the subgraph of G_θ with only those edges incident to an element of T. (e) For this choice of T, G_θ and G_{θ_T} produce the same equivalence classes on the edges of G.

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 6.2. *Let $G = (V(G), E(G), w_G)$ be a minimal weighted graph with spanning tree T. Consider a step in the execution of Algorithm 2 on G, where the current tree T is transformed into a new spanning tree T' by removing an edge uv and adding a new edge ab. If edges u_1v_1 and u_2v_2 are 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 with edge set restricted to those edges incident to vertices that correspond to edges in T and T', respectively. Thus, by swapping uv for ab in the tree, the only edges that may have been deleted from G_{θ_T} to form $G_{\theta_{T'}}$ are those incident to u_{uv}. Since two vertices are adjacent in G_{θ_T} if and only if they correspond to two edges of G 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 6.2, we can introduce a new picture of what G_{θ_T} looks like relative to G_θ, represented in Figure 11. In particular, if we take G_θ and color black all the vertices corresponding to edges in T, deleting the edges not incident to at least one black vertex produces G_{θ_T}. This helps provide a visual representation for the relationship between θ^* and θ^*_T.

Lemma 6.3. *Let $G = (V(G), E(G), w_G)$ be a minimal weighted graph and consider a step in the execution of Algorithm 2 on G where spanning tree T is updated to spanning tree T' by removing edge uv and adding edge ab. If u_{st} is a vertex in G_{θ_T} that has been discovered prior to this step, then $[st]_{\theta_T} \subseteq [st]_{\theta_{T'}}$.*

Proof. We will show that $C_{\theta_T}(u_{st}) \subseteq C_{\theta_{T'}}(u_{st})$, and since the connected components of an edge-relation graph are exactly in correspondence to the equivalence classes of said relation, this will prove the lemma. Take $u_{xy} \in C_{\theta_T}(u_{st})$ and consider a path from u_{xy} to u_{st}. If $u_{xy} \notin C_{\theta_{T'}}$, then on all paths from u_{xy} to u_{st}, some edge must have been present in G_{θ_T} but not $G_{\theta_{T'}}$. By Lemma 6.2, all edges present in G_{θ_T} but not $G_{\theta_{T'}}$ must be adjacent to u_{uv}. Thus, edge u_{uv} is in the same connected component as u_{st} in G_{θ_T}.

Updates to T are always done in lines 14-15, and line 13 ensures that uv has not yet been marked with any class number. Note that when a vertex is first added to *reachable* in line 9 or line 18, the corresponding edge has already been marked with a class number in line 8 or line 17, respectively. Thus, if u_{uv} has no associated class number, then it has never been added to the set *reachable*. We consider two cases: u_{uv} and u_{st} were in the same connected component since the beginning of the algorithm, and at some point the tree was changed such that u_{uv} and u_{st} became part of the same connected component.

- Case 1: u_{uv} and u_{st} were in the same connected component continuously from the beginning to the point that G_{θ_T} was created.

  Vertices are only removed from *undiscovered* in line 19, and to be removed they must have been in *reachable* at that time. Thus, since u_{st} is discovered, st must have been placed in *reachable* at some point in line 9 or 17. At this time, u_{uv} was reachable from u_{st} due to the fact it was continuously reachable up until T' was formed, so at this time uv was added to *reachable* and given a class number.

- Case 2: At some point, u_{uv} and u_{st} were not in the same connected component of the edge-relation graph.

  Consider the most recent step in which u_{uv} and u_{st} were made adjacent in the edge-relation graph, prior to creating T'. Edges are only added in lines 14-15, and immediately after such an update, lines 16-18 update the edge-relation graph and mark all vertices reachable from the vertex that may have gained new edges as *reachable*. Thus, since u_{uv} and u_{st} were made adjacent at this time, they were both added to *reachable* and uv received a class number.

Thus, in any case, if u_{st} is a discovered vertex at this step, then no edge in u_{st}'s connected component could be removed from its connected component in the edge-relation graph and we get that u_{uv} and u_{xy} are still connected in $G_{\theta_{T'}}$ if they were in G_{θ_T}. □

It remains to show that if xy is the edge discovered in line 6 of Algorithm 2 and T is the tree at the end of that iteration of the while loop, then $[xy]_{\theta^*_T} = [xy]_{\theta^*}$. Note that Lemma 6.3 implies that if at any point in the algorithm we obtain a tree T such that $[xy]_{\theta^*_T} = [xy]_{\theta^*}$, then for any tree T^* that we obtain later in the execution of the algorithm, we get $[xy]_{\theta^*_{T^*}} = [xy]_{\theta^*}$. 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 6.4. *Consider a minimal graph $G = (V(G), E(G), w_G)$ and the execution of Algorithm 2 on G. If at some step in the execution, the current spanning tree T is updated to a new spanning 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 corresponding to vertices (of G_θ) that have already been assigned a class number.*

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 $\theta^*_{T^*}$ are those of θ^*, as desired.

Lemma 6.5. *Let $G = (V(G), E(G), w_G)$ be a weighted minimal graph and consider the execution of Algorithm 2 on G. If xy is an edge discovered at the beginning of a particular iteration of the while loop at line 6 and T is the tree at the end of that iteration of the while loop, then $[xy]_{\theta^*_T} = [xy]_{\theta^*}$.*

Proof. Consider the first iteration of the loop beginning at line 5 for which this lemma does not hold. From Lemma 6.4 and the fact that we process every edge in u_{xy}'s connected component of G_{θ_T}, we know that at the end of an iteration of the outer while loop, all edges corresponding to vertices (of G_{θ_T}) in $C_{\theta_T}(u_{xy})$ have paths through T that include only marked edges, which are edges corresponding to vertices that are in this connected component or some previously processed connected component in G_{θ_T}. Now, assume $[xy]_{\theta^*_T} \neq [xy]_{\theta^*}$. Since we know that $[xy]_{\theta^*_T} \subseteq [xy]_{\theta^*}$ from Claim 6.1, this means that $[xy]_{\theta^*_T} \subsetneq [xy]_{\theta^*}$. Pick $uv \in [xy]_{\theta^*}, uv \notin [xy]_{\theta^*_T}$. We know that because these edges are in the same θ^* 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 $uv \not\theta^*_T xy$, there exists j such that u_jv_j and $u_{j+1}v_{j+1}$ are not related by θ^*_T but $u_jv_j \in [xy]_{\theta^*_T}$. Pick the first such j and to simplify notation, we will call u_jv_j and $u_{j+1}v_{j+1}$ by the names ab and st respectively. We have $ab \,\theta\, st$, but $ab \in [xy]_{\theta^*_T}$ and $st \notin [xy]_{\theta^*_T}$.

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 $s^*t^* \in [st]_{\theta^*_T}$. Thus, on this path there exists $s^*t^* \in [st]_{\theta^*_T} \neq [xy]_{\theta^*_T}$.

By Lemma 6.3 and the fact that ab has been processed (as it is in $[xy]_{\theta^*_T}$), 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 at line 5 (i.e. an iteration where the algorithm chose an edge other than xy at line 6). However, since we are considering the first iteration of the while loop in line 5 for which the lemma does not hold, we get $[s^*t^*]_{\theta^*_T} = [s^*t^*]_{\theta^*} = [xy]_{\theta^*} = [st]_\theta$ (since $xy \,\theta\, s^*t^* \theta\, st$ and s^*t^*'s equivalence class did not change over the course of this execution of the loop at line 5).

Thus, we know that at the end of the round, every edge not in *undiscovered* has $[xy]_{\theta^*_T} = [xy]_{\theta^*}$. 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 θ^*_T are the same as those of θ^*. □

Thus Algorithm 1 on the input $(G, \theta^*_{T^*})$, where T^* is the output of Algorithm 2 on G, produces an irreducible pseudofactorization of G. Next, we analyze the runtime of Algorithm 2 to determine if it improves the time complexity of pseudofactorization.

6.2 Runtime of Algorithm 2

First, we note that because the number of equivalence classes of $\theta^*_{T^*}$ for any T^* is at most $n-1$ for an n-vertex, m-edge minimal graph, Algorithm 1 on input $(G, \theta^*_{T^*})$ (where T^* is the output of Algorithm 2 on G) takes only $O(mn)$time, as finding the set of equivalence classes for θ^*_T in line 2 of Algorithm 1 takes only $O(mn)$ time, and the loop beginning at line 4 of Algorithm 1 executes only $O(n)$ times (Since the equivalence classes of θ^* and $\theta^*_{T^*}$ are the same, this implies that line 4 executes at most $O(n)$ times when the input is (G, θ^*) as well, but line 2 could still take $O(m^2)$ time to execute, so the real benefit is the speed-up in finding equivalence classes.)

This leaves us with analyzing the runtime to find an appropriate T^* using Algorithm 2. To do so, we break the algorithm's work into parts.

- First, consider the work done by lines 14-18 across the entire run of the algorithm. If the inside of this statement is executed, then some new edge ab is added to the tree.

  Claim 6.6. *Consider an execution of Algorithm 2 on input G. If at some point during the algorithm edge $ab \in E(G)$ is added to the current spanning tree, it is never removed after that point.*

  Proof. Note that the edge ab that is added to the tree is taken from *reachable*, but in order to be placed in *reachable* in line 9, it had to have been marked with a class number in line 8. The algorithm only ever removes edges from the tree in line 14, and in order to remove an edge there, the edge cannot be marked with a class number. Thus, once ab is added to the tree in line 14, it can never be removed. □

  This implies that lines 14-18 only execute at most $n-1$ times, as there can be at most $n-1$ nodes in a tree. The bulk of the work in these lines comes from lines 16-18. In updating G_{θ_T}, we may need to remove some edges adjacent to u_{uv} and add some edges adjacent to u_{ab}. Finding which of these edges should stay or remain takes $O(m)$ time total.

  This leaves us with considering lines 17-18. In particular, in these lines we run BFS on the newly updated graph starting from u_{ab}. However, it is not a full BFS run, as some nodes have already been marked with a class number and thus discovered. In running this BFS, we explore all $O(m)$ edges out of u_{ab}, as well as all edges adjacent to *undiscovered* nodes. However, for the edges adjacent to *undiscovered* nodes, this is the first time exploring such edges. Thus, we will count this work toward a single run of BFS on the entire graph rather than count it here. Thus, in total lines 14-18 take $O(mn)$ time plus the work it contributes to initial edge discovery in the full BFS run.

- Consider the work done in lines 11-13. It takes $O(n)$ time to find the unique simple path from a to b in the given tree and traverse it, and once w have done this for an edge $ab \in E(G)$, we remove ab from *reachable* and *undiscovered*, which means it can never be in *reachable* again, and thus we never repeat this process of looking for the path from a to b again. Thus, it takes at most $O(n)$ time per edge in $E(G)$ to run lines 11-13.

- Consider the work done in lines 6-9. In these lines, we pick an edge in $E(G)$ whose corresponding vertex has not yet been discovered in the edge-relation graph and we run BFS from it. Because the connected components of G_{θ_T} are updated in lines 16-18 whenever T itself is updated, we know that xy being *undiscovered* implies that all edges *reachable* from xy at this point are also *undiscovered*. Thus, the work required to run BFS and update the *reachable* nodes in lines 6-9 involves discovering

nodes and edges in G_{θ_T} for the first time. Like the partial BFS run in lines 14-18, we will count this towards the cost of a single run of BFS.

- A single run of BFS on this graph takes at most $O(mn)$ time, as there are at most $n-1$ edges initially in the tree and at most $n-1$ edges ever added to the tree. Even if every vertex in every iteration of the spanning tree was related to all other edges in the graph by θ, this would imply that there were at most $O(mn)$ unique edges across all iterations of G_{θ_T}. Thus, since BFS runs in linear time, this is an upper bound on how long the BFS run on *unique* edges takes. (We have already addressed the edges that are re-traversed in lines 14-18 and counted that time in our analysis of lines 14-18.)

Thus, we get an overall runtime of $O(mn)$ for Algorithm 2 on an n-vertex, m-edge graph G.

7 Conclusion

In areas like molecular engineering, distance metrics are complex and frequently not representable in unweighted graphs. Isometric embeddings of weighted graphs, while more difficult than for unweighted graphs, open up new avenues for solving problems in these areas. Here, we extended factorization and pseudofactorization to minimal weighted graphs and provided polynomial-time algorithms for computing both, in $O(n^2 \log\log n + m^2)$ time and $O(n^2 \log\log n + mn)$ time, respectively for n-vertex, m-edge weighted graphs. Several open questions remain, including the following:

- Can the efficiency of weighted graph factorization be improved, as weighted graph pseudofactorization was in Section 6, to $O(mn)$ when distances are precomputed?
- Can we place lower bounds on the time needed to find a graph's prime factorization or irreducible pseudofactorization? In particular, can we lower bound these processes by $O(mn)$ time?
- Can pseudofactorization aid in isometrically embedding weighted graphs into particular destination graphs of interest, such as hypercube graphs or arbitrary unweighted graph products?

In partial response to the first question, we can see by the proofs given in Section 7 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.

In partial response to the third question, Berleant *et al.* [3] has studied isometric embeddings of weighted graphs into Cartesian products of unweighted complete graphs, or Hamming graphs. In analogy to results on unweighted graphs [12], Berleant *et al.* [3] found that every isometric embedding of G into an unweighted Hamming graph can be formed by concatenating individual embeddings of the pseudofactors of G. Their findings apply also to hypercube graphs, which are a subset of Hamming graphs. This connection hints at a deeper link between pseudofactorization and isometric embeddings of weighted graphs that we hope will continue to be explored in future research.

Acknowledgements

The authors would like to acknowledge the anonymous reviewer for their helpful comments and suggestions.

M.B. and J.B. were supported by the Office of Naval Research (N00014-21-1-4013), the Army Research Office (ICB Subaward under W911NF-19-2-0026), and the National Science Foundation (CBET-1729397 and

OAC-1940231). M.B., J.B., and K.S. were supported by the National Science Foundation (CCF-1956054). Additional funding to J.B. was provided through a National Science Foundation Graduate Research Fellowship (grant no. 1122374). A.C. was supported by a Natural Sciences and Engineering Research Council of Canada Discovery Grant. V.V.W. was supported by the National Science Foundation (CAREER Award 1651838 and grants CCF-2129139 and CCF-1909429), the Binational Science Foundation (BSF:2012338), a Google Research Fellowship, and a Sloan Research Fellowship. This research was also supported by a core center grant from the National Institute of Environmental Health Sciences, National Institutes of Health (P30-ES002109).

Declarations of Interest

The authors declare that they have no competing interests.

References

[1] Yaron E Antebi, James M Linton, Heidi Klumpe, Bogdan Bintu, Mengsha Gong, Christina Su, Reed McCardell, and Michael B Elowitz. Combinatorial signal perception in the BMP pathway. *Cell*, 170(6):1184–1196, 2017.

[2] Eric B Baum. Building an associative memory vastly larger than the brain. *Science*, 268(5210):583–585, 1995.

[3] Joseph Berleant, Kristin Sheridan, Anne Condon, Virginia Vassilevska Williams, and Mark Bathe. Isometric hamming embeddings of weighted graphs. *Discrete Applied Mathematics*, 332:119–128, 2023.

[4] Kevin M Cherry and Lulu Qian. Scaling up molecular pattern recognition with DNA-based winner-take-all neural networks. *Nature*, 559(7714):370–376, 2018.

[5] Michel Deza and Monique Laurent. Additional notes. In R. L. Graham, B. Korte, Bonn L. Lovasz, A.Wigderson, and G.M. Ziegler, editors, *Geometry of Cuts and Metrics*, chapter 20.4, pages 308–311. Springer, 1997.

[6] 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.

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

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

[9] Joan Feigenbaum, John Hershberger, and Alejandro A. Schäffer. A polynomial time algorithm for finding the prime factors of cartesian-product graphs. *Discrete Applied Mathematics*, 12(2):123–138, 1985.

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

[11] Ronald L Graham and Henry O Pollak. On the addressing problem for loop switching. *The Bell System Technical Journal*, 50(8):2495–2519, 1971.

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

[13] 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.

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

[15] W. H. Kautz. Unit-distance error-checking codes. *IRE Transactions on Electronic Computers*, EC-7(2):179–180, 1958.

[16] Tomas Malinauskas and E Yvonne Jones. Extracellular modulators of Wnt signalling. *Current opinion in structural biology*, 29:77–84, 2014.

[17] Andrew Neel and Max Garzon. Semantic retrieval in DNA-based memories with Gibbs energy models. *Biotechnology progress*, 22(1):86–90, 2006.

[18] Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theor. Comput. Sci.*, 312(1):47–74, 2004.

[19] 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.

[20] Lulu Qian and Erik Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332(6034):1196–1201, 2011.

[21] Lulu Qian, Erik Winfree, and Jehoshua Bruck. Neural network computation with DNA strand displacement cascades. *Nature*, 475(7356):368–372, 2011.

[22] Gert Sabidussi. Graph multiplication. *Mathematische Zeitschrift*, 72:446–457, 1959/60.

[23] Francesca L Short, Tim R Blower, and George PC Salmond. A promiscuous antitoxin of bacteriophage T4 ensures successful viral replication. *Molecular microbiology*, 83(4):665–668, 2012.

[24] Elke Wilkeit. Isometric embeddings in hamming graphs. *Journal of Combinatorial Theory, Series B*, 50(2):179–197, 1990.

[25] Peter Winkler. Factoring a graph in polynomial time. *Eur. J. Comb.*, 8:209–212, 1987.

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

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

[28] David Yu Zhang and Georg Seelig. Dynamic DNA nanotechnology using strand-displacement reactions. *Nature chemistry*, 3(2):103–113, 2011.

[29] Ling Zhu, Jared D Sharp, Hiroshi Kobayashi, Nancy A Woychik, and Masayori Inouye. Noncognate Mycobacterium tuberculosis toxin-antitoxins can physically and functionally interact. *Journal of Biological Chemistry*, 285(51):39732–39738, 2010.