Embedding Graphs as Euclidean kNN-Graphs

Thomas Schibler
University of California, Santa Barbara, CA, USA

Subhash Suri
University of California Santa Barbara, CA, USA

Jie Xue
New York University Shanghai, China

Abstract

Let $G = (V, E)$ be a directed graph on n vertices where each vertex has out-degree k. We say that G is kNN-realizable in d-dimensional Euclidean space if there exists a point set $P = \{p_1, p_2, \dots, p_n\}$ in $\mathbb{R}^d$ along with a one-to-one mapping $\phi : V \to P$ such that for any $u, v \in V$, u is an out-neighbor of v in G if and only if $\phi(u)$ is one of the k nearest neighbors of $\phi(v)$; we call the map ϕ a *kNN-realization* of G in $\mathbb{R}^d$. The kNN-realization problem, which aims to compute a kNN-realization of an input graph in $\mathbb{R}^d$, is known to be NP-hard already for $d = 2$ and $k = 1$ [Eades and Whitesides, Theoretical Computer Science, 1996], and to the best of our knowledge has not been studied in dimension $d = 1$. The main results of this paper are the following:

- For any fixed dimension $d \geq 2$, we can efficiently compute an embedding realizing at least a $1 - \varepsilon$ fraction of G's edges, or conclude that G is not kNN-realizable in $\mathbb{R}^d$.
- For $d = 1$, we can decide in $O(kn)$ time whether G is kNN-realizable and, if so, compute a realization in $O(n^{2.5}\mathsf{poly}(\log n))$ time.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Geometric graphs, k-nearest neighbors, graph embedding, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.SoCG.2025.73

Related Version *Full Version*: `https://arxiv.org/pdf/2504.06503`

1 Introduction

The *kNN-graph* of a set P of points in $\mathbb{R}^d$ is a directed graph with vertex set P and edges defined as follows: there is a directed edge from $a \in P$ to $b \in P$ if and only if b is one of the k-nearest neighbors of a in P (under the Euclidean distance). A directed graph $G = (V, E)$ is *kNN-realizable* in $\mathbb{R}^d$ if it is isomorphic to the kNN-graph of a set of points in $\mathbb{R}^d$. In this paper, we consider the following natural problems regarding kNN-graphs: Can we efficiently check whether a given directed graph is kNN-realizable in $\mathbb{R}^d$? If so, can we efficiently find a kNN-realization of G? More formally, a kNN-*realization* (or kNN-*embedding*) of G in $\mathbb{R}^d$ is a one-to-one mapping $\phi : V \to P$ from the vertex set V of G to a set P of points in $\mathbb{R}^d$ that induces an isomorphism between G and the kNN-graph of P. Throughout the paper, we use the terms embedding and realization interchangeably.

The kNN-graph is a member of the well-known family of *proximity* graphs in computational geometry that includes minimum spanning trees, relative neighborhood graphs, Gabriel graphs, and Delaunay triangulations [11]. Over the past several decades, a substantial research effort has been directed to design efficient algorithms to compute these structures for an input set of points. The *inverse* problem – the focus of our paper – where we want to recover the points that produce a given proximity graph, however, remains less well understood.

An early example of a positive result in this direction is on Euclidean minimum spanning trees. Monma and Suri [22] show that any tree with maximum node degree 5 can be realized as a minimum spanning tree of points in the plane and, additionally, every planar point set admits a minimum spanning tree with degree at most 5. This settles the above problem for *non-degenerate* planar point sets but if we allow co-circularities, then a planar minimum spanning tree can have maximum node degree 6; in that case, the problem of computing a 1NN-realization was proved to be NP-complete by Eades and Whitesides [13]. Dillencourt [12] considers the problem of recognizing triangulations that can be realized as Delaunay triangulations in the plane, and it is also known that all outerplanar triangulations are realizable [24]. (These proofs are non-constructive, however, and only exponential time algorithms are known for computing the coordinates of the points in the realization [1].) Among other related results, Bose et al. [5] give a characterization of *trees* that can be realized in the plane as relative neighborhood or Gabriel graphs, and Eppstein et al. [15] establish some structural properties of kNN graphs of random points in the plane.

The kNN-realization is not directly related to the *metric embedding* problem but there are some obvious similarities. The input to metric embedding is a weighted graph satisfying triangle inequalities and the goal is to find an embedding realizing all pairwise distances. This is not always possible [20] – there are simple metrics that cannot be embedded in any finite dimensional Euclidean space. However, if we allow some *distortion* (i.e. approximation) of distances, then any n-node metric graph can be embedded in $O(\log n)$ dimensional space with polylog distortion [6], or using the celebrated Johnson-Lindenstrauss theorem [18] in dimension $O(\log n/\varepsilon^2)$ with distortion $1+\varepsilon$ for any fixed constant $\varepsilon > 0$.

In metric embedding the goal is to realize all pairwise distances (approximately), while in kNN-realization the goal is only to preserve all *ordinal* neighbor relations in a *directed* graph: a neighbor of each vertex must be closer than any of its non-neighbors but the choice of specific distances does not matter. Although there is some work in embedding ordinal relations as well, the main focus is specialized metric spaces and *bounds* on *ordinal distortion.* In particular, Alon et al. [2] consider embeddings with ordinal relaxations into ultrametric spaces, and Bădoiu et al. [7] improve their bounds for tree and line embeddings. A different line of research concerns *triplets* embedding, where the input is a set of triples (a, b, c) specifying constraints of the form $d(a, b) < d(a, c)$. Even for embedding in one dimension $\mathbb{R}^1$, the general triplet problem as well as the related “betweenness” problem is MAXSNP-hard [9], and a FPTAS is known for maximizing the number of satisfied triplet constraints [17]. (If all $\binom{n}{3}$ triplet constraints are given, then the problem is trivial to solve – indeed, once the leftmost point is determined, the rest of the ordering can be easily decided.) In contrast with the general triplet constraints, the pairwise relations in a kNN-realization problem in $\mathbb{R}^1$ have a richer structure that allows us to solve the problem efficiently.

The kNN-realization is also related to the *sphericity* of graphs [19], where the goal is to embed a (undirected) graph $G = (V, E)$ into an Euclidean space $\mathbb{R}^k$ such that there is a point $p(u)$ for each vertex $u \in V$, and $d(p(u), p(v)) \leq 1$ iff $(u, v) \in E$. The smallest dimension k that admits such an embedding is called the *sphericity* of G [19]. (Another related problem is the *dimension* of a graph, introduced by Erdős, Harary and Tutte [16]: it is the smallest number k such that G can be embedded into Euclidean k-space with every edge of G having length 1.) Along these lines, Chatziafratis and Indyk [8] also show that if we want to preserve the relative distances among the k nearest neighbors of each point, then the embedding dimension must grow linearly with k. Unlike these results, our work is algorithmic – we design efficient algorithms to realize a given graph G in a specified dimension d.

Euclidean embedding of neighbor relations is also studied in social sciences for geometric realization of preference orderings. Given a set V of n voters, a set C of m candidates, and a rank ordering of the candidates by each voter, we say that the preference graph can be realized in Euclidean d-space if each voter's preferences are consistent with its Euclidean distances to all the candidates. In this case, the smallest dimension must satisfy $d \geq \min\{n, m-1\}$ [4].

1.1 Main results

We study the *kNN-realization* problem, which takes a directed graph G as an input and aims to compute a kNN-realization of G in $\mathbb{R}^d$ (or decide the nonexistence of such a realization). Throughout we focus on the *unranked* kNN-realization problem where nearest neighbors are realized as an *unordered set* but our results also hold (with minor caveat) for the ranked version where edges of G specify each of the k neighbors in order (see Concluding Remarks). Our two main results are the following.

1. Assuming that G is kNN-realizable in $\mathbb{R}^d$, we can find a d-dimensional realization in polynomial time that preserves at least a $1-\varepsilon$ fraction of the edges of G, for any fixed $\varepsilon > 0$. If our algorithm fails, we can also conclude that G is *not* kNN-realizable in $\mathbb{R}^d$. In particular, our algorithm is an EPTAS (*efficient polynomial-time approximation scheme*) for the kNN-realization problem for fixed k and d.
2. We give a *linear-time* algorithm to decide wether G is kNN-realizable in $\mathbb{R}^1$. Specifically, the algorithm runs in $O(kn)$ time, which is linear in the size of the input graph. If G is realizable, our algorithm can compute a realization in $O(n^{2.5}\mathsf{poly}(\log n))$ time.

1.2 Basic definitions

For a (directed or undirected) graph G, we use $V(G)$ and $E(G)$ to denote the set of its vertices and edges, respectively. If G is a directed graph, we say G is *k-regular* if the out-degree of every vertex of G is exactly equal to k. We use $\mathsf{in}[v]$ (resp., $\mathsf{out}[v]$) to denote the set consisting of v itself and all in-neighbors (resp., out-neighbors) of v. A *kNN-realization* of $G = (V, E)$ in a metric space $\mathcal{M} = (M, \mathsf{dist})$ is an injective map $\phi : V \to M$ such that $\mathsf{dist}(\phi(v), \phi(u)) < \mathsf{dist}(\phi(v), \phi(u'))$, for any distinct triple $v, u, u' \in V$ where $(v, u) \in E$ and $(v, u') \notin E$. We say G is *kNN-realizable* in $\mathcal{M}$ if there exists a kNN-realization of G in $\mathcal{M}$.

2 Approximate kNN-realization in $\mathbb{R}^d$

We first observe that it is easy to decide in polynomial time whether a given graph G is kNN-realizable in a finite-dimensional Euclidean space. We just have to check the acyclicity of an auxiliary graph defined as follows. Let Λ_G be a directed graph whose vertices correspond to pairs of vertices in $V(G)$ with a directed edge $(\{v, u\}, \{v, u'\})$ for every distinct triple $v, u, u' \in V(G)$ where $(v, u) \in E(G)$ and $(v, u') \notin E(G)$. Intuitively, Λ_G encodes the $\leq$-relation among the pairwise distances of points in any (potential) kNN-realization of G. If Λ_G has a directed cycle, then clearly G is not kNN-realizable. Otherwise, a topological sort of Λ_G gives a total ordering of the vertex pairs in G. With this ordering, we can find a kNN-realization of G in $\mathbb{R}^n$ using the result of Bilu and Linial [3].

On the other hand, deciding whether G is kNN-realizable in $\mathbb{R}^d$, for a specific dimension d, is NP-hard. This is shown by Eades and Whitesides [14] who proved the hardness for $d = 2$ and $k = 1$. Therefore, in this section, we explore an *approximation* algorithm for the realization problem in any fixed dimension d. We first need to define an approximate solution to our problem. There is a natural way to measure how well a map $\phi : V(G) \to \mathbb{R}^d$ approximates a kNN-realization: randomly sample an edge $(u, v) \in E(G)$ and consider the probability that $\phi(v)$ is among the k nearest neighbors of $\phi(u)$. Formally, we introduce the following definition.

▶ **Definition 1** (approximate kNN-realization). *Let G be a k-regular directed graph. For $c \in [0, 1]$, a map $\phi : V(G) \to \mathbb{R}^d$ is a **c-approximate kNN-realization** of G in $\mathbb{R}^d$ if*

$$\sum_{(u,v)\in E(G)} \sigma_\phi(u, v) \geq c \cdot |E(G)|,$$

where $\sigma_\phi : V(G) \times V(G) \to \{0, 1\}$ is the indicator function defined as $\sigma_\phi(u, v) = 1$ if we have $|\{v' \in V(G)\backslash\{u\} : \|\phi(u) - \phi(v')\|_2 \leq \|\phi(u) - \phi(v)\|_2\}| \leq k$ and $\sigma_\phi(u, v) = 0$ otherwise.

Our main result is an algorithm for computing a $(1 - \varepsilon)$-approximate kNN-realization of G in $\mathbb{R}^d$ with time complexity $f(k, d, \varepsilon) \cdot n^{O(1)}$, for any given $\varepsilon > 0$ (provided that G is kNN-realizable in $\mathbb{R}^d$), where f is some computable function. In other words, for fixed k and d, we obtain an *efficient polynomial-time approximation scheme* (EPTAS) for the kNN-realization problem in $\mathbb{R}^d$.

At a high level, our algorithm consists of two main steps. In the first step, it computes a set $E \subseteq E(G)$ of edges such that $|E| \leq \varepsilon|E(G)|$ and each weakly-connected component of $G - E$ contains $O_\varepsilon(1)$ vertices. The existence of E follows from the result of Miller et al. [21] on graph separators, and the computation of E relies on the approximate minimum cut algorithm of Chuzhoy et al. [10]. The edges in E are the ones we sacrifice in our approximation and so, in the second step, our algorithm computes a map $\phi : V(G) \to \mathbb{R}^d$ satisfying $\sigma_\phi(u, v) = 1$ for all edges $(u, v) \in E(G)\backslash E$. This turns out to be easy, since the weakly-connected components of $G - E$ are of size $O_\varepsilon(1)$ and we can work on these components individually. These two steps will be presented in Sections 2.1 and 2.2, respectively.

2.1 Splitting the graph by removing few edges

A *balanced cut* of an *undirected* graph H is a subset $E \subseteq E(H)$ such that every connected component of $H - E$ contains at most $|V(H)|/2$ vertices. Every directed graph G naturally corresponds to an undirected graph G_0 defined as $V(G_0) = V(G)$ and $E(G_0) = \{\{u, v\} : (u, v) \in E(G) \text{ or } (v, u) \in E(G)\}$. We need the following important result.

▶ **Lemma 2.** *Let G be a directed graph that is kNN-realizable in $\mathbb{R}^d$, and G_0 be the undirected graph corresponding to G. Then any subgraph H of G_0 with $|V(H)| \geq 2$ admits a balanced cut of size $O(|V(H)|^{1-\frac{1}{d}})$, where the constant hidden in $O(\cdot)$ only depends on k and d.*

Proof sketch. Let $k, d \in \mathbb{N}$ be fixed numbers. The lemma follows from two results in [21]. Specifically, it was shown in [21] that if a graph G is kNN-realizable in $\mathbb{R}^d$, then its corresponding undirected graph G_0 satisfies the following conditions.

1. The degree of every vertex in G_0 is $O(1)$.
2. For every subgraph H of G_0, there exists $S \subseteq V(H)$ such that $|S| = O(|V(H)|^{1-\frac{1}{d}})$ and every connected component of $H - S$ contains at most $\frac{d+1}{d+2} \cdot |V(H)|$ vertices.

We remark that [21] only claimed condition 2 above for the case $H = G_0$, but the argument extends to any subgraph H of G_0. Using the two conditions above, it is fairly easy to construct the desired balanced cut for any subgraph of G_0. ◀

A *minimum* balanced cut refers to a balanced cut consisting of the minimum number of edges. A *c-approximate* minimum balanced cut refers to a balanced cut whose size is at most $c \cdot \mathsf{opt}$, where opt is the size of a minimum balanced cut. The above lemma implies that if G is kNN-realizable in $\mathbb{R}^d$, then the size of a minimum balanced cut of any subgraph H of G_0 is $O(|V(H)|^{1-\frac{1}{d}})$. We need the following algorithm by Chuzhoy et al. for computing approximate minimum balanced cuts.

▶ **Theorem 3** ([10]). *For any fixed number $\alpha > 0$, there exists an algorithm that, given an undirected graph H of n vertices and m edges, computes a $\log^{O(1/\alpha)} n$-approximate minimum balanced cut of H in $O(m^{1+\alpha})$ time.*

The following lemma is the main result of this section, which states that one can remove a small fraction of edges from G to split G into small components.

▶ **Lemma 4.** *Let $k, d \in \mathbb{N}$ be fixed numbers. Given a k-regular directed graph G and a number $\varepsilon \in (0,1]$, one can either compute a subset $E \subseteq E(G)$ such that $|E| \leq \varepsilon|E(G)|$ and each weakly-connected component of $G - E$ contains at most $(\frac{1}{\varepsilon})^{O(1)}$ vertices, or conclude that G is not kNN-realizable in $\mathbb{R}^d$, in $O(|V(G)|^{1+\alpha})$ time for any constant $\alpha > 0$. Here the constants hidden in $O(\cdot)$ depend on k, d, and α.*

Proof. Let G_0 be the corresponding undirected graph of G, and $\alpha > 0$ be a constant. It suffices to compute $E \subseteq E(G_0)$ such that $|E| \leq \varepsilon|E(G_0)|$ and each connected component of $G_0 - E$ contains at most $(\frac{1}{\varepsilon})^{O(1)}$ vertices. Indeed, the edges in G corresponding to E form a subset of $E(G)$ of size $O(\varepsilon|E(G)|)$ which satisfies the desired property. This is fine since our algorithm works for an arbitrary $\varepsilon > 0$.

Pick a constant $\alpha' \in (0, \alpha)$ and let ApprxMinCut(H) denote the algorithm in Theorem 3, which returns an approximate minimum balanced cut of H in $O(|V(H)|^{1+\alpha'})$ time. Our algorithm, presented in Algorithm 1, computes $E \subseteq E(G_0)$ by recursively applying ApprxMinCut as a sub-routine. We fix some threshold $\delta = (\frac{1}{\varepsilon})^c$ for a sufficiently large c (depending on k, d, and α). If $|V(G_0)| \leq \delta$, then we simply return $E = \emptyset$. Otherwise, we apply ApprxMinCut(G_0) to obtain an approximate minimum balanced cut of G_0, and add the edges in the cut to E. Then for every connected component C of $G_0 - E$, we recursively apply our algorithm on C, which returns a subset of $E_C \subseteq E(C)$, and we add the edges in E_C to E. Finally, we return E as the output of our algorithm.

Algorithm 1 CutEdge(G_0).

```
1: if $|V(G_0)| \leq \delta$ then return $\emptyset$
2: $E \leftarrow$ ApprxMinCut$(G_0)$
3: $\mathcal{C} \leftarrow$ set of connected components of $G_0 - E$
4: for every $C \in \mathcal{C}$ do
5:     $E \leftarrow E \cup$ CutEdge$(C)$
6: return $E$
```

Clearly, every connected component of $G_0 - E$ contains at most $\delta = (\frac{1}{\varepsilon})^{O(1)}$ vertices. So it suffices to show $|E| \leq \varepsilon|E(G_0)|$ and analyze the running time of the algorithm. To bound $|E|$, we observe that when applying ApprxMinCut on any subgraph H of G_0, the size of the edge set obtained is of size $|V(H)|^{1-\frac{1}{d}} \log^{O(1/\alpha')} |V(H)|$. Indeed, a minimum balanced cut of H has size $O(|V(H)|^{1-\frac{1}{d}})$ by Lemma 2, and ApprxMinCut(H) returns a $\log^{O(1/\alpha')} |V(H)|$-approximate minimum balanced cut of H. Since we defined $\delta = (\frac{1}{\varepsilon})^c$ for a sufficiently large c, we may assume that the size of ApprxMinCut(H) is at most $|V(H)|^{1-\frac{1}{2d}}$ for any subgraph H of G_0 such that $|V(H)| > \delta$. Also, we may assume that $\varepsilon(2^{\frac{1}{3d}} - 1)n^{1-\frac{1}{3d}} > n^{1-\frac{1}{2d}}$ for all $n > \delta$. We shall prove that when applying our algorithm on a subgraph H of G_0 with $|V(H)| = n$, the size of $E \subseteq E(H)$ returned is at most $\varepsilon(n - n^{1-\frac{1}{3d}})$.

We apply induction on n. If $V(H) \leq \delta$, then our algorithm returns an empty set and thus the statement trivially holds. Assume the statement holds for $|V(H)| \in [n-1]$, and we consider the case $|V(H)| = n$ (where $n > \delta$). Our algorithm applies $\textsc{ApprxMinCut}(H)$ to obtain an approximate minimum balanced cut $E_0 \subseteq E(H)$ of H. As aforementioned, $|E_0| \leq n^{1-\frac{1}{2d}}$. Let $C_1, \dots, C_r$ be the connected components of $H - E_0$. Set $n_i = |V(C_i)|$ for $i \in [r]$. We have $n_i \leq n/2$ for all $i \in [r]$, by the definition of a balanced cut. We recursively call our algorithm on each C_i to obtain a subset $E_i \subseteq E(C_i)$. By our induction hypothesis, $|E_i| \leq \varepsilon(n_i - n_i^{1-\frac{1}{3d}})$. Finally, the algorithm returns $E = \bigcup_{i=0}^{r} E_i$. Thus, we have $|E| = |E_0| + \sum_{i=1}^{r} |E_i|$. As $\sum_{i=1}^{r} n_i = n$, $\sum_{i=1}^{r} |E_i| \leq \varepsilon n - \varepsilon \sum_{i=1}^{r} n_i^{1-\frac{1}{3d}}$. Furthermore, since $n_i \leq n/2$ for all $i \in [r]$ and $\sum_{i=1}^{r} n_i = n$, it holds $\sum_{i=1}^{r} n_i^{1-\frac{1}{3d}} \geq 2(\frac{n}{2})^{1-\frac{1}{3d}} = 2^{\frac{1}{3d}} n^{1-\frac{1}{3d}}$. Combining the bounds for $|E_0|$ and $\sum_{i=1}^{r} |E_i|$, we have

$$|E| \leq n^{1-\frac{1}{2d}} + \varepsilon n - \varepsilon \cdot 2^{\frac{1}{3d}} n^{1-\frac{1}{3d}}.$$

Recall our assumption $\varepsilon(2^{\frac{1}{3d}} - 1)n^{1-\frac{1}{3d}} \geq n^{1-\frac{1}{2d}}$. Together with the inequality above, this gives us $|E| \leq \varepsilon(n - n^{1-\frac{1}{3d}})$, so our induction works. In particular, when we apply Algorithm 1 on G_0, the output $E \subseteq E(G_0)$ satisfies $|E| \leq \varepsilon(|V(G_0)| - |V(G_0)|^{1-\frac{1}{3d}}) \leq \varepsilon |V(G_0)|$.

To analyze the time complexity of Algorithm 1, we consider the recursion tree T of our algorithm when applying on G_0. Each node $t \in T$ corresponds to a recursive call, and denote by $G_0(t)$ the corresponding graph handled in that call (which is a subgraph of G_0). The time cost at a node $t \in T$ is $O(|E(G_0(t))|^{1+\alpha'})$, which is just $O(|V(G_0(t))|^{1+\alpha'})$ because $|E(G_0(t))| \leq k|V(G_0(t))|$. Since $\textsc{ApprxMinCut}$ always returns a balanced cut, we know that $|V(G_0(t))| \leq |V(G_0(t'))|/2$ if t' is the parent of t in T. This implies that the depth of T is $O(\log |V(G_0)|)$. Also, the sum of $|V(G_0(t))|$ for all nodes $t \in T$ at one level of T is at most $|V(G_0)|$. Thus $\sum_{t \in T} |V(G_0(t))| \leq |V(G_0)| \log |V(G_0)|$ and $\sum_{t \in T} |V(G_0(t))|^{1+\alpha'} \leq |V(G_0)|^{1+\alpha'} \log |V(G_0)|$. The latter implies that the time complexity of Algorithm 1 is $O(|V(G_0)|^{1+\alpha'} \log |V(G_0)|)$, which is $O(|V(G_0)|^{1+\alpha})$ since $\alpha > \alpha'$. We remark that a more careful analysis can be applied to show that the running time of our algorithm is actually $O(|V(G_0)|^{1+\alpha'})$ instead of $O(|V(G_0)|^{1+\alpha'} \log |V(G_0)|)$. But for simplicity, we chose this looser analysis, which is already sufficient for our purpose. ◀

2.2 Computing the approximate realization

After computing the set $E \subseteq E(G)$ of edges using Lemma 4, we consider the graph $G - E$. Let $C_1, \dots, C_r$ be the weakly-connected components of $G - E$. We have $|V(C_i)| = (\frac{1}{\varepsilon})^{O(1)}$ by Lemma 4. For each $i \in [r]$, we want to compute a "kNN-realization" of C_i in $\mathbb{R}^d$. Of course, here each C_i is not necessarily k-regular, and thus a kNN-realization of C_i is not defined. But we can slightly generalize the definition of kNN-realization as follows. For a directed graph C, we say a map $\phi : V(C) \to \mathbb{R}^d$ is a *quasi-kNN-realization* of C in $\mathbb{R}^d$ if for every edge $(u, v) \in E(C)$, $|\{v' \in V(C) \backslash \{u\} : \|\phi(u) - \phi(v')\|_2 \leq \|\phi(u) - \phi(v)\|_2\}| \leq k$. By this definition, a kNN-realization is also a quasi-kNN-realization. Also, it is clear that if $\phi : V(H) \to \mathbb{R}^d$ is a quasi-kNN-realization of a directed graph H in $\mathbb{R}^d$, then for any subgraph C of H, the map $\phi_{|V(C)}$ is a quasi-kNN-realization of C in $\mathbb{R}^d$. Therefore, if G is kNN-realizable in $\mathbb{R}^d$, then each of $C_1, \dots, C_r$ admits a quasi-kNN-realization in $\mathbb{R}^d$.

Next, we discuss how to compute a quasi-kNN-realization of each C_i in $\mathbb{R}^d$ (or conclude it does not exist). To this end, we need the following lemma.

▶ **Lemma 5.** *Let C be a directed graph where $|V(C)| \geq k + 1$. If C admits a quasi-kNN-realization in $\mathbb{R}^d$, then there exists a k-regular supergraph C' of C with $V(C') = V(C)$ such that C' is kNN-realizable in $\mathbb{R}^d$.*

Proof. Assume $\phi : V(C) \to \mathbb{R}^d$ is a quasi-kNN-realization of C in $\mathbb{R}^d$. We can slightly perturb ϕ so that for any distinct $u, v, v' \in V(C)$, $\|\phi(u) - \phi(v')\|_2 \neq \|\phi(u) - \phi(v)\|_2\}$. Now define C' as a directed graph with $V(C') = V(C)$ and $E(C') = \{(u, v) \in V(C') \times V(C') : u \neq v \text{ and } \mathsf{rank}_\phi(u, v) \leq k\}$, where

$$\mathsf{rank}_\phi(u, v) = |\{v' \in V(C')\backslash\{u\} : \|\phi(u) - \phi(v')\|_2 \leq \|\phi(u) - \phi(v)\|_2\}|.$$

The construction guarantees that C' is k-regular and ϕ is a kNN-realization of C' in $\mathbb{R}^d$. Since ϕ is a quasi-kNN-realization of C, $E(C) \subseteq E(C')$ and thus C' is a supergraph of C. ◀

If $|V(C_i)| \leq k$, then any map from $V(C_i)$ to $\mathbb{R}^d$ is a quasi-kNN-realization of C_i. Otherwise, by the above lemma, to compute a quasi-kNN-realization of C_i in $\mathbb{R}^d$, it suffices to consider every supergraph C_i' of C_i with $V(C_i') = V(C_i)$ and try to compute a kNN-realization of C_i' (or conclude that C_i' is not kNN-realizable). Note that the number of such supergraphs is at most $\exp((\frac{1}{\varepsilon})^{O(1)})$ because $|V(C_i)| = (\frac{1}{\varepsilon})^{O(1)}$. If we obtain a kNN-realization of some C_i', then it is a quasi-kNN-realization of C_i. To compute a kNN-realization $\phi : V(C_i') \to \mathbb{R}^d$ of C_i', we formulate the problem as finding a solution to a system of $(\frac{1}{\varepsilon})^{O(1)}$ degree-2 polynomial inequalities on $(\frac{1}{\varepsilon})^{O(1)}$ variables. For each $v \in V(C_i')$, we represent the coordinates of $\phi(v)$ by d variables $x_1(v), \dots, x_d(v)$. Then for all distinct $u, v, v' \in V(C_i')$ such that $(u, v) \in E(C_i')$ and $(u, v') \notin E(C_i')$, we introduce a degree-2 polynomial inequality $\sum_{j=1}^d (x_j(u) - x_j(v))^2 \leq \sum_{j=1}^d (x_j(u) - x_j(v'))^2$, which expresses $\|\phi(u) - \phi(v)\|_2 < \|\phi(u) - \phi(v')\|_2$. Clearly, the solutions to this system of inequalities one-to-one correspond to the kNN-realizations of C_i' in $\mathbb{R}^d$. Renegar [23] showed that a system of p degree-2 polynomial inequalities on q variables can be solved in $p^{O(q)}$ time. Therefore, we can compute in $(\frac{1}{\varepsilon})^{(\frac{1}{\varepsilon})^{O(1)}}$ time a kNN-realization of C_i' in $\mathbb{R}^d$ (or decide its non-existence). This further implies that we can compute in $(\frac{1}{\varepsilon})^{(\frac{1}{\varepsilon})^{O(1)}}$ time a quasi-kNN-realization of C_i in $\mathbb{R}^d$ (or decide its non-existence). The total time cost for all C_i is then $(\frac{1}{\varepsilon})^{(\frac{1}{\varepsilon})^{O(1)}} \cdot n$, since $r \leq n$.

If C_i does not admit a quasi-kNN-realization in $\mathbb{R}^d$ for some $i \in [r]$, then we can directly conclude that G is not kNN-realizable in $\mathbb{R}^d$. Otherwise we construct a $(1 - \varepsilon)$-approximate kNN-realization of G in $\mathbb{R}^d$ as follows. Let $\phi_i : V(C_i) \to \mathbb{R}^d$ be the quasi-kNN-realization of C_i in $\mathbb{R}^d$ we compute, for $i \in [r]$. Define a map $\phi : V(G) \to \mathbb{R}^d$ as follows. Pick a vector $\vec{x} \in \mathbb{R}^d$ such that $\|\vec{x}\|_2 >> \max_{i \in [r]} \max_{u,v \in V(C_i)} \|\phi_i(u) - \phi_i(v)\|_2$. Then for each $i \in [r]$ and each $v \in V(C_i)$, set $\phi(v) = i\vec{x} + \phi_i(v)$. The choice of $\vec{x}$ guarantees that for a vertex $v \in V(C_i)$, the ϕ-images of the vertices in C_i are closer to $\phi(v)$ than the ϕ-images of the vertices outside C_i. Also, for any $u, v \in V(C_i)$, $\|\phi(u) - \phi(v)\|_2 = \|\phi_i(u) - \phi_i(v)\|_2$. Therefore, for any $u, v \in V(C_i)$, we have

$$\begin{aligned}
&\{v' \in V(G)\backslash\{u\} : \|\phi(u) - \phi(v')\|_2 \leq \|\phi(u) - \phi(v)\|_2\} \\
= &\{v' \in V(C_i)\backslash\{u\} : \|\phi(u) - \phi(v')\|_2 \leq \|\phi(u) - \phi(v)\|_2\} \\
= &\{v' \in V(C_i)\backslash\{u\} : \|\phi_i(u) - \phi_i(v')\|_2 \leq \|\phi_i(u) - \phi_i(v)\|_2\}.
\end{aligned}$$

Recall the function $\sigma_\phi : V(G) \times V(G) \to \{0, 1\}$ in Definition 1. The above equality shows that for $u, v \in V(C_i)$, if $|\{v' \in V(C_i)\backslash\{u\} : \|\phi_i(u) - \phi_i(v')\|_2 \leq \|\phi_i(u) - \phi_i(v)\|_2\}| \leq k$, then $\sigma_\phi(u, v) = 1$. It follows that $\sigma_\phi(u, v) = 1$ for any $(u, v) \in E(C_i)$ because ϕ_i is a quasi-kNN-realization of C_i, so we have

$$\sum_{(u,v) \in E(G)} \sigma_\phi(u, v) \geq \sum_{i=1}^r |E(C_i)| = |E(G)| - |E| \geq (1 - \varepsilon) \cdot |E(G)|.$$

Therefore, ϕ is a $(1-\varepsilon)$-approximate kNN-realization of G in $\mathbb{R}^d$. Combining the time costs for computing E and the maps $\phi_1, \ldots, \phi_r$, the overall time complexity of our algorithm is $O(n^{1+\alpha})$, where $O(\cdot)$ hides a constant depending on k, d, α, and ε.

▶ **Theorem 6.** *Let $\alpha > 0$ be any fixed number. Given a k-regular directed graph G of n vertices and a number $\varepsilon > 0$, one can compute in $f(k, d, \varepsilon) \cdot n^{1+\alpha}$ time a $(1-\varepsilon)$-approximate kNN-realization of G in $\mathbb{R}^d$, or conclude that G is not kNN-realizable in $\mathbb{R}^d$, where f is some computable function depending on α.*

3 kNN-Realization in $\mathbb{R}^1$

To the best of our knowledge, the kNN-realization problem on the line does not seem to have been studied, and it is the focus of this section. A number of line embedding problems are NP-hard as mentioned earlier, including triplet constraints and betweenness problems [9]. In the former, we are given a set of triplet constraints of the form $d(a, b) < d(a, c)$, while in the latter each constraint (a, b, c) requires the ordering to satisfy $a < b < c$. Given the hardness result, the focus in these problems is on approximating the maximum number of satisfied constraints in the linear ordering [17].

Unlike these intractable embedding problems on the line, we show that the partial order constraints implied by a kNN-realization problem have sufficiently rich structure to admit a polynomial time solution. Most of the difficulty is in *deciding* whether a k-regular directed graph G is kNN-realizable on the line; if the answer is yes, then one can easily compute the embedding in polynomial time using linear programming. (It is also worth pointing out that the decision problem for kNN-realization on the line is significantly more complicated than the special case of triplet constraints in [17] when *all $\binom{n}{3}$ triples are specified*. Indeed, in that case, one can guess the leftmost point, and then all the remaining points are immediately determined by the triplet constraints. This is not the case in kNN-realization: fixing the leftmost point does not fix its neighbors' order.)

The *decision* version of the kNN-realization problem, of course, is easy *if* we are given a permutation of G's vertices $(v_1, \ldots, v_n)$. In this case, we can easily compute a kNN-realization $\phi : V(G) \to \mathbb{R}^1$ satisfying $\phi(v_1) < \cdots < \phi(v_n)$, or decide that a feasible realization does not exist, by formulating the problem as a linear program (LP) with n variables $x_1, \ldots, x_n$ and the following set of constraints

- $x_i \le x_j$ for all $i, j \in [n]$ with $i \le j$,
- $\Delta_{i,j} < \Delta_{i,k}$ for all distinct $i, j, k \in [n]$ such that $(v_i, v_j) \in E(G)$ and $(v_i, v_k) \notin E(G)$, where $\Delta_{i,j} = x_{\max\{i,j\}} - x_{\min\{i,j\}}$ and $\Delta_{i,k} = x_{\max\{i,k\}} - x_{\min\{i,k\}}$.

Clearly, if $x_1, \ldots, x_n$ satisfy these constraints, then setting $\phi(v_i) = x_i$ gives us the desired realization. On the other hand, if ϕ is the realization we want, then the numbers $x_1, \ldots, x_n$ where $x_i = \phi(v_i)$ must satisfy the constraints. Therefore, we begin with this key problem: efficiently computing an ordering of the images of the vertices on $\mathbb{R}^1$.

3.1 Finding the vertex ordering

We say an ordering $(v_1, \ldots, v_n)$ of $V(G)$ is a *feasible vertex ordering* of G if there exists a kNN-realization $\phi : V(G) \to \mathbb{R}^1$ of G in $\mathbb{R}^1$ satisfying $\phi(v_1) < \cdots < \phi(v_n)$. The goal of this section is to give an $O(kn)$-time algorithm for computing a feasible vertex ordering of G (assuming it exists). We may assume, without loss of generality, that G is weakly-connected; otherwise we can consider each weakly-connected component of G individually.

Our first observation states two key properties of a feasible vertex ordering: (1) for each v_i its k out-neighbors form a contiguous block, and for different v_i's their blocks have the same linear ordering as the vertex ordering, and (2) for each v_i its in-neighbors also form a continuous block, which extends at most k vertices to the left and at most k to the right of v_i. (Recall that both $\mathsf{out}[v]$ and $\mathsf{in}[v]$ include v, for all vertices $v \in V(G)$.) See figure 1.

▶ **Observation 7.** *A feasible vertex ordering $(v_1, \dots, v_n)$ of G satisfies the following.*

1. *There exist $p_1, \dots, p_n \in [n-k]$ such that* **(i)** $\mathsf{out}[v_i] = \{v_{p_i}, \dots, v_{p_i+k}\}$ *for all $i \in [n]$,* **(ii)** $p_1 \leq \cdots \leq p_n$, *and* **(iii)** $p_i \leq i \leq p_i + k$ *for all $i \in n$.*
2. *There exist $q_1, \dots, q_n, q'_1, \dots, q'_n \in [n]$ such that* **(i)** $\mathsf{in}[v_i] = \{v_{q_i}, \dots, v_{q'_i}\}$ *for all $i \in [n]$,* **(ii)** $q_1 \leq \cdots \leq q_n$, **(iii)** $q'_1 \leq \cdots \leq q'_n$, *and* **(iv)** $q_i \leq q'_i \leq q_i + 2k$ *for all $i \in [n]$.*

Proof. Let $\phi : V(G) \to \mathbb{R}^1$ be a kNN-realization of G in $\mathbb{R}^1$ satisfying $\phi(v_1) < \cdots < \phi(v_n)$. Clearly, for each $i \in [n]$, the $k+1$ points in $\{\phi(v_1), \dots, \phi(v_n)\}$ closest to $\phi(v_i)$ are $\{\phi(v_{p_i}), \dots, \phi(v_{p_i+k})\}$ for some $p_i \in [n-k]$ such that $p_i \leq i \leq p_i + k$. Furthermore, one can easily verify that $p_1 \leq \cdots \leq p_n$. Assume $p_i > p_j$ for some $i, j \in [n]$ with $i < j$. Then we have $\phi(v_{p_j}) < \phi(v_{p_i}) \leq \phi(v_i) < \phi(v_j) \leq \phi(v_{p_j+k}) < \phi(v_{p_i+k})$. On the other hand, since $v_{p_i+k} \notin \{\phi(v_{p_j}), \dots, \phi(v_{p_j+k})\}$ and $v_{p_j} \notin \{\phi(v_{p_i}), \dots, \phi(v_{p_i+k})\}$,

$$\phi(v_j) - \phi(v_{p_j}) \leq \phi(v_{p_i+k}) - \phi(v_j) \leq \phi(v_{p_i+k}) - \phi(v_i) \leq \phi(v_i) - \phi(v_{p_j}),$$

which contradicts the fact $\phi(v_{p_j}) < \phi(v_{p_i}) \leq \phi(v_i) < \phi(v_j)$. By the definition of kNN-realization, we see $\mathsf{out}[v_i] = \{v_{p_i}, \dots, v_{p_i+k}\}$ for all $i \in [n]$. This proves condition 1.

Condition 2 follows from condition 1. Observe that for each $i \in [n]$, it holds that $\mathsf{in}[v_i] = \{v_j : p_j \leq i \leq p_j + k\} = \{v_j : i - k \leq p_j \leq i\}$. Since $p_1 \leq \cdots \leq p_n$, this implies $\mathsf{in}[v_i] = \{v_{q_i}, \dots, v_{q'_i}\}$ where $q_i = \min\{j : i - k \leq p_j \leq i\}$ and $q'_i = \max\{j : i - k \leq p_j \leq i\}$. The monotoncities $q_1 \leq \cdots \leq q_n$ and $q'_1 \leq \cdots \leq q'_n$ follow directly from the monotoncity $p_1 \leq \cdots \leq p_n$. It suffices to show that $q_i \leq q'_i \leq q_i + 2k$ for all $i \in [n]$. The inequality $q_i \leq q'_i$ is trivial. To see $q'_i \leq q_i + 2k$, assume that $q'_i > q_i + 2k$. Then there exists $j, j' \in [n]$ with $j' > j + 2k$ such that $i - k \leq p_j \leq i$ and $i - k \leq p_{j'} \leq i$. By condition 1, we have $j \geq p_j$ and $p_{j'} \geq j' - k > j + k$. Therefore, $p_{j'} > j + k \geq p_j + k \geq (i-k) + k = i$, which contradicts the fact $i - k \leq p_{j'} \leq i$. This proves condition 2. ◀

Consider the equivalence relation $\sim$ defined on $V(G)$ as $u \sim v$ iff $\mathsf{out}[u] = \mathsf{out}[v]$. Let $\mathcal{C}_G$ be the set of equivalence classes of this relation, which is a partition of $V(G)$ into groups of vertices with the same set of out neighbors. One can easily compute $\mathcal{C}_G$ in $O(k^2 n)$ time as follows. Given a vertex $v \in V(G)$, we check whether $\mathsf{out}[u] = \mathsf{out}[v]$, for all $u \in \mathsf{out}[v]$, which takes $O(k^2)$ time since $|\mathsf{out}[v]| = k + 1$ and each equality test takes $O(k)$ time. Thus, the time for computing all classes is $O(k^2 n)$. Interestingly, we use Observation 7 to compute $\mathcal{C}_G$ in $O(kn)$ time, if G is kNN-realizable in $\mathbb{R}^1$, as shown in the following Lemma. Due to space constraints the proof is included in the full version.

▶ **Lemma 8.** *Let G be a k-regular directed graph of n vertices. One can compute in $O(kn)$ time a partition $\mathcal{C}$ of $V(G)$ such that $\mathcal{C} = \mathcal{C}_G$ if G is kNN-realizable in $\mathbb{R}^1$.*

Write $r = |\mathcal{C}_G|$. Let $(v_1, \dots, v_n)$ be a feasible vertex ordering of G. Observation 7 implies that each class $C \in \mathcal{C}_G$ is a set of consecutive vertices in the sequence $(v_1, \dots, v_n)$, i.e., $C = \{v_\alpha, \dots, v_\beta\}$ for some $\alpha, \beta \in [n]$. Define $\mathsf{out}[C] = \mathsf{out}[v]$ for an arbitrary vertex $v \in C$. Therefore, there is a natural ordering $(C_1, \dots, C_r)$ of the classes in $\mathcal{C}_G$, in which the indices of the vertices in C_i are smaller than the indices of the vertices in C_j for all $i, j \in [r]$ with $i < j$. We call $(C_1, \dots, C_r)$ a *feasible class ordering* of G.

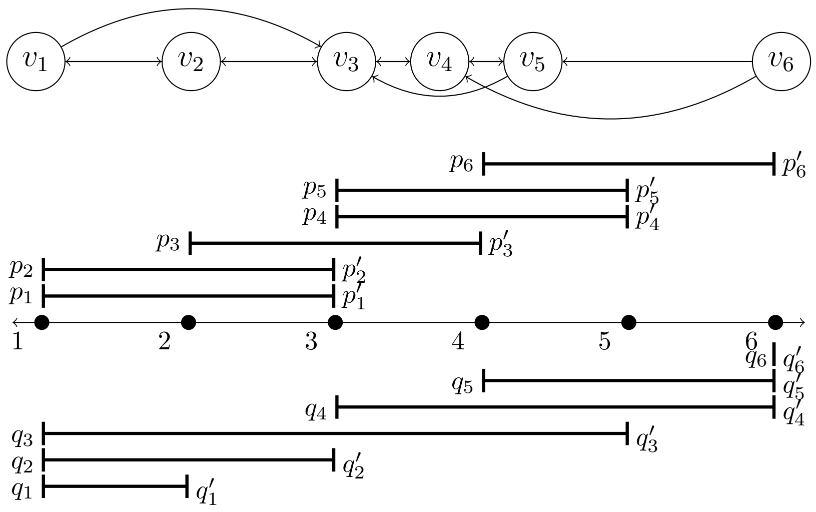

Figure 1 A 2-regular graph on 6 vertices and its realization (top). The values of p, p', q, q' for each vertex illustrate a natural ordering of the in and out-neighborhoods corresponding to the feasible vertex ordering $v_1, \cdots, v_6$ (bottom).

▶ **Observation 9.** *If $(C_1, \ldots, C_r)$ is the feasible class ordering of G corresponding to a feasible vertex ordering $(v_1, \ldots, v_n)$ of G, then there exist $c_1, \ldots, c_r \in [n-k]$ satisfying*

- $\mathsf{out}[C_i] = \{v_{c_i}, \ldots, v_{c_i+k}\}$ *for all $i \in [r]$,*
- $c_1 < \cdots < c_r$*,*
- *if G is weakly-connected, then $c_{i+1} \le c_i + k$ for all $i \in [r-1]$.*

Proof. Let $p_1, \ldots, p_n \in [n-k]$ be the indices in condition 1 of Observation 7, for the feasible vertex ordering $(v_1, \ldots, v_n)$. For each $i \in [r]$, set $c_i = p_s$ where $s = 1 + \sum_{j=1}^{i-1} |C_j|$, and we then have $\mathsf{out}[C_i] = \mathsf{out}[v_s] = \{v_{c_i}, \ldots, v_{c_i+k}\}$. Since $p_1 \le \cdots \le p_n$, we have $c_1 \le \cdots \le c_r$. Furthermore, for all $i \in [r-1]$, we have $c_i \ne c_{i+1}$, simply because $\mathsf{out}[C_i] \ne \mathsf{out}[C_{i+1}]$. Thus, $c_1 < \cdots < c_r$. To see the last property, assume $c_{i+1} > c_i + k$ for some $i \in [r-1]$. Then $(\bigcup_{j=1}^{i} \mathsf{out}[C_j]) \cap (\bigcup_{j=i+1}^{r} \mathsf{out}[C_j]) = \emptyset$. Note that $\bigcup_{j=1}^{i} C_j \subseteq \bigcup_{j=1}^{i} \mathsf{out}[C_j]$ and $\bigcup_{j=i+1}^{r} C_j \subseteq \bigcup_{j=i+1}^{r} \mathsf{out}[C_j]$. So there is no edge between $\bigcup_{j=1}^{i} C_j$ and $\bigcup_{j=i+1}^{r} C_j$, which implies that G is not weakly connected. ◀

To compute a feasible vertex ordering of G, we first compute a feasible class ordering, using the previous observation. Due to space constraints the proof of the following lemma is included in the full version.

▶ **Lemma 10.** *Let G be a weakly-connected k-regular directed graph of n vertices. Given $\mathcal{C}_G$, one can compute in $O(kn)$ time an ordering of $\mathcal{C}_G$, which is a feasible class ordering of G if G is kNN-realizable in $\mathbb{R}^1$.*

Next, we show how to compute a corresponding feasible vertex ordering of G given a feasible class ordering $(C_1, \ldots, C_r)$. By definition, we know that in the feasible vertex ordering, the vertices in C_i must appear before the vertices in C_j for all $i, j \in [r]$ with $i < j$. So it suffices to figure out the "local" ordering of the vertices in each class C_i. We do this by considering the in-neighbors of each vertex. As observed before, for each $v \in V(G)$, $\mathsf{in}[v]$ is the union of several classes in $\mathcal{C}_G$, and in addition, $\mathsf{in}[v] = \bigcup_{i=p}^{q} C_i$ for some $p, q \in [r]$, by condition 2 of Observation 7. It turns out that we can sort the vertices in each class according to the values of p and q. Formally, we prove the following lemma.

▶ **Lemma 11.** *Let G be a weakly-connected k-regular directed graph of n vertices, and $r = |\mathcal{C}_G|$. Given an ordering $(C_1, \dots, C_r)$ of $\mathcal{C}_G$, one can compute in $O(kn)$ time an ordering $(v_1, \dots, v_n)$ of $V(G)$ such that if $(C_1, \dots, C_r)$ is a feasible class ordering of G, then $(v_1, \dots, v_n)$ is a feasible vertex ordering of G.*

Proof. Our algorithm for computing a feasible vertex ordering of G is presented in Algorithm 2. First, for each $v \in V(G)$, we compute a pair $R(v) = (p, q)$ where $p \in [r]$ (resp., $q \in [r]$) is the minimum (resp., maximum) index such that $C_p \subseteq \mathsf{in}[v]$ (resp., $C_q \subseteq \mathsf{in}[v]$). We define a partial order $\preceq$ on index-pairs by setting $(p, q) \preceq (p', q')$ if $p + q \leq p' + q'$. Then we order the vertices in each class C_i as $(u_1, \dots, u_{|C_i|})$ such that $R(u_1) \preceq \cdots \preceq R(u_{|C_i|})$. If there exist $u, v \in C_i$ such that $R(u)$ and $R(v)$ are incomparable under the order $\preceq$, then we just arbitrarily order the vertices in C_i. We will see later that in this case, $(C_1, \dots, C_r)$ is not a feasible class ordering of G. By concatenating the orderings of $C_1, \dots, C_r$, we obtain an ordering $(v_1, \dots, v_n)$ of $V(G)$.

We first show that Algorithm 2 can be implemented in $O(kn)$ time. Computing $R(v)$ for all $v \in V(G)$ can be done in $O(kn)$ time, because $|\mathsf{in}[v]| = O(k)$ by (ii) of Observation 7. To see the time cost for ordering the vertices in each C_i, observe that $|C_i| \leq k$. Indeed, $v \in \mathsf{out}[v] = \mathsf{out}[C_i]$ for all $v \in C_i$, which implies $C_i \subseteq \mathsf{out}[C_i]$ and thus $|C_i| \leq |\mathsf{out}[C_i]| = k$. If we sort the vertices in C_i directly, it takes $O(k \log k)$ time. One can improve the time cost to $O(k)$ by observing that for any $v \in C_i$, the pair $R(v) = (p, q)$ satisfies $p \geq i - k$ and $q \leq i + k$, by condition 2 of Observation 7, which implies $2i - 2k \leq p + q \leq 2i + 2k$. In other words, in the sorting task, the keys are all in the range $\{2i - 2k, \dots, 2i + 2k\}$, whose size is $O(k)$. Thus, the task can be done in $O(k)$ time using bucket sorting. It follows that Algorithm 2 can be implemented in $O(kn)$ time.

To see the correctness of our algorithm, suppose $(C_1, \dots, C_r)$ is a feasible class ordering of G, and let $(v_1^*, \dots, v_n^*)$ be a corresponding feasible vertex ordering of G. We do not necessarily have $(v_1, \dots, v_n) = (v_1^*, \dots, v_n^*)$. However, as we will see, it holds that $(R(v_1), \dots, R(v_n)) = (R(v_1^*), \dots, R(v_n^*))$, which turns out to be sufficient. By (ii) of Observation 7, for each $v \in V(G)$ with $R(v) = (p, q)$, we have $\mathsf{in}[v] = \bigcup_{i=p}^{q} C_i$, and furthermore, $R(v_1^*) \preceq \cdots \preceq R(v_n^*)$. Now consider a class C_i. Note that $C_i = \{v_\alpha, \dots, v_\beta\} = \{v_\alpha^*, \dots, v_\beta^*\}$, where $\alpha = 1 + \sum_{j=1}^{i-1} |C_j|$ and $\beta = \sum_{j=1}^{i} |C_j|$. We have $R(v_\alpha^*) \preceq \cdots \preceq R(v_\beta^*)$, and our algorithm guarantees that $R(v_\alpha) \preceq \cdots \preceq R(v_\beta)$. Therefore, $(R(v_\alpha), \dots, R(v_\beta)) = (R(v_\alpha^*), \dots, R(v_\beta^*))$. It then follows that $(R(v_1), \dots, R(v_n)) = (R(v_1^*), \dots, R(v_n^*))$.

To see $(v_1, \dots, v_n)$ is a feasible vertex ordering of G, we further observe that the function $\pi : V(G) \to V(G)$ defined as $\pi(v_i) = v_i^*$ for $i \in [n]$ is an automorphism of G. Consider indices $i, j \in [n]$. We have $\mathsf{out}[v_i] = \mathsf{out}[v_i^*]$, since v_i and v_i^* belong to the same class in $\mathcal{C}_G$. Thus, $(v_i, v_j) \in E(G)$ iff $(v_i^*, v_j) \in E(G)$. On the other hand, because $R(v_j) = R(v_j^*)$, we have $\mathsf{in}[v_j] = \mathsf{in}[v_j^*]$ and hence $(v_i^*, v_j) \in E(G)$ iff $(v_i^*, v_j^*) \in E(G)$. As such, $(v_i, v_j) \in E(G)$ iff $(v_i^*, v_j^*) \in E(G)$, which implies that π is an automorphism of G. Now consider a kNN-realization $\phi : V(G) \to \mathbb{R}^1$ of G in $\mathbb{R}^1$ satisfying $\phi(v_1^*) < \cdots < \phi(v_n^*)$. Set $\phi' = \phi \circ \pi$, which is a map from $V(G)$ to $\mathbb{R}^1$. As π is an automorphism of G, ϕ' is also a kNN-realization of G. Furthermore, $\phi'(v_i) = \phi(\pi(v_i)) = \phi(v_i^*)$ for all $i \in [n]$, which implies $\phi'(v_1) < \cdots < \phi'(v_n)$. The existence of ϕ' shows that $(v_1, \dots, v_n)$ is a feasible vertex ordering of G. ◀

▶ **Theorem 12.** *Given a k-regular directed graph G of n vertices, one can compute in $O(kn)$ time an ordering $(v_1, \dots, v_n)$ of $V(G)$, which is a feasible vertex ordering of G if G is kNN-realizable in $\mathbb{R}^1$.*

Proof. For the case where G is weakly-connected, the theorem follows directly from Lemmas 8, 10 and 11. Otherwise, we compute the orderings for the weakly-connected components of G individually and concatenate them; this gives us the desired ordering of $V(G)$. ◀

Algorithm 2 $\textsc{VertexOrder}(G, (C_1, \dots, C_r))$.

```
1: for $v \in V(G)$ do
2:     $p \leftarrow \min\{i \in [r] : C_i \subseteq \mathsf{in}[v]\}$
3:     $q \leftarrow \max\{i \in [r] : C_i \subseteq \mathsf{in}[v]\}$
4:     $R(v) \leftarrow (p, q)$
5: for $i = 1, \dots, r$ do
6:     $L_i \leftarrow$ an ordering $(u_1, \dots, u_{|C_i|})$ of $C_i$ satisfying $R(u_1) \preceq \dots \preceq R(u_{|C_i|})$
7: return $L_1 + \dots + L_r$
```

3.2 Deciding the realizability and computing the realization

To decide the kNN-realizability of G, we first run the algorithm of Theorem 12, which returns the vertex ordering $(v_1, \dots, v_n)$ of $V(G)$, and then run the linear program given at the beginning of Section 3 for this ordering. Either the LP is feasible, in which case the solution gives a kNN-realization of G in $\mathbb{R}^1$, or LP is infeasible, in which case we can conclude that $(v_1, \dots, v_n)$ is not a feasible vertex ordering and thus G is not kNN-realizable in $\mathbb{R}^1$. Thus, the kNN-realization problem in $\mathbb{R}^1$ can be solved in polynomial time.

Interestingly, we can show that if we are only interested in the *decision* problem (and not actual embedding), then solving the LP is not necessary. Specifically, we can decide whether $(v_1, \dots, v_n)$ is a feasible vertex by simply checking condition 1 of Observation 7. If the ordering satisfies that condition, then the LP is *always feasible*. The proof of the following lemma is somewhat technical and is presented in the full version.

▶ **Lemma 13.** *Let G be a k-regular directed graph of n vertices. An ordering $(v_1, \dots, v_n)$ of $V(G)$ is a feasible vertex ordering of G iff it satisfies condition 1 of Observation 7. In particular, one can decide whether a given ordering of $V(G)$ is a feasible vertex ordering of G or not in $O(kn)$ time.*

We can now state the main result of this section.

▶ **Theorem 14.** *Given a k-regular directed graph G of n vertices, one can decide in $O(kn)$ time whether G is kNN-realizable in $\mathbb{R}^1$, and if so, a kNN-realization of G in $\mathbb{R}^1$ can be computed in $O(n^{2.5}\mathsf{poly}(\log n))$ time.*

4 Concluding remarks and extensions

We considered the problem of realizing a directed graph G as an Euclidean kNN graph. Our key results are: (1) for any fixed d, we can efficiently embed at least a $1 - \varepsilon$ fraction of G's edges in $\mathbb{R}^d$ or conclude that G is not realizable, and (2) a linear time algorithm to decide if G is realizable in $\mathbb{R}^1$. Our theorems extend to the case where the neighbors of each vertex in G are given as a *ranked* list, meaning that the embedding must satisfy $||\phi(v) - \phi(u_i)|| < ||\phi(v) - \phi(u_{i+1})||$, for $i = 1, \dots, k-1$, where u_i is the ith nearest neighbor of v (except in $\mathbb{R}^1$ where we need to solve the LP to decide if it is feasible). Our approximation scheme also applies to other proximity graphs that meet the following conditions: (1) the graph can be partitioned into constant-sized components using a sublinear size separator, and (2) each component's edges can be embedded independently. For example, we can approximately embed Delaunay triangulations in the plane with maximum degree $k = O(n^{\frac{1}{3}})$.

References

1 A. Agrawal, S. Saurabh, and M. Zehavi. A finite algorithm for the realizabilty of a delaunay triangulation. In *17th International Symposium on Parameterized and Exact Computation*, volume 249, pages 1:1–1:16, 2022. doi:10.4230/LIPICS.IPEC.2022.1.

2 N. Alon, M. Bădoiu, E. D. Demaine, M. Farach-Colton, M. Hajiaghayi, and A. Sidiropoulos. Ordinal embeddings of minimum relaxation: General properties, trees, and ultrametrics. *ACM Trans. Algorithms*, 4(4), 2008. doi:10.1145/1383369.1383377.

3 Y. Bilu and N. Linial. Monotone maps, sphericity and bounded second eigenvalue. *Journal of Combinatorial Theory, Series B*, 95(2):283–299, November 2005. doi:10.1016/J.JCTB.2005.04.005.

4 A. Bogomolnaia and J.-F. Laslier. Euclidean preferences. *Journal of Mathematical Economics*, 43(2):87–98, 2007.

5 P. Bose, W. J. Lenhart, and G. Liotta. Characterizing proximity trees. *Algorithmica*, 16:83–110, 1996. doi:10.1007/BF02086609.

6 J. Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel J. Math*, pages 46–52, 1985.

7 M. Bădoiu, E. D. Demaine, M. Hajiaghayi, A. Sidiropoulos, and M. Zadimoghaddam. Ordinal embedding: Approximation algorithms and dimensionality reduction. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 21–34, 2008.

8 V. Chatziafratis and P. Indyk. Dimension-accuracy tradeoffs in contrastive embeddings for triplets, terminals & top-k nearest neighbors. In *Symposium on Simplicity in Algorithms*, pages 230–243. 2024.

9 B. Chor and M. Sudan. A geometric approach to betweenness. *SIAM J. Discret. Math.*, 11(4):511–523, 1998. doi:10.1137/S0895480195296221.

10 J. Chuzhoy, Y. Gao, J. Li, D. Nanongkai, R. Peng, and T. Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1158–1167. IEEE, 2020.

11 M. de Berg, O. Cheong, M. van Kreveld, and M. H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2008.

12 M. Dillencourt. Toughness and delaunay triangulations. In *Proceedings of the Third Annual Symposium on Computational Geometry*, pages 186–194, 1987.

13 P. Eades and S. Whitesides. The realization problem for euclidean minimum spanning trees is np-hard. In *Proceedings of the Tenth Annual Symposium on Computational Geometry*, pages 49–56, 1994.

14 P. Eades and S. Whitesides. The logic engine and the realization problem for nearest neighbor graphs. *Theoretical Computer Science*, 169(1):23–37, 1996. doi:10.1016/S0304-3975(97)84223-5.

15 D. Eppstein, M. Paterson, and F. F. Yao. On nearest-neighbor graphs. *Discret. Comput. Geom.*, 17:263–282, 1997. doi:10.1007/PL00009293.

16 P. Erdős, F. Harari, and W. T. Tutte. On the dimension of a graph. *Mathematika*, 12:118–122, 1965.

17 B. Fan, D. Ihara, N. Mohammadi, F Sgherzi, A. Sidiropoulos, and M. Valizadeh. Learning Lines with Ordinal Constraints. *APPROX/RANDOM*, 176:45:1–45:15, 2020. doi:10.4230/LIPICS.APPROX/RANDOM.2020.45.

18 W. Johnson and J. Lindenstrauss. Extensions of lipschitz maps into a hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

19 H. Maehara. Space graphs and sphericity. *Discrete Applied Mathematics*, 7(1):55–64, 1984. doi:10.1016/0166-218X(84)90113-6.

20 J. Matousek. *Lectures on Discrete Geometry*. Springer, 2002.

21 G. L. Miller, S.H. Teng, W. P. Thurston, and S. A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44:1–29, 1997. doi:10.1145/256292.256294.

22 C. L. Monma and S. Suri. Transitions in geometric minimum spanning trees. *Discrete & Computational Geometry*, 8:265–293, 1992. doi:10.1007/BF02293049.

23 J. Renegar. On the computational complexity and geometry of the first-order theory of the reals. *Journal of Symbolic Computation*, 13(3):255–299, 1992.

24 K. Sugihara. Simpler proof of a realizability theorem on delaunay triangulations. *Information Processing Letters*, 50(4):173–176, 1994. doi:10.1016/0020-0190(94)00027-1.