Universidade Estadual de Londrina

LARISSA CAPOBIANCO SHIMOMURA

PROXIMITY GRAPHS FOR SIMILARITY SEARCHES: EXPERIMENTAL SURVEY AND THE NEW CONNECTED-PARTITION APPROACH HGRAPH

Londrina
2019

LARISSA CAPOBIANCO SHIMOMURA

PROXIMITY GRAPHS FOR SIMILARITY SEARCHES: EXPERIMENTAL SURVEY AND THE NEW CONNECTED-PARTITION APPROACH HGRAPH

A thesis presented to the Graduate Program in Computer Science at the State University of Londrina to obtain the degree of Master of Science in Computer Science.

Orientador: Prof. Dr. Daniel dos Santos Kaster

Londrina
2019

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

Shimomura, Larissa Capobianco.
Proximity Graphs for Similarity Searches: Experimental Survey and the New Connected-Partition Approach HGraph / Larissa Capobianco Shimomura. - Londrina, 2019.
114 f. : il.

Orientador: Daniel dos Santos Kaster.
Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2019.
Inclui bibliografia.

1. Similarity Search - Tese. 2. Proximity Graphs - Tese. 3. Experimental Survey - Tese. 4. Connected-Partition Approach - Tese. I. Kaster, Daniel dos Santos. II. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. III. Título.

LARISSA CAPOBIANCO SHIMOMURA

PROXIMITY GRAPHS FOR SIMILARITY SEARCHES: EXPERIMENTAL SURVEY AND THE NEW CONNECTED-PARTITION APPROACH HGRAPH

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

Orientador: Prof. Dr. Daniel dos Santos Kaster
Universidade Estadual de Londrina – UEL

Prof. Dr. José Fernando Rodrigues Junior
Universidade de São Paulo – USP

Prof. Dr. Alan Salvany Felinto
Universidade Estadual de Londrina – UEL

Londrina, 16 de abril de 2019.

To my parents for their love, support and encouragement.

ACKNOWLEDGEMENTS

I would like to thank the National Council for Scientific and Technological Development (CNPq) and the National Council for the Improvement of Higher Education (CAPES) for the financial support.

In addition, I would like to thank my advisor Daniel dos Santos Kaster for the guidance during the past years. I am grateful to all the colleagues of the CROSS laboratory with whom I had the pleasure to work with during these past couple years.

At last, but definitely not least I would like to thank my family, boyfriend and all of my close friends for their support. You are always there for me when I need.

"The greatest challenge to any thinker is stating the problem in a way that will allow a solution."
(Bertrand Russel)

SHIMOMURA, L. C. **Grafos de proximidade para consultas por similaridade:** análise experimental e a nova abordagem de partições conectadas HGraph. 2019. 114 f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2019.

RESUMO

O desenvolvimento tecnológico acelerou o crescimento do volume de dados complexos como imagens, vídeos, séries temporais e dados geográficos. Uma abordagem bastante utilizada para a recuperação de dados complexos são as consultas por similaridade. As consultas por similaridade têm como objetivo principal recuperar dados similares a partir de características intrínsecas dos dados. Assim, para facilitar a recuperação de dados complexos usando consultas por similaridade é necessário organizar grande quantidade de dados de forma que dados similares possam ser recuperados da forma mais rápida possível. Diversos métodos de acesso foram propostos na literatura para tornar a recuperação por similaridade de grandes bases de dados mais rápida. Artigos publicados recentemente indicam que métodos que utilizam grafos são bastante eficientes e superam o desempenho de métodos de outras categorias em várias situações. Porém, de acordo com nosso conhecimento, nenhum trabalho se dedicou a realizar uma análise experimental em um número abrangente de métodos baseados em grafos utilizando os mesmos algoritmos de busca e o mesmo ambiente. Esta dissertação apresenta uma revisão bibliográfica sobre os principais tipos de grafos utilizados para consultas por similaridade, também foi realizado uma avaliação experimental utilizando os mesmos algoritmos de consulta com resposta exata e com resposta aproximada. O métodos foram avaliados conforme seu comportamento considerando os principais parâmetros de construção e consulta para uma variedade de bases de dados reais. A partir dos resultados desta avaliação foi proposto o método baseado em grafos, HGraph. O HGraph é um método baseado em partições conectadas proposto para construir grafos de proximidade e responder consultas por similaridade. O HGraph utiliza uma estratégia de divisão e conquista para construir tipos de grafos propostos na literatura. Para conectar as diferentes partições do processo de divisão e conquista arestas longas foram adicionadas ao HGraph. Foi avaliado o comportamento dos principais parâmetros do HGraph e seu desempenho quanto a tempo de construção e consultas por similaridade foram comparados com os métodos k-NNG (k-Nearest Neighbors Graph), NSW (Navigable Small World Graph) e SAT (Spatial Approximation Tree). Como resultado, o HGraph foi capaz de melhorar o k-NNG em termos de tempo de construção, tempo de consulta e qualidade de resposta. Além disso, o HGraph obteve melhor desempenho na busca em alguns datasets quando comparado ao NSW e a SAT.

Palavras-chave: Consultas por similaridade. Grafos de proximidade. Espaços métricos. Análise experimental. Abordagem de partições conectadas.

SHIMOMURA, L. C. **Proximity graphs for similarity searches:** experimental survey and the new connected-partition approach HGraph. 2019. 114 p. Dissertation (Master's Deegre in Computer Science) – Universidade Estadual de Londrina, Londrina, 2019.

ABSTRACT

The technology development has accelerated the growth of the volume of complex data, such as images, videos, time series, and georeferenced data. A widely used approach to retrieve complex data are the similarity searches. The similarity searches aim at retrieving similar data according to intrinsic characteristics of the data. Therefore, in order to facilitate the retrieval of complex data using similarity searches, it is necessary to organize large collections of data in a way that similar data can be retrieved in the shortest time as possible. Several access methods were proposed in the literature to speed up similarity data retrieval from large databases. Recently, graph-based methods have emerged as a very efficient alternative for similarity retrieval, with reports indicating they have outperformed methods of other categories in several situations. However, to the best of our knowledge, there is no previous work with experimental analysis on a comprehensive number of graph-based methods using the same search algorithm and execution environment. This work presents two main contributions. The first contribution is a survey on the main graph types currently employed for similarity searches and an experimental evaluation of the most representative graphs in a common platform, for exact and approximate search algorithms. We evaluated the relative performance behavior of these graphs with respect to the main construction and query parameters for a variety of real-world datasets. According to the evaluation of the results, we propose a new graph-based method called HGraph, the second contribution of this work. HGraph is a connected partition approach to build graph-based methods and answer similarity searches. In HGraph we use a divide and conquer strategy to build graph-based methods proposed in the literature and add long-range edges to connect the different partitions. These long-range edges are added in order to increase the answer quality compared to their "base" graph type. We evaluated the HGraph main parameters behavior and compared the HGraph construction time and similarity search performance for approximate searches to the k-NNG, SWG and the SAT. As a result, the HGraph method was able to accelerate the k-NNG graph construction and improve the k-NNG query time and query recall. Thus, the HGraph performed better in similarity searches than the SWG and the SAT in some datasets.

Keywords: Similarity search. Proximity graphs. Metric spaces. Experimental survey. Connected-partition approach.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

BFS	Breadth first Search
DBMS	Database Management System
DG	Delaunay Graph
DFS	Depth First Search
DCT	Division Control Tree
DLG	Double Layer Neighborhood Graph
FANNG	Fast Approximate Nearest Neighbor Graph
GG	Gabriel Graph
GNNS	Graph Nearest Neighbor Search
GS	Greedy Search
HRG	Hyperspherical Region Graph
ILS	Iterated Local Search
k-DRG	k-Degree Reduced Graph
k-DG	k-Delaunay Graph
k-GG	k-Gabriel Graph
k-NNG	k-Nearest Neighbor Graph
k-NN	k-Nearest Neighbor Query
k-RNG	k-Relative Neighborhood Graph
LSH	Locality Sensitive Hashing
MST	Minimum Spanning Tree
NSW	Navigable Small World Graph
NNG	Nearest Neighbor Graph
PGS	Parallel Greedy Search
PBKNNG	Pruned Bi-Directed Graph

Rq	Range Query
RNDF	Relative Neighborhood Density Factor
RNG	Relative Neighborhood Graph
SWG	Small World Graph
SAT	Spatial Approximation Tree

CONTENTS

1 INTRODUCTION

The rapid technological development has enabled users to produce every day complex data such as images, videos, long texts, etc. As the volume of complex data grows it also grows the necessity of storing and retrieving this kind of data efficiently to support modern applications [1].

Unlike traditional data (e.g., numeric values and short character strings), most complex data cannot be retrieved using classical comparison approaches such as ordering or equality. Complex data do not have a **total order relation**, thus they cannot be simply retrieved using comparison operators such as $<, >, \leq, \geq$ [2]. Identity comparisons are also of little help. For example, two images would be retrieved as equals if they are equal pixel-wise what is very difficult when they are from different sources. A more suitable and used approach to retrieve complex data are the similarity searches [3].

Similarity searches are based on retrieving similar data of one or more data used as a reference according to an intrinsic characteristic of the data. For example, from an image, it can be extracted characteristics based on their color, shape, texture, etc [4]. Similarity retrieval of data is employed on a wide range of modern applications, for example, content-based image retrieval, pattern recognition, and recommender systems, to name a few [5, 6, 7].

To search by similarity the characteristics that will be used to compare the complex data are translated into a set of features, called feature vector [2]. The similarity between two complex data can be measured by applying a (dis)similarity function to the feature vectors of the compared data. Usually, a distance function is used to calculate the (dis)similarity among complex data [8, 3].

Similarity searches can have a high computational cost due to the big volume of data and the high complexity of calculating distance functions [3, 9]. So, how complex data is modeled in databases is very important for similarity searches since it can speed up similar data retrieval.

Most of the traditional Database Management Systems (DBMS) includes indexing methods such as the B-tree and hashing methods, however, these methods are not suited for complex data similarity retrieval. As a result, new indexing structures were proposed in the literature [4]. Some of these methods are: Slim-tree [10], CM-tree [11], Locality Sensitive Hashing (LSH) [12] and *proximity graphs* [13, 9].

Graphs allow big interconnectivity of data, due to its structure, enabling to explore relationships and neighbors in an agile way. Graphs are very used not only as data models but also as indexing structures in recommendation systems and social networks [14].

Recently, graph-based methods have emerged as a very efficient option to execute similarity queries in metric and non-metric spaces [15, 16, 17]. This thesis addresses the use of graphs to accelerate similarity searches.

1.1 Contributions

When using graphs to model the similarity space, it can be faster to retrieve similar data by exploring the neighborhood (adjacent vertices) of a vertex or even to perform other operations such as relevance feedback [18] compared to other methods in the literature.

To model the similarity space as a graph, a common approach is to use vertices to represent complex data and edges connecting two vertices as the similarity relationship between the pair of complex data [16, 9, 17, 13, 19]. Pairs of vertices can be connected by edges according to special conditions, for instance, in *k-NN* graphs, each vertex has an edge connecting it to each of its k-Nearest Neighbors. Notice that in this approach the type of graph is defined by how the vertices are connected. In this thesis we use this approach.

Other existing approaches of using graphs in similarity search consists of searching complex data by their structural similarity. Such an approach is related to modeling complex data by using graph-based representations. These representations of complex data are usually based on the structure of the data and the similarity can be computed by graph similarity algorithm [20]. In this case, the query element is represented by a graph instead of a feature vector. Graph similarity is a challenging problem and several graph indexing methods have been proposed. Examples of these methods are the gIndex [21] and Graph-Grep [22]. Despite this is a relevant issue, our focus in this work is to use graphs to represent the similarity space itself, with vertices and edges representing, respectively, complex data elements and the similarity (or proximity) relationship between them. Graph-based structure representations of complex data are out of the scope of the work.

Some graph-based methods proposed have already demonstrated superior efficiency when compared to other types of similarity approximate search methods, such as LSH (Locality Sensitive Hashing) [17], and exact methods, like AESA [9]. However, there are no survey articles comparing the main graph-based methods of the literature. Survey articles on similarity search only cite graphs as a theoretical background to other methods or for specific domains [3, 23]. Moreover, some properties of types of graphs seem to be very effective to enhance the precision of approximate queries, such as *long edges*. Therefore, another open problem is to address whether and how these properties can be successfully integrated into other well-known proximity graphs to improve them.

In this context, we focus on two main contributions in this thesis:

1. A comparison of graph-based methods – This contribution aims at providing a comprehensive performance analysis of the graph-based methods presented in the literature according to the type of graph, graph construction, and search algorithm execution and applicability. We describe the main graph types and analyze how the parameters affect both construction and search execution
2. *HGraph* method – A connected partition approach to build types of graphs used for similarity searches. The *HGraph* objective is to accelerate the construction of the graph-based methods using a divide and conquer approach and increase the similarity search quality (query time and recall) by adding what we call long-range edges.

The results of the comparison of graph-based methods provide a quantitative view of the exact search compared to accurate setups for approximate search. These results reinforce the tradeoff between graph construction cost and search performance according to the construction and search parameters. With respect to the approximate methods, the *NSW* (Navigable Small World Graph) presented the highest recall rates. Nevertheless, given a recall rate, there is no winner graph for query performance. The results of the evaluation of the parameters of the *HGraph* showed that the use of long-range edges could increase the search recall and the *HGraph* method can build a *k-NNG* (k-Nearest Neighbors Graph) with high accuracy. When comparing the *HGraph* to other graph-based methods given a recall rate = 1 (exact answer) the *HGraph* was able to outperform or have approximate query time compared to other graph-based methods.

1.2 Outline

This master's thesis is organized into 5 chapters. The summary of the remaining chapters is as follows:

- Chapter 2, *Basic Concepts* presents the necessary background to understand this thesis. More specifically background on metric spaces, similarity search methods, and proximity graphs;
- Chapter 3, *Graph-based methods for similarity search* contains an overview of existing graph-based methods for similarity search, including the main types of graphs and search algorithms;
- Chapter 4, *Comparative analysis of graph-based methods* presents the first part of this thesis contribution, experimental evaluation of existing graph-based methods for similarity search, its results and conclusions;

- Chapter 5, presents the *HGraph*, our proposed method for similarity search and experimental results of the *HGraph* parameters behavior and comparison to other graph-based methods.

2 BASIC CONCEPTS

In this chapter, we give the necessary background on similarity search and proximity graphs to read the thesis. Section 2.1 presents the basic concepts of similarity searches while Section 2.2 introduce the proximity graphs, which is a class of graphs that have been used for similarity searches.

2.1 Similarity Search

Similarity searches retrieve similar data to one or more reference elements according to an intrinsic characteristic. Similarity searches can be applied to different domains of data: images, audio, video, geographic data, etc [3].

Figure 1 shows a diagram on the similarity search process, taking an image database as an example. As it is illustrated in the figure, the similarity search process can be divided into 3 major sections:

- Feature extraction – responsible for automatic feature extraction of the raw data into numerical representations called feature vectors;
- (dis)Similarity measurement – responsible for comparing feature vectors, usually a distance function;
- Search – responsible for accessing and retrieving similar data from the database.

The feature extraction is responsible for transforming the complex data content of interest into representations of it, called feature vectors [24]. Feature vectors are used to

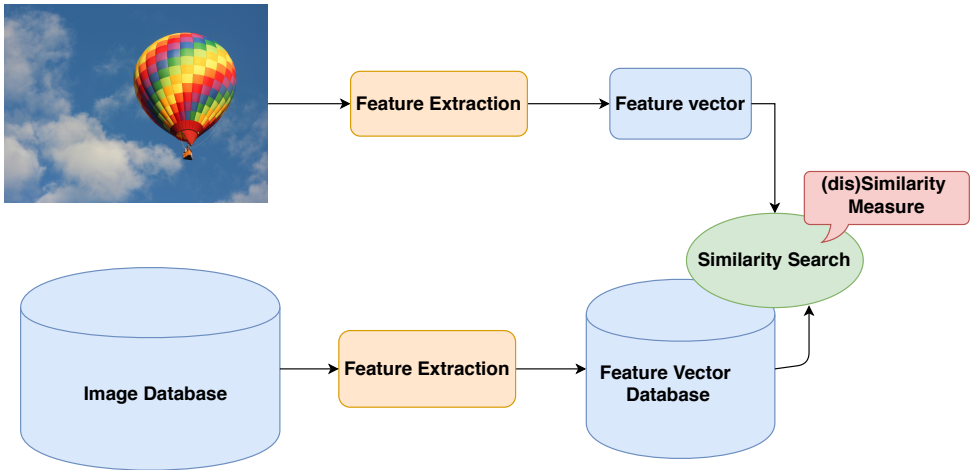

Figure 1 – Similarity search process.

index and compare the complex data. Each different data domain has relevant features that can be used to represent the data. The extracted feature depends on the necessity and on the context of the problem the similarity search is being applied to. An example is Context-Based Image Retrieval (CBIR) systems for Computer Aided Diagnosis (CAD). In this case, similarity searches can be used to retrieve images to assist diseases diagnoses, such as Lung and Breast Cancer through computed tomography and mammography scans using texture features extracted from them [25, 6].

2.1.1 Similarity Measures

To compare two or more different feature vectors from complex data it is necessary to have a measure that can report how much these data are similar or dissimilar. Usually, this measure is a distance function δ that returns a quantitative result. The resulting distance represents how much the compared feature vectors are dissimilar from each other. The bigger the distance the more dissimilar are the compared data and the closer to zero the more similar they are [8].

The combination of the feature vector and the (dis)similarity function used is called similarity space. Since distance functions are commonly used as a (dis)similarity function, the similarity space can be modeled as a metric space. Considering $x, y, z \in$ domain $\mathbb{S}$ and a distance function $\delta : \mathbb{S} \times \mathbb{S} \to \mathbb{R}$, the metric space is an unordered pair $\langle \mathbb{S}, \delta \rangle$ which the following properties hold [8]:

- Non-negativity – $\delta(x, y) \geq 0$;
- Symmetry – $\delta(x, y) = \delta(y, x)$;
- Identity – $\delta(x, y) = 0 \Leftrightarrow x = y$;
- Triangle inequality – $\delta(x, y) \leq \delta(x, z) + \delta(z, y)$.

There are several distance functions that can be used to measure the similarity between complex data. The most commonly used metric distance functions are the so-called L_p norms (or Minkowski norms) for any $1 \leq p \leq \infty$. The Minkowski norms are given in Equation 2.1 in which n is the feature vector length [26].

$$\delta(x, y) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \tag{2.1}$$

Commonly used distances from the L_p norms are the L_1 (Manhattan distance), L_2 (Euclidean distance) and L_∞ (Chebyshev distance). The L_1 distance, also called Manhattan or city block distance, computes the distance between two points in a grid-like path. The L_2 distance, the Euclidean distance, is close to our sense of distance as it computes

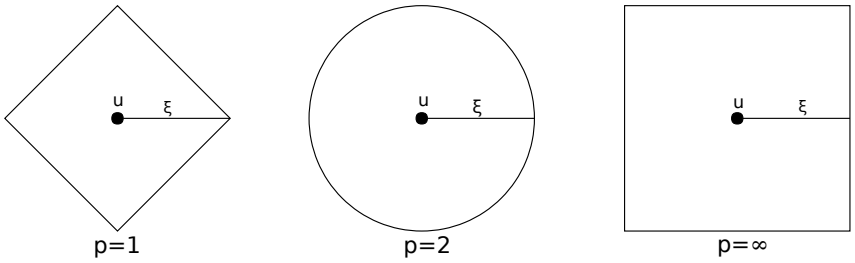

Figure 2 – L_1, L_2 and L_∞ distances.

the distance between two points in a straight line. The L_∞ distance, also called Chebyshev distance, between two points is the maximum difference on a coordinate. Figure 2 shows the difference of these distances in space from a center element u to points that have the same distance in each metric.

Aside from the metric distance functions sometimes it is necessary to use (dis)similarity functions that do not hold all of the metric space properties. When this kind of (dis)similarity functions are used the similarity space is modeled as a *non-metric space* [7]. Examples of non-metric distance functions are L_p norms for $p < 1$ and the Cosine distance (a distance that calculates the cosine of the angle between two feature vectors) [27].

2.1.2 Similarity Queries

Similarity queries are queries that retrieve elements from a database according to a query operator that is based on similarity. There are two basic types of similarity queries, the k-Nearest Neighbor query ($kNNq$) and the Range query (Rq). The formal definitions to these two types of queries are as follows [3, 28, 29].

- *Range* Query – *Rq*: Given a query element $q \in \mathbb{S}$ the $Rq(q, \xi)$ retrieves all the elements from $S \subset \mathbb{S}$ that are within a parameter defined distance ξ to q according to a distance function δ. The result set of this query can be expressed as $\{s_i \in S | \delta(s_i, q) \leq \xi\}$.

- k-Nearest Neighbor (k-*NN*) Query – *kNNq*: Given a query element $q \in \mathbb{S}$ the $kNNq(q, k)$ retrieves the k closest elements to q from $S \subset \mathbb{S}$ according to a distance function δ. The result set $S_k \subset S$ of this query can be expressed as $\{s_i \in S \mid |S_k| = k \text{ and } \forall s_k \in S_k, s_j \in S - S_k, \delta(q, s_k) \leq \delta(q, s_j)\}$

Figures 3 and 4 show, respectively, examples of a Rq and of a $kNNq$ ($k = 5$) in a 2-dimensional Euclidean space. There also are many other types of similarity operations, for example, the disjunction and conjunction of *Range* and *k-NN* query [30], similarity joins [31] and similarity group-by [32].

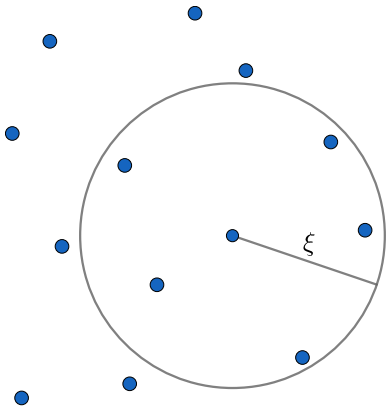

Figure 3 – *Range* query.

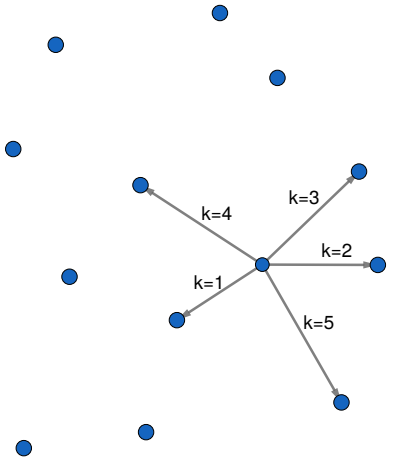

Figure 4 – *k-NN* query.

2.1.3 Similarity Search Methods

A naive strategy to solve similarity searches is to make a sequential search in the dataset comparing each element to the query element. However, this strategy is inefficient because of the high computational cost of calculating similarity functions, especially in high dimensional spaces with a big volume of data. Indexing methods for traditional types of data, e.g. B-trees, are available in Database Management Systems (DBMS) to speed up the searching process. However, indexing structures proposed for traditional types of data are not suitable for complex data and similarity searches as these structures are based on the total order relation [4]. As a consequence new methods were proposed to make similarity-based retrieval faster. The main types of methods are:

- Tree-based methods – Use trees as a structure to index complex data. Tree-based methods differ from each other on how to partition data hierarchically to build the structure, how to choose the pivots and if the structures are dynamic or static. The indexes can use, for example, ball partitioning (e.g. Vantage-Point tree – VP-tree [33]) or generalized hyperplane partitioning (e.g. GH-tree [34]) [4]. Other than the already cited, a wide range of tree structures for similarity search was proposed in the literature [3]. Some examples are M-tree [1], Slim-tree [10], DBM-tree [11], etc.

- Permutation-based methods – In permutation-based methods, every dataset element is represented as a permutation of a set of pivots sorted by the distance to the element. The similarity of objects is based on their relative distances to pivots [15, 35]. Methods that use permutations include MI-File [36] and PP-Index [37].

- Hashing-based methods – According to [Wang et. al.] [38], there are two main approaches to use hashing for similarity search: indexing data items using hash tables (i.e., store items with the same hash code in the same hash bucket), and using hash

codes to approximate the distance. A well-known hashing algorithm for approximate similarity search is the Locality Sensitive Hashing (LSH) [12].

- Graph-based methods – In graph-based methods, the similarity space is modeled as a graph. A common approach is to use vertices as complex data and edges as the similarity relationship between the connecting pair of vertices. Graph-based methods include *k-NNG* [9], *RNG* [13] and *k-DRG* [39].

According to Harwood e Drummond[40] tree-based methods and graph-based methods are able to partition the dataset in a very similar way. However, search algorithms in graph-based methods can be more computationally efficient. That is because in tree-based methods the propagation of the search algorithm works from the root to the bottom of the tree. Even though this process is computationally efficient the average recall that one single propagation achieves can be very low. In order to increase the recall rates backtracking algorithms are used. Every time a wrong path is taken during the search backtracking is needed in order to take the search to another path. When an error is made in the lower layers of the tree the only solution is to re-traverse the tree choosing each time a different path. In graph-based methods, we do not use a global hierarchical structure so when executing a search algorithm that uses backtracking (similar to the tree-based methods) the backtracking cost is uniform independently of the point the wrong path is taken.

Thus, according to Ocsa, Bedregal e Cuadros-Vargas[13], the selection of a bad root in a tree-based method can lead to excessive node exploration while graph-based methods can start from a vertex closer to the query element since it does not use a global hierarchical structure. Starting from a closer vertex can reduce significantly the number of distance computations in queries as the number of vertices to be evaluated during the search are also reduced.

Other results in the literature also showed that graph-based methods can perform better than hashing and tree-based methods[41, 17]. Section 2.2 presents a theoretical background on graph-based methods and Chapter 3 gives more details on existing graph-based methods in the literature and how it performed against other methods. Therefore, considering the mentioned advantages of graph-based methods and the results in the literature, the focus of this work is on graph-based methods.

2.2 Proximity Graphs and the Spatial Approximation Property

A graph is defined as $G = (V, E)$, where V is the set of vertices (nodes) and E is the set of edges that connect pairs of vertices in V. The most common type of graphs used for complex data retrieval using similarity search is the **Proximity Graph** [13, 16, 9].

A proximity graph is a graph where each pair of vertices $(v, u) \in V$ is connected by an edge $e = (u, v)$, $e \in E$, if and only if u and v fulfill a defined property P [13]. Property P is called *neighborhood criterion* and it defines the type of proximity graph. The edges in E can be weighted or not. The weight is usually the proximity measurement between the connecting vertices, such as the distance between them ($\delta(u, v)$) [9].

The fundamental approach to perform similarity queries on a proximity graph are to employ the so-called *spatial approximation*, introduced by Navarro in [42]. Given a query element q and proximity measure δ, this approach consists in starting from a given vertex $u \in V$ and iteratively traverse the graph in a way to get spatially closer and closer to the elements that are the most similar to q. Every iteration propagates the search from a vertex u to its "neighbors" ($N(u)$) that are closer to q and consequently more likely to reach the answer, where vertices in $N(u)$ are adjacent to v in the graph. According to the author, this approach can answer exact results for metric spaces.

Given a metric space $\langle \mathbb{S}, \delta \rangle$ and a graph $G = (V, E)$, where $V \subseteq \mathbb{S}$ and every $e \in E$ has the form $e = (u, v)$ such that $v \in N(u)$, G must fulfill the Property 2.2 to correctly answer similarity queries using a search algorithm based on the spatial approximation approach, for any query element $q \in \mathbb{S}$:

$$\forall u \in V, \text{ if } \quad \forall v \in N(u), \delta(q, u) \leq \delta(q, v), \text{then} \quad \forall v' \in V, \delta(q, u) \leq \delta(q, v') \tag{2.2}$$

Property 2.2 means that if it is not possible to get closer q than u going to its neighbors, then u is the closest element to q in the graph. The trivial graph that satisfies this property is the complete graph (i.e., every vertex is connected to all vertices in the graph), however in this case the search ends up in a sequential scan. Therefore, it is necessary to use graphs with fewer edges, as the other proximity graphs described in the following subsections and in Chapter 3.

2.2.1 Delaunay Graph and its Subgraphs

One of the most important proximity graphs in the literature is the Delaunay Graph (DG). The Delaunay Graph is dual to the Voronoi Diagram [43]; it is a planar graph in which the points of each adjacent Voronoi region are connected by an edge [44]. For a better understanding of the Delaunay Graph it is important to define the Voronoi Diagrams.

Considering a set $P = p_1, p_2, p_3, \ldots, p_n$ of n points in $\mathbb{R}^2$, the general idea of the Voronoi diagram is to make subdivisions on the plane in n regions associating each point from P to a region, called Voronoi region [45]. The Voronoi region of a point $p_i \in P$, denoted by $V(p_i)$, is a subdivision of the Voronoi diagram in which any point $x \in \mathbb{R}^2$ is

closest to p_i than to any other point p_j, as noted in Equation 2.3, where δ is the Euclidean (L_2) distance [46].

$$V(p_i) = \{x \in \mathbb{R}^2 : \delta(p_i, x) \leq \delta(p_j, x), 1 \leq j \leq n\} \tag{2.3}$$

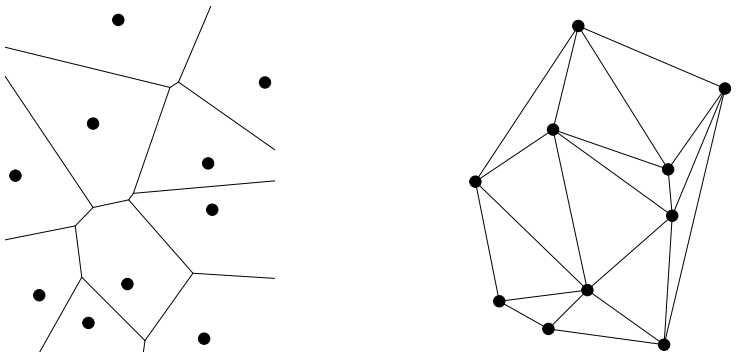

Figure 5 – Voronoi Diagram and Delaunay Graph for the same point set.

Figure 5 shows a Voronoi Diagram and its dual, the Delaunay Graph. As shown by Navarro[42], the definition of the Voronoi regions is correspondent to the spatial approximation property. Hence, using the DG, search algorithms based on the spatial approximation can return exact results. However, DG has two major limitations: (1) it is not possible to compute the DG for a generic metric space given only the set of dataset distances, since it is required extra information on the internal structure of the metric space [16, 13, 42]; and (2) despite previous works have extended the DG for high dimensional data [47, 48, 49, 50], the existing solutions suffer from the curse of dimensionality [16], as the graph can become a complete graph very quickly as the dimension of the data increases [40]. The DG limitations will be further discussed in subsection 2.2.3.

There are previous works that define proximity graphs as subgraphs of DG [51]. The most representative graph types and their proximity property P, considering Euclidean spaces, are [13]:

- Gabriel Graph (GG): $(u, v) \in E$ if and only if there is no other vertex in the graph that is inside the ball with radius $\frac{\delta(u,v)}{2}$ which u and v are exactly in opposite extremes;
- Relative Neighborhood Graph (RNG): $(u, v) \in E$ if and only if there is no vertex in the intersection between balls $B_u(u, \delta(u, v))$ and $B_v(v, \delta(v, u))$;
- Minimum Spanning Tree (MST): a connected acyclic subgraph of a graph that connects all the vertices with the minimal sum of weights of its edges;
- Nearest Neighbor Graph (NNG): $(u, v) \in E$ if and only if v is the nearest neighbor to u.

The hierarchy relation between DG and its subgraphs is shown in Equation 2.4 [51].

$$NNG \subseteq MST \subseteq RNG \subseteq GG \subseteq DG \tag{2.4}$$

A particular case of a DG subgraph to solve similarity searches is the Spatial Approximation Tree (SAT), proposed by Navarro[42]. SAT is a particular case of a DG subgraph in which the resulting structure is a tree with a fixed root to start the search, which returns exact results. The Spatial Approximation Tree (SAT) is constructed by consecutively inserting nodes according to spatial approximation. First, an element is randomly selected as the root and the rest of the elements are sorted according to their distance to this root element. Then an element a is added as a child node to the root if a is closer to the root element than to the any already added child node. This process is done recursively to each child node of the root element. SAT was later extended to the dynamic SAT (DSAT) and to the DiSAT (Distal SAT) [52]. The SAT has proven to execute significantly fewer distance computations during its construction when compared to the M-tree in both static and dynamic versions [53, 54].

Despite the SAT being a method that has fast construction compared to brute force graph-based methods construction (see Chapter 4) Ocsa, Bedregal e Cuadros-Vargas[13] stated that the selection of a bad root in SAT can lead to excessive node exploration during query execution. Ocsa, Bedregal e Cuadros-Vargas[13] also showed that a RNG can reduce significantly the number of distance computations in queries when compared to SAT.

2.2.2 Order-k Proximity Graphs

Besides proximity graphs, there is also **Higher Order Proximity Graphs**, also called **Order-k Proximity Graphs**. The Order-k Proximity Graphs generalize the proximity property P of their corresponding order-0 graphs ("classic" proximity graph definition) to an Order-k property P. As a result, not only the graph changes but also the proximity property P.

For example, the proximity property P for the k-RNG graph is $(v, u) \in E$ if and only if the intersection between $B_u(u, \delta(u, v))$ and $B_v(v, \delta(v, u))$ contains at most k points. In the k-DG graph $(v, u) \in E$ if a ball through v and u contains at most k points [55, 56]. Another example is the k-NNG, k-Nearest Neighbor Graph, which each vertex has an undirected edge to its k-nearest neighbors, not only to the nearest neighbor as occurs in the NNG. The k-NNG with directed edges to its k-nearest neighbors is denoted as k-NNG_u. Alike the order-0, the Order-k Proximity Graphs also follows a hierarchy, as

shown in [57]:

$$k\text{-}NNG_u \subseteq k\text{-}NNG \subseteq k\text{-}RNG \subseteq k\text{-}GG \subseteq k\text{-}DG \tag{2.5}$$

Subgraphs of DG and the order-k graphs have common edges with the DG for the same vertex set. However, the lacking edges and the edges that exist in other proximity graphs but do not exist in the DG can lead a search algorithm based on spatial approximation to a local approximate answer. For example, the k-NNG has common edges with the DG, however, every k-NNG vertex is limited to having at most k edges incident from it. Since the number of edges for each vertex in the DG is not limited by a parameter, a vertex can have more or fewer edges than k.

2.2.3 Limitations of Delaunay Graph

For the reason that the Delaunay Graph is dual to the Voronoi diagram, by definition, the DG fulfills the spatial approximation property described in Equation 2.2 [42]. However, the DG has some limitations which prevent it to be used as a structure for complex data storage and retrieval.

First of all, the Delaunay Graph is only defined for Euclidean spaces so is not possible to compute the Delaunay graph for a general metric space given only the set of distances of the dataset elements [13, 42]. To retrieve complex data using similarity searches it is necessary a structure that can be used with any (dis)similarity function.

Despite there are some works that extended the Delaunay Graph for high dimensional data [47, 48, 49, 50], a few authors, such as Harwood e Drummond[40], state that the Delaunay Graph can become a complete graph very quickly as the dimension of the data increases. To illustrate this statement, we did an experiment using a synthetic dataset. The synthetic dataset used is based on Gaussian distribution and contained 150 elements of 100 dimensions. For each variation of the dataset, from 2 to 100 dimensions, the Delaunay Graph (DG), the Gabriel Graph (GG) and the Relative Neighborhood Graph (RNG) were built, and the average number of neighbors per vertex was measured. The graphs that were constructed and measured in this experiment do not have duplicated edges, meaning that if the average number of edges reaches the maximum number of edges of a vertex ($|V| - 1$), the resulting graph is a complete graph.

The result of the experiment is shown in Figure 6. As shown in the figure, the average number of neighbors per vertex of the Delaunay Graph quickly becomes 150 as the dataset dimension increases, meaning that it has turned into a complete graph. It is also noticeable that the Gabriel Graph also quickly becomes a complete graph, however, the RNG does not have such a behavior. Moreover, this pattern is independent of the dataset size. To point out that, the same experiment was done for datasets with larger

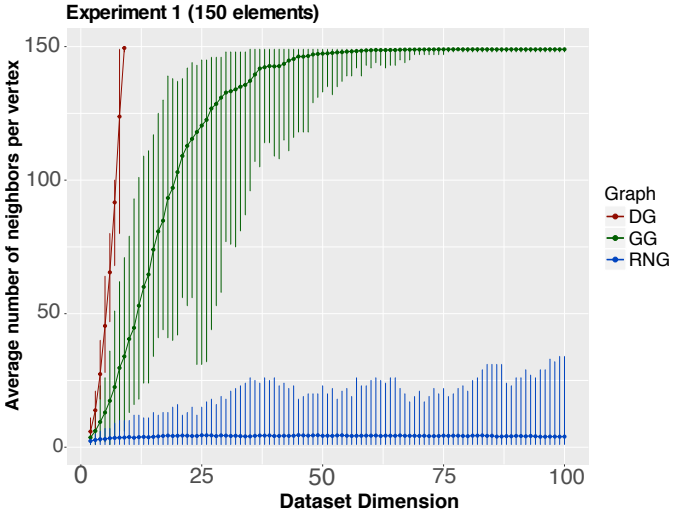

Figure 6 – Average number of neighbors per vertex by the dimension of the dataset for a 150 elements synthetic dataset.

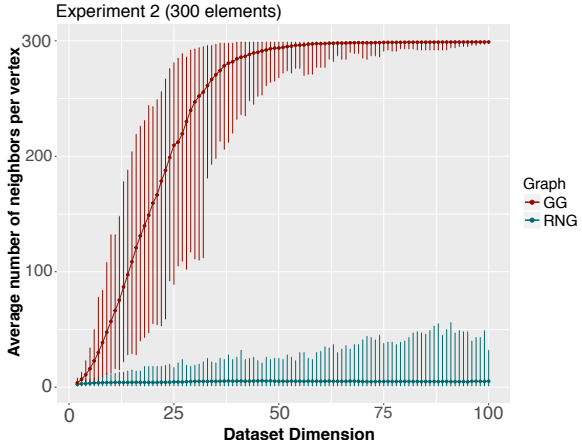

Figure 7 – Average number of neighbors per vertex by the dimension of the dataset for a 300 elements synthetic dataset.

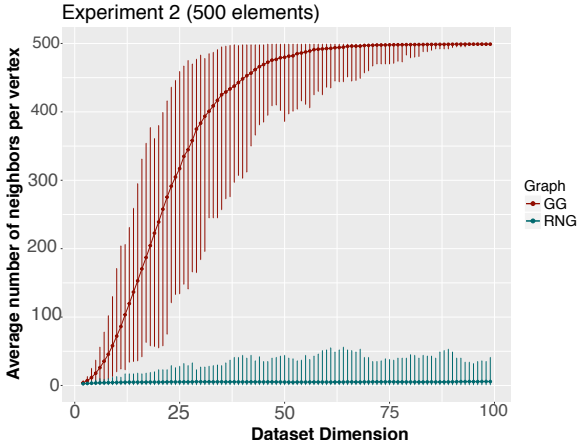

Figure 8 – Average number of neighbors per vertex by the dimension of the dataset for a 500 elements synthetic dataset.

numbers of elements: one dataset with 300 elements and another with 500 elements. Both of these datasets were generated by a Gaussian distribution the same way as the one used for the previous experiment. The results of the second experiment using the dataset of 300 elements are shown in Figure 7 and using the dataset of 500 elements are shown in Figure 8 (the Delaunay Graph was omitted). In both of the datasets, the Gabriel Graph turns into a complete graph while the Relative Neighborhood Graph maintains a low average of neighbors per vertex with a slight increasing pattern on the maximum value.

By turning into a complete graph, there are no advantages of using this kind of structure to represent the similarity space since it is necessary to calculate the (dis)similarity to all the nodes (data) when inserting a new element to the graph or editing a vertex of the graph. When comparing the execution of a $kNNq$ or a Rq on a complete graph to a

sequential search in the database, both require to compare all the elements in the database to the query element when the query element q is not part of the graph vertex set V.

Hence, to be more compelling in answering similarity queries using graphs, previous works focused on how to use graphs with a **minimized** number of edges. This study area includes different types of graph structures to model the similarity space, and search algorithms over graphs to answer similarity queries or other operations that involve complex data. The next chapter reviews proposed graph-based methods for similarity search in the literature.

3 GRAPH-BASED METHODS AND SEARCH ALGORITHMS

For graphs to answer similarity searches it is necessary to use a graph that has a *minimized* number of edges and can be constructed according to any similarity function δ. However, graphs that fulfill the spatial approximation property (e.g., complete graph and *DG*) have critical limitations, as previously discussed. Nevertheless, a large amount of research has focused on using graphs with a minimized number of edges for **approximate similarity search** [16].

In Section 3.1 we describe the search algorithms proposed in the literature for exact and approximate search and in the following (Section 3.2) we describe several types of graphs used for similarity search.

3.1 Exact and Approximate Search in Proximity Graphs

There are several similarity search algorithms for exact and approximate results. As previously discussed, for a search algorithm based on spatial approximation to return exact results the graph structure needs to have some particular properties. Since it is impractical to use *DG* in general high dimensional spaces, search algorithms for exact search employ different techniques to always return the correct answer in graph-based methods.

For **exact similarity search**, which means a search that produces exact results, in proximity graphs, Paredes and Chavez [9] proposed algorithms for *k-NN* and *Range* queries using weighted directed *k-NN* graphs. Their main contribution is to use weighted *k-NNG* to estimate lower and upper bounds between query and dataset elements. These bounds are used to discard elements (vertices) in the dataset that are not in the result set according to properties of metric spaces. Since this algorithm uses metric space properties to prune vertices that cannot be in the result set, such as triangle inequality and symmetry, this algorithm may not give exact result answers in general non-metric spaces.

The upper bound distance of two vertices u and v of the *k-NNG* is the weight/cost of the shortest path between u and v, $d_G(u, v) \geq \delta(u, v)$. The lower bound is calculated as $max_p = \delta(p, q) - d_G(p, u)$, where p is any previously selected vertex by the algorithm. $d_G(p, u)$ is the weight of shortest path between vertex p and u according to Dijkstra shortest path algorithm. A generic graph-based approach to solve *Range* queries is to consider a set of candidate elements and then, iteratively, extract each element u from this set and if $d(u, q) \leq r$, being q the query element and r the range, then u is reported as part of the result. Else, all elements v are deleted from the candidate set if $d_G(u, v) \leq \delta(u, q) - r$.

The authors improve this generic algorithm by using *k-NNG* properties:

1. Use the covering radius of each vertex (the distance from the vertex to its k^{th} nearest neighbor). If the query element q and the range r is contained in the covering radius of a vertex u, the candidate elements are $u \cup NN_k(u)$;

2. Propagate to the neighborhood $NN_k(u)$ of every candidate u, employing upper and lower bounds to discard elements that are not in the query result without having to calculate their distance to the query element. If a vertex u is part of the result set of the query, then it is likely that $NN_k(u)$ is also part of the result.

3. Working evenly in all graph regions to avoid concentrating efforts in only one graph region and to avoid computing a path in the graph several times.

The properties used in 1 and 2 are calculated according to the distance between elements. To work evenly in different graph regions (property 3) the authors proposed two heuristics to select the next vertex to be evaluated as part of the query result set. The first heuristic proposed selects the vertices that have the biggest number of already discarded neighbors from the result set because these vertices will probably be discarded as well, reducing the number of candidate elements. The second heuristic selects the vertices that have the biggest graph distance from the previously selected vertex. This second heuristic is based on on the assumption that if two vertices are distant in the graph they are also distant in the similarity space.

To compute a k-Nearest Neighbors query the authors use a similar algorithm to the Range Query but using the well-known approach of maintaining a dynamic decreasing radius initially defined as infinity (∞). Another modification to compute *k-NN* queries is that the algorithm maintains an auxiliary result set containing the k-Nearest Neighbors vertices evaluated to the moment. The radius of the search is the distance from the query element to the k^{th} nearest neighbor of this auxiliary set. Vertices with a lower bound bigger than the query range are eliminated as candidates to the result set. The lower bound is calculated as $max_p = \delta(p, q) - d_G(p, u)$, in which p is any previously selected vertex by the algorithm, $d_G(p, u)$ is the weight of shortest path between vertex p and u according to Dijkstra shortest path algorithm.

Because of the high cost to produce an exact answer to similarity searches, a large amount of efforts has concentrated in **approximate similarity searches, which are searches with approximate results**. According to Patella e Ciaccia[58] the **objective of approximate searches** is to reduce the search time compared to the exact search with the minimum error possible. The user is commonly offered a quality/time trade-off in similarity searches. In these cases, in order to speed up the search results, the algorithms are allowed to degrade in the query answer quality, eventually returning elements that are

not part of the correct/exact result. This trade-off happens in graph-based methods as it will be discussed in Chapter 4. **In this work**, we will be referring as approximate search results the search results given when executing approximate search algorithms proposed in the literature.

A strategy used to search over proximity graphs is to select initial vertices and use a best-first search process based on the spatial approximation which produces approximate results for most proximity graphs. [59]. The Greedy Search algorithm (*GS*) [60] shares the same basic idea of other greedy implementations found in previous works. *GS* starts by randomly choosing a vertex of the graph as the start vertex v_i; then, it computes the similarity between the query q and each neighbor of v_i; the neighbor vertex with the highest similarity to q is then selected to be v_i. This process continues until there is no neighbor with a higher similarity value to q than v_i.

The *GNNS* algorithm is described as a "best-first search method to answer a *k-NN* query" [17]. The *GNNS* search algorithm can be found in the algorithm 1. The δ is the distance function used and $NN_F(y_{t-1})$ is a function that returns the $F \leq k$ nearest neighbors to the y_{t-1} in the graph, in which k is the parameter k in the graph. The set U is the set that contains the calculated distances δ along with the algorithm execution.

The algorithm starts with a randomly initial vertex $u \in V$ and (line 5 in algorithm 1), in each iteration, u swapped with $v \in V$, which the neighbor of u that is most similar to q. This iterative process ends after a maximum number of swaps T. The number of swaps limits the search in case the user wants to, however, if $T = |V|$ in which V is the set of vertices in the graph, the search will calculate the best result possible. As stated before, the greedy search will return a local optimal answer depending on the graph structure. A strategy to enforce the algorithm to explore other vertices is to start the search from another initial vertex.

Thus, the above greedy search process is initialized R times with different starting vertex. The final result is the set of k most similar vertices to q evaluated in all random restarts R. It was shown that *GNNS* can outperform the **LSH** [12] and the **KD-tree** [61] when comparing speedup over linear search and number of distance computations for high dimensional real and synthetic datasets. This result supports the claim that a graph-based approximate search can be an interesting approach to perform similarity searches.

A major drawback of search algorithms based on spatial approximation is when the starting node is "far away" from the ideal result, thus leading to very long running time that depends on the graph size. To minimize the execution time and, at the same time, increase the probability of finding the expected answer, *seed nodes* can be used as starting vertices. These seed nodes can be selected by random sampling, using a second index structure [62] or selecting well-separated elements in a dataset [59].

Algorithm 1 GNNS Algorithm.

```
1: function GNNS(q, K, R, T, F)
2:     S ← {}
3:     U ← {}
4:     for r ← 1, ..., R do
5:         y_0 ← random vertex v ∈ V from graph
6:         for t ← 1, ..., T do
7:             y_t ← argmin_{y∈NN_F(y_{t-1})} δ(y, q)
8:             S ← S ∪ NN_F(y_{t-1})
9:             U ← U ∪ δ(y, q)|y ∈ NN_F(y_{t-1})
10:    Sort U according to δ function
11:    Pick the closest K elements from U
12:    Return the corresponding elements in S
```

The majority of other existing algorithms for approximate search use variations of the basic spatial approximation and/or greedy search algorithm described above. For example, the author in in [39] proposes the Parallel Greedy Search (PGS) algorithm to increase the probability to find the exact nearest neighbor to the query element. This algorithm performs parallel processes of the greedy search, each one starting from a different initial vertex. In their subsequent paper, the authors proposed a breadth-first search (BFS) algorithm to improve the results of the *k-NN* query on a *k-DRG* [63]. In this new proposed algorithm, the first step is to execute the *GS* algorithm with multiple starting vertices. The set of resulting vertices of each greedy search with a different start is used as the set of initial vertices for the breadth-first search (BFS). This initial vertex set is called by the authors as *attractors*. Other examples on spatial approximation based algorithms include the proposed search algorithms for *SWG* [16](subsection 3.2.4), *RNG* [13](subsection 3.2.3) and FANNG [40](subsection 3.2.4).

3.2 Types of Graphs used for Similarity Search

In this section, we introduce the main types of graphs used for similarity search.

The next subsections are organized by the type of graph proposed for similarity search, the employed strategy for graph construction, and further advances.

For better visualization of the different structures of graphs, a sample of 32 instances of the USCities dataset will be used. The USCities dataset contains information of the 2001 census of the United States of America. It has approximately 100 numeric attributes that represent demographic data and geographic coordinates of the cities that took part in the census. From these 100 numeric attributes of the USCities dataset, only the geographic coordinates(latitude and longitude) of the cities will be used, to easily visualize the graph's structure. The used instances can be visualized in Figure 9 and the Delaunay Graph for instances number 1 to 30 can be visualized in Figure 10. The two remaining

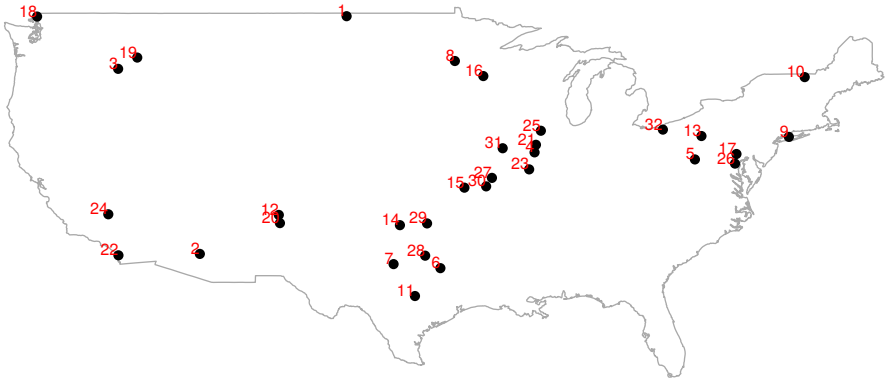

Figure 9 – 32 samples of the USCities database plotted according to their geographic coordinates.

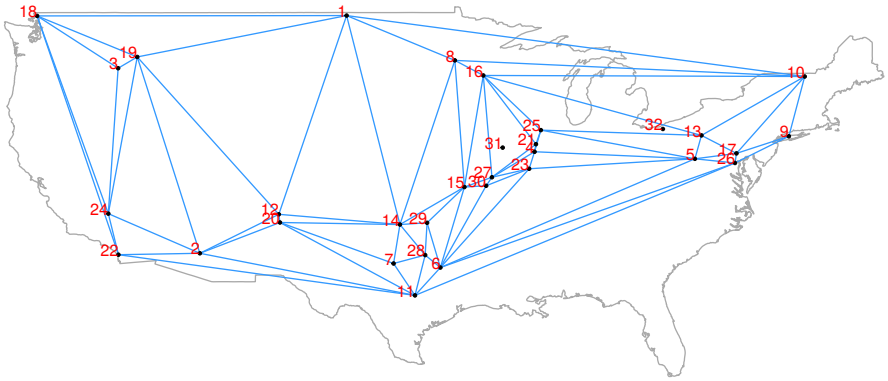

Figure 10 – Delaunay Graph of the USCities database samples.

samples (number 31 and 32) will be used as query data for the discussion regarding the search process in the different graph types.

3.2.1 *k-NN* Graphs

The k-Nearest Neighbor Graph (*k-NNG*) [59] is defined as a graph $G = (V, E)$, where $E = \{(u, v), v \in NN_k(u)_\delta\}$ such that $NN_k(u)_\delta$ is the set containing the k nearest neighbors of u in the set of vertices V regarding the similarity function δ. The edges of *k-NNG* can be undirected or directed, in this work we denote *k-NNG*$_u$ as undirected *k-NNG*. Figure 11 shows the directed 2-NNG built from the USCities sample. The *k-NNG* can be weighted in which the weight is usually the distance between the connecting pair of vertices ($\delta(u, v)$).

The trivial brute force algorithm for constructing *k-NNG* (denoted *k-NNG-BF*) has cost $O(n^2)$ and, consequently, it is not feasible to be used for large datasets [19]. Thus, some research works focused on developing better algorithms for *k-NNG* construction.

Paredes et al. [64] proposed two algorithms for *k-NNG* construction, the Recursive Partition Based Algorithm and the Pivot Based Algorithm. Although these two methods

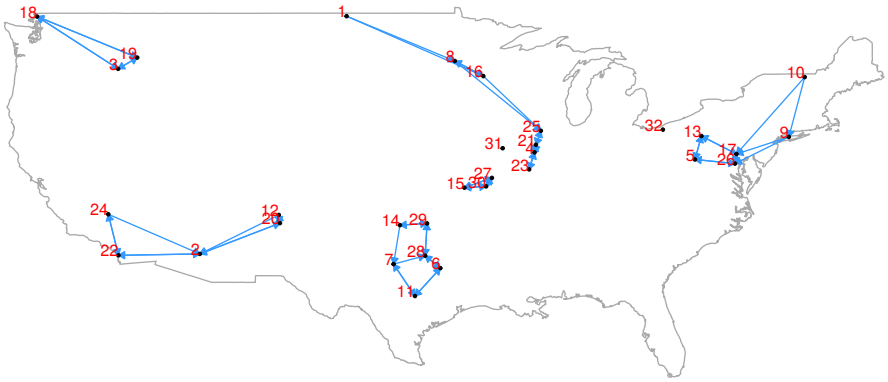

Figure 11 – 2-NN directed graph of the 32 USCities sample.

presented good results in the experiments when compared to the *k-NNG-BF* algorithm, both methods require a secondary global structure to support the *k-NNG*.

The Recursive Partition Based Algorithm requires the construction of the structure Division Control Tree (DCT). The DCT is a binary tree that represents the partitions that are used The intent to build the DCT is to make the computations of the $NN_k(v)$ faster.

The DCT is constructed by partitioning recursively the dataset. Considering the dataset S, two elements $r, l \in S$ are selected to be the children nodes. The rest of the dataset $S - r, l$ is partitioned in two subsets: the S_r set that contains elements that are closest to r; and the S_l set that contains elements that are closest to l. And this process is done recursively to the subsets S_l, S_r and to each children node selected and its set of closest elements. The recursive procedure to build the DCT stops when $\mid S \mid < 2$. The recursive procedure ensures that every node has already computed distances to their "ancestor" nodes.

The Pivot Based Algorithm uses a pivot index. A pivot index selects a set of elements from the original dataset P, the pivots, and then stores a table of the distances of each one of the pivots to each one of the elements in the dataset $(\delta(p, v), p \in P$ and $v \in V)$. The pivot index facilitates the $NN_k(v)$ searches computation. The main goal of building these secondary structures is to make the $NN_k(v)$ computation faster.

A second proposed *k-NNG* construction algorithm is the *NN-Descent* algorithm [19]. In this approach, an approximate *k-NNG* is built with arbitrary similarity measures, without the need of a secondary global structure, and with a lower computational cost of $O(n^{1.14})$. The basic idea of *NN-Descent* algorithm is "the neighbor of a neighbor" is also probably a "neighbor". It starts by first computing a random approximation of *k-NN* for each element V, and then iteratively improve the first approximation comparing each element to its neighbor's neighbors, including its *k-NN* and reverse *k-NN*. This algorithm stops when no improvement in the result set is made [19].

The NN-Descent algorithm uses local join, sampling, and early termination to optimize the comparison between each element to its neighbor's neighbors. Local joins are used to compute the similarity between each pair of different neighbors a vertex v has. Since these local joins can compute the similarity between the same pair of vertices every iteration and in each iteration fewer and fewer new vertices and edges are inserted in the *k-NNG*. The authors use a boolean flag in the implementation that makes possible for the algorithm to check if a vertex is new or not. The similarity is computed only when it is new. If the parameter k is large the cost of a local join can be very high. In order to minimize it, the local join is operated only on a sample of neighbors. The sample is done based on the sample rate defined by parameter ρ on neighbor vertices that were marked with the boolean flag as true (new vertices). After the local join, the boolean flag of these vertices is marked as false. In each iteration, the algorithm counts the number of updates in the *k-NNG* and the algorithm terminates when it becomes less than ηNK, in which η is a parameter, K is the parameter k of the built *k-NNG* and N the number of updated vertices. In the last few algorithm iterations, very little modifications in the approximate *k-NNG* are done, so the authors used an early termination condition to avoid a bigger computational cost for just little modifications.

Besides the above methods, some authors focused on divide and conquer algorithms to build a *k-NNG* [65, 66]. Chen et. al [65] proposed two algorithms to build a *k-NNG*, the overlap method and the glue method. Both of these algorithms use the Lanczos procedure [67] to execute the dataset division based on a spectral bisection [65, 68]. Each recursion divides the elements into two subsets. The Lanczos procedure is used to find the most valuable eigenvectors and eigenvalue of a matrix. The spectral bisection is done by using the eigenvalues signs to partition the graph. The overlap method divides into two overlapping subsets while the glue method divides into two disjoint subsets and a third subset is used to merge them. In both methods, when the size of the subset is smaller than a parameter defined threshold a *k-NNG* of this subset is built using the brute force construction algorithm. Next, the conquer procedure is called to merge the graphs. The final step of the proposed methods is to refine each element *k-NN* by comparing its current neighbors and its neighbor neighbors and update the edges in case there is a vertex that should be in this element neighborhood but is not.

Wang et. al [66] proposed a multiple random divide and conquer algorithm to build an approximated neighborhood graph. The main idea of this approach is to randomly and hierarchically partition the dataset until the size of a subset is smaller than a parameter defined threshold so that similar elements have a high probability to be in the same subset. This random partition process is repeated in order to increase the chance that close elements are connected in at least one partition/subset. Because the dataset is partitioned many times into many subsets some of the partitions in the process works as an overlap to make connections among subgraphs. In the authors implementation of the algorithm

random directions are chosen to perform random divisions to partition the dataset. The principal directions are obtained by using Principal Component Analysis (PCA) of a sample of elements from each subset. In the experimental results, the introduced multiple random divisions algorithm outperformed the methods proposed by Chen et. al [65] in construction time when achieving the same or even better accuracy (the accuracy was measured comparing the constructed graph to the exact *k-NN* graph).

The remainder of other approaches either build exact or approximated *k-NNG* with lower complexity than *k-NNG-BF*, where some methods work specifically in metric spaces and others for general similarity functions [69, 70, 71].

3.2.2 *k*-NNG-based Graphs

Other methods can be classified by the type of graph employed for similarity search, either built from a *k-NNG* or with the same properties as *k-NNG*.

A related graph to the *k-NNG* is the quasi proximity graph. The quasi proximity graph proposed by Chavez et. al. [72] is a proximity graph induced by the $k\text{-}NNG_u$ (directed *k-NNG*). The neighborhood of a vertex in the $k\text{-}NNG_u$ is not symmetrical, that means, a vertex v can be part of the $NNk(u)$ of the vertex u but u is not necessarily in $NNk(v)$. This property can lead a greedy search result to a local minimum for a 1-NN search. The quasi proximity graph is a $G(V, E)$ where $(u, v) \in E \leftrightarrow |NN_k(u) \cap NN_k(v)| \geq k$, so if the vertices u and v have k neighbors in common, u and v are also neighbors. According to the results, the quasi proximity graph finds the solution for the 1-NN search with a high recall rate especially for higher values of k.

Another variation of the *k-NNG* is the k-Degree Reduced Graph (*k-DRG*)) [60]. As the name suggests, the main idea of the *k-DRG* is to build a graph from the *k-NNG* with a *reduced number* of edges that guarantees the k nearest neighbors of a vertex can be reached using the *GS* algorithm.

It was proposed two construction algorithms for the *k-DRG*, the first one was proposed in [60] is used when the k parameter for the *k-DRG* is defined and the second was proposed in [39] for when the parameter k is not defined.

The construction used when the parameter k is provided in advance is done by incrementing k' until $k' = k$ in which k' is a counter to control the number neighbors in each vertex starting from 1 until k. In each iteration an edge will connect vertices $v \in V$ with its k' nearest neighbor $u \in V$ only if is not possible to reach u from v using the already inserted edges in the *k-DRG* and a greedy search algorithm. This will reduce the number of edges in the *k-DRG* compared to the original *k-NNG* but maintains the navigability of the *k-NNG* [60].

When the parameter k is not provided in advance a search success probability error

will serve as the stop condition of the algorithm [39]. This search success probability is calculated based on the graph basins. The basin of a query element q in a graph is the set of vertices on the graph from which a greedy search algorithm can find the closest vertex to q. The search success probability is given by the ratio of the size of the basin to all of the vertices $|V|$. The calculation of the exact search success probability in each iteration for each query element q starting the search from each vertex in the graph demands a high computational cost. As a result, the authors used the Monte Carlo method to make an approximation of this success probability by sampling the initial vertices for the searches and the query elements, called quasi queries, in the construction algorithm.

Additional variations include the Hierarchical *k-DRG* [73] and the Double Layer Neighborhood Graph (DLG) [74], which were proposed by the authors of *k-DRG* in subsequent papers. Both of these hierarchical graphs uses as the base graph the *k-DRG*. In DLG, the search space is reduced by using representative vertices of the base layer as starting vertices (seeds) for the search algorithm. The representative vertices are selected from the base layer based on their betweenness centrality, the number of shortest paths between all pairs of vertices in the graph that the evaluated node appears.

Finally, another *k-NNG* based graph is the Pruned Bi-Directed k-Nearest Neighbor Graph (*PBKNNG*) [75]. The *PBKNNG* adds reversely directed edges to all the directed edges of the *k-NNG* since the search performance of the k-NNG improves as the k parameter increases. Some of the added edges are pruned according to heuristics because as the k increases more memory is necessary to store the edges.

3.2.3 Relative Neighborhood Graph – RNG

The Relative Neighborhood Graph (*RNG*) is a subgraph of *DG* with guarantees that for any pair of vertices there exist one or more paths that connect the pair of vertices (i.e., *RNG* is a connected graph [13, 51]). Formally, the *RNG* is a graph $G = (V, E)$ whose set of edges E is determined by a proximity property P that $(u, v) \in E$ if and only if $B_u(u, \delta(u, v)) \cap B_v(v, \delta(v, u)) = \emptyset$, where $u, v \in V$ and $B(x, r)$ is a ball defined by x and distance threshold (radius) r [13]. Figure 13 show the proximity property P of the Relative Neighborhood Graph. A very important property of *RNG* is that it is parameter-free, i.e., the number of edges in the graph is defined by the dataset properties and, therefore, it is "auto-adjusted". Figure 12 shows the *RNG* built from the sample from the Figure 9.

Distinctly from *DG*, *RNG* can be built given only the distances between the dataset elements. However, the complexity to build *RNG* in general spaces is $O(n^3)$ [76], which is even higher than the complexity of the brute force algorithm to build the *k-NNG*. There exist methods to construct *RNG* with lower complexity, such as $O(n \cdot \log n)$ [77] and $O(n)$ [78]. However, the main drawback of these methods is they require to have a Delaunay triangulation in order to then build *RNG* which makes the total cost very high.

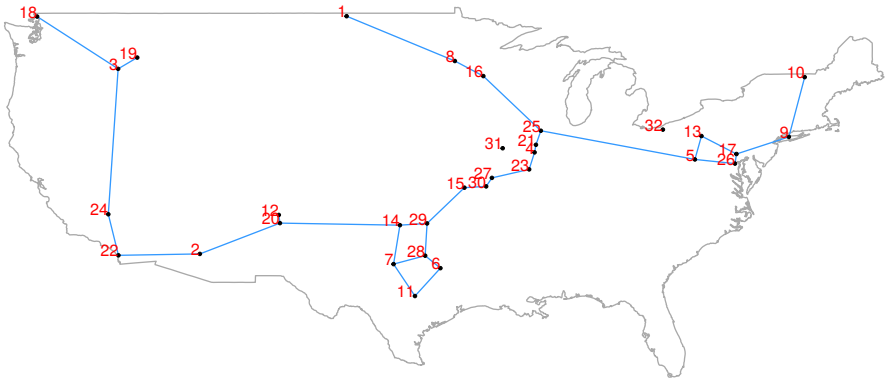

Figure 12 – *RNG* of the 32 USCities sample.

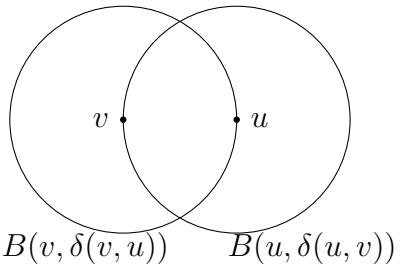

Figure 13 – Proximity property of the Relative Neighborhood Graph (RNG). Since there are no vertices in $B(v, \delta(v, u)) \cap B(u, \delta(u, v))$, the pair of vertices $(v, u) \in E$.

A similar algorithm to the *GS* [63] could be used to search over the *RNG*. The algorithm starts by randomly selecting a vertex (node) of the graph to start the search and the neighborhood (adjacent vertices) of this start vertex is visited until the nearest neighbor to the query element is found. If the selected starting node is too far from the result the searching process can have very long execution time depending on the size of the graph. To minimize this problem the author in [13] proposed the use of the Relative Neighborhood Density Factor (RNDF) for *seed* selection. The RNDF is a value based on the neighborhood of each vertex of the graph. Considering a vertex $v \in V$, the RNDF value of this vertex is 1 if v is the nearest neighbor to all of its adjacent neighbors, if this is not true the RNDF value for v will be less than 1. The selected vertices will be the ones with the highest RNDF and have more than 1 neighbor (adjacent vertex). So to make the nearest neighbor query in the *RNG* more efficient the closest seed vertex to the query element is selected as a starting node for the search. Rq and $kNNq$ can be computed over the *RNG* by exploring iteratively the neighborhood of the seed vertex chosen as the nearest neighbor. As mentioned before the *RNG* reduced significantly the number of distance computations during the search when compared to the *SAT*. Thus, the *RNG* also performed better when compared to the Slim-tree [10] and DBM-Tree [11] for high dimensional metric spaces.

Variations of *RNG* include hierarchical graphs, Hyperspherical Region Graph (HRG) [41]

and its improved version MOBHRG [79], which aim at improving the search through an initial search in the upper graph level.

The HRG is a graph $G_H = (V, E)$ where $H(v, u)$ for any pair of vertices $v, u \in V$ is a hypersphere in which the radio is the distance between v and u is $\delta(v, u)$ and the center of the hypersphere is the vertex v. The HRG consists of only vertices that are centers of hyperspheres. A hypersphere is defined by the authors as $H = (v, c, O)$, v is the center of the hypersphere, c is the maximum of elements that a hypersphere can have and O is the set of elements that are included in the hypersphere. The HRG is a type of a *RNG* so the proximity property is very similar to the *RNG* but considering the hyperspheres. The proximity property that must be fulfilled to connect two vertices $v, u \in V$ is $(H(v, u) \sqcap H(u, v)) \cap V = \emptyset$.

The MOBHRG makes improvements on the HRG graph construction and updates algorithm by reducing the overlap region between hyperspheres. The improvement consists of using the minimum spanning tree (MST) algorithm to choose the center of a new split hypersphere. A hypersphere is split when the number of elements in it exceeds c, the maximum number of elements that a hypersphere can have. By using the MST algorithm the resulting graph will have more separated regions minimizing the number of vertices evaluated in the search when comparing to the HRG, contributing to search algorithms.

Experimental results in the paper showed that the MOBHRG always takes less time to build than the M-tree, and when comparing query response time the MOBHRG performs better than the M-tree [1] and the *SAT* for synthetic and real datasets.

3.2.4 Other Graphs for Approximate Similarity Search

Recently, new types of graphs have been proposed for approximate similarity search. These proposed graphs achieved high recall rates and performed reasonably well in high dimensional metric spaces, as reported in [16, 40].

One representative of this type of graphs is the Navigable Small World graph (*NSW*) [16, 80]. A small world graph (*SWG*) is a random graph, very sparse, with high local clustering property, and with a short distance value among vertices in the graph. In other words, a pair of vertices is connected by a path that is considered *small* compared to the size of the graph. The proposed *NSW* graph is based on an approximation of *DG* and has a *SWG* topology, due to its long-range links to keep the navigation property. The authors use a basic greedy search algorithm for nearest neighbor search and also presents a $kNNq$ algorithm with approximate results.

The *NSW* graph can be defined as $G = (V, E)$ in which the vertices (nodes) represents the complex data and the edges links between them. Vertices of the *NSW* are connected by two types of edges: *short-range links*, used for the greedy search, and

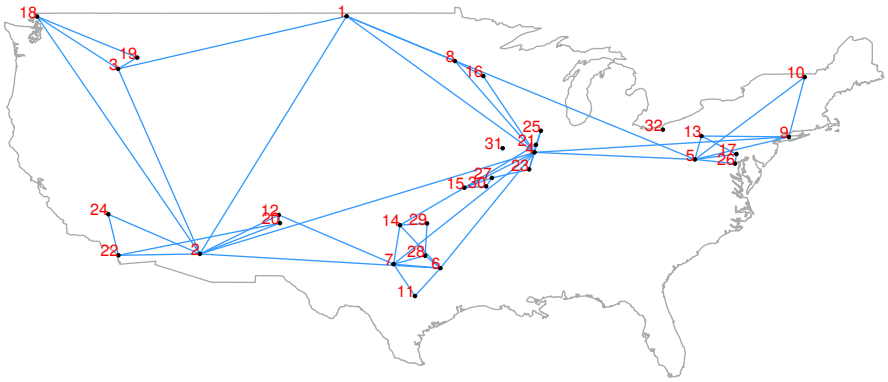

Figure 14 – Navigable Small World Graph ($k = 2$).

long-range links, which define the *small world properties* in the graph. The construction algorithm of the *NSW* is based on inserting iteratively the complex data (vertices). For each newly inserted vertex, a search **over the graph** is done to identify and connect its k nearest neighbors to the newly inserted vertex. This algorithm idea is to build in the *NSW* an approximation to the Delaunay graph. The newly created edges are short-range links because it connects the vertex to their current k nearest neighbors. In each construction algorithm iteration, the edges that were previously short-range become long-range links. Figure 14 shows an *NSW* with $k = 2$. The number of long-range links is not controllable by any parameter and the excessive number of edges per vertex in *NSW* can slow down the query execution using a best-first approach. This will be further discussed in Chapter 4.

In [16, 80], it is proposed a modified version of *k-NN* queries for *NSW*, called *multiple start greedy search*. The two main contributions of this algorithm are: (1) a shared list of size *efSearch*, between m searches of visited vertices, is maintained to avoid distance computation; and (2) a new stop condition is defined to maintain the algorithm exploring the neighborhood of result, aiming to improve the result set (*k-NN*).

The Fast Approximate Nearest Neighbour graph [40] (FANNG), which is built with particular properties which search intent is to find the closest vertex as possible as to the exact result. The FANNG uses the occlusion rule to connect vertices, this rule is similar to *RNG* neighborhood criterion. The occlusion rule states that if there is an edge connecting the vertices v_1 and v_2, it's not necessary to have an edge connecting the vertex v_1 to any other vertex v_3 that is closer to v_2 than to v_1. However, the proposed structure only gives exact results to a greedy search if the query element is identical to a vertex inserted in the graph. As a result, the property defined was modified in order to deal with situations in which the main interest is to search within a distance r from the query element (*Range* queries). The property considering the distance r from the query element q, if $\delta(q, v_2) < \delta(q, v_1)$ it is not necessary to have an edge connecting from v_1 to v_3 where $\delta(q, v_3) < r$.

As the FANNG is modified, the search algorithm is also modified. Instead of stopping the nearest neighbor search when no progress can be made, the search uses a version of depth-first search (DFS) to backtrack to the second closest vertex and considers any edges that have not been explored yet [40]. And recursively, if all the neighbors to the second closest vertices have already been explored, the algorithm will backtrack to the third closest vertex and so on.

Other important graph-based methods in the literature are: the Pivot Neighborhood graph [81], which is a graph-based method for approximate search where vertices are used as pivots of the dataset; the Randomized Neighborhood graph [82], in which the neighborhood of a vertex is based on *cones* of the space calculated by the algorithm described in [83].

Even though there are some works in the literature that addresses updates in proximity graphs [84, 85], to the best of our knowledge, the graph-based methods for similarity searches in the literature are mainly static methods. The paper by Hacid et. al [84] introduced algorithms for local insertion and local deletes of vertices for neighborhood graph adaptation for data indexing. Suppose we need to insert or delete a vertex q in a graph $G(V, E)$. The proposed algorithms in the paper defines a hyper-sphere $SR(q, r)$ in which the center is q and includes all vertices w in which $\delta(q, w) \leq r$. Instead of reconstructing the whole neighborhood graph G only the vertices and edges contained in the defined hyper-sphere SR are reconstructed after inserting or deleting q.

4 COMPARATIVE ANALYSIS OF GRAPH-BASED METHODS

As already mentioned, the main objectives of this thesis are: (1) to compare and provide a deep analysis of the tradeoffs between construction time and query time for the main graph-based methods; and (2) to develop a new graph-based approach for similarity search, based on divide-and-conquer construction with the addition of long-range edges, which can be used with several types of graphs previously proposed in the literature.

These two objectives are complementary since it is from the comparison results that we can analyze what aspects can be incorporated in a new method to improve the performance compared to the already proposed graph-based methods in the literature. The first objective is addressed in this chapter, and part of this contribution was published in [86]. The work regarding the second objective is described in Chapter 5.

Survey articles by Chávez et al. [3] and Hjaltason and Samet [4] describe search in metric spaces and cover indexing methods used for similarity searches. The work of Skopal and Bustos [7] provides a broad view of works addressing searches using non-metric similarity functions for different complex data domains. Nevertheless, none of these works include graph-based methods. Graphs are cited only to introduce theoretical foundations or to present applications on graph datasets. A recent survey by Naidan et al. [15] covered permutation-based methods for similarity search and presented an experimental analysis of some methods used for similarity searches. The tested methods included the graphs Small World Graph (*SWG*) [16] and the Approximate *k-NN* Graph built with the *NN-Descent* [19]. However, they neither explore the impact of different parameters nor address the exact search in graph-based methods.

In this chapter, we present a comparative analysis of the existing graph-based methods. This chapter is divided into two main sections: Section 4.1 presents how the search algorithms can be applied to the different types of graphs, especially in terms of the spatial approximation property, and; Section 4.2 presents an experimental performance analysis of the most representative types of graphs according to their main construction and search parameters.

4.1 Search Algorithms Applicability in Different Types of Graphs

As discussed before, the spatial approximation property does not give exact answers for the main graph types used for similarity search in the literature. This section presents a discussion on the graph types structure according to the spatial approximation property applicability in different types of graphs. The discussion is carried out by showing counter-

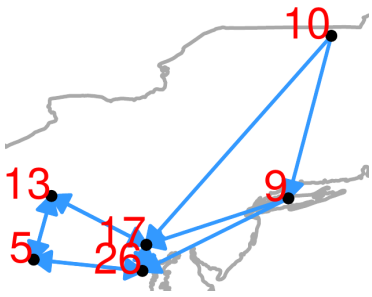

Figure 15 – Any algorithm based on neighborhood expansion would not be able to evaluate vertex 10 as these algorithms follows the edge direction and there are no incoming edges to number 10.

examples on how the spatial approximation property does not hold for the main graph types using the USCities sample dataset introduced in the previous chapter (section 3.2), and in which situations the graphs do not perform as expected. At the end of the section, we also discuss why Paredes and Chavez[9] proposed search algorithm implementation does not give exact answers in a weighted *RNG*, compared to *k-NNG*.

In the *k-NNG* and *k-NNG*-based graphs vertices are more connected or less connected according to the parameter k. Two main limitations of the *k-NNG* can be observed: (1) The edge direction can limit the search algorithm execution and; (2) The graph can be disconnected. Figure 15 shows the problem of the directed edges. When using a directed *k-NNG* the spatial approximation based search algorithm needs to follow the edge direction. Suppose the closest vertex to a query is vertex number 10, it is not possible to reach number 10 by a spatial approximation from any vertex since number 10 is not part of the 2-nearest neighbor of any vertex. In this case, the returned answer will not be correct.

However, the *k-NNG* does not hold the spatial approximation property even when using undirected edges. Since *k-NNG* is dependent on the parameter k, the resulting graph can generate disconnected regions and a simple search algorithm such as the greedy search will not be able to evaluate different graph regions. This evaluation will be possible only when using multiple restarts like *GNNS*. Still, even if *GNNS* or a similar search algorithm is used it is not guaranteed that it will always return the exact answers to the query as the starting elements are randomly selected.

An alternative for exact answers using *k-NNG* are the search algorithms proposed by Paredes and Chavez [9]. However, this algorithm has two drawbacks: (1)the properties used to prune the vertices in the search algorithm are based on metric spaces properties and it may not give the expected answer in non metric spaces; and (2)the algorithm execution takes longer than the best-first approaches as it will be shown in Section 4. *k-NNG* based graphs share the same drawback of the *k-NNG*.

The *RNG*, as defined before, is a connected graph. The fact that *RNG* is a con-

(a) Blue circle radius is $\delta(27, 31) = 2.067$ while red circle radius is $\delta(4, 31) = 2.107$, δ represents the L_2 distance. Nearest neighbor of 31 is 27.

(b) From vertex 4 the *GS* evaluates only vertices 23 and 21 that are farther from 31 than 4. 4 is wrongly returned as 31 nearest neighbor.

Figure 16 – Counter example on the spatial approximation property in *RNG*.

nected graph does eliminate the problem of the disconnected regions in *k-NNG*, but still cannot guarantee exact answers when using a Spatial Approximation-based search algorithm. Even though *RNG* is a Delaunay Graph subgraph, it does not fulfill the Spatial Approximation property. When a using greedy search algorithm it can return a local minimum as the answer. This happens when the query element is not part of the *RNG* and there is a vertex that is more similar to the query element than its neighbors, however, it is not the expected answer. Figure 16 shows this problem. Suppose the greedy search starting vertex is the number 21 and the query data is the start (number 31), by spatial approximation the next evaluated vertex will be number 4 since it is the most similar neighbor to the query. Thereafter, when evaluating the neighbors of number 4, both of them (number 23 and 21) are less similar than the current evaluated number 4, so 4 is returned as the closest neighbor to 31. But if when observing the close-up of the searched region in Figure 16(a), the actual nearest neighbor to 31 is number 27. The counter-example showed in Figure 16(b) can prove how the *RNG* does not hold the Spatial Approximation property. The same problem of stopping at a local minimum occurs in other proximity graphs, such as *k-NNG*, even when there is a path in the graph between the starting vertex and the element(s) that should be the query answer.

The Navigable Small World Graph [16] contains small world properties and an approximation to the Delaunay Graph. The long-range links of the navigable small world graph are used as "shortcuts" to get to the closest neighbors of the query element. For example, in Figure 14 using a simple greedy search algorithm starting from vertex number 1 and searching for the number 31 it is possible to hop from number 1 directly to number 21, without needing to evaluate the neighborhood of the vertices 8 and 16 that would be evaluated in *RNG* (Figure 12) and 2-NNG(Figure 11). The "shortcuts" can reduce the number of distance computations during the search. Still, since it is an approximation to the Delaunay Graph it does not guarantee an exact answer when using a Spatial Approximation algorithm approach, however, the answer quality is very good, with high

recall rates.

The main drawback of the Navigable Small World Graphs is the number of edges when compared to the *k-NNG* with the same k value of *NSW* or to the *RNG*. As a result of the long-range links in *NSW*, the number of edge per vertex is higher than a *k-NNG* or *RNG* and consequently, the *NSW* requires more memory to store the graph edges when compared to the other presented graphs (with the same k value).

The search algorithms proposed by [9] are based on metric spaces properties and was proposed for *k-NN* graphs, nevertheless, they can be used on other types of graphs with some modifications. One of the pruning methods used by the algorithm is when the query radius is contained in a vertex covering radius (distance between the vertex and its farthest neighbor), then the elements that are not part of the vertex covering radius can be pruned. According to metric spaces properties, the pruning method is correct and can be used in the search algorithm.

However, the **implementation** used of this pruning method is done by deleting from candidate vertices those that are not in the neighborhood of the currently evaluated vertex. This implementation works only on *k-NNG*. It is guaranteed that in the *k-NNG* all elements of the dataset, graph vertices, that have smaller distance than the k^{th} nearest neighbor of a vertex u is contained in the u neighborhood. But for other graphs, eg. *RNG*, the mentioned property is not guaranteed as *k-NNG* and *RNG* have different construction properties. In *RNG* it is possible to have a vertex u that is contained in the covering radius of a vertex v but is not in v's neighborhood. Figure 17 shows an example of how the proposed implementation cannot be used in a weighted *RNG*. In Figure 17 the query element is "inside" of the covering radius of vertex 25 (red circle) but the nearest neighbors are not connected by an edge to vertex 25. In the original algorithm, vertices that are not connected to 25 would be discarded as answer candidates and the returned query answer would not be exact. The implementation of the algorithm needs modifications to work on other types of graphs.

4.2 Experimental Analysis of the Main Graph-based Methods

This section presents an extensive experimental evaluation of graph-based methods for similarity search in metric spaces.

Our analysis comprehends the exact and approximate search using the most representative graphs mentioned in Chapter 3. The search methods include the exact algorithm of Paredes and Chavez and the greedy algorithms *GS* and *GNNS* (Section 3.1). The exact algorithm requires a weighted graph to be pre-computed, hence, we implemented the algorithm using weighted versions of *RNG* and *k-NNG*, denoted herein as *RNGW* and *k-NNGW*, respectively (both algorithms were constructed using the brute

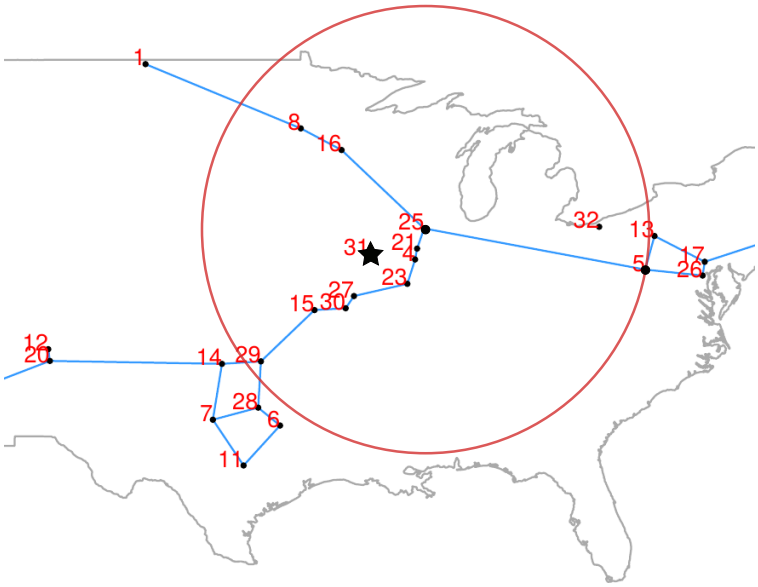

Figure 17 – The query 31 is contained in 25 covering radius. By executing Paredes and Chavez search algorithm, vertices that are not connected to 25 would be discarded as answer candidates, including 31 real nearest neighbors.

force approach). *GS* and *GNNS* algorithms were employed for approximate searching in graphs *RNG*, *NSW*, *k-NNG* using brute force construction denoted as *k-NNG-BF*, and *k-NNG* using *NN-Descent* construction referred to in this analysis as just *NN-Descent*. We also tested the *SAT* to serve as a baseline for the spatial approximation search. Table 1 shows a summary of evaluated methods, search methodology (exact or approximate), and construction algorithms. Such a wide combination of search methods and proximity graphs implemented in a common platform allows us to study, in details, the performance behavior of several factors of graph structure, search method and parameters, in order to understand the inherent tradeoffs among methods. The main goal is to provide a report with a baseline analysis that can be further used and extended in future research works. By analyzing in which cases each type of graph performs better, the results give hints on what approaches can be used in our proposed graph method to improve the search quality compared to the literature methods.

We use an assortment of real datasets that vary in dimensionality and cardinality to give a detailed experimental analysis in a uniform platform. To the best of our knowledge, our work is the first to cover analysis of exact and approximate search on different graph types for similarity search in metric spaces. The results provide a quantitative view of exact and approximate search as well as the tradeoff between graph construction cost and search performance according to different construction and search parameters.

The datasets employed in the analysis are: *USCities*, with geographic coordina-

tes of American cities[1]; *Color Moments*, co-occurrence *Texture* and *Color Histogram*, which contain the respective feature vectors extracted from photo images obtained from *Corel* [87]; *MNIST* [88], a collection of binary images of handwritten digits[2]; and *ANN-SIFT1M*, a million SIFT feature vectors used in [89]. For *ANN-SIFT1M* we used only the database set, the learning and query set was not employed. Table 2 shows the size and dimensionality of every dataset. All methods were tested in metric spaces using the Euclidean distance (L_2) as the similarity measure. From each dataset, we removed 100 random elements to serve as query elements and used the remaining ones to build the graphs.

Table 1 – Evaluated methods.

Method	Search Methodology	Construction Algorithm
RNGW	Exact [9]	Brute-force $O(n^3)$
k-NNGW	Exact [9]	Brute-force $O(n^2)$
RNG	Approximate	Brute-force $O(n^3)$
NSW	Approximate	Consecutive vertex insertion with edges based on approximated *k-NN* search
k-NNG	Approximate	Brute-force $O(n^2)$ and *NN-Descent* (approximate *k-NNG*)
SAT	Exact [42]	Consecutive node insertion $O(\frac{n \log^2 n}{\log \log n})$

Table 2 – Datasets used for the experiments.

Dataset	Size	Dimensions
USCities	25,374	2
Color Moments	68,040	9
Texture	68,040	16
Color Histogram	68,040	32
MNIST	70,000	784
ANN-SIFT1M(base vectors)	1,000,000	128

We employed the C++ library NMSLib (Non-Metric Space Library) [90] to develop all tested methods. The library originally includes the *NN-Descent* and *NSW* graphs. Therefore, we extended it by implementing *RNG*, Brute Force *k-NNG*, and weighted graphs *RNGW* and *k-NNGW*. We also implemented the search algorithms *GS*, *GNNS* and exact, whenever applicable, to ensure that every structure employs exactly the same code for searching. As for the *SAT* we used the implementation available in the NMSLib.

[1] <http://www.census.gov/main/www/cen2000.html>
[2] <http://yann.lecun.com/exdb/mnist>

The experiments were carried out on an Intel Core i5 (4GB RAM) with a single thread for all methods on an Ubuntu GNU/Linux 16.04 64 bits.

The evaluated metrics for each method were: graph structure construction time, average query execution time, number of distance calculations during query and recall. Recall is the fraction of correct query answers retrieved. Suppose C is the set of correct query answers and C^A is the set of answers retrieved from the queries execution, the recall measure is given by equation 4.1 [91].

$$\text{Recall} = \frac{|C \cap C^A|}{|C|} \tag{4.1}$$

4.2.1 Exact Search Evaluation

This section discusses how proximity graphs behave to produce exact answers for similarity queries, specifically *k-NN* queries. The exact algorithm of Paredes and Chavez is based on metric space properties and was proposed for *k-NNG*, nevertheless, the algorithm can be adapted to other types of graphs. As mentioned at the end of Section 4.1, the property in the implementation for one of the pruning methods does not hold for all types of graphs. Therefore, our implementation of the exact algorithm in the weighted graph *k-NNGW* includes all pruning techniques of the original proposal while the implementation in *RNGW* excludes the aforementioned technique.

Firstly, we analyze how the performance of the exact algorithm in *k-NNGW* is affected by the number of neighbors used to build the graph, the parameter k that will be denoted as *NN* in this Chapter (to differ from *k-NN* queries parameter k). Figure 18 shows the average query time to answer *k-NN* queries, for $k \in \{1, 5, 10, 30\}$, in the datasets *USCities* and *Moments* for increasing *NN* values. It can be noticed that in general, the query time drops abruptly as the *NN* parameter increases, and after reaching the best *NN* value it grows again. Moreover, different values for k in queries demand different *NN* to achieve the best performance. Regarding this aspect, the dataset properties are of major impact. In Figure 18(a), the lowest query times for different k in *USCities* are reached for directly proportional values for k and *NN*, that is, the greater is k the greater is the best *NN* value. On the contrary, for *Moments*, such a relation is inversely proportional as, for example, for $k = 30$ the best *NN* is 25, and for $k = 1$ the best *NN* is 150 (Figure 18(b)).

Then, we compare in Figure 19 the performance of the exact algorithm using *k-NNGW* and *RNGW* with the performance of the approximate *GNNS* algorithm with configurations of *k-NNG-BF*, *NN-Descent* and *NSW* that provide highly accurate results (recall of at least 0.99) in the smallest query time for 10-NN queries. The first and second columns of Figure 19 show, respectively, the average number of distance computations and query time regarding the datasets, both in $\log_{10}$ scale. All graphs employed *NN* =100

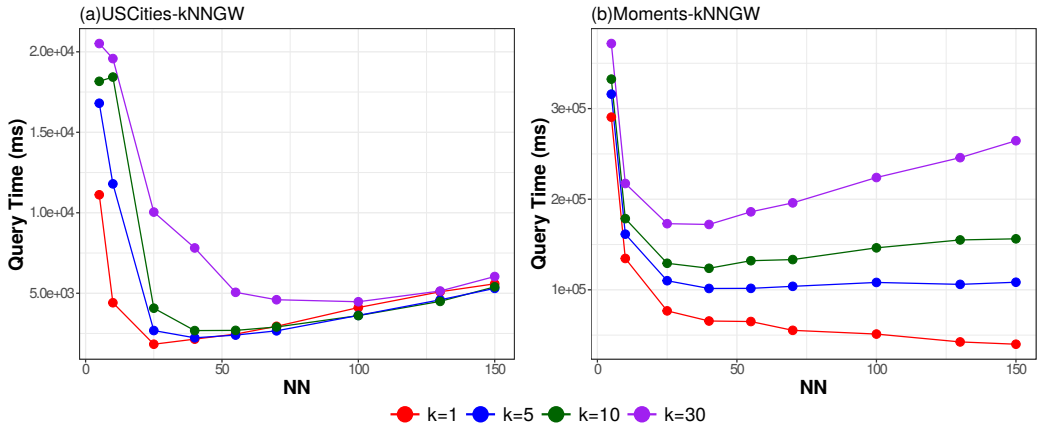

Figure 18 – Query time (ms) for different number of neighbors in graph *vs.* k values in k-*NN* queries for *USCities* and *Moments* datasets.

(except *RNGW* that is parameter-free). The best number of restarts for the *GNNS* is denoted as R in the figure.

For the *USCities* dataset, it is noticeable that the *RNG* exact algorithm performed significantly fewer distance computations than the other methods (Figure 18(a)). However, regarding query time (Figure 18(b)), the behavior is the opposite, being *RNGW* the slowest option by orders of magnitude. This huge performance degradation is due to internal computations of the algorithm, such as the Dijkstra shortest path algorithm used to calculate the lower and upper bounds that are used by the pruning methods. *k-NNGW* follows a similar behavior due to the same reason, although being less extreme than *RNGW*. *NSW* and *SAT* performed an intermediate number of distance computations, and *NSW* was the fastest method for this dataset, closely followed by *SAT*. *k-NNG* and *NN-Descent* demanded a huge amount of distance computations, as a result, their execution time was intermediate.

Regarding the *Moments* dataset, *k-NNGW* demanded the lowest number of distance computations (Figure 18(c)), nevertheless it was by far the slowest method regarding time (Figure 18(d)). *RNGW* was omitted because its execution time was too high. In this dataset, *SAT* degraded quickly with the increase in the dataset size, being up to 3 times slower than *k-NNG* and *NN-Descent*, and up to 5 times slower than *NSW*, which again was the fastest method.

These results confirm that approximate search is much more feasible for cheap metrics, when it is enough to provide answers close to the exact ones. The next sections concentrate on evaluating the parameters regarding approximate methods.

4.2.2 Analysis of Construction Parameters

Aside from *RNG* and *SAT*, all other tested methods have parameters to tune. Table 3 shows the construction parameters evaluated in this section, the tested values

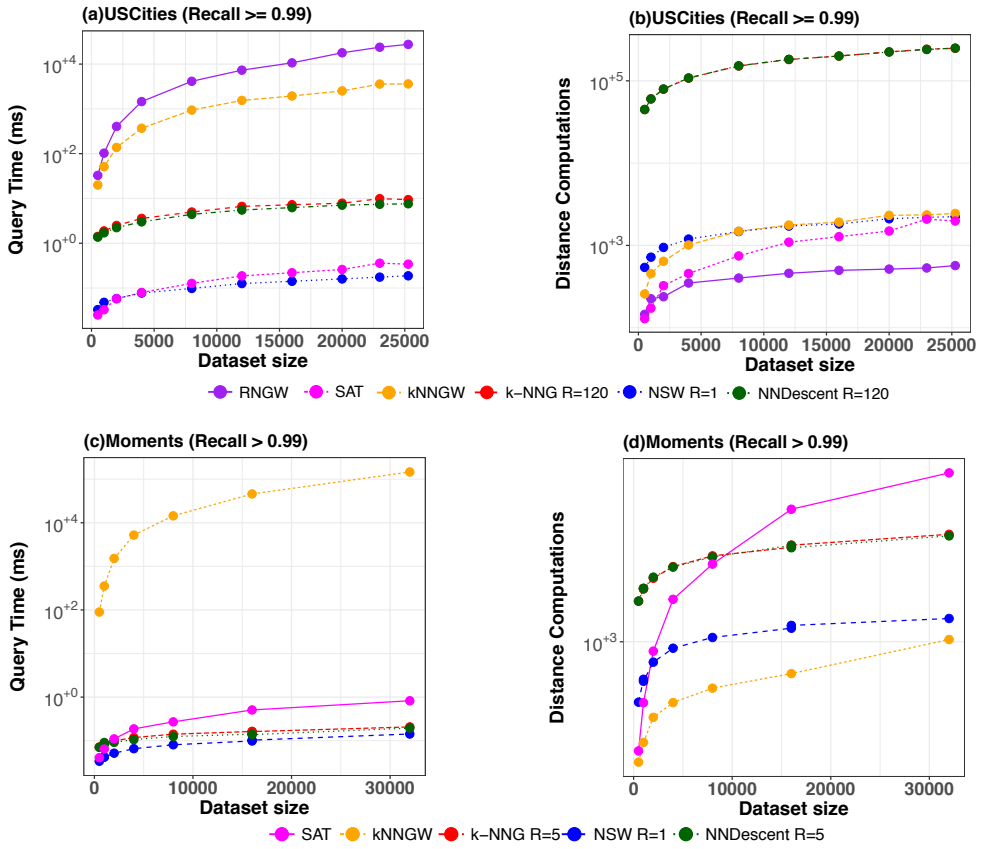

Figure 19 – Performance comparison between graph-based precise search and approximate search with at least 0.99 of recall for 10-NN queries in the *USCities* and *Moments* dataset.

and their description. To limit the number of variations, η and w were set to defaults. After having constructed the graphs, we executed batteries of 1-NN queries using the *GS* algorithm.

Table 3 – Construction parameters tested.

Graph	Parameter	Tested values	Description
k-NNG-BF, *NN-Descent* and *NSW*	*NN*	$\{5, 10, 25, 40, 55, 70, 100, 130, 150\}$	Number of neighbors per vertex (k for *k-NN* searches in *NSW*)
NN-Descent	ρ	$\{1.0, 0.5\}$	Sample rate of neighbor's neighbors to be checked for each vertex
NSW	*efConstruction*	$\{20, 100\}$	Size of the candidate list (ef) of the *k-NN* search in vertex insertion
NN-Descent	η	0.001 (default)	Early termination threshold
NSW	w	1 (default)	Number of starts for *k-NN* search

Figure 20 presents the results (construction time and query time plots are in $\log_{10}$ scale). It is visible in plots (a)-(c) in the figure that both *NSW* and *NN-Descent* demand more indexing time as the *NN* increases for all datasets. In contrast, our *k-NNG-BF* implementation executes a sequential scan over the dataset to identify the k-neighbors of each vertex and, consequently, the variation in the construction time with the increase of *NN* is insignificant. Even though the *NN-Descent* method has lower complexity than the brute force algorithm, for large *NN* *NN-Descent* takes close to longer time to

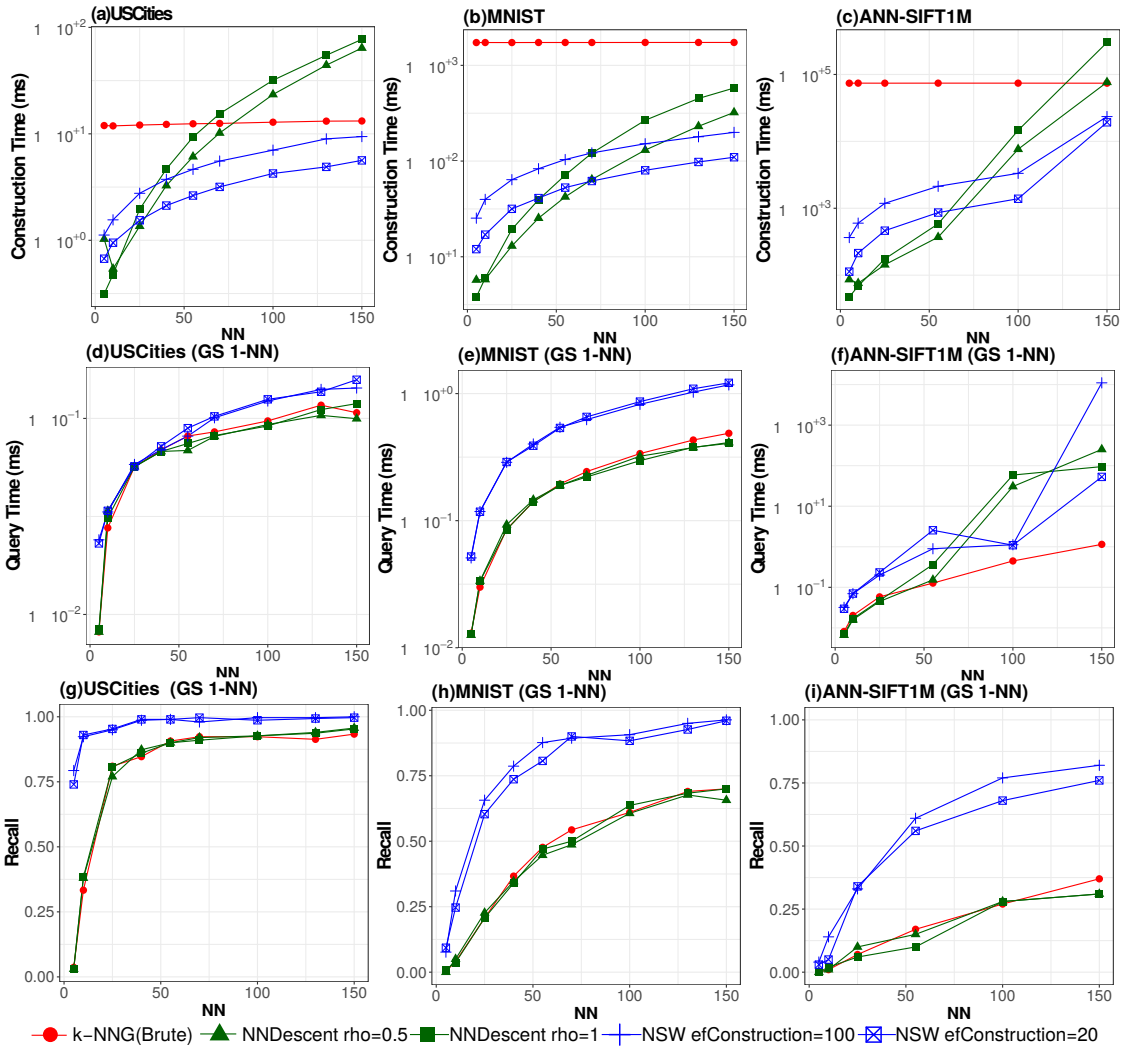

Figure 20 – Construction time (first row), query time (second row), and recall (third row) for increasing *NN* values.

build the graph if compared to *k-NNG-BF*. The exception, in our tests, is the *MNIST* high dimensional dataset that even for $NN = 150$ was still faster than *k-NNG-BF*, nevertheless presenting a monotonically increasing pattern. *NSW* demands more time than *NN-Descent* to be constructed for small *NN*, however, the behavior for large *NN* is the opposite. Even though *NN* increases the construction time in *NSW* because of the search process in vertices insertion, a large *NN* has more impact on *NN-Descent* as it increases the number of neighbors to compare in each iteration. Notice that both construction time and query time for *ANN-SIFT1M* were degraded by swap operations for large *NN*, as for some configurations the available RAM was exhausted.

Both the query execution time (Figure 20(d)-(f)) and recall rate (Figure 20(g)-(i)) consistently rise with the increase of the *NN* parameter for all datasets. As a role of thumb, the more precise a method is the more query time it demands. If the *NN* parameter is large, the constructed graph is more connected and *GS* has greater chances of returning the correct query answer regardless of the starting vertex. However, the larger the *NN*

the more adjacent vertices are evaluated in each greedy step, thus consuming more time. Remember that the *GS* algorithm is a simple greedy search algorithm that highly depends on the starting vertex. *NSW* was the slowest method in the majority of the cases, however, it was the most precise even for *MNIST* and *ANN-SIFT1M*, the tested datasets with the highest dimensionality and cardinality, respectively. Another result is that *NN-Descent* yields recall rates very close to the recall rates of *k-NNG-BF*, however demanding less query time for the same *NN* in several cases, with the exception of the *ANN-SIFT1M* due to swap overhead consequent from memory consumption.

The additional parameters of *NN-Descent* and *NSW* also have an impact on the construction time. As it can be observed in the plots, to build a *NN-Descent* with $\rho = 1$ demanded in average 30% more time than to build it for the same dataset and *NN* value with $\rho = 0.5$. Unexpectedly, such a rise in construction time did not reflect corresponding gains in query time or recall as they were close for both tested ρ values. With regard to *NSW*, the time elapsed to build a graph for the same dataset and *NN* value for *efConstruction*= 100 was in average 40% larger than for *efConstruction*= 20. Such a cost increase resulted in higher recall rates for some datasets, and overall negligible query overhead, except for *ANN-SIFT1M* in which some configurations suffered from swap. Nevertheless, the dominating parameter, according to our tests, in terms of cost-benefit for *NSW* is *NN*. For instance, the time elapsed to build *NSW* with *NN* =25 and *efConstruction*= 100 is comparable to the time to build *NSW* with *NN* =70 and *efConstruction*= 20 for *MNIST*, however the recall of the second configuration is around 30% superior than that of the first.

4.2.3 Scalability Evaluation

As discussed before, the construction of *NSW* graph and the *NN-Descent* method are dependent of the *NN* and additional parameters, while the construction algorithm for *k-NNG* and *RNG* have $O(n^2)$ and $O(n^3)$ complexity, respectively. For scalability evaluation of the different types of graphs we constructed and searched over the graphs by increasing continuously the dataset size from a very small number of elements to the full dataset. Figure 21 shows the construction time, the query time and recall for *GS* algorithm for according to the increase of elements in log_10 scale. Due to time limitations, we were not able to construct *RNG* for more than $100,000$ elements in *ANN-SIFT1M* dataset.

As expected by the algorithm complexity, *RNG* has the biggest construction among the tested methods. For the full dataset, is around one order of magnitude higher than the Brute-force *k-NNG* as it can be observed from Figure 21(a)–(c). The construction time difference is noticeable especially for the *ANN-SIFT1M* dataset in which *RNG* construction time for $100,000$ elements is just approximately 20% slower compared to a 100-NNG

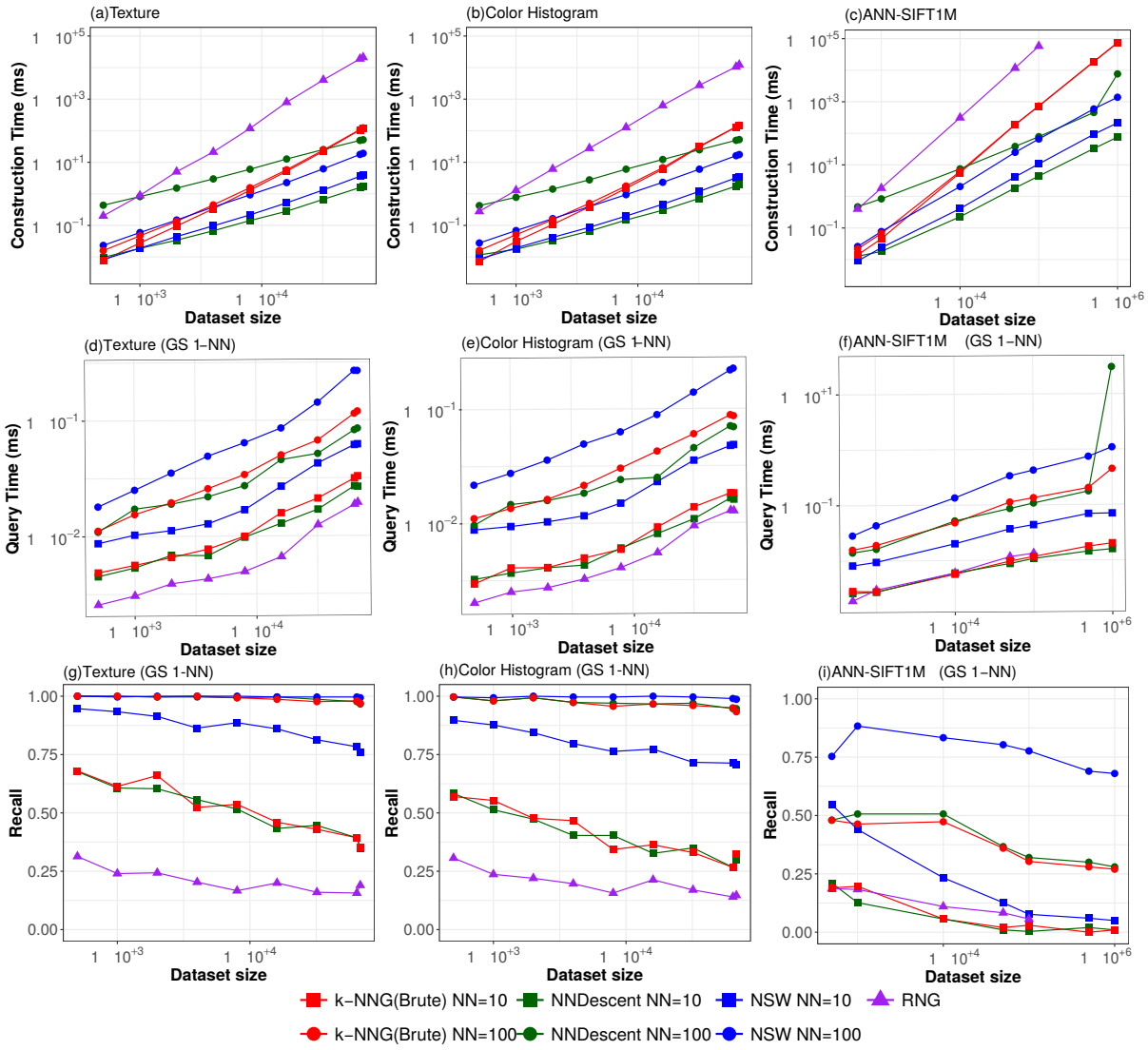

Figure 21 – Index time *vs.* number of elements in dataset for NN =10, 100, $\rho = 0.5$ for *NN-Descent* and *efConstruction*= 20 for *NSW*.

for the entire dataset (10 times more elements). Regardless of the number of elements, *NN-Descent* for $NN = 10$ has the fastest construction time, while, for graphs with NN =100 *NSW* has the best construction time. *RNG* has the worst scalability as the construction time increase one magnitude order in each 10 times the number of elements in all datasets in Figure 21. The *k-NNG* increases around half of the magnitude order from 10^3 to 10^4 elements in *Texture* and *Color Histogram* datasets, however from 10^4 to the full size of the dataset, it increases around one magnitude order. *NN-Descent* and *NSW* grow approximately half of the magnitude order in each 10 times the number of elements in all datasets. For graphs *k-NNG*, *NN-Descent* and *NSW*, both with NN =10, when the number of elements doubles the construction time increases in average 280%, 100% and 130%, respectively.

Figure 21(e),(f),(g) and (h) shows the query time scalability of the tested graphs for the greedy search algorithm for 1-NN search while Figure 21(i),(j),(k) and (l) shows

the corresponding recall. Even though *RNG* has the best query time the produced recall is the worst. Although *RNG* tests for the entire *ANN-SIFT1M* dataset were not completed, in the tested cases the recall rates are also close to *NN-Descent*, *NSW*, and *k-NNG* for *NN* =10. Indicating that for very high dimensional datasets *RNG* can perform better than the other tested graphs with small *NN*.

NSW has the highest query time when compared to *k-NNG* and *NN-Descent* for the same *NN* value but also has the best recall rates in all cases. This happens especially because of the long-range edges in *NSW*. Each vertex in *NSW* is not restricted to only *NN* connections like *k-NNG* and *NN-Descent*, and since *NSW* has more connections, consequently, more neighbors to be evaluated in each greedy step, impacting on the query time. Despite the fact that long-range links worsen the query time it raises the answer quality.

Surprisingly, *NN-Descent* had better query time than *k-NNG* for *NN* =100 and approximate query time for *NN* =10, except for the *ANN-SIFT1M* dataset due to swap overhead as described in Figure 4.2.2.

As expected, as the number of elements increases the recall decreases. For *Texture* and *Color Histogram* dataset *NN-Descent* and *k-NNG* for *NN* =100 achieved very high recall rates while for datasets with higher dimensions, *ANN-SIFT1M*, the *NSW NN* =100 outperforms significantly. Similarly, in *Texture* and *Color Histogram* datasets, the recall decrease of each method is not very accentuated when compared to the *ANN-SIFT1M* dataset.

Distinctly, graphs with a low degree (*NN* =10, in this case) present a very small recall rate for very high dimensional datasets even with a small number of elements. In datasets with lower dimensionality like *Color Histogram*, these graphs presented better recall rates for the same number of elements (around 0.55 0.6 for *NN-Descent* and *k-NNG*) and for even lower dimensionality the recall are higher (around 0.7 for *NN-Descent* and *k-NNG*).

4.2.4 Analysis of the Number of Restarts in the *GNNS* Search

To improve the result quality of the *GS* algorithm, we employed the *GNNS* algorithm, which uses three parameters: T for the number of greedy steps, R for the number of restarts of the greedy process, and E for the number of evaluated neighbors of a vertex in each greedy step. We do not employ the parameter T, instead, we terminate the greedy process when the algorithm reached the best answer (vertex) when compared to its neighbors. We have alternatively setted varied parameter R in $\{1, 5, 10, 20, 40, 80, 120, 160, 200, 240\}$ and used E=number of neighbors of a vertex for all searches.

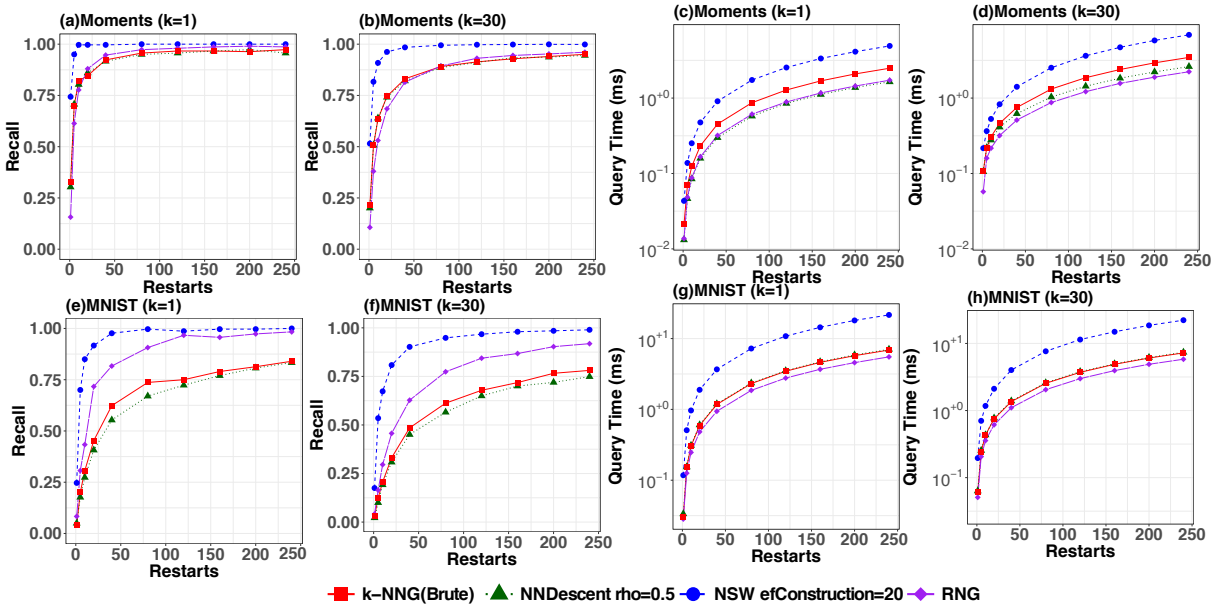

Figure 22 – Recall and query time for Moments feature (Corel dataset) and MNIST dataset with the construction parameters for *k-NNG*, *NN-Descent* and *NSW* graph: $NN = 10$, $\rho = 0.5$ for *NN-Descent* and *efConstruction*= 20 for *NSW* graph.

Figure 22 shows the recall and query time according to the numbers of restarts for *GNNS* algorithm for $k = 1$ and $k = 30$ *k-NN* queries.

In previous Figure 4.2.3 results showed that *RNG* has the worst recall among the tested graph for the *GS* algorithm. By varying the number of restarts for the *GNNS* algorithm, *RNG* has similar or better recall to *k-NNG* (Brute force and *NN-Descent*) for 1-NN queries, for higher $k = 30$ the search recall for *RNG* decreases and the query time increases (Figure 22) though it has the fastest query time when compared to the other graphs. Particularly for *MNIST* dataset (Figure 22(i) and (k)) *RNG* presented better recall and better query time when compared to *k-NNG* and *NN-Descent* for $k = 1$ and $k = 30$. For a recall of approximately 0.75 for $k = 30$ in *MNIST* dataset, *RNG* needed around 80 restarts while *k-NNG* and *NN-Descent* needed 240 restarts. As for the Texture and Moments dataset, *RNG* presented similar recall to *k-NNG* and *NN-Descent* for 1-NN searches (Figure 22(a) and (e)) with a query time approximate to *NN-Descent*. However, for 30-NN searches to achieve the same 0.75 recall in Texture(Figure 22(g)) dataset *RNG* needs 40 restarts while *k-NNG* and *NN-Descent* needs 20 restarts. Even though *RNG* needs more restarts to achieve the same recall when comparing the query time in Figure 22(h) *k-NNG* for 20 restarts is executed in approximately the same query time than *RNG* for 40 restarts. This indicates that tuning the number of restarts for high recall rates, *RNG* is a good option compared to small *NN k-NNG* and *NN-Descent* for 1-NN searches and in some cases for 30-NN searches, especially for high dimensional datasets.

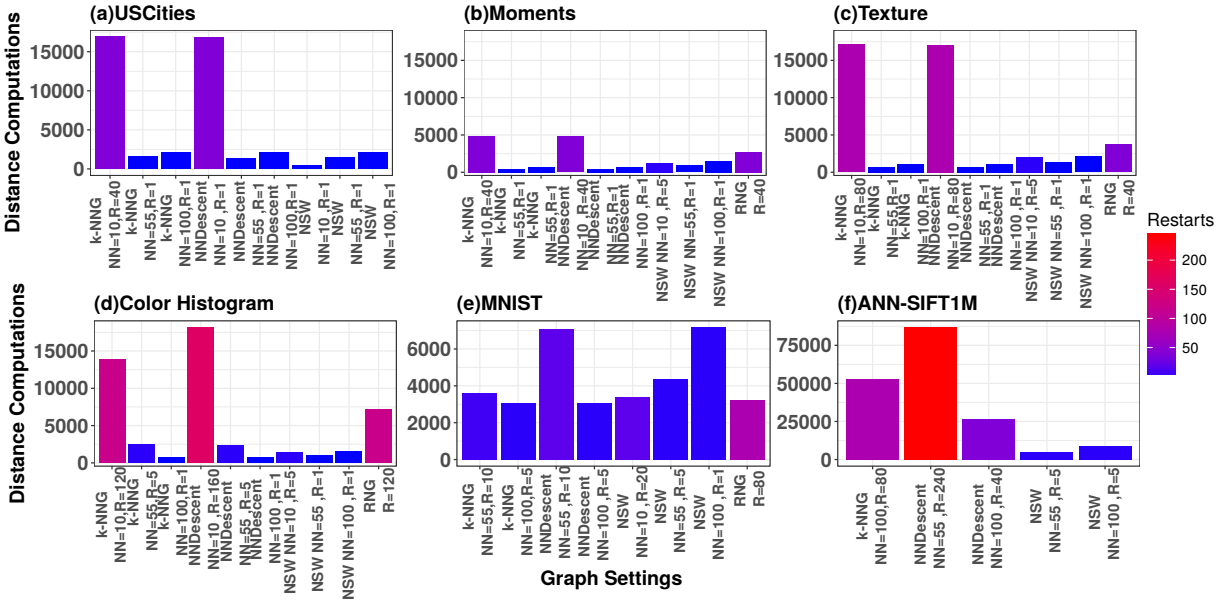

Figure 23 – Distance Computations and parameters for *GNNS* search algorithm for *RNG*, *k-NNG*, *NSW* and *NN-Descent* to achieve a recall rate bigger than 0.9 for NN parameters: [10, 55, 100] for 1-NN search.

NSW graph produces the best recall rates when compared to other graphs regardless of the number of restarts but also needs more time to execute the query. Overall, *NSW* needs a small number of restarts to achieve high recall rates in *Moments* dataset. To achieve recall=1 *NSW* needs 5 restarts for 1-NN searches in *Moments* dataset (Figure 22(a),(b)) while *RNG* needs 80 restarts. In this case, the *NSW* also performed better in query time. For 30-NN searches in *MNIST* (Figure 22(k),(l)) *NSW* and *RNG* achieved approximately the same recall with 40 and 200 restarts, respectively; unexpectedly, the query time for these two points is also approximate. Each vertex in the *NSW* graph can have more neighbors than the set *NN* parameter (long-range edges), hence, it will have more vertices to compare in each greedy step.

As we are interested in results as close to exact results as possible, we selected the search parameter settings for each method that achieved recall rates bigger than 0.9 and compared it by distance computations, as this is correlated to the query time in the *GNNS* algorithm. To avoid too many settings we fixed the following construction parameters: $NN \in \{10, 55, 100\}$, $\rho = 0.5$ and *efConstruction*= 20. The results are in Figure 23 and the needed number of restarts (R) is displayed in the method label. Graphs with the selected parameters that did not achieve recall ≥ 0.9 are not displayed. Except for the *RNG* in the *ANN-SIFT1M* dataset that we were not able to test for the entire dataset due to time limitations and *RNG* high construction cost.

To achieve the same recall rate, graphs built with bigger *NN* parameter values need a small number of restarts. For example in *Texture*(Figure 23(c)) dataset, *k-NNG*, *NN-Descent* built with $NN = 10$ needs 80 restarts while for $NN = 55$ (around 5 times

more connected) only R=1 is enough to achieve 0.9 recall. In cases in which the same number of restarts is needed to achieve a determined recall but the *NN* parameter of the graph is different, the search is faster (fewer distance computations) in the graph with the smaller *NN* parameter. This happens for *k-NN*, *NN-Descent* and *NSW* R=1, $NN = 55$ and $NN = 100$ in *Moments*, *Texture* and *USCities* datasets.

As discussed beforehand, *NSW* needs a very small number of restarts to achieve high recall rates, as long-range edges can act as shortcuts for data that are far from each other. There are cases that the *NSW* number of computations is comparable to other methods. For example, for $NN = 55$ in *USCities* dataset, *NSW* has approximately the same number of distance computations than *k-NNG* and *NN-Descent*. Another example is in *MNIST* in which NN=10 *NSW* and *RNG* have approximate distance computations. However, for large datasets such as *ANN-SIFT1M*, a *NSW* with the same *NN* outperforms *k-NNG* and *NN-Descent* by far.

Figure 23(b),(c),(d) shows *RNG* can outperform *NN* =10 *k-NNG* and *NN-Descent*, but needs more restarts. Even though *RNG* needed 80 restarts in (e), 16 times more than *NN* =100 *k-NNG*, *NN-Descent* and *NSW* needed, the number of distance calculations are comparable. Another interesting point of Figure 23(e) is that the *NSW* had the worst results compared to *k-NNG* and *NN-Descent* for the same *NN* parameter, with a visible difference for *NN* =100. The restarts can be tuned to achieve high recall, however, the query time and distance computations needed to achieve it relies on the graph structure.

4.3 Final Remarks

We surveyed and evaluated the performance of several representative graph-based methods used for exact and approximate similarity search, according to their main construction and search parameters for a variety of real-world datasets.

The experimental evaluation results showed that *NSW* outperforms other methods in construction time. However, when comparing the *NSW* to *k-NNG* and *NN-Descent* for the same *NN* value, the *NSW* has the worst query time. This is a consequence of the large number of neighbors per vertex in *NSW*. When comparing graph settings for a given recall rate, we were not able to point out a winner method for every condition tested. In our analysis, the search performance for *k-NNG* and *NN-Descent* was similar, since *NN-Descent* is an approximate method to build the *k-NNG*. Nevertheless, despite the similar search performance for *k-NNG* and *NN-Descent*, the *NN-Descent* construction time is much smaller for small *NN* values. We concluded that *RNG* is competitive or outperforms other tested methods for small *NN*, especially in high dimensional datasets. Although *RNG* has the major drawback of having a construction algorithm with high complexity, its construction is parameter free. When compared graph settings for a given

recall rate, no graph was the winner for all cases.

For *k-NNG*, *NN-Descent* and *NSW* methods, we found that there is a tradeoff between construction and query time. As the number of neighbors per vertex increases, query time and the number of restarts needed to return query answers close to exact answers decreases. Observing the results, we believe that adding edges to connect selected vertices from different graph regions can increase the recall in different types of graphs. From this hypothesis, we formulated a new proposal which will be presented in the next chapter.

5 HGRAPH: A CONNECTED-PARTITION APPROACH TO PROXIMITY GRAPHS

From the comparison presented in Chapter 4 we can point two main findings for query performance: (1) one of the drawbacks of the tested graph-based methods is the slow construction time, especially for *RNG* and *k-NNG* (both brute force and *NN-Descent* with large *NN* parameter); and (2) as the counter-examples showed, for a search algorithm based on spatial approximation return high-quality results the graph structure must provide ways so the search algorithm can approximate to the query answer in wherever graph region the correct answer is. The search must not concentrate efforts in only one region because it will probably result in a local optimal answer.

The *NSW* has achieved, in general, the best results in terms of recall, and one the main properties that lead *NSW* to reach high accurate results is that it connects different graph regions by using the long-range edges.

Definition 1. ***Long-range edges****: Long-range edges are edges that do not follow the proximity property of the graph and link vertices that are not close to each other according to the proximity property implied to build the graph.*

Based on these findings we propose the *HGraph* method. *HGraph* is a method that can be used with any type of graph proposed in the literature. *HGraph* speeds up the graph construction time and increases the approximate search results quality by adding long-range edges connecting selected vertices. These selected vertices are the pivots used in the partitioning process of the *HGraph*, as a consequence the long-range edges will connect different graph regions, reducing the local answer problem.

In this context, the *HGraph* foundation is twofold.

1. Divide and conquer strategy to build any type of graph. This strategy not only speeds up the graph construction but also can run in parallel. The *HGraph* method divides the dataset into smaller overlapping subsets according to elements selected as pivots. The division process is done recursively until the cardinality of a subset is smaller than a fixed value of m.

2. Addition of long-range edges to only selected vertices of the graph. The selected vertices are the pivots in each set recursive partition. Whenever a set is divided, the pivots are connected following the neighborhood criterion of a parameter defined graph type. The added long-range edges are responsible for connecting different graph regions.

The idea is to connect pivots that are far away in similarity space and as the partition process continues the distance between the pivots gets smaller in comparison to the first selected pivots. When the partition process ends, a graph is built locally in each subset. Even though our objective is to have independent subsets, at the end of the partition process we duplicate subset overlapping elements in order to not disconnect close elements that belong to different subsets. The overlap elements of each subset approximate the built *HGraph* to the exact type of graph chosen for the *HGraph* construction. Figure 24 shows an overview of the construction of a graph using the *HGraph* method. In the Figure the dataset is partitioned into overlapping subsets S_1, S_2 and S_3 considering overlapping elements (dashed lines). After the set is divided, the subset pivots are connected between them. The edges in black that connect the pivots are the long-range edges. As the final step, a graph is built in each subset (vertices with the same color). The vertices from a subset that connect to an element with another color, a different subset are the overlapping elements.

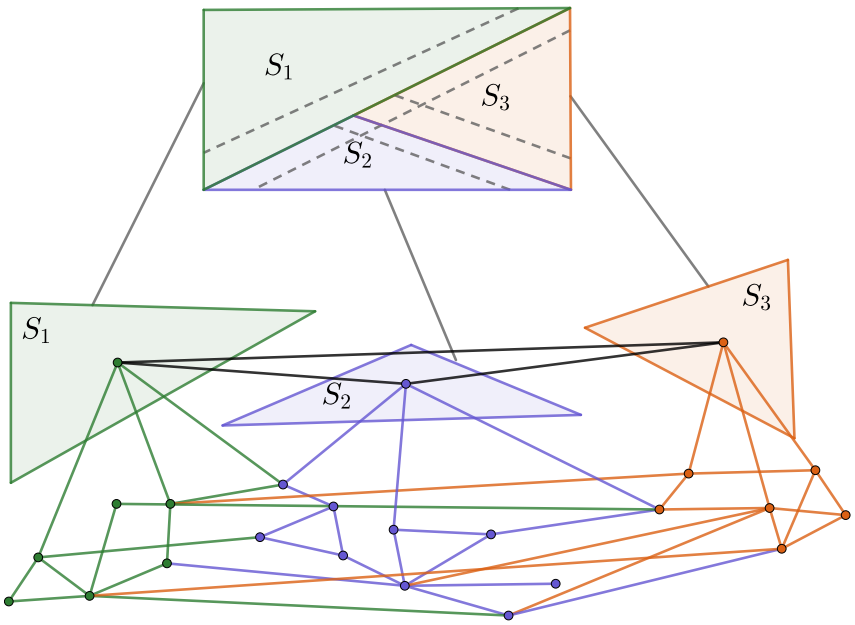

Figure 24 – Overview of the *HGraph* method: The dataset is partitioned into overlapping subsets S_1, S_2 and S_3. The pivots are connected between them and in each subset a graph is built (different colors of edges) considering overlapping elements.

The *HGraph* method differs from other algorithms that use divide and conquer strategy to build proximity graphs because its main objective is not to build an accurate graph. Even though, we prove in Section 5.2 that the *HGraph* is able to build an accurate graph, the main objective of the *HGraph* is increase the search quality (time and recall) compared to the "base" graph adding long-range edges. Another point of the *HGraph* is that the proposed method was designed to work with different types of proximity graphs while the proposed divide and conquer algorithms in the literature focus on accelerating

the construction of one specific graph type.

The *HGraph* is also different from the hierarchical graphs mentioned because the *HGraph* construction follows a Top-Down strategy while the hierarchical proximity graphs reviewed builds a "base" graph first and then connects selected vertices in an upper layer. One example is the DLG introduced in 3.2.2.

5.1 HGraph Method

In this section, we present details of the *HGraph* method. The parameters for the *HGraph* method are: number of pivots n_P and pivots selection strategy, the minimum number of elements m, overlap rate o, type of graph and graph construction parameters g_1 and the second type of graph g_2 for a final refinement procedure applied only to the pivots. For better visualization, the parameters and their definitions are presented in Table 4.

Table 4 – Construction parameters of HGraph.

Parameter	Definition
n_P	Number of Pivots
m	Minimum partition size
o	Overlap rate
g_1	Type of graph and parameters
g_2	Type of graph and parameters to be built – RefineGraph function (only pivots)
ptype	Pivot selection strategy

The experimental analysis in Section 5.2 suggests default values for some of the parameters, for example, overlap rate, pivots selection strategy, and type of graph g_1 and shows the behavior of each parameter in the *HGraph* construction algorithm and search performance.

Algorithms 2 and 3 presents the basic *HGraph* method. The main procedure of the *HGraph* method is the recursiveConstruction (algorithm 3). The recursiveConstruction is responsible for partition the dataset recursively, add long-range edges to the chosen pivots and build the graph type g_1 in each subset. The refineHGraph method is a final refining method to connect pivots from different graph regions. The next subsections explain in more details about each step of the *HGraph* method. Subsection 5.1.1 presents the adopted approach to partition a dataset, subsection 5.1.2 presents how the overlap elements in the dataset partition is calculated and subsection 5.1.3 gives more details on how to connect the pivots and build the graph in each subset.

Algorithm 2 *HGraph* Method.

Input Dataset S, number of pivots n_P, minimum subset size m, overlap rate o, graph types g_1 and g_2

Output *HGraph* G

1: **function** HGRAPH(S, n_P, m, o, g_1, g_2)
2: $G, P \leftarrow \{\}$
3: $G, P \leftarrow$ recursiveConstruction($S, n_P, l = 0, o, g_1, g_2, m, G, P$) ▷ G: built HGraph; P: set of pivots
4: $G \leftarrow$ refineHGraph(G, P, g_2)▷ g_2 is the type of graph to be built in the pivots set

Algorithm 3 HGraph Recursive Construction Algorithm.

1: **function** RECURSIVECONSTRUCTION($S, p, l, o, g_1, g_2, m, G(V, E), P$)
2: $l \leftarrow l + 1$
3: **if** $|S| \leq m$ **then**
4: $G'(V', E') \leftarrow$createGraph(S, g_1) ▷ Builds graph of type and parameters g_1 from set S
5: $V \leftarrow V \cup V'$
6: $E \leftarrow E \cup E'$
7: **else**
8: $P_S \leftarrow$ choosePivots(S, p) ▷ Selects p pivots from S
9: $P \leftarrow P \cup P_S$
10: $S_1, S_2 \ldots S_p \leftarrow$ partitionSet(S, P_S, o)
11: $G_p(V_p, E_p) \leftarrow$ connectPivots(P_S, g'_2) ▷ Connect pivots with undirected edges (g')
12: $V \leftarrow V \cup V_p$
13: $E \leftarrow E \cup E_p$
14: **for** $i \leftarrow 1, \ldots, |P_S|$ **do**
15: $s \leftarrow \frac{|S|}{{n_P}^l}$
16: $p \leftarrow \frac{|S_i|}{s}$
17: recursiveConstruction($S_i, p, l, o, g_1, g_2, m, G, P$)
18: **return** G, P ▷ Graph and set of Pivots

5.1.1 Dataset Partition

In this section, we will present the details of the dataset partition step (partitionSet in line 10 of algorithm 3). First of all the *HGraph* selects from the dataset n_P elements that will act as pivots according to the pivot selection strategy *ptype*. Then the dataset is divided into n_P subsets using the generalized hyperplane partition approach [92]. The generalized hyperplane partition is done by calculating the distance of each element of the dataset to each one of the pivots. Each element is addressed to the subset of its closest pivot. Suppose $n_P = 2$, and the pivots are elements p_1, p_2 selected from dataset S. The generalized hyperplane partition uses the following rules (Equation 5.1) to divide dataset

elements $s_i \in S$ in S_1, S_2 subsets:

$$\begin{aligned} S_1 &\leftarrow \{s_i | \delta(s_i, p_1) \leq \delta(s_i, p_2)\} \\ S_2 &\leftarrow \{s_i | \delta(s_i, p_1) \geq \delta(s_i, p_2)\} \end{aligned} \tag{5.1}$$

In this example, the elements that are closer to pivot p_1 than p_2 will belong to subset S_1 while elements that are closer to p_2 than p_1 will belong to subset S_2. This rule can be generalized to partitioning into n_P subsets.

Using the generalized hyperplane partition approach the resulting subsets will be disjoint from each other. If we use disjoint subsets the final graph built by the *HGraph* method will not connect two elements from different subsets even if they are really close in the similarity space. As mentioned in Chapter 4, disconnected regions or cases in which two elements that are very similar but are not connected can restrain spatial approximation based search algorithms to find exact or high recall approximate similarity search answers. For this reason, our proposal considers an overlap region between the subsets during the dataset partition. This problem remains even when adding the long-range edges to the pivots.

Figure 25 shows a 2-NN graph built with the *HGraph* method with 2 pivots (vertices C and G) without considering an overlap region between the subsets. In this graph, suppose a 1-NN search that should reach the vertex (i.e., I is the closest vertex to the query element). If this search started on vertex K, the result would be vertex E as the search would stop after having traversed the path from K to E because E is closer to the query element than all its adjacents (J and C). On the other hand, if we duplicate elements in an overlap region close to the border between two neighbor partitions, the border vertices can be connected by edges while the processing of the partitions continues to be independent. The figure shows two main approaches to define the overlap region: balls or hyperplanes. The approach based on balls would define the overlap between balls centered at the neighbor region pivots, being the amount of overlap given by the balls' radii. In the example of 25, using this approach vertices B, E and J would be connected to vertices F and M in the final graph. The approach based on hyperplanes defines the overlap as the area between the hyperplane that divides the partitions (main hyperplane) and another two hyperplanes whose distances from the main hyperplane give the amount of overlap. This approach allows to connect vertices close to the border in the whole extension of the partition border instead of only elements that are close to the pivots as occurs in the ball-based overlap. In 25, the overlap based in hyperplanes would also allow connecting vertices H and O to the vertices of the left-hand subgraph.

The next subsection will discuss the overlap calculating method.

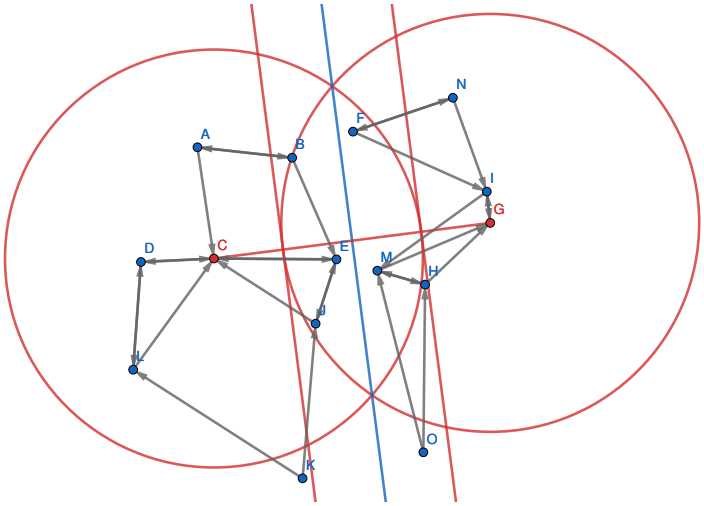

Figure 25 – HGraph construction of a 2-NN without considering an overlap region.

5.1.2 Overlap

An approach to select the overlap region elements is to use the covering radius from the pivots. However, depending on the distance used as the (dis)similarity measure some elements that are in the overlap region are not identified. Thus, there are no guarantees that at least one overlap region element will be selected depending on the dataset distribution and the defined covering radius and shape of the used distance measure. That happens because we are using a generalized hyperplane partition.

For this reason, to correctly define the overlap region elements we propose to use the distance according to the hyperplane. Since we are dealing with metric spaces the hyperplane does not actually exist, as a result, instead of calculating the distance of an element/point to the hyperplane we approximate it by using the distance difference between the element and the pivots. Figure 26 shows in orange the curvature of the overlap region considering this approach. This curvature was generated by setting points that have the same distance difference to the pivots (C and G) in different positions of the space. Observe also that in Figure the element O could be selected as an overlap region element.

There are two main alternatives to determine the overlap size: based on the distance to the pivots/hyperplane, or based on the number of elements in the overlap. We chose to use the second approach to guarantee that we always give chances to vertices close to the border of a partition to connect to vertices in the neighbor partition, nevertheless, the actual connection will only happen if the pair of vertices satisfies the graph proximity property. Therefore, our approach for overlap consists of estimating the number of elements in the overlap region using the parameter o and use the distance difference to select them. The overlap region size between two subsets is calculated as the proportion

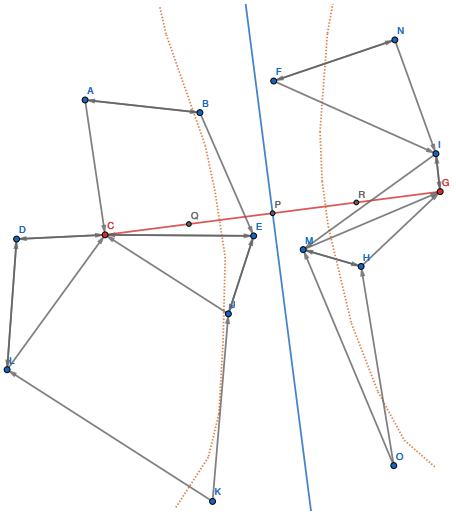

Figure 26 – Hyperplane curvature in orange.

o of the whole set that is being partitioned (Equation 5.2).

$$|S_1 \cap S_2| = \lceil |o * S| \rceil \tag{5.2}$$

By using ceil of the proportion to define the overlap region size we can guarantee that at least one element will be selected as an overlap region element.

As mentioned, in order to determine if an element $x \in S_1$ is in the overlap region of S_2 we consider the distance difference between the element and the pivots to simulate the distance to the hyperplane. The elements $s_i \in S_1$ are sorted according to the difference of the distance to its subset pivot p_1 and the distance to the pivot p_2 that we are currently calculating the overlap region, $\delta(s_i, p_1) - \delta(s_i, p_2)$.

After having sorted the elements the top-$|o * S|$ that has the smallest distance difference between the pivots are selected as overlap region elements. The overlap region elements are calculated for every pair of pivots. The overlap region elements of a subset are duplicated and added to the subset. An important point is that since the distance δ is calculated for all pivots during the generalized hyperplane partition it is not necessary any extra distance calculations for the overlap region computation.

The proposed approach to calculate the overlap guarantees a little diversification of selected elements without additional execution costs, which did not happen when using only the covering radius. This diversification is important to not only select elements that are in a very dense part of the border of the subset. Observe in Figure 27, the elements in green are those selected as overlap region elements. In Figure 27 the pivots of the first partition are in blue while the pivots of the second partition are in red. The elements in green are the overlap elements of the first partition and the elements in pink are the overlap elements of the second partition.

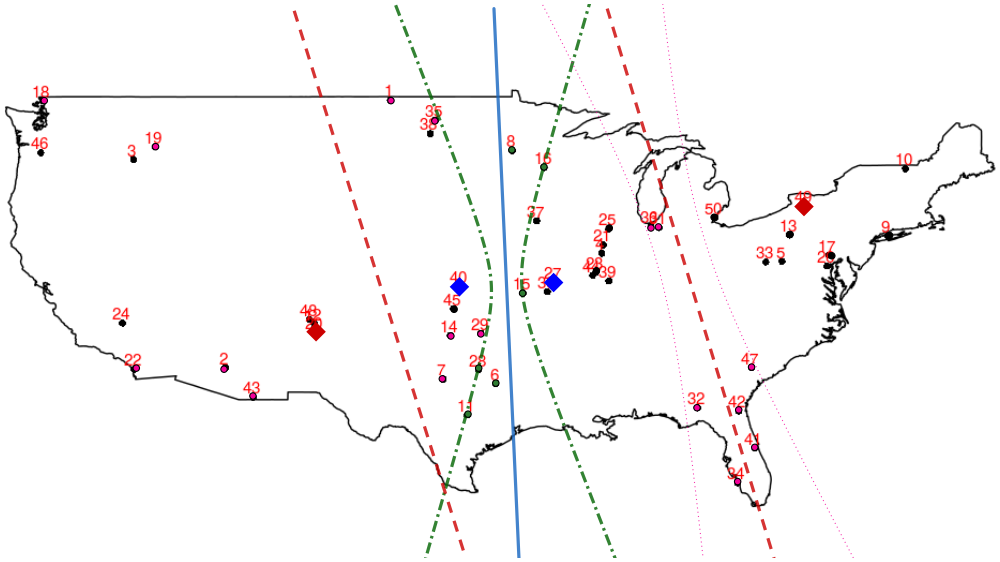

Figure 27 – Partitions of the *HGraph* method. The diversification is not ideal but helps better than when using only the distance.

5.1.3 HGraph Construction

The described partition process is recursively repeated for each S_i subset until $|S_i| \leq m$. When the partition process stops, the *HGraph* algorithm builds a graph g_1 (defined in the parameter settings) in each subset (line 4 of algorithm 3). We will call in this text each division step of *HGraph* as level l. The total number of divisions needed to reach a subset of size less or equal m will be represented by L and can be estimated by:

$$\frac{|S|}{{n_P}^L} \leq m$$

The number of distance calculations for each level is $n_P \cdot |S|$. In each level, this number decreases as the size of S also decreases to the number of elements in each subset $S_i + |oS|$, the minimum number of elements in the overlap region.

Depending on the dataset distribution, after the partitioning, the subsets may not be balanced. Balancing the number of elements in each subset is important for load balancing when running in parallel. Furthermore, when balancing the number of elements it is guaranteed that the size of each subset will be significantly smaller than the size of the whole dataset. With a smaller number of elements in each subset, the *HGraph* will need lesser number of recursion calls than when dealing with a very big subset. The number of recursion calls affects construction time.

In order to balance the size of the subsets in each level, we calculate the proportion of the actual subset size according to the expected size of the subset in that level and address the number of pivots according to this proportion. In the first division step, level $l = 1$, the dataset will be divided into parameter defined n_P subsets. In the second level $l = 2$ the expected total number of pivots is ${n_P}^l$ and the expected size of each subset is

$s = \frac{|S|}{{nP}^l}$. According to the expected size of the subsets, the number of pivots addressed to each subset is calculated by $\frac{|S_i|}{s}$. Figure 28 shows an example of how the number of pivots is addressed to each subset.

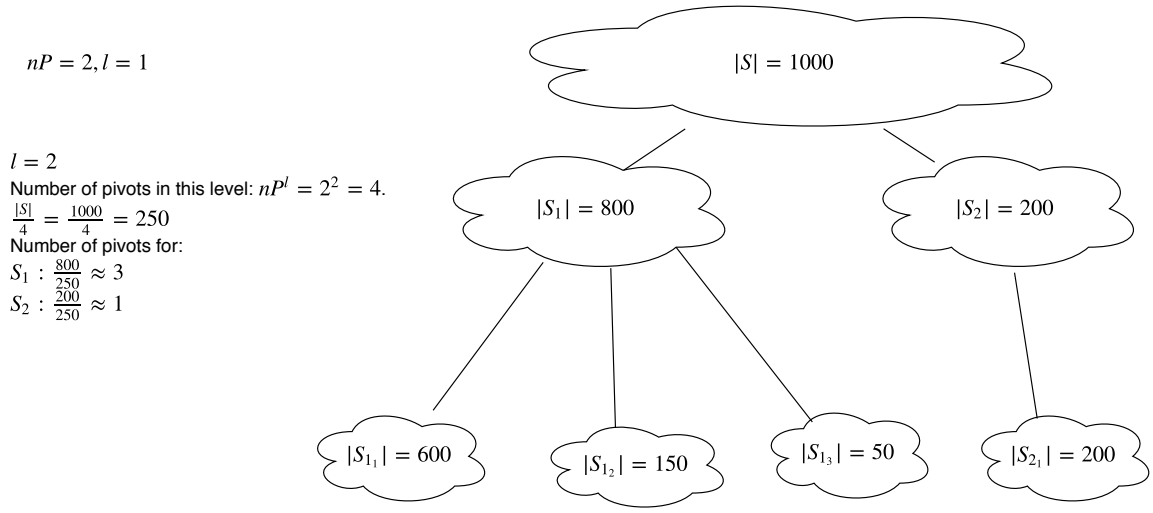

Figure 28 – Example on the number of pivots used in each subset division for $n_P = 2$ and a dataset $|S| = 1000$.

For better visualization of the *HGraph* method, Figures 29 and 30 show an example on the *HGraph* algorithm for the parameters $n_P = 2$ using random selection of pivots,$m = 6$, g_1= (*k-NNG*, NN =2),g_2=(*k-NNG*, NN =4), $o = 0.1$ in a dataset of 24 elements. From these parameters we can estimate $L = 2$. In Figure 29(a) the dataset is partitioned according to the vertices F and M (in red). Following the overlap rate formula, each subset has at least two elements in the overlap area (elements in purple). Notice that the overlap region sizes regarding the partitions are different due to the definition of the overlap according to the proportion of the number of elements in each partition. Thus, the overlap limit touches the overlapping element which is the farthest to the (main) hyperplane. The same process is repeated for each partition (in green) in Figure 29(b).

After partitioning the dataset until the subset size is less or equal to parameter m the chosen graph type and parameters g_1 is built in the final subsets. Figure 30(a) shows the graph construction for the first subset while in Figure 30(b) shows the graph construction completed for all subsets. The elements of the overlap region of each subset are in at least two subsets and are responsible for connecting the graphs built from each subset.

Even though the overlap region elements connect the graphs they only help to build an approximate graph to the "original" graph type construction algorithm. Without the overlap elements, the *HGraph* would not be able to answer correctly some similarity queries in which similar vertices are close but not in the same subset (see Figure 25). Before calling recursively the *HGraph* partition process, the pivots are connected with

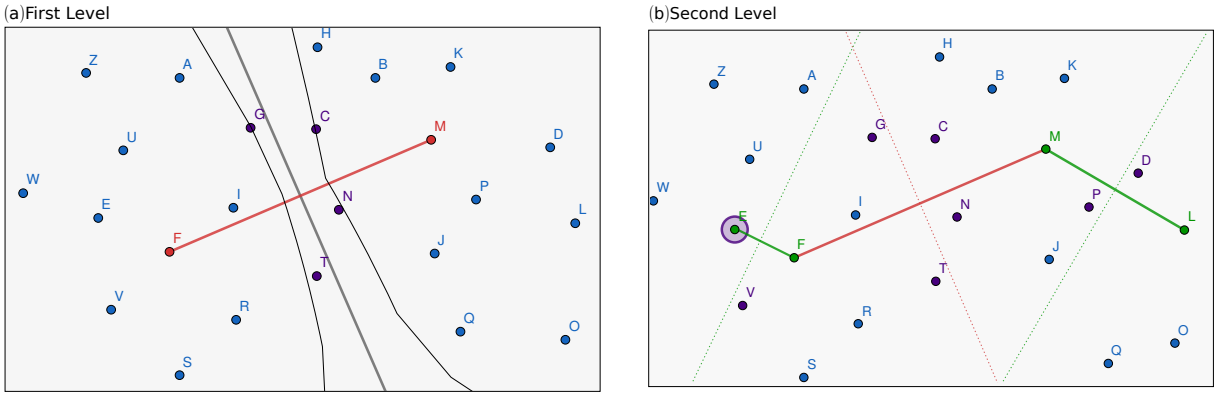

Figure 29 – Example on the *HGraph* algorithm. The dataset is recursively partitioned according to the divide and conquer strategy. The elements in purple are the elements in the overlap region of each partition.

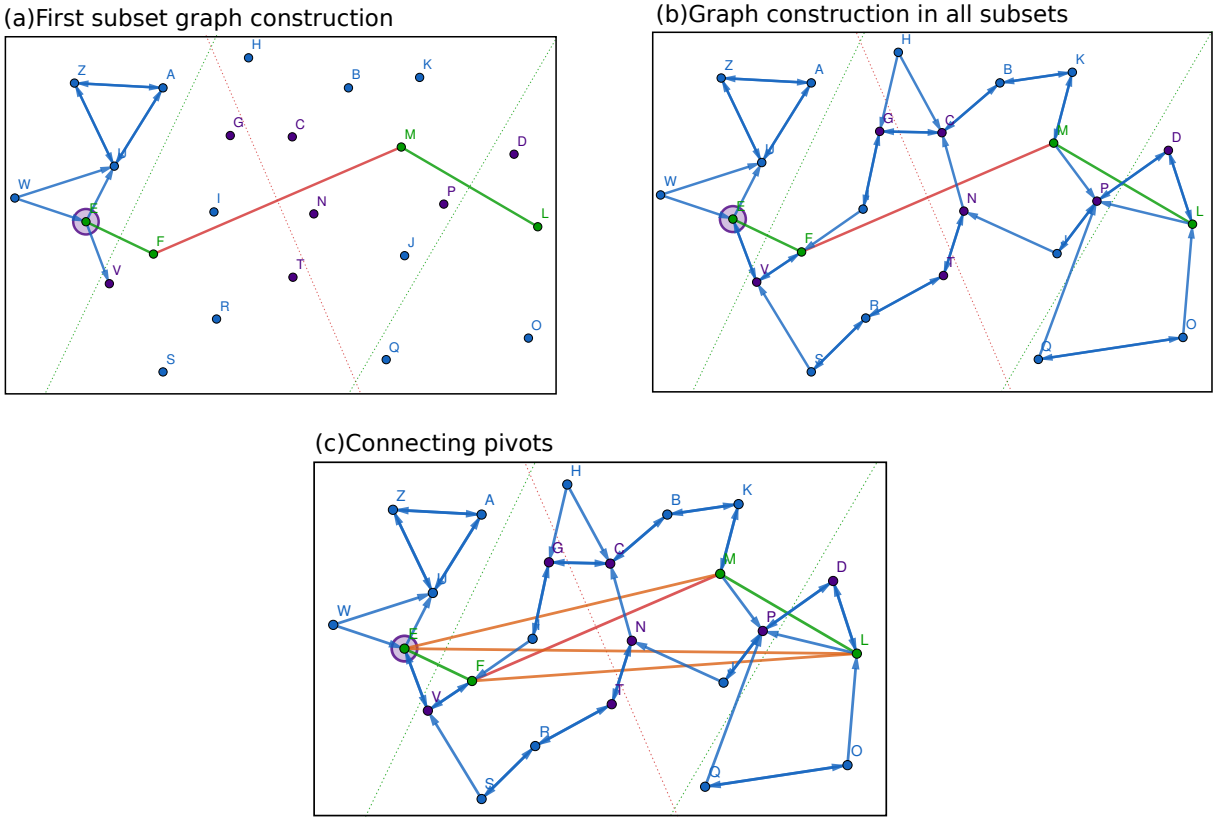

Figure 30 – (a)Graph construction in the first subset, (b)Graph construction for all subsets.

undirected edges according to the neighborhood criterion of g_2. The edges added to the pivots in this step are called long-range edges.

The pivots chosen in the previous level are selected to be pivots in the next level as well. For instance, observe that vertices F and M are also chosen as pivots for the next level in Figure 29(b) and (c). This way the pivots will become hubs in the graph that connects different subsets as the dataset partitioning recursively continues. Using the pivots as the vertices with long-range edges has two advantages: (1) no additional computations are needed to select vertices to add long-range edges, and (2) the pivots act as "connectors" to other graph regions as there will be at least one connection from one subset to another.

In Section 4.1 we mentioned the search problem when using a directed graph. To reduce this problem and assure that a neighbor vertex of a pivot can reach the pivot vertex and use its long-range edges during the search, all edges added to the pivots (in any partition level) are always undirected even when g_2 is a directed graph. Observe that in Figure 29(a) the pivots F and M are connected, in Figure 29(b) the pivots of the first subset E, F and of the second subset M and L are also connected. Since in this example we have only 2 pivots on each level they are just connected with each other. If there were more pivots, the pivots would be connected following the type of graph defined by g_2.

Observing Figure 30(b) it is perceptive that pivots have very few long range edges that connect graph regions that are far from each other. This is a result of connecting pivots only locally in each level subset. To fix this problem, a final RefineHGraph (line 4 in algorithm 2) procedure is called to the entire set of pivots of *HGraph*. This procedure adds undirected edges between pivots following the neighborhood criterion of the type of graph g_2. If the chosen type of graph used for the RefineHGraph procedure is dependent of the *NN* parameter the chosen *NN* must be bigger than the *NN* used for the graph constructed in the last level. The bigger *NN* parameter will create more long-range edges among the pivots and increase the graph connectivity. In Figure 30(c) we can observe the completed construction of the 2-*NNG* using the proposed *HGraph* method. In the Figure, we use an undirected *k-NNG* for the refine procedure. Edges (in orange in the Figure) were added to the pivots following a 4-NN graph (2**NN*).

The overlap elements belong to more than one subset. As a consequence, the overlap elements can have an excessive number of edges compared to the other elements. The excessive edges can impact the query time. In order to reduce the number of edges, the *HGraph* checks in each edge insertion if the edge really should be inserted or not according to the type of graph g_1. For example, if g_1 =*k-NNG* in each new edge insertion connecting the pair of vertices (u, v), the *HGraph* checks if the vertex v belongs to the $NN_k(u)$.

From algorithm 3 we can observe that the resulting graph is highly dependent of

the three main functions, createGraph (the type of graph to build), choosePivots (pivot selection method) and partitionSet (strategy to divide into subsets).

The createGraph and refineHGraph functions can build any type of graph used for similarity searches, for example, *k-NNG*, *RNG*, *NSW*, etc. There are numerous methods in the literature that can be used to choose pivots [35, 8] and can be adapted to *HGraph*. In the following Section 5.2, we will show evaluations on how each parameter of the *HGraph* affects both construction time and search performance.

5.2 Analysis of the Effect of the HGraph Parameters

The construction algorithm and main parameters for *HGraph* were presented in the last section. In this section, we discuss how the parameters interfere in construction and search performance and show experimental results regarding the choice of parameters.

HGraph was implemented in C++ as an extension of the Non-Metric Space Library [90]. For the experiments, we used the same datasets as in Chapter 4 shown in Table 2. The experiments were carried out on an Intel Core i7 (32GB RAM) with a single thread for all methods on an Ubuntu GNU/Linux 18.04.1 64 bits.

5.2.1 Long-Range Edges

In this subsection, we show how much the long-range edges can increase the average recall rate for similarity searches. To evaluate only the effect of long-range edges in *HGraph* we set the overlap proportion parameter o to 1. By setting $o = 1$ we can build an exact base graph g_1 but with the addition of long-range edges. Considering that $o = 1$, regardless of how many times we partition the dataset the size of the set will not reduce, in consequence, the stop condition of the *HGraph* was altered to run the experiments in this subsection. Instead of building the graph when the subset size is smaller than m, in this case, we limited the number of recursive calls, the levels, in algorithm 3. We used a small number of levels for the stop condition because of the long execution time when setting $o = 1$. The modification on the stop condition allowed us to evaluate how much long-range edges and the number of pivots n_P affects the similarity search performance compared to a graph-based method used in the literature.

In the experiments of this subsection, we tested $n_P = 2, 5, 10$ and $g_1 = k\text{-}NNG$ with parameter $NN = 5, 10, 55$ for 1 and 10-Nearest Neighbor queries using *GNNS* search algorithm. Figure31 show the performance of the search regarding query time and recall according to the graph parameter *NN* and Figure 32 show the performance of the search regarding query time and recall according to the number of Restarts for the *GNNS* search algorithm. The query time plots are in log_{10} scale. Figure 31 presented the results for the

USCities, Texture and MNIST datasets and Figure 32 shows the results for the Texture, Moments and Color Histogram datasets.

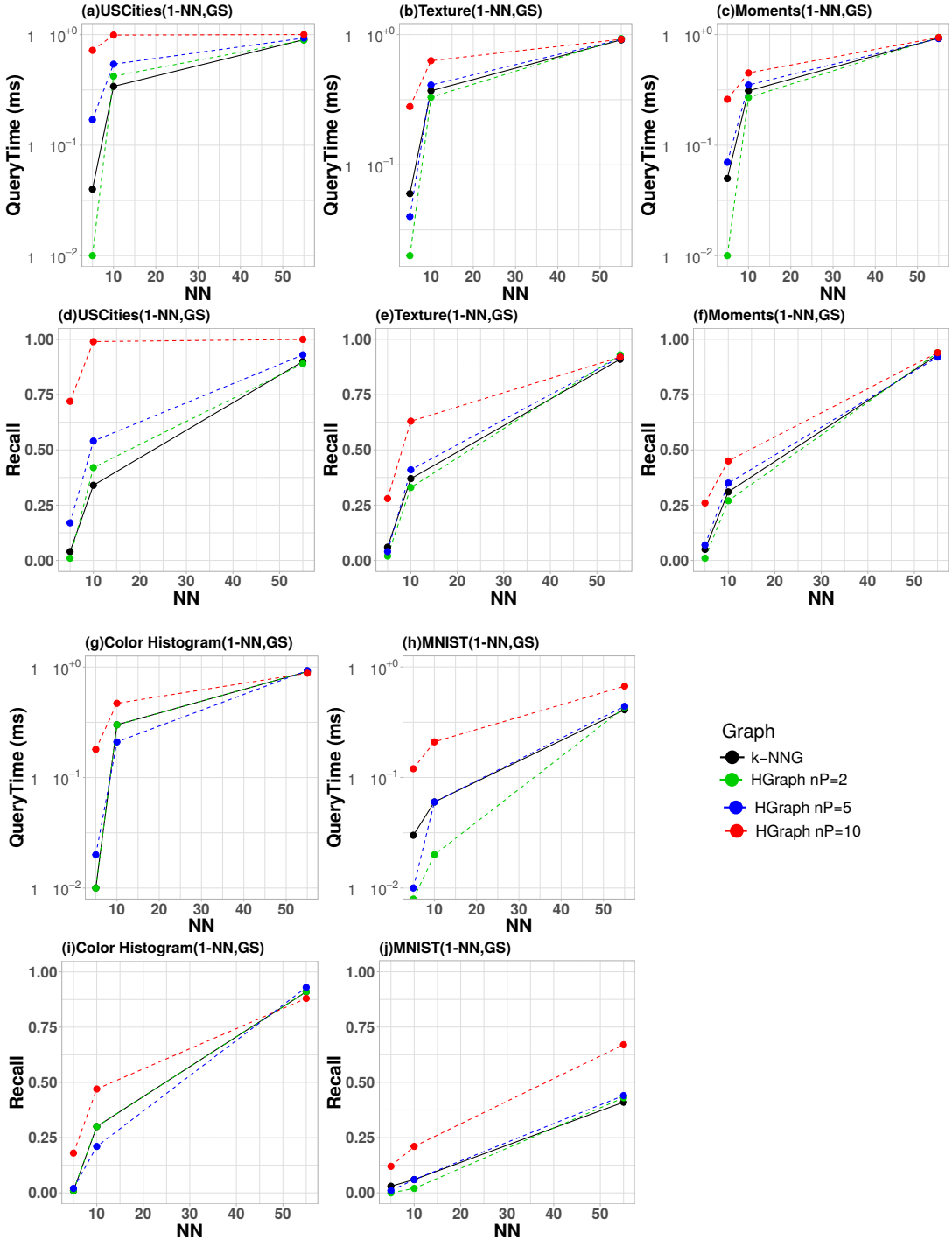

Figure 31 – Long-range edges for 1-NN *GS* algorithm.

As expected, as the number of pivots increases the more impact it has on the query time and the recall. As a pattern, the Figures 31 and 32 shows that even for different datasets the behavior is the same, in which increasing the number of long-range edges the recall increases, however, the query time also increases. Thus, we can also clearly observe the trade-off between the query time and recall for *HGraph* when adding long-range edges.

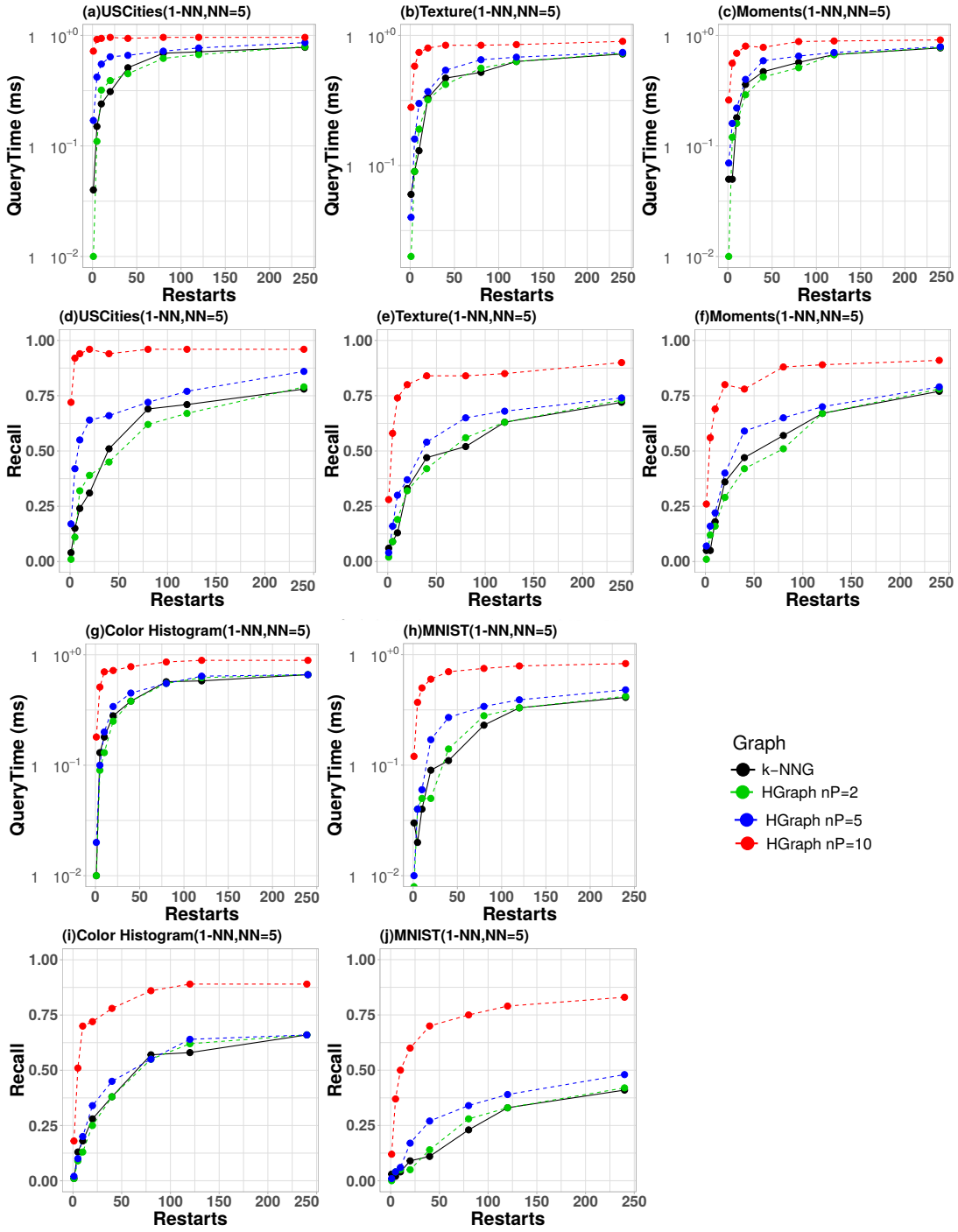

Figure 32 – Long-range edges for 1-NN and $NN = 5$ according to the number of restarts.

For smaller datasets, for example, USCities using only 2 pivots can increase the Recall in 20% when NN =10 with the *GS* search algorithm (Figure 31(d)). While in the features from the Corel dataset and the MNIST dataset we can observe that the increase in search answer quality is very modest when using only 2 pivots, and in some cases, for example in the MNIST (Figure 31(j)) dataset in which the dimensionality is bigger, the improvement in recall is also small when using 5 pivots. However, when using 10 pivots for Texture when $NN = 10$ the Recall increases approximately 70% compared to the *k-NNG*

as it can be observed in Figure 31(e). For the MNIST dataset the Recall increased from 0.06 for the *k-NNG* to 0.21 using 10 pivots and *NN* =10 (Figure 31(j)).

The drawback of adding more long-range edges is that the query time also increases as it can be easily observed in both Figures 31(a) to (c),(g) and (h) and 32(a) to (c),(g) and (h). The query time for *HGraph* using 10 pivots can take one order magnitude more time to execute the query compared to the *k-NNG*, as it can be observed in the USCities dataset. Unexpectedly, the query time for 2 and 5 pivots was better or approximate compared to the *k-NNG* in several cases when using a small *NN* (Figure 32). For the *GNNS* algorithm, when the number of restarts increases the difference in query time between *HGraph* and *k-NNG* decreases and it can be observed in Figure 32(a) to (c),(g) and (h) for restarts = 240.

5.2.2 Overlap Size

In this subsection, we evaluate the parameter o of the *HGraph* construction. The main objectives of the experiments in this subsection are: (1)Analyse how the overlap rate o affects the construction time in *HGraph* and (2)Investigate how much a *k-NN* graph built with the *HGraph* approximates to the exact construction of *k-NN* using the brute-force algorithm according to the overlap size. We used the *k-NN* graph because it is one of the most used graphs in the literature.

In order to isolate the parameter o, we did not add any long-range edges to the graphs built in this experiment. The o values tested were $0.05, 0.1$ and 0.2. This means that in each partition step $5\%, 10\%$ and 20%, respectively, of the set to be partitioned in each recursive step are used as overlap elements.

Figures 33 and 34 show the construction time for *HGraph* in comparison to the brute force *k-NNG* for the Color Histogram and Moments features of the Corel dataset. From these figures, we can observe that when increasing the overlap rate o the construction time is affected and even takes more time to construct the graph than the brute force *k-NNG*. The bigger the value of the overlap rate o the bigger is each resulting subset of the partition process. As a consequence, the number of recursive calls in the partitioning step is also bigger to reach the minimum value of m elements to build the graph. This happens when overlap rate $o = 0.2$ while when $o = 0.05$ or 0.1 the construction difference can be of at most 1 order magnitude faster compared to the *k-NNG* brute force construction.

Another point we can observe in Figures 33 and 34 is that the construction time for *HGraph* with any value of o grows along with the increase of the *NN* parameter, also strong in the Figures is that the smaller the overlap the better. And the construction time for a big number of pivots and *NN* can get close to the *k-NNG* construction time even when using a small overlap rate of 0.1.

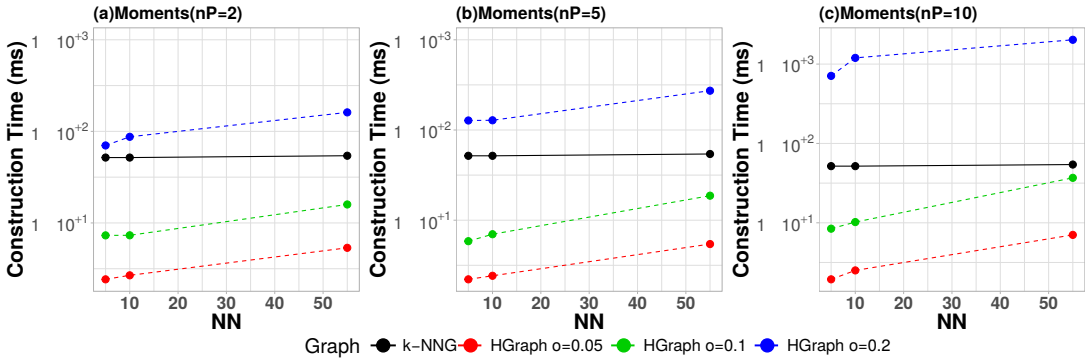

Figure 33 – Construction Time of HGraph according to the overlap rate values for Color Moments feature (Corel dataset) for $m = 1000$.

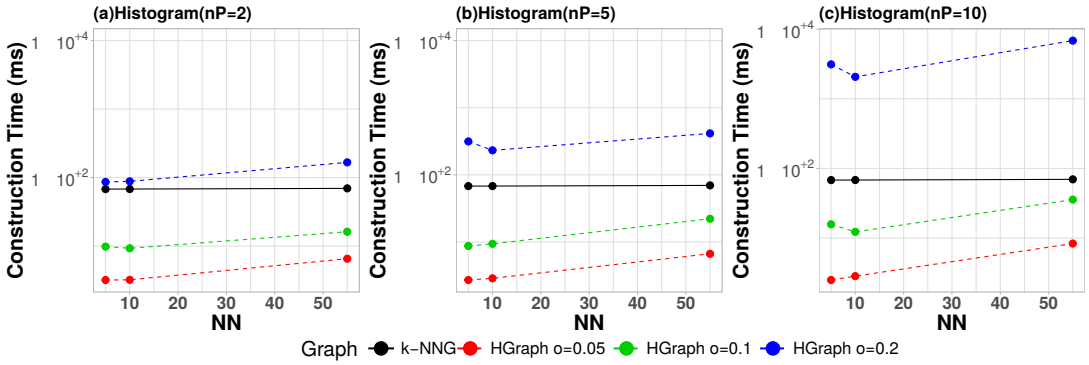

Figure 34 – Construction Time of HGraph according to the overlap rate values for Color Histogram feature (Corel dataset) for $m = 1000$.

Since no long-range edges were added, all of the vertices built with *HGraph* will have the same number of edges per vertex compared to the exact *k-NNG*. Since the *GNNS* algorithm is dependent of the number of edges to evaluate, as already discussed previously, the query time of the *k-NNG* and *HGraph-k-NNG* are similar as it can be observed in Figure 35.

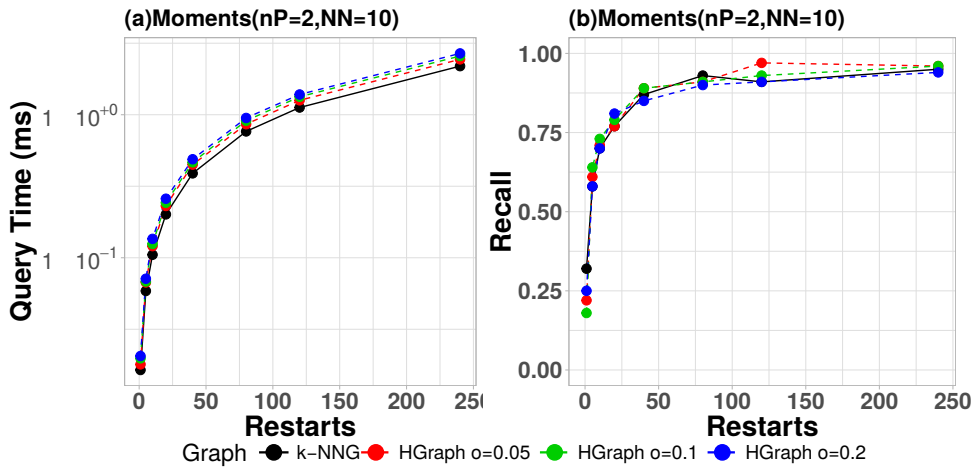

Figure 35 – Moments Dataset: Query Time and Recall according to number of Restarts for HGraph-*k-NNG* built with different overlap rate values.

A second analysis is how much the proposed *HGraph* can approximate to the actual graph construction and how does the overlap rate affects it. So to measure how much the HGraph-*k-NNG* approximates to the brute force *k-NNG* construction we used the accuracy in Equation 5.3 [65]

$$\frac{|E_{HGraph} \cap E_{k-NNG}|}{|E_{k-NNG}|} \tag{5.3}$$

which calculates the ratio between the number of edges in *HGraph* (E_{HGraph}) that match to edges in *k-NNG* (E_{k-NNG}) and the number of edges in *k-NNG*.

Figures 36 and 37 show the calculated accuracy of the HGraph compared to the exact *k-NNG* for Color Moments and Color Histogram feature of the Corel dataset, respectively. In these Figures, the color represents the values for overlap rate and the different shapes represent the number of pivots n_P.

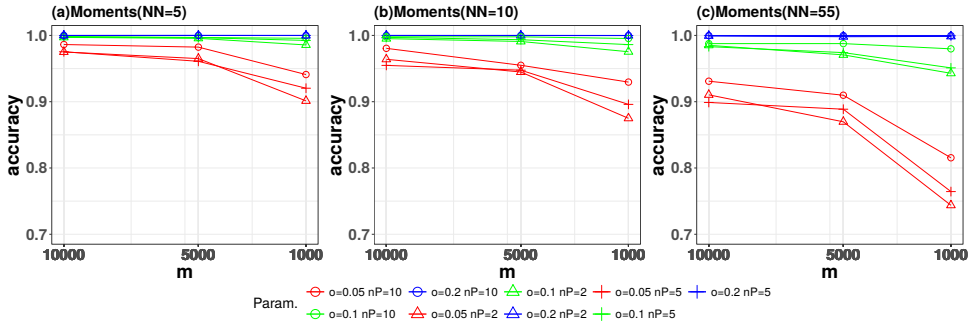

Figure 36 – Accuracy of the HGraph for Color Moments feature compared to the exact *k-NNG*.

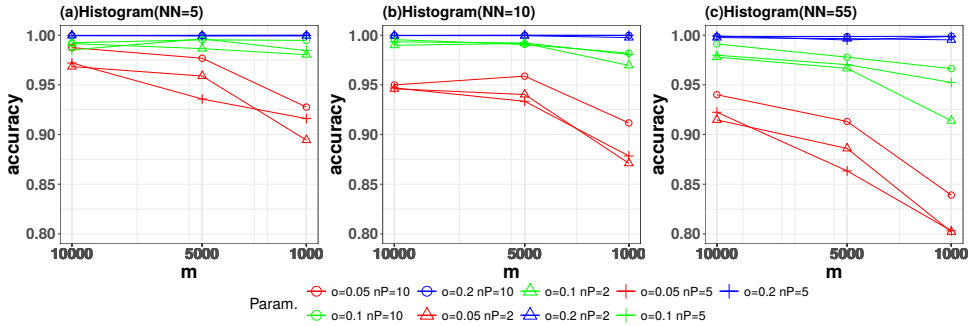

Figure 37 – Accuracy of the HGraph for Color Histogram feature compared to the exact *k-NNG*.

In both features the use of $o = 0.2$ was enough to give accuracy above 0.99 regardless of the number of pivots (n_P) used as a construction parameter. However, as discussed previously, $o = 0.2$ and above this value causes the construction of the *HGraph* slower than the brute-force construction, considering non-parallel construction. The overlap rate of $o = 0.1$ achieved accuracy above 0.9 in all parameters settings while $o = 0.05$ had the

worst performance among the tested values. The bigger the overlap rate value the more accurate the resulting *HGraph* when compared to the base graph type, in this case, the *k-NNG*.

Another interesting point of the results is that the more pivots the better the accuracy in both datasets (Figures 36 and 37). Comparing the results that use the same overlap rate value for construction, the *HGraph* in which the number of seeds $n_P = 10$ has better accuracy compared to $n_P = 2$ or $n_P = 5$. Thus, the bigger the m value the better the accuracy. This happens for the reason that the bigger the m value the number of elements to build the graph is also bigger and minimizes the case in which the number of overlap elements was not sufficient to build an accurate graph.

Considering that $o = 0.1$ gave a satisfying accuracy of above 0.9 in all tested parameter settings, accelerated the construction of the *HGraph* compared to the exact *k-NNG* (Figures 33,34) and did not degenerate the recall (Figure 35) or query time we fixed the overlap rate to $o = 0.1$ in the experiments in this section.

5.2.3 Pivot Selection Techniques

There are numerous methods in the literature that can be used for pivot selection and can be adopted in *HGraph*. Survey articles by the authors Amato, Esuli e Falchi[35] and Bustos, Navarro e Chavez[93] introduced pivot selection techniques for similarity search in metric spaces. Some of the pivot selection techniques that were implemented to work on *HGraph* are:

- Random Selection[93] – selects n_P pivots randomly from the dataset;
- Hull of Foci (HF) algorithm[94] – The algorithm starts by selecting an arbitrary element from the dataset. In each iteration, the farthest element from the previously selected element is chosen to be part of the pivots set. This process is executed until n_P pivots are selected.
- k-Medoids[35] – Partition based clustering algorithm that tries to minimize the average distance between elements and cluster medoids. First of all n_P elements of the dataset are selected to act as medoids then the algorithm follows two steps: (1)Assign each element of the dataset to the cluster of its most similar medoid (pivot) and (2) In case there is another element a in which the average dissimilarity to all objects in the cluster is less than the previously chosen medoid, the element a is the new medoid of that cluster. This process is done until there are no changes in the medoids (pivots) set.

We chose the random selection because of its low computational cost. Meanwhile, we chose the HF algorithm and the k-Medoids (PAM) algorithm taking into consideration

that both these algorithms minimize the distance of the dataset elements to its pivots. However, it is known in the literature that the PAM algorithm works inefficiently for large datasets [95, 96] and HF algorithm has $O(n)$ [94] complexity. As a result, both algorithms can affect *HGraph* construction time. In order to minimize the pivot selection execution time for the HF algorithm and the k-Medoids, in every partition step, we selected the pivots from a random sample of the whole set in that step. Considering that in each step the set size changes we used a rate from 0.05 to 0.5 to determine the sample size. The sample size is the rate values times the set size in that step.

For the experiments in this subsection we searched for 1-NN neighbors of 100 randomly selected query elements using the *GS* and the *GNNS* algorithm. Figures 38 and 39 show the results for the Color Moments and USCities datasets for the construction parameters of HGraph: $m = 1000, n_P = 5, o = 0.1$, and g_1=k-NNG. The figures show the Construction Time, Query Time and Recall for 1-NN searches using the *GS* algorithm.

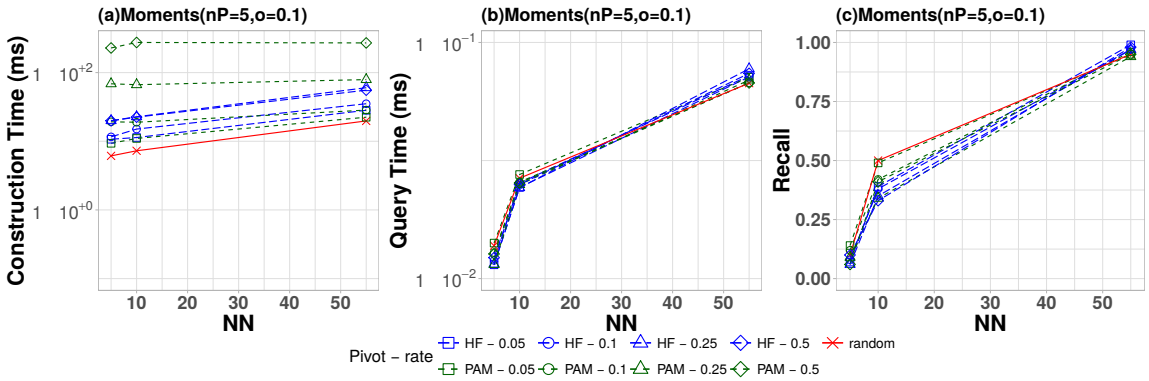

Figure 38 – Comparison of Pivot Selection techniques for the Color Moments dataset.

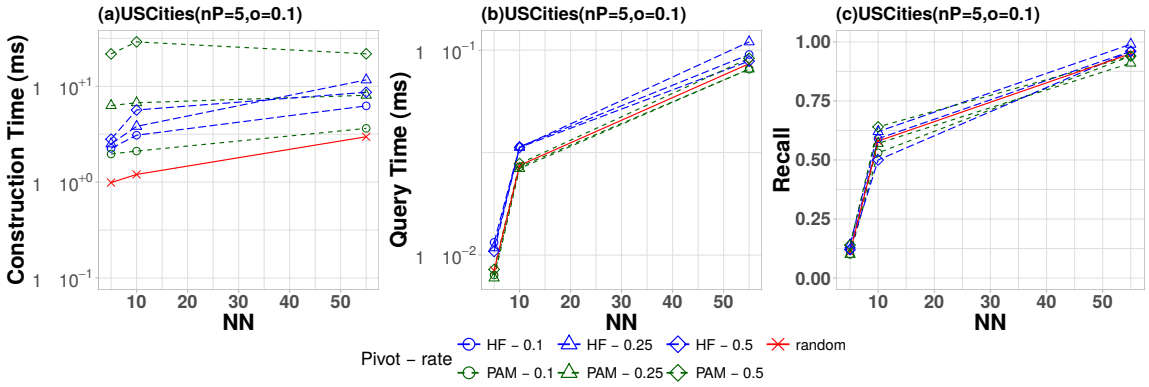

Figure 39 – Comparison of Pivot Selection techniques for the USCities dataset.

As expected the use of HF and PAM algorithms to select pivots affects the graph construction time. We can observe from Figures 38(a) and 39(a) that in both datasets the PAM algorithm resulted in the slowest HGraph construction because of its computational complexity.

In the USCities dataset, the HF algorithm took longer to execute queries when compared to the PAM and random pivot selection for all rate values used in the experiment. However, in the Color Moments dataset the query time for all pivot selection methods was similar for NN =5 and 10. Unexpectedly, the recall in the Moments dataset using random pivot selection was better or very close to the other pivot selection algorithms, while in the USCities datasets the random pivot selection recall stands in between PAM and HF algorithms according to the rate values.

The next Figure 40 shows the results for pivot selection techniques according to the number of Restarts in the *GNNS* algorithm.

In both datasets, the HF algorithm for pivot selection performed the best using 0.5 sample rate according to the increase of restarts and unexpectedly the PAM algorithm resulted in worst to close recall to the random pivot selection. In the Figure 40 we can see clearly the tradeoff of query time and recall for Moments and USCities datasets. The *HGraph* constructions that took the longest to execute the search algorithm had the best resulting recall while the fastest ones had worst recall. In both cases the random pivot selection had intermediate results, however, it is the fastest algorithm to use in HGraph construction as already shown. An exception is in the Color Histogram dataset (Figure 40(g) and (h)) the random pivot selection in comparison to the PAM algorithm had a better recall in most of the cases, however, it takes less time to execute the search. According to this experiment, there are no big advantages in using the PAM algorithm for pivot selection as it takes a longer time to construct the graph and the search recall was close or worst when compared to the HF algorithm and the random pivot selection.

In Figure 40 all the experiments used the same parameters for the *k-NNG* construction NN =5. Figures 42 and 41 show the behavior of the tested pivot selection algorithms when the parameter NN increases to NN =10 according to the number of restarts in the *GNNS* algorithm.

In Figure 41 we can observe that in the Moments dataset the HF algorithm with sample rate= 0.5 took the longest to execute the queries while using the other values it was faster than random and PAM. In Figure 42 we can observe that in the Color Histogram dataset the query time of the different pivot selection algorithms is close for all values of restart. Regarding the recall, the HF algorithm is slightly better for some cases, however, the recall is also close for all values.

For a better visualization on how much the pivot selection algorithm affects the HGraph construction time according to the m parameter and the number of pivots, we fixed the sample rate of the pivot selection algorithms to 0.1. Figure 43 shows the results.

As the parameter m increase the construction time increases in *HGraph* construction. This happens because the bigger the parameter m the more elements are contained

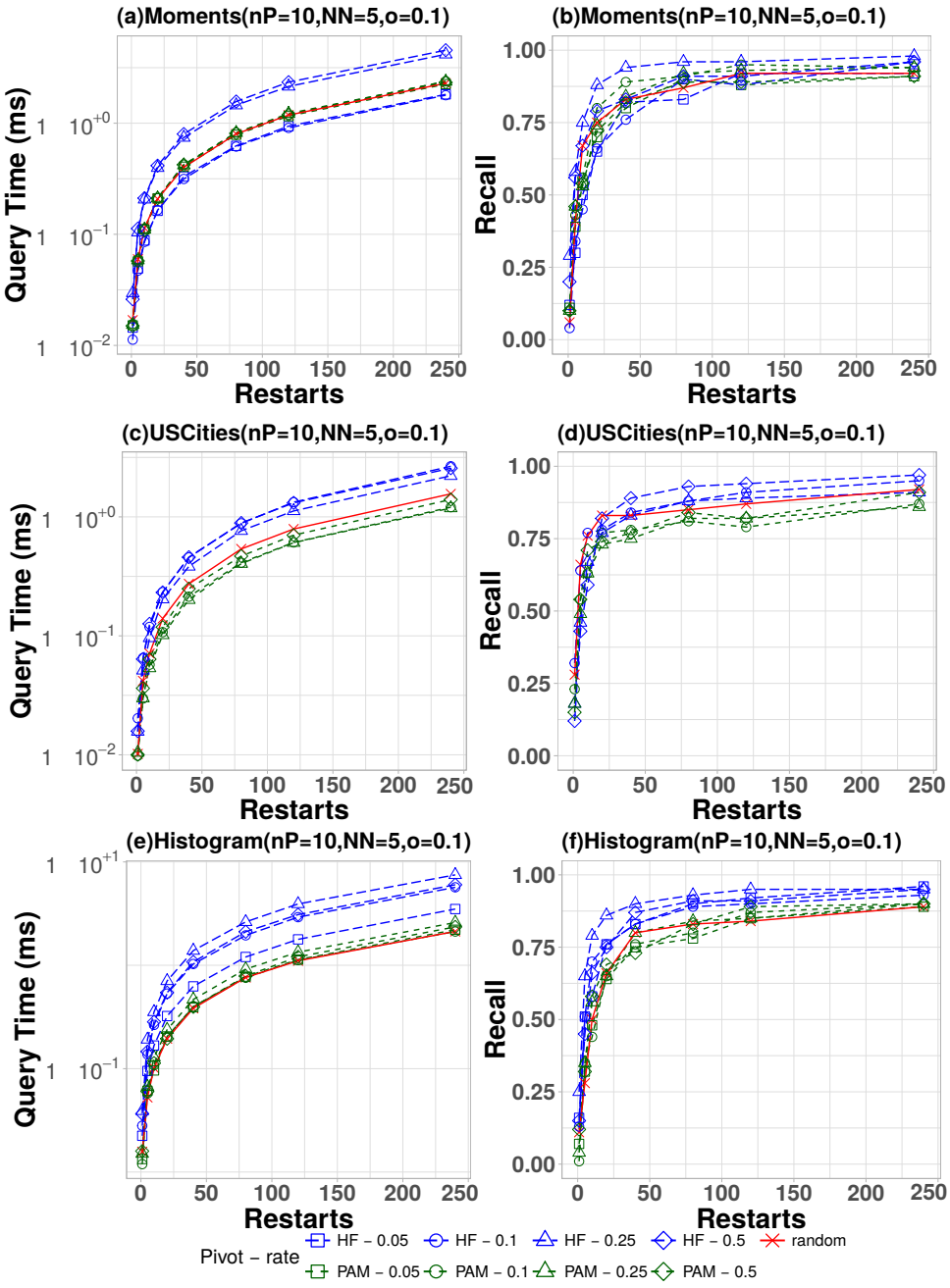

Figure 40 – Pivot Selection techniques and sample rate according to Restarts for Color Moments, USCities, Color Histogram dataset for $n_P = 10$ and $g_1 = k\text{-}NNG$, $NN = 5$.

in a subset to build the graph.

We can clearly see that the PAM algorithm takes the longest to construct closely followed by the HF algorithm. The fastest pivot selection is using random pivot selection as expected by their respective computational costs. Analysing the number of pivots used, the n_P value, for HF and PAM pivot selection the index time increases as the number of pivots also increases. While for random selection the intermediate value $n_P = 5$ had the best index time.

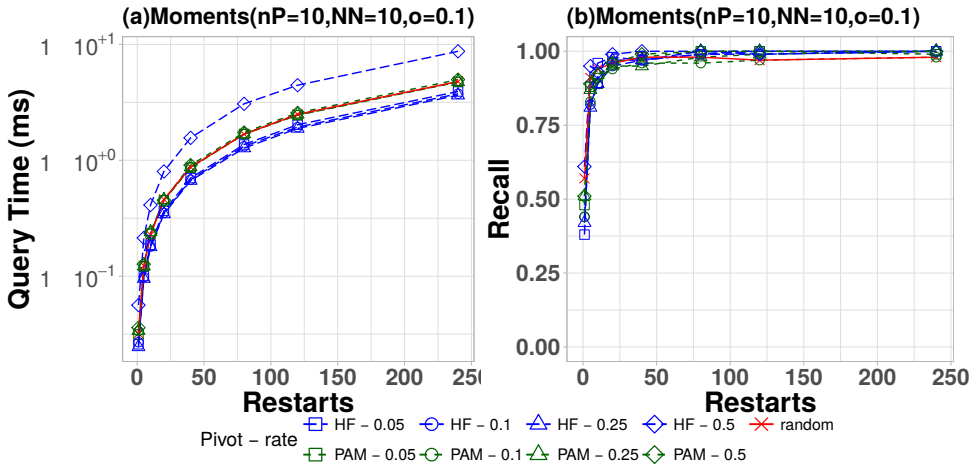

Figure 41 – Pivot Selection techniques according to Restarts for Color Moments dataset for $n_P = 10$ and $gtype_1$ =k-NNG, $NN = 10$.

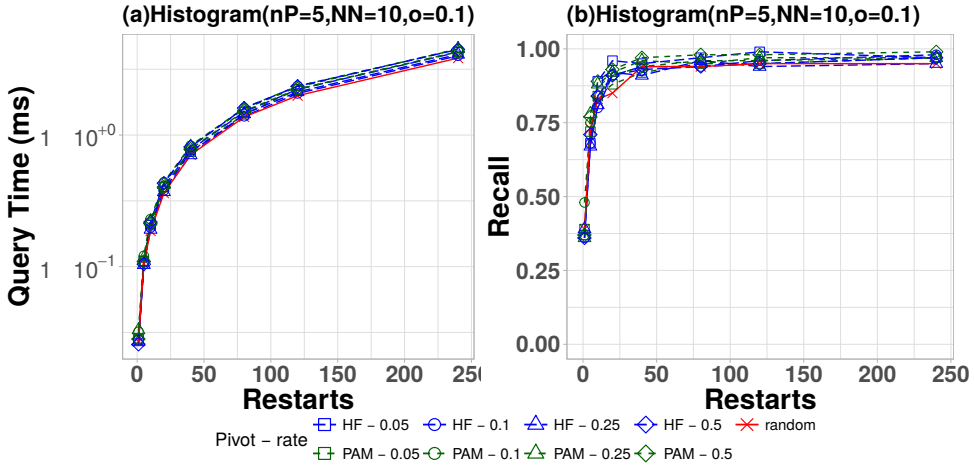

Figure 42 – Pivot Selection techniques according to Restarts for Color Histogram dataset for $n_P = 5$ and $gtype_1$ =k-NNG, $NN = 10$.

In conclusion, the HF algorithm was the best choice among the tested methods for search recall, however, the algorithm affects the graph construction time as the computational cost of this algorithm is higher than pivot random selection. The HF algorithm also takes a longer time to execute a search algorithm in the graph. On the other hand, the random pivot selection promotes the fastest graph construction and average performance regarding query time and recall. Therefore, for the experiments in the next sections, we will employ both the HF algorithm with 0.1 sample rate and the random selection of pivots.

5.2.4 Type of Graph

Another important parameter is the type of graph that *HGraph* will build, the parameter g_1. We have implemented mainly two graph types for g_1, the k-NNG and the NSW. We chose these two types of graphs because NSW is the current state of art and

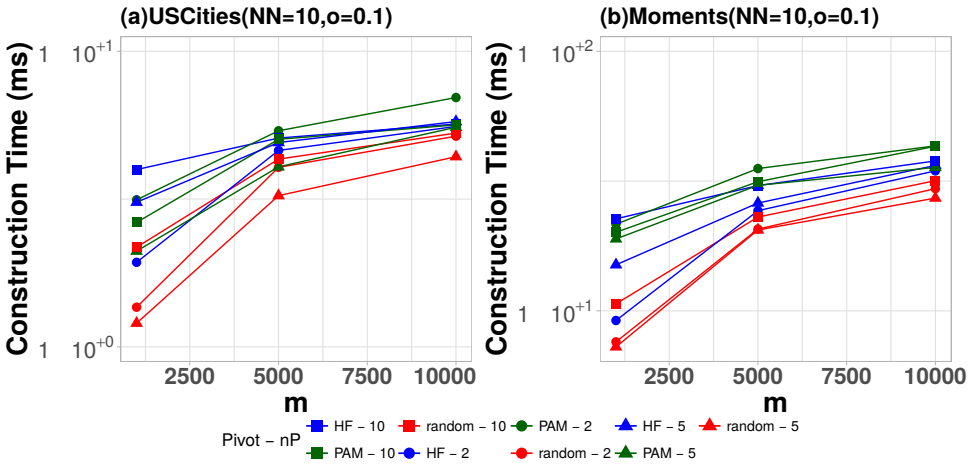

Figure 43 – Index Time according to parameter m, pivot selection algorithm and number of seeds n_P.

because *k-NNG* is one of the most used types of graphs in the literature along with its variations.

After the end of the recursive partition process, the chosen type of graph is built in each subset separately. Since the elements in the overlap area of each partitioning process can belong to more than one subset, to not add an excessive number of edges in these elements we need to control the number of edges of these vertices. The " control" is done according to the type of graph. For example, if it is a *k-NNG* the number of edges will have the fixed number *NN* while for the *RNG* each new edge to be inserted needs to follow the *RNG* neighborhood criterion. Reminding that this "control" on the number of edges is only done to vertices that are not pivots.

The number of edges per vertex in the *NSW* is not limited, every vertex has at least *NN* undirected edges. However, to avoid excessive edges in the *HGraph* we analysed in Figure 44 the average number of edges of a *NSW* according to its *NN* for different datasets. From Figure 44 we can easily observe that independently of the dataset the average number of edges per vertex is correlated to the *NN* parameter and is approximately two times the *NN* parameter. In conclusion, for the *HGraph-NSW* we limited the number of edges per vertex in $2*NN$.

Figures 45 and 46 show the performance of the *HGraph* using g_1 =*NSW* and *k-NNG* for $n_P = 5$ and $n_P = 10$, respectively, using the *GS* algorithm for 1-*NN* queries. From the figures we can observe that the *HGraph* drops significantly the construction time compared to *k-NNG*, however, it takes more time to construct when using g_1 =*NSW* when compared to *NSW*. The same pattern occurs in other datasets too. There are cases when using a dataset with a lower dimensionality than the MNIST dataset, such as the Color Histogram dataset, the *HGraph-NSW* (Figure 47) can take more time to build than the brute force *k-NNG*. This happens because it takes more time to partition the

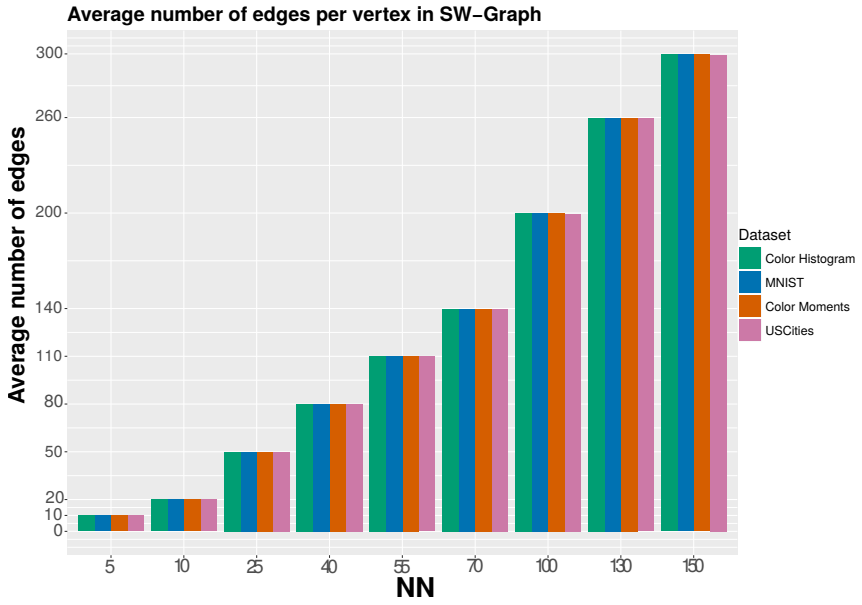

Figure 44 – Average number of edges per vertex in *NSW*.

dataset into n_P subsets using the generalized hyperplane partition strategy than using a greedy-based search algorithm to insert the vertices. Thus, after partitioning, the search for vertex insertion is executed to build a *NSW* in each resulting subset.

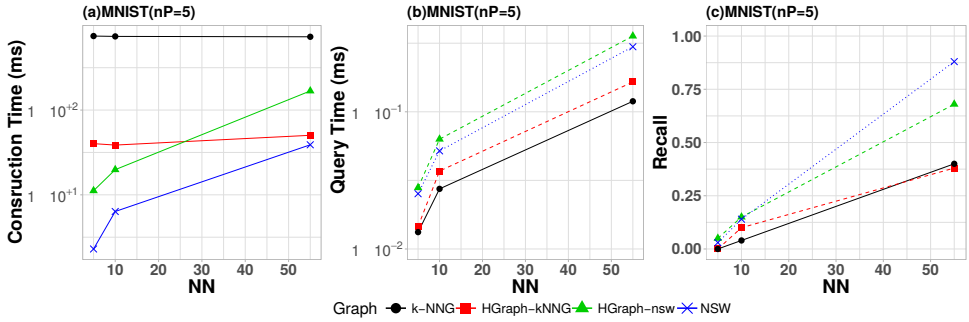

Figure 45 – Construction time, Query time and Recall for MNIST dataset compared to *NSW* and *k-NNG* for $n_P = 5$.

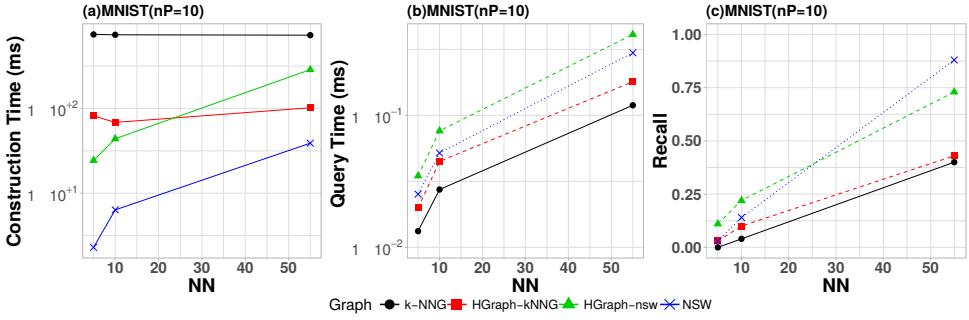

Figure 46 – Construction time, Query time and Recall for MNIST dataset compared to *NSW* and *k-NNG* for $n_P = 10$.

As for search performance, as expected the *HGraph- NSW* and the *NSW* were the slowest to execute the queries compared to the other methods due to the number of long-

range edges. In the MNIST dataset for $NN \leq 10$ the *HGraph- NSW* gives better recall than *NSW*, and as a trade-off, the query time is also bigger. Unexpectedly for $NN = 55$ the *NSW* performed better than *HGraph- NSW* in both query time and recall. Comparing the *HGraph- k-NNG* to the *k-NNG*, the *HGraph- k-NNG* did reduce the execution time and increased or gave approximate answer quality (recall), however, as already discussed the long-range edges increased the query time. For high dimensional dataset like MNIST, when the parameter *NN* increases from 5 to 55, that means approximately 10 times more, the construction time has a slight increase of 25% while in the Color Histogram the increase was of 200% for $n_P = 10$.

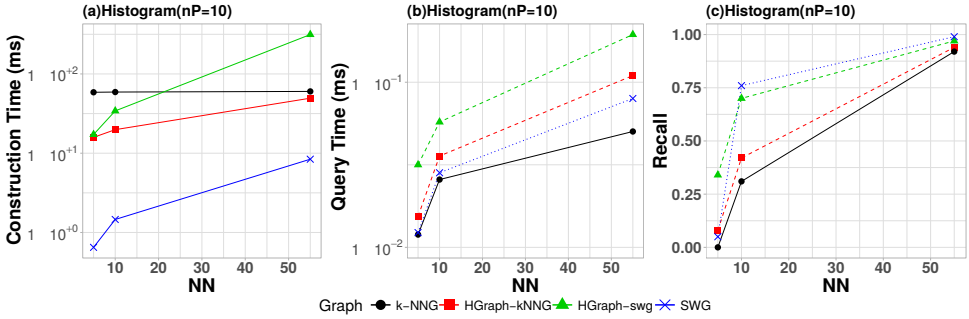

Figure 47 – Construction time, Query time and Recall for Color Histogram dataset compared to *NSW* and *k-NNG* for $n_P = 10$.

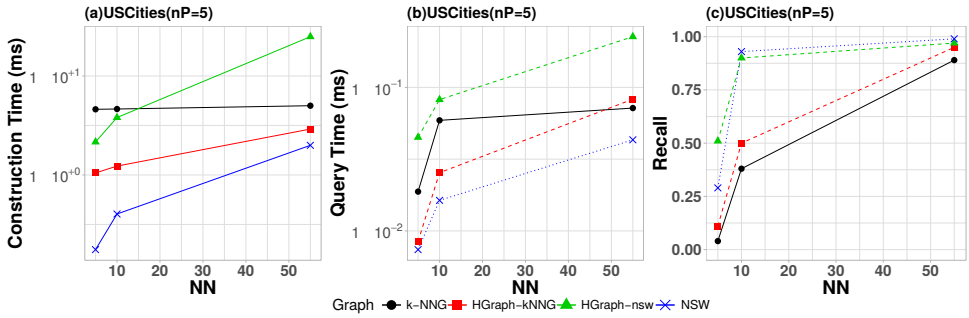

Figure 48 – Construction time, Query time and Recall for USCities dataset compared to *NSW* and *k-NNG* for $n_P = 5$.

In the USCities dataset presented Figure 48, a low dimensional and low cardinality dataset, we detected an unexpected pattern for the query time for *HGraph-k-NNG*. Even with the long-range edges the query execution took less time than the *k-NNG* and resulted in better recall rates.

Next, we compared the query time and recall according to the number of restarts for the *GNNS* algorithm for 1-*NN* queries. Through this comparison we can analyse the search performance of *HGraph* according to other parameters without limiting to the *GS* algorithm.

For better visualization, we compared the search performance of *HGraph* compared to each of its base graph g_1. Figure 49 shows the query time and recall of the

HGraph-k-NNG compared to the *k-NNG* according to the number of Restarts of the *GNNS* algorithm. We can observe from these results that the *HGraph* was able to increase the Recall in all datasets. In the MNIST dataset the results showed a trade-off between query time and recall as the query time also increased for *HGraph-k-NNG*, however, in the datasets with lower dimensionality, USCities and Texture, the *HGraph* was able to improve the query time as well.

Figure 50 shows the query time and recall of the *HGraph-NSW* compared to *NSW*. In this case, we can observe from the results a clear trade-off between query time and recall. In the MNIST dataset, the *HGraph-NSW* had the closest query time to the *NSW* and the largest recall increase. Unexpectedly, for bigger *NN* values ($NN \geq 10$) the *HGraph-NSW* did not increase the resulting recall or decrease the query time compared to the *NSW*.

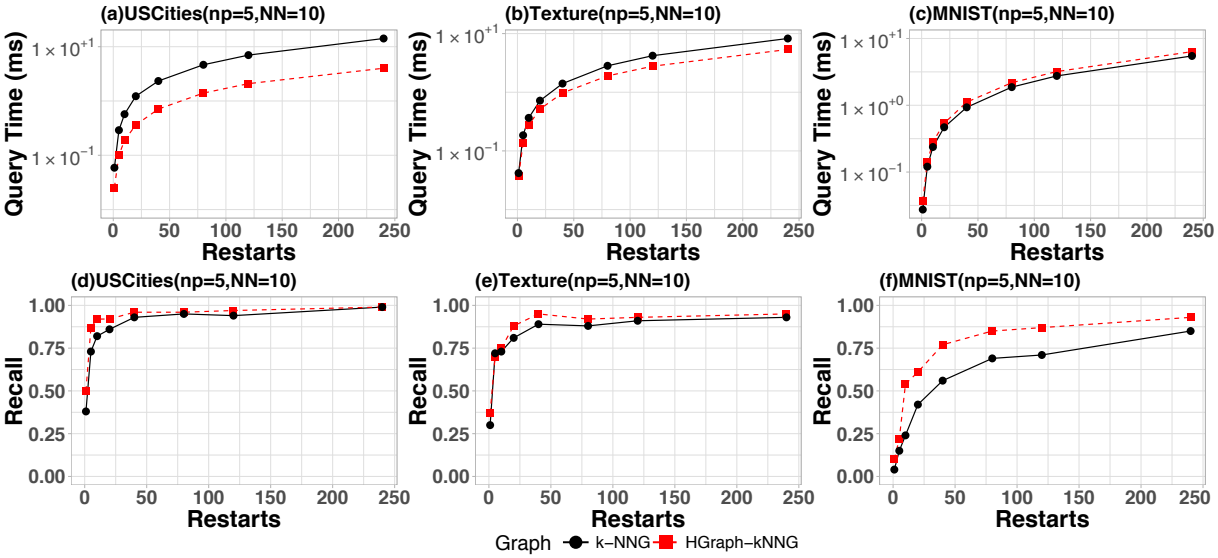

Figure 49 – Query time (log_{10} scale) and Recall for USCities, Texture and MNIST datasets comparing *HGraph-k-NNG* to *k-NNG* for $x = 5$,NN =10 and $m = 1000$.

Figures 51 and 52 show the query time and recall according to the number of restarts for the Color Histogram dataset when NN =5 and 1-NN queries using random pivot selection and the HF algorithm respectively. Even though the recall rate was quite similar using these two pivot selection algorithms, the main change was on the query time in this case. Using the HF algorithm the *HGraph-k-NNG* took longer to execute the search compared to random pivot selection in Figure 51.

In Figures 53 and 54 we show the search performance for the Color Moments dataset for the *GNNS* algorithm. We can observe in this dataset also the trade-off between query time and recall according to the number of restarts. In Figure 53 the *HGraph-k-NNG* had better results than *NSW*, with a smaller number of restarts, for instance when restarts= 40 *HGraph-NSW* achieved equal to approximate recall and query time as *NSW* with restarts= 240.

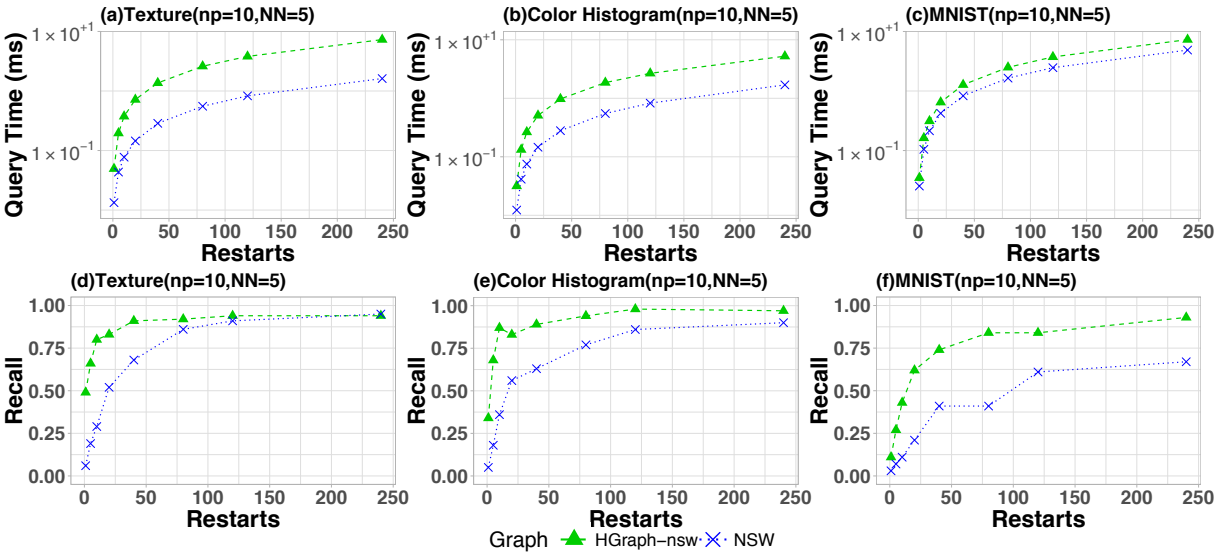

Figure 50 – Query time (log_{10} scale) and Recall for Color Histogram,MNIST and Texture datasets comparing *HGraph-NSW* to *NSW* for $x = 10$,NN =5 and $m = 1000$.

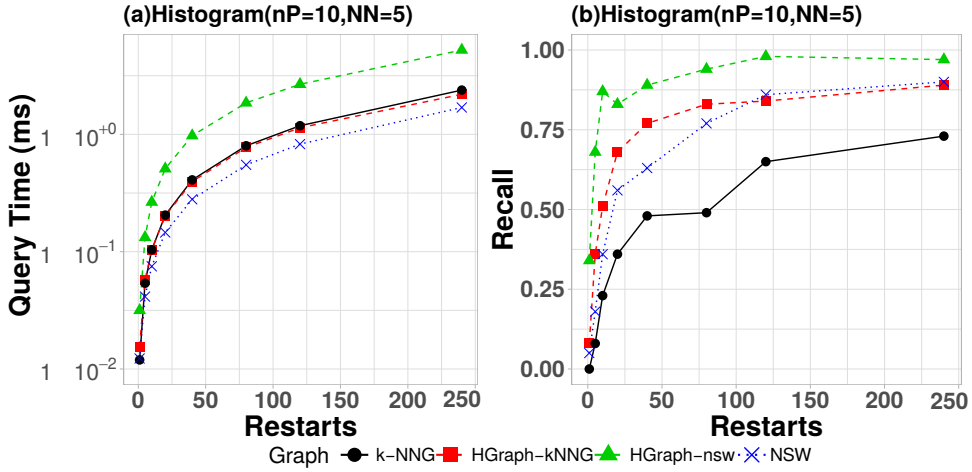

Figure 51 – $NN = 5$ and $n_P = 10$ for Color Histogram dataset using random pivot selection.

k-NNG had the worst performance compared to the other graphs. Comparing to *HGraph-k-NNG* the results confirm that the *HGraph* method increased the answer quality compared to its base graph-based method, the *k-NNG*. For $NN = 10$ the *NSW* performs better than most of the cases, however, if we equal the recall, in restarts= 80 the *HGraph-k-NNG* had lower query time for equal recall compared to the *NSW*. Even though the *HGraph-NSW* had the best recall for a lower number of restarts its construction time is a disadvantage. We can also observe that the query time can be too large when compared to its base graph, the *NSW*. On the other hand, the *HGraph-k-NNG* can produce good results without compromising the query time compared to other methods.

In Figure 55 we show how the query time and recall perform according to the *k-NN* search k parameter. As expected, as the k increases the query time increases and

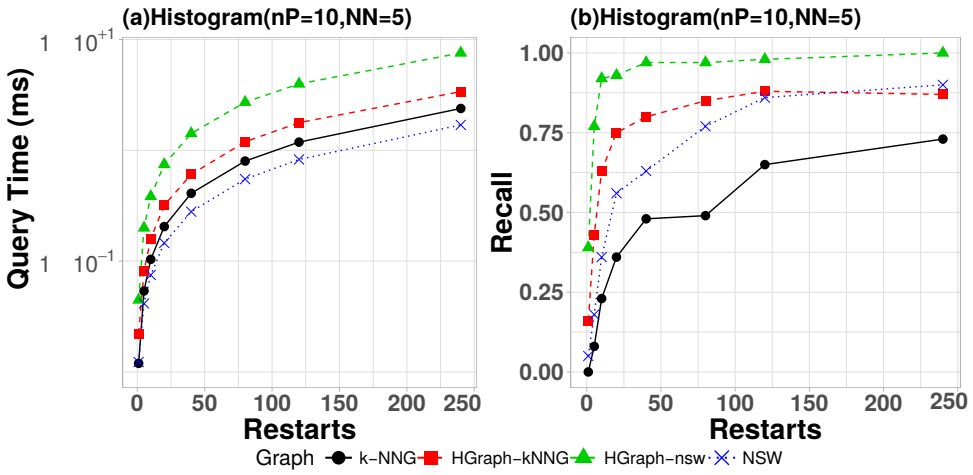

Figure 52 – $NN = 5$ and $n_P = 10$ for Color Histogram dataset using the HF algorithm for pivot selection.

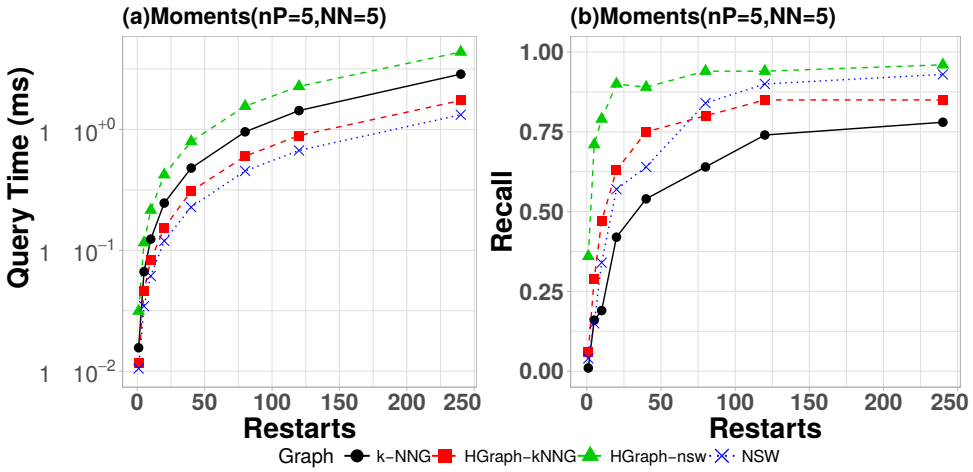

Figure 53 – Comparison to *k-NNG* and *NSW* for Moments dataset for the *HGraph* settings $NN = 5$ and $n_P = 5$.

the recall decreases. An interesting point is that the *NSW* recall decreases more than the *HGraph- NSW*.

According to these results, we did an analysis similar to the one that was done in Chapter 4. We selected top-30 best graph settings in query time that resulted in exact answers (Recall= 1) and analysed the query time and the number of distance computations for 1-*NN* queries. We also compared the search results to the *SAT*. The graph settings were ordered according to the query time. Appendix A shows the bar plots presented in this section in landscape mode for better visualization of the results. Figures 56 shows the query time while figure 57 shows the correspondent number of distance computations.

Comparing the number of distance and the query time we can observe that for the methods *NSW* and *HGraph-NSW* both measures are not completely related. From this

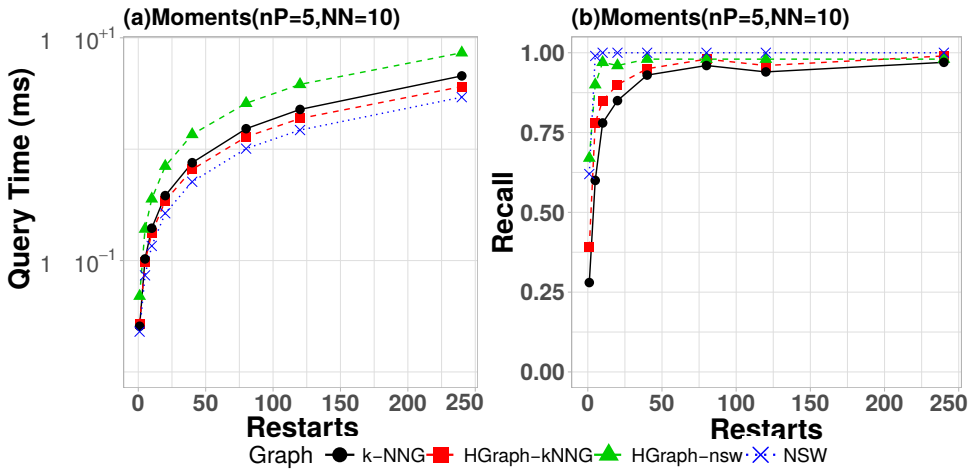

Figure 54 – Comparison to *k-NNG* and *NSW* for Moments dataset for the *HGraph* settings $NN = 10$ and $n_P = 5$.

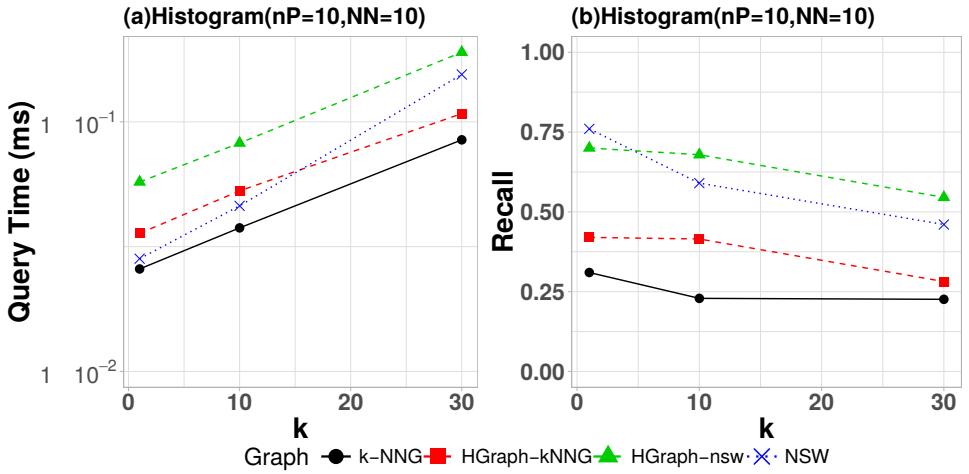

Figure 55 – Query Time and Recall according to number of k in *k-NN* queries.

experiment we can also observe that some settings of the graph-based methods resulted in less query time and distance computations compared to the *SAT*, proving that graph-based methods that use spatial approximation can be more efficient for precise queries than a tree-based method that uses spatial approximation.

To achieve the exact answer the best configurations were with $NN = 55$ and a small number of restarts. For the Color Histogram dataset, the *HGraph- k-NNG* had the best performance in both query time and distance computations. Followed by the *NSW* and the *k-NNG*. This reinforces what has been shown in previous results, the *HGraph* method improves the *k-NNG* in construction time and in query time.

However, the *NSW* can also outperform the *HGraph* in some cases. For example, in Figure 58 we selected for the same recall = 1 the top-30 graph settings and search parameters. The *NSW* showed the best performance, followed by the *HGraph-k-NNG* and the *HGraph-NSW*.

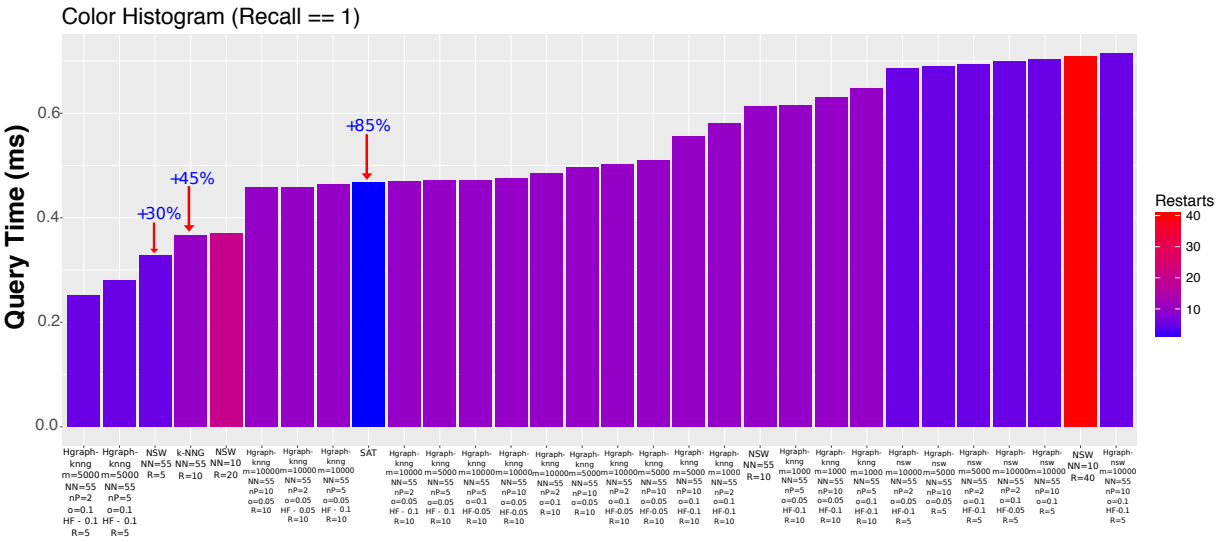

Figure 56 – Query time for Top-30 graph settings (recall = 1) for 1-*NN* queries – Color Histogram dataset.

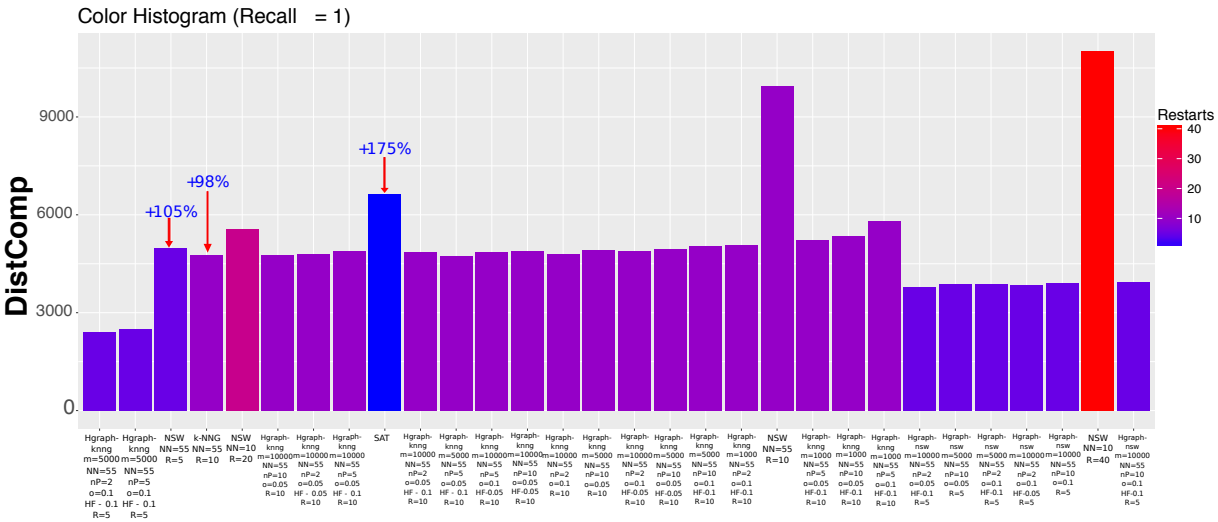

Figure 57 – Distance Computations for Top-30 graph settings (recall = 1) for 1-*NN* queries – Color Histogram dataset.

Next, we evaluated the best graph configurations in query time for Recall >= 0.9. Figures 59, 60 and 61 shows the results.

In Figures 59 and 60 the *NN* parameter had the most impact in the results, all selected graph settings $NN = 55$. Thus, except for the *NSW* $NN = 10$ configuration in Figure 59 all the presented methods has the same *NN* and number of restarts. In this case, since the long-range edges increase the query time, the fastest method was *k-NNG* for Color Histogram, Texture and Moments, however, followed by the *HGraph-k-NNG*. In these datasets, the big value for *NN* was enough to achieve the recall of 0.9 in just one restart, except for *NSW* $NN = 10$. An important point is that in Figure 59 the *HGraph-k-NNG* method was able to outperform the *NSW* in several different graph settings.

When analysing the MNIST dataset (Figure 61) for Recall ≥ 0.9 the *HGraph-k-*

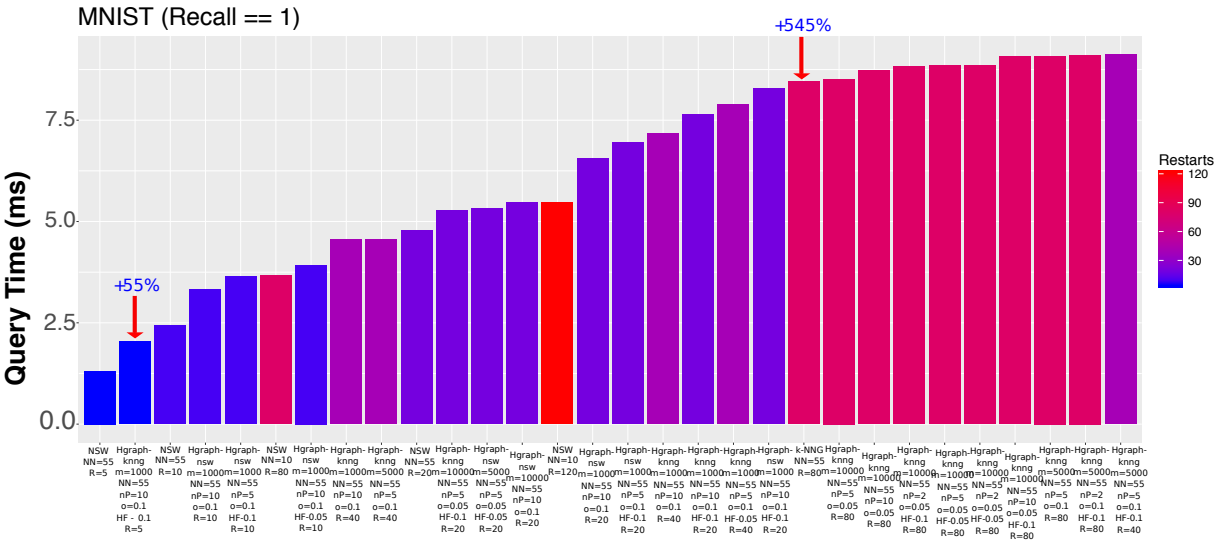

Figure 58 – Query Time for Top-30 graph settings (recall = 1) for 1-*NN* queries – MNIST dataset.

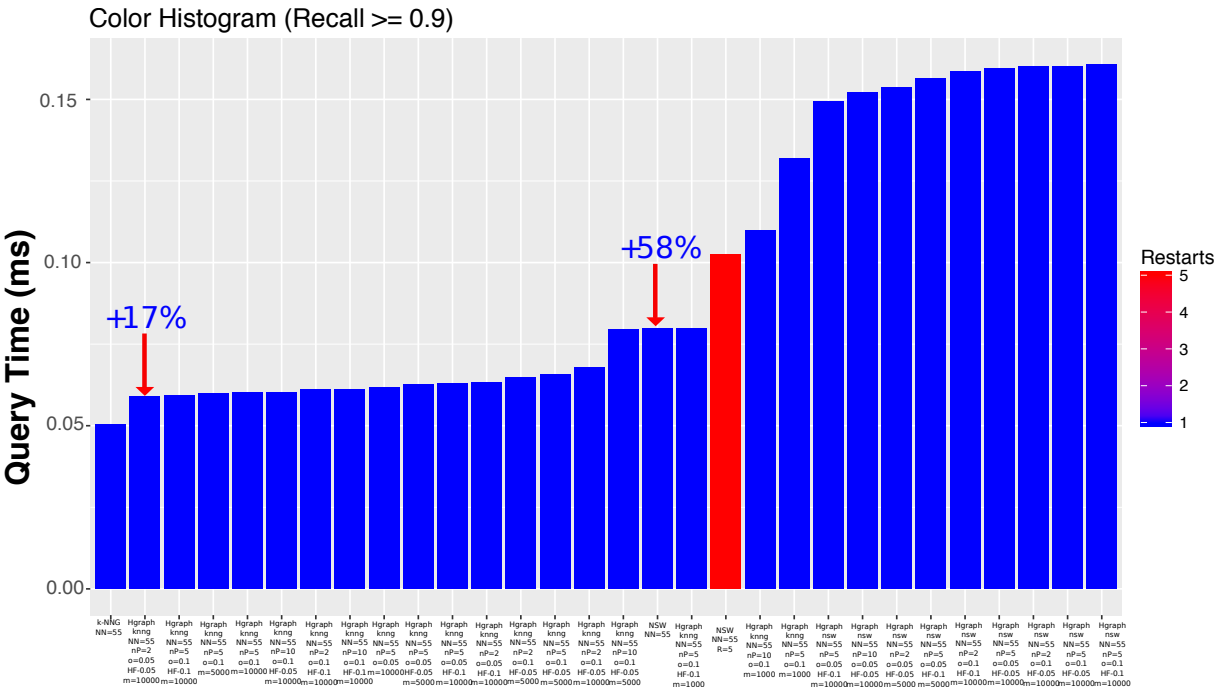

Figure 59 – Query Time for Top-30 graph settings (recall $\geq$ 0.9) for 1-*NN* queries – Color Histogram dataset.

NNG resulted in the lowest query time even when using the same *NN* and number of restarts for the *GNNS* algorithm. In the MNIST dataset, the *NSW* configuration is the sixth best query time while the top-5 are *HGraph-k-NNG* configurations.

5.3 Final Remarks

As a second contribution of this thesis, we proposed the *HGraph* method, a connected partition approach for similarity searches.

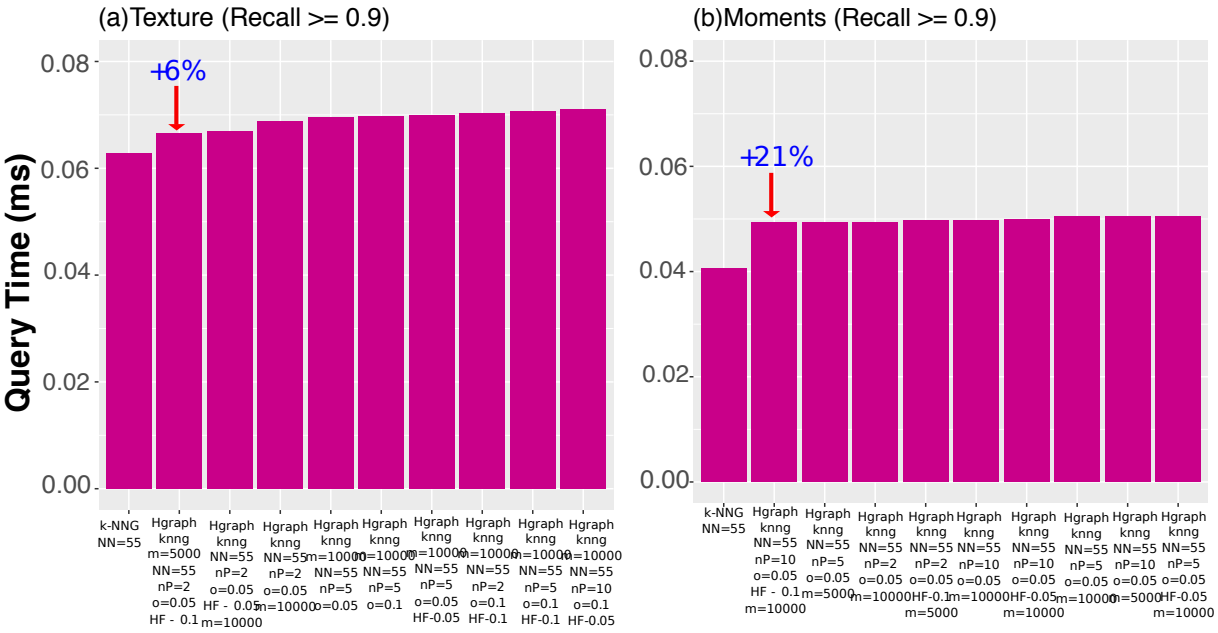

Figure 60 – Query Time for Top-10 graph settings (recall $\geq$ 0.9) for 1-*NN* queries – Texture and Moments dataset. The number of restarts for all of the selected graph settings in this Figure is 1.

We evaluated the behavior of the main parameters of the *HGraph* and from the results, we can suggest default values for some of the *HGraph* parameters. We showed through the experiments that the long-range edges do increase the recall in approximate searches in graph-based methods. In some specific datasets, the use of the long-range edges could increase the recall up to 70% compared to the *k-NNG* (see Figure 31(e)). However, the query time also increases when setting the same search parameters.

We also showed how the *HGraph-k-NNG* construction algorithms strategy can build a graph that is approximate to the *k-NNG* according to the overlap size. We concluded from the overlap size analysis that values for overlap (parameter o) that are bigger than 0.1 take a long time to build the *HGraph*, in some cases more time than the quadratic *k-NNG* brute force construction. The default values suggested for parameter o are $0.05, 0.1$.

We analysed three different pivot selection algorithms in the literature: the random selection, the Hull of Foci algorithm and the k-Medoids. According to the results, the algorithm that gave the best answer quality was the HF algorithm, however, the random pivots followed closely. The random pivot selection was the best in terms of construction time. The k-Medoids algorithm takes a long time to execute and did not bring any improvements compared to the other two algorithms. As a conclusion, we suggest the use of the HF algorithm or the random selection for parameter *ptype*.

The *HGraph* construction time increases as the *NN* parameters increases, thus, the construction time for *HGraph-NSW* can be very expensive when compared to the *NSW*.

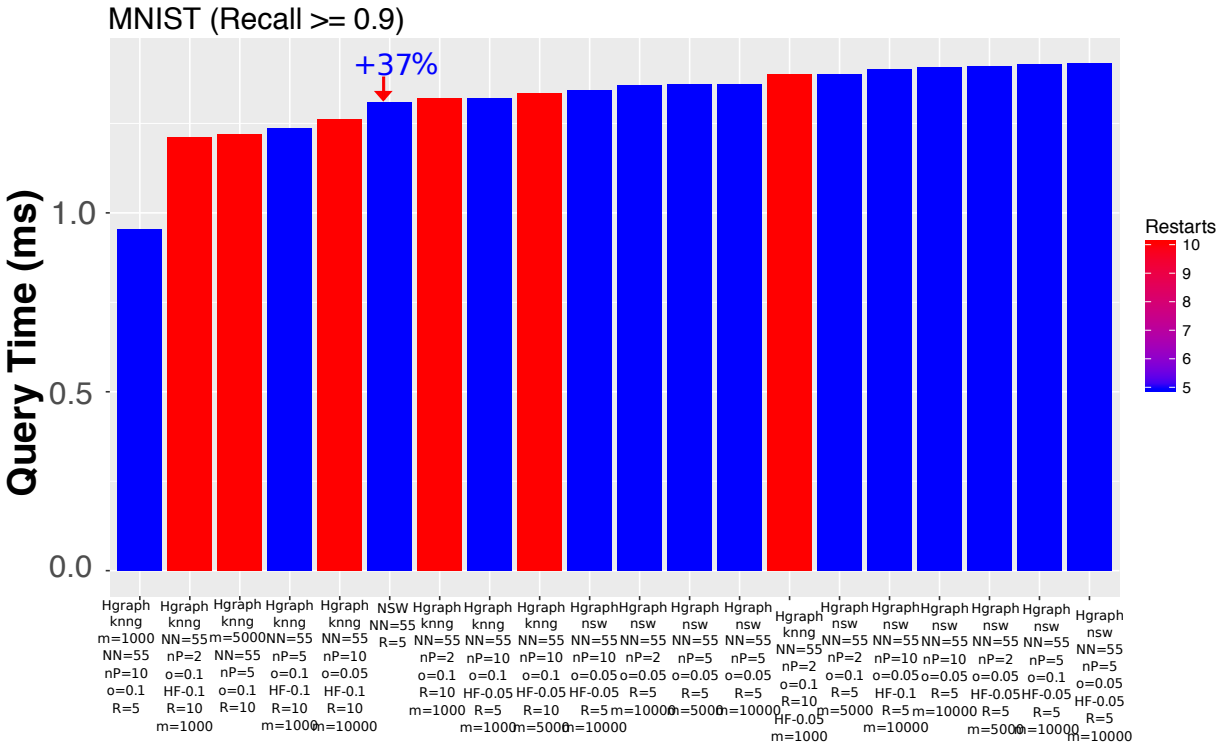

Figure 61 – Query Time for Top-20 graph settings (recall $\geq$ 0.9) for 1-*NN* queries – MNIST dataset.

That happens because the partition process takes more time to execute than the insertion algorithm proposed by the authors. In order to use $g_1 = NSW$ and have advantages over the proposed *NSW* in the literature, we would need to change the partition strategy from the generalized hyperplane partition to a faster strategy.

At last, we compared the *HGraph* performance to the types of graphs used as g_1 and to the *SAT*, a tree-based method that uses spatial approximation for queries. To achieve exact answer the *NN* parameter had the most impact in all of the datasets for both analyses recall $\geq$ 0.9 and recall = 1. For recall $\geq$ 0.9 for the same *NN* parameters and number of restarts, the *k-NNG* performed better in Color Histogram, Texture and Moments dataset. However, for the MNIST dataset, the *HGraph-k-NNG* had the best query time for recall $\geq$ 0.9 (approximate result). We showed that the *HGraph-k-NNG* was able to perform the best in terms of query time for the Color Histogram while for the MNIST dataset the *NSW* was the best for an exact answer (recall = 1). Nevertheless, followed by the *HGraph-k-NNG*. We also showed through these experiments that the graph-based methods *NSW* and *HGraph* can outperform the *SAT*.

6 CONCLUSION

The technology development has accelerated the growth of the volume of complex data, such as images, videos, time series, and georeferenced data. As the volume of complex data grows, also grows the necessity of storing and retrieving this kind of data efficiently to support modern applications [1]. A widely used approach to retrieve complex data are the similarity searches.

In order to improve the retrieval of complex data using similarity searches, it is necessary to organize large collections of data in a way that similar data can be retrieved in the shortest time as possible. Most of the traditional Database Management Systems (DBMS) includes indexing methods such as the B-tree and hashing methods, however, these methods are not suited for complex data similarity retrieval. As a result, new indexing structures were proposed in the literature [4]. Some of these methods are: Slim-tree [10], CM-tree [11], Locality Sensitive Hashing (LSH) [12] and *proximity graphs* [13, 9].

Graph-based methods have emerged as a very efficient alternative for similarity retrieval, with reports indicating they have outperformed methods of other categories in several situations. However, to the best of our knowledge, there is no previous work with experimental analysis on a comprehensive number of graph-based methods using the same search algorithm and execution environment. In this context, this thesis presented two main contributions: (1) A performance analysis of the main graph-based methods in the literature for both exact and approximate similarity searches in metric spaces and; (2)HGraph, a connected-partition approach method to build proposed types of graphs in the literature to answer similarity searches.

We presented a survey on graph-based methods categorizing the methods by its type of graph and presented the search algorithms used for exact and approximate searches on metric spaces. We also discussed the applicability of the spatial approximation property in the different types of graphs. In this discussion, we showed counter-examples on how the spatial approximation property does not hold for some of the main types of graphs used for similarity searches. We also approached on how the exact search algorithm proposed by Paredes and Chavez[9] implementations does not give exact answers in a weighted *RNG* compared to *k-NNG*.

Next, we did an experimental analysis of the performance behavior of the tested graph-based methods with respect to the main construction and query parameters for a variety of real-world datasets. Our experimental results provided a quantitative view of the exact search compared to accurate setups for approximate search. The experimental

evaluation results showed that *NSW* outperforms other methods in construction time. However, as the number of neighbors per vertex in *NSW* increases, for the same *NN* value, the query time increase as well compared to *k-NNG* and *NN-Descent*.

For *k-NNG*, *NN-Descent* and *NSW* methods, we found that there is a trade-off between construction and query time. The more edges the best query time, but the highest construction time. Thus, as the number of neighbors per vertex increases, the query time and the number of restarts needed to return query answers close to exact answers decreases. The *NSW* connects different graph regions by using the long-range edges and achieves the highest recall rates among the tested methods even for a small *NN*. The drawback is that the excessive number of edges compared to the other graph-based methods slows down the query execution. When comparing graph settings for a given recall rate, we were not able to point out a winner method for every condition tested. We also showed that the methods can achieve the same recall rate in approximate query time depending on the parameter settings. In order to understand better the graph-based methods according to the dataset distribution we plan to run experiments using synthetic datasets as future works.

Considering the results of the experimental analysis of the performance behavior of the tested graph-based methods we proposed the *HGraph* method. The *HGraph* is a connected partition approach to build proposed graph types in the literature. The *HGraph* uses a divide and conquer approach to speed up the graph construction and adds edges to selected vertices (long-range edges) in order to increase approximate similarity search recall.

We presented an experimental evaluation of the *HGraph* main parameters. Through this evaluation we concluded: (1) Long-range edges increased the answer quality compared to the "base" graph type; (2) The *HGraph-k-NNG* algorithm can build a graph that is approximate to the *k-NNG* with a small overlap rate of 0.1; (3) The random pivot selection algorithm and the HF algorithm had the best performance while the k-Medoids algorithm did not show advantages over the other two methods.

At last, we compared the *HGraph-k-NNG* and the *HGraph-NSW* to the *k-NNG*, *NSW* and *SAT*. The *HGraph-k-NNG* was able to improve the *k-NNG* method in construction time, query time and recall. Thus, we showed cases in which the *HGraph-k-NNG* and the *HGraph-NSW* were able to outperform *NSW* and *SAT* when Recall= 1.

As future works, we intend to run experiments of the *HGraph* method using other graph types such as the *RNG* as g_1 and test a faster dataset partition strategy in order to improve the construction time for *HGraph- NSW*. The divide and conquer construction strategy used in the *HGraph* method builds the graph using independent subsets, in consequence, it is possible to run the methods in parallel. Future works include the implementation of the *HGraph* in an architecture that supports running the *HGraph* cons-

truction in parallel. We believe that the parallel implementation can support similarity searches using the *HGraph* method in bigger scale datasets.

REFERENCES

[1] CIACCIA, P.; PATELLA, M.; ZEZULA, P. M-Tree: An Efficient Access Method for Similarity Search in Metric Spaces. In: *23rd International Conference on Very Large Data Bases (VLDB)*. Athen, Greece: [s.n.], 1997. p. 426–435.

[2] BARIONI, M. C. N. et al. *Advanced Database Query Systems*. [S.l.]: IGI Global, 2011. 323–359 p. ISBN 9781609604752.

[3] CHáVEZ, E. et al. Searching in metric spaces. *ACM Comput. Surv.*, v. 33, n. 3, p. 273–321, 2001. ISSN 0360-0300.

[4] HJALTASON, G. R.; SAMET, H. Index-driven similarity search in metric spaces. *ACM Trans. Database Syst.*, v. 28, n. 4, p. 517–580, 2003. ISSN 0362-5915.

[5] JOUILI, S.; TABBONE, S. Hypergraph-based image retrieval for graph-based representation. *Pattern Recognition*, v. 45, n. 11, p. 4054–4068, 2012.

[6] TSOCHATZIDIS, L. et al. Computer-aided diagnosis of mammographic masses based on a supervised content-based image retrieval approach. *Pattern Recognition*, v. 71, p. 106 – 117, 2017. ISSN 0031-3203. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0031320317302194>.

[7] SKOPAL, T.; BUSTOS, B. On nonmetric similarity search problems in complex domains. *ACM Comput. Surv.*, v. 43, n. 4, p. 1–50, 2011.

[8] ZEZULA, P. et al. *Similarity Search: The Metric Space Approach*. 1st. ed. [S.l.]: Springer, 2010. ISBN 1441939725, 9781441939722.

[9] PAREDES, R.; CHÁVEZ, E. Using the k-nearest neighbor graph for proximity searching in metric spaces. In: _____. *String Processing and Information Retrieval SPIRE*. [S.l.]: Springer Berlin Heidelberg, 2005. p. 127–138.

[10] TRAINA, C. et al. Slim-trees: High performance metric trees minimizing overlap between nodes. In: _____. *Advances in Database Technology — EDBT 2000: 7th Int'l Conf. on Extending Database Technology Konstanz, Germany, March 27–31, 2000 Proc.* [S.l.]: Springer Berlin Heidelberg, 2000. p. 51–65. ISBN 978-3-540-46439-6.

[11] VIEIRA, M. R. et al. Dbm-tree: A dynamic metric access method sensitive to local density data. In: CITESEER. *In SBBD*. [S.l.], 2004.

[12] INDYK, P.; MOTWANI, R. Approximate nearest neighbors: Towards removing the curse of dimensionality. In: *Proc. of the ACM Symp. on Theory of Computing*. [S.l.: s.n.], 1998. p. 604–613.

[13] OCSA, A.; BEDREGAL, C.; CUADROS-VARGAS, E. A new approach for similarity queries using neighborhood graphs. In: *Brazilian Symp. on Databases*. [S.l.: s.n.], 2007. p. 131–142.

[14] LEE, J. K. H. Efficient recommender system based on graph data for multimedia application. *Int'l Journal of Multimedia and Ubiquitous Engineering*, v. 8, n. 4, 2013.

[15] NAIDAN, B.; BOYTSOV, L.; NYBERG, E. Permutation search methods are efficient, yet faster search is possible. *Proc. VLDB Endow.*, VLDB Endowment, v. 8, n. 12, p. 1618–1629, 2015. ISSN 2150-8097.

[16] MALKOV, Y. et al. Approximate nearest neighbor algorithm based on navigable small world graphs. *Inf. Syst.*, v. 45, p. 61–68, 2014. ISSN 0306-4379.

[17] HAJEBI, K. et al. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In: *Int'l Joint Conf. on Artificial Intelligence IJCAI*. [S.l.: s.n.], 2011. p. 1312–1317.

[18] BULò, S. R.; RABBI, M.; PELILLO, M. Content-based image retrieval with relevance feedback using random walks. *Pattern Recognition*, v. 44, n. 9, p. 2109 – 2122, 2011. ISSN 0031-3203. Computer Analysis of Images and Patterns.

[19] DONG, W.; MOSES, C.; LI, K. Efficient k-nearest neighbor graph construction for generic similarity measures. In: *Proc. of the WWW*. [S.l.: s.n.], 2011. p. 577. ISBN 9781450306324.

[20] YAN, X. et al. Feature-based similarity search in graph structures. *ACM Trans. Database Syst.*, v. 31, n. 4, p. 1418–1453, 2006. ISSN 0362-5915.

[21] YAN, X.; YU, P. S.; HAN, J. Graph indexing: A frequent structure-based approach. In: *Proc. of the ACM SIGMOD*. [S.l.: s.n.], 2004. p. 335–346.

[22] GIUGNO, R.; SHASHA, D. Graphgrep: A fast and universal method for querying graphs. In: *Object recognition supported by user interaction for service robots*. [S.l.: s.n.], 2002. v. 2, p. 112–115.

[23] HJALTASON, G. R.; SAMET, H. Index-driven similarity search in metric spaces. *ACM Trans. Database Syst.*, v. 28, n. 4, p. 517–580, 2003. ISSN 0362-5915.

[24] GIONIS, A.; INDYK, P.; MOTWANI, R. Similarity search in high dimensions via hashing. In: *Proceedings of the 25th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. (VLDB '99), p. 518–529. ISBN 1-55860-615-7. Disponível em: <http://dl.acm.org/citation.cfm?id=645925.671516>.

[25] HAN, F. et al. Texture feature analysis for computer-aided diagnosis on pulmonary nodules. *Journal of Digital Imaging*, v. 28, n. 1, p. 99–115, Feb 2015. ISSN 1618-727X. Disponível em: <https://doi.org/10.1007/s10278-014-9718-8>.

[26] DEZA, M. M.; DEZA, E. *Encyclopedia of Distances*. [S.l.: s.n.], 2009. 350 p. ISBN 9783642002335.

[27] WEINSHALL, D.; JACOBS, D. W.; GDALYAHU, Y. Classification in non-metric spaces. In: *Proceedings of the 11th International Conference on Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 1998. (NIPS'98), p. 838–844. Disponível em: <http://dl.acm.org/citation.cfm?id=3009055.3009172>.

[28] BöHM, C.; BERCHTOLD, S.; KEIM, D. A. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 33, n. 3, p. 322–373, set. 2001. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/502807.502809>.

[29] BARIONI, M. C. N. *Operações de consulta por similaridade em grandes bases de dados complexos*. Tese (Doutorado) — Universidade de São Paulo, São Carlos, SP, 2006.

[30] ARANTES, A. S. et al. Efficient algorithms to execute complex similarity queries in RDBMS. *Journal of the Brazilian Computer Society*, scielo, v. 9, p. 5 – 24, 04 2004. ISSN 0104-6500. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0104-65002004000100002&nrm=iso>.

[31] SILVA, Y. N.; AREF, W. G.; ALI, M. H. The similarity join database operator. *Proc. of the IEEE ICDE*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 892–903, 2010.

[32] SILVA, Y. N.; AREF, W. G.; ALI, M. H. Similarity group-by. In: *2009 IEEE 25th International Conference on Data Engineering*. [S.l.: s.n.], 2009. p. 904–915. ISSN 1063-6382.

[33] YIANILOS, P. N. Data structures and algorithms for nearest neighbor search in general metric spaces. In: *SODA*. [S.l.: s.n.], 1993. v. 93, n. 194, p. 311–321.

[34] UHLMANN, J. K. Satisfying general proximity/similarity queries with metric trees. *Information processing letters*, Elsevier, v. 40, n. 4, p. 175–179, 1991.

[35] AMATO, G.; ESULI, A.; FALCHI, F. A comparison of pivot selection techniques for permutation-based indexing. *Inf. Syst.*, Elsevier Science Ltd., Oxford, UK, UK, v. 52, n. C, p. 176–188, ago. 2015. ISSN 0306-4379. Disponível em: <http://dx.doi.org/10.1016/j.is.2015.01.010>.

[36] AMATO, G.; GENNARO, C.; SAVINO, P. Mi-file: using inverted files for scalable approximate similarity search. *Multimedia Tools and Applications*, v. 71, n. 3, p. 1333–1362, Aug 2014. ISSN 1573-7721. Disponível em: <https://doi.org/10.1007/s11042-012-1271-1>.

[37] ESULI, A. Use of permutation prefixes for efficient and scalable approximate similarity search. *Information Processing & Management*, v. 48, n. 5, p. 889 – 902, 2012. ISSN 0306-4573. Large-Scale and Distributed Systems for Information Retrieval. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306457310001019>.

[38] WANG, J. et al. Hashing for similarity search: A survey. *CoRR*, abs/1408.2927, 2014. Disponível em: <http://arxiv.org/abs/1408.2927>.

[39] AOYAMA, K. et al. Fast Approximate Similarity Search Based on Degree-Reduced Neighborhood Graphs Categories and Subject Descriptors. In: *ACM SIGKDD*. [S.l.: s.n.], 2011. p. 1055–1063. ISBN 9781450308137.

[40] HARWOOD, B.; DRUMMOND, T. Fanng: Fast approximate nearest neighbour graphs. In: *Proc. of the IEEE CVPR*. [S.l.: s.n.], 2016. p. 5713–5722.

[41] FLOREZ, O. U.; LIM, S. HRG: A Graph Structure for Fast Similarity Search in Metric Spaces. In: _____. *Database and Expert Systems Applications: 19th Int'l Conf., DEXA 2008, Turin, Italy, September 1-5, 2008. Proc.* [S.l.]: Springer Berlin Heidelberg, 2008. p. 57–64. ISBN 978-3-540-85654-2.

[42] NAVARRO, G. Searching in metric spaces by spatial approximation. *The VLDB Journal The Int'l Journal on Very Large Data Bases*, Springer-Verlag, v. 11, n. 1, p. 28–46, 2002. ISSN 10668888.

[43] FORTUNE, S. Voronoi diagrams and delaunay triangulations. *Computing in Euclidean geometry*, v. 1, n. 193-233, p. 2, 1992.

[44] AURENHAMMER, F. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, v. 23, n. 3, p. 345–405, 1991. ISSN 0360-0300.

[45] AGGARWAL, A. et al. A linear-time algorithm for computing the voronoi diagram of a convex polygon. *Discrete & Computational Geometry*, v. 4, n. 6, p. 591–604, 1989. ISSN 1432-0444.

[46] OKABE, A.; BOOTS, B.; SUGIHARA, K. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. [S.l.]: John Wiley & Sons, Inc., 1992. ISBN 0-471-93430-5.

[47] B, B. C.; DOBKIN, D. P.; HUHDANPAA, H. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, v. 22, n. 4, p. 469–483, 1996. ISSN 0098-3500.

[48] LAWSON, C. L. Properties of n-dimensional triangulations. *Computer Aided Geometric Design*, v. 3, n. 4, p. 231 – 246, 1986. ISSN 0167-8396.

[49] WATSON, D. F. Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *The Computer J.*, v. 24, n. 2, p. 167–172, 1981.

[50] BOWYER, A. Computing dirichlet tessellations. *The Computer J.*, v. 24, n. 2, p. 162–166, 1981.

[51] TOUSSAINT, G. Proximity graphs for nearest neighbor decision rules: Recent progress. In: *Proc. of the Symp. on the INTERFACE*. [S.l.: s.n.], 2002. p. 17–20.

[52] CHáVEZ, E. et al. Faster proximity searching with the distal sat. *Information Systems*, v. 59, p. 15 – 47, 2016. ISSN 0306-4379. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306437916000028>.

[53] NAVARRO, G.; REYES, N. Dynamic spatial approximation trees. *J. Exp. Algorithmics*, ACM, New York, NY, USA, v. 12, p. 1.5:1–1.5:68, jun. 2008. ISSN 1084-6654. Disponível em: <http://doi.acm.org/10.1145/1227161.1322337>.

[54] NAVARRO, G.; REYES, N. Dynamic spatial approximation trees for massive data. In: *2009 Second International Workshop on Similarity Search and Applications*. [S.l.: s.n.], 2009. p. 81–88.

[55] BOSE, P. et al. Some properties of k-delaunay and k-gabriel graphs. *Computational Geometry*, v. 46, n. 2, p. 131 – 139, 2013. ISSN 0925-7721. Canadian Conference on Computational Geometry, Winnipeg, Aug. 9-11.2010. Disponível em: <http://www.sciencedirect.com/science/article/pii/S092577211200079X>.

[56] ABELLANAS, M. et al. On properties of higher-order delaunay graphs with applications. In: . [S.l.: s.n.], 2005. p. 119 – 122. EuroCG.

[57] BOSE, P. et al. Proximity graphs: E, δ, Δ, χ AND ω. *Int. J. Comput. Geometry Appl.*, v. 22, n. 5, p. 439–470, 2012.

[58] PATELLA, M.; CIACCIA, P. Approximate similarity search: A multi-faceted problem. *Journal of Discrete Algorithms*, v. 7, n. 1, p. 36 – 48, 2009. ISSN 1570-8667. Selected papers from the 1st International Workshop on Similarity Search and Applications (SISAP). Disponível em: <http://www.sciencedirect.com/science/article/pii/S1570866708000762>.

[59] ARYA, S. et al. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, v. 45, n. 6, p. 891–923, 1998. ISSN 0004-5411.

[60] AOYAMA, K. et al. Fast Similarity Search in Small-World Networks. In: . [S.l.]: Springer Berlin Heidelberg, 2009. p. 185–196.

[61] BEIS, J. S.; LOWE, D. G. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In: *Proc. of the IEEE CVPR*. [S.l.: s.n.], 1997. p. 1000–1006.

[62] WANG, J.; LI, S. Query-driven iterated neighborhood graph search for large scale indexing. In: *Proc. of ACM Int'l Conf. on Multimedia*. [S.l.: s.n.], 2012. p. 179–188. ISBN 978-1-4503-1089-5.

[63] AOYAMA, K. et al. Graph index based query-by-example search on a large speech data set. In: *Proc. of the IEEE ICASSP*. [S.l.]: IEEE, 2013. p. 8520–8524. ISBN 978-1-4799-0356-6. ISSN 1520-6149.

[64] PAREDES, R. et al. Practical construction of k-nearest neighbor graphs in metric spaces. In: _____. *Experimental Algorithms: 5th International Workshop, WEA 2006, Cala Galdana, Menorca, Spain, May 24-27, 2006. Proc.* [S.l.]: Springer Berlin Heidelberg, 2006. p. 85–97. ISBN 978-3-540-34598-5.

[65] CHEN, J.; FANGAND, H.-R.; SAAD, Y. Fast approximate knn graph construction for high dimensional data via recursive lanczos bisection. *Journal of Machine Learning Research*, v. 10, p. 1989–2012, 2009.

[66] WANG, J. et al. Scalable k-nn graph construction for visual descriptors. In: *Proc. of the IEEE CVPR*. [S.l.: s.n.], 2012. p. 1106–1113.

[67] LANCZOS, C. r an iteration method for the solution of the eigenvalue i problem. of linear differential and integral operatorsl. *Journal of Research of the National Bureau of Standards*, Citeseer, v. 45, n. 4, 1950.

[68] BOLEY, D. Principal direction divisive partitioning. *Data Min. Knowl. Discov.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 2, n. 4, p. 325–344, dez. 1998. ISSN 1384-5810. Disponível em: <https://doi.org/10.1023/A:1009740529316>.

[69] ZHANG, Y.-M. et al. Fast knn graph construction with locality sensitive hashing. In: *Proc. of the ECML/PKDD*. [S.l.: s.n.], 2013. p. 660–674.

[70] CONNOR, M.; KUMAR, P. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE Trans. Vis. Comput. Graph.*, v. 16, n. 4, p. 599–608, 2010.

[71] BOUTET, A. et al. Being prepared in a sparse world: The case of knn graph construction. In: *Proc. of the IEEE ICDE*. [S.l.: s.n.], 2016. p. 241–252.

[72] CHÁVEZ, E.; TELLEZ, E. S. Navigating k-nearest neighbor graphs to solve nearest neighbor searches. In: _____. *Advances in Pattern Recognition: Second Mexican Conference on Pattern Recognition, MCPR 2010, Puebla, Mexico, September 27-29, 2010. Proc.* [S.l.]: Springer Berlin Heidelberg, 2010. p. 270–280. ISBN 978-3-642-15992-3.

[73] AOYAMA, K. et al. Zero-resource spoken term detection using hierarchical graph-based similarity search. In: *Proc. of the IEEE ICASSP*. [S.l.]: IEEE, 2014. p. 7093–7097. ISBN 978-1-4799-2893-4.

[74] AOYAMA, K. et al. Double-layer neighborhood graph based similarity search for fast query-by-example spoken term detection. In: *Proc. of the IEEE ICASSP*. [S.l.: s.n.], 2015. p. 5216–5220.

[75] IWASAKI, M. Pruned bi-directed k-nearest neighbor graph for proximity search. In: _____. *Proc. of the SISAP*. [S.l.: s.n.], 2016. p. 20–33.

[76] JAROMCZYK, J. W.; TOUSSAINT, G. T. Relative neighborhood graphs and their relatives. *Proc. of the IEEE*, v. 80, n. 9, p. 1502–1517, 1992.

[77] SUPOWIT, K. J. The relative neighborhood graph, with an application to minimum spanning trees. *J. ACM*, v. 30, n. 3, p. 428–448, 1983. ISSN 0004-5411.

[78] LINGAS, A. A linear-time construction of the relative neighborhood graph from the delaunay triangulation. *Comput. Geom.*, v. 4, p. 199 – 208, 1994. ISSN 0925-7721.

[79] FLOREZ, O. U.; QI, X.; OCSA, A. Mobhrg: Fast k-nearest-neighbor search by overlap reduction of hyperspherical regions. In: *Proc. of the IEEE ICASSP*. [S.l.: s.n.], 2009. p. 1133–1136.

[80] MALKOV, Y. et al. Scalable distributed algorithm for approximate nearest neighbor search problem in high dimensional general metric spaces. In: *Proc. of the SISAP*. [S.l.: s.n.], 2012. p. 132–147.

[81] ZHOU, W. et al. Large scale nearest neighbors search based on neighborhood graph. In: *Int'l Conf. on Advanced Cloud and Big Data*. [S.l.: s.n.], 2013. p. 181–186.

[82] ARYA, S.; MOUNT, D. M. Approximate nearest neighbor queries in fixed dimensions. In: *Proc. of the Fourth Annual ACM-SIAM Symp. on Discrete Algorithms*. Philadelphia, PA, USA: SIAM, 1993. (SODA '93), p. 271–280. ISBN 0-89871-313-7.

[83] YAO, A. C.-C. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. on Computing*, v. 11, n. 4, p. 721–736, 1982.

[84] HACID, H.; YOSHIDA, T. Neighborhood graphs for indexing and retrieving multi-dimensional data. *Journal of Intelligent Information Systems*, v. 34, n. 1, p. 93–111, Feb 2010. ISSN 1573-7675. Disponível em: <https://doi.org/10.1007/s10844-009-0081-z>.

[85] HACID, H.; ZIGHED, A. D. An effective method for locally neighborhood graphs updating. In: ANDERSEN, K. V.; DEBENHAM, J.; WAGNER, R. (Ed.). *Database and Expert Systems Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 930–939. ISBN 978-3-540-31729-6.

[86] SHIMOMURA, L. C.; VIEIRA, M. R.; KASTER, D. S. Performance analysis of graph-based methods for exact and approximate similarity search in metric spaces. In: MARCHAND-MAILLET, S.; SILVA, Y. N.; CHÁVEZ, E. (Ed.). *Similarity Search and Applications*. Cham: Springer International Publishing, 2018. p. 18–32. ISBN 978-3-030-02224-2.

[87] ORTEGA, M. et al. Supporting ranked boolean similarity queries in mars. *IEEE Trans. Knowl. Data Eng.*, v. 10, n. 6, p. 905–925, 1998.

[88] LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, v. 86, n. 11, p. 2278–2324, 1998.

[89] JEGOU, H.; DOUZE, M.; SCHMID, C. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, v. 33, n. 1, p. 117–128, 2011.

[90] BOYTSOV, L.; NAIDAN, B. Engineering efficient and effective non-metric space library. In: BRISABOA, N.; PEDREIRA, O.; ZEZULA, P. (Ed.). *Similarity Search and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 280–293. ISBN 978-3-642-41062-8.

[91] AMATO, G. et al. Region proximity in metric spaces and its use for approximate similarity search. *ACM Trans. Inf. Syst.*, ACM, New York, NY, USA, v. 21, n. 2, p. 192–227, abr. 2003. ISSN 1046-8188. Disponível em: <http://doi.acm.org/10.1145/763693.763696>.

[92] UHLMANN, J. K. Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, v. 40, n. 4, p. 175 – 179, 1991. ISSN 0020-0190. Disponível em: <http://www.sciencedirect.com/science/article/pii/002001909190074R>.

[93] BUSTOS, B.; NAVARRO, G.; CHAVEZ, E. Pivot selection techniques for proximity searching in metric spaces. In: *SCCC 2001. 21st International Conference of the Chilean Computer Science Society*. [S.l.: s.n.], 2001. p. 33–40. ISSN 1522-4902.

[94] TRAINA JR., C. et al. The omni-family of all-purpose access methods: A simple and effective way to make similarity search more efficient. *The VLDB Journal*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 16, n. 4, p. 483–505, out. 2007. ISSN 1066-8888. Disponível em: <http://dx.doi.org/10.1007/s00778-005-0178-0>.

[95] PARK, H.-S.; JUN, C.-H. A simple and fast algorithm for k-medoids clustering. *Expert Systems with Applications*, v. 36, n. 2, Part 2, p. 3336 – 3341, 2009. ISSN 0957-4174. Disponível em: <http://www.sciencedirect.com/science/article/pii/S095741740800081X>.

[96] Jiang, Y.; Zhang, J. Parallel k-medoids clustering algorithm based on hadoop. In: *2014 IEEE 5th International Conference on Software Engineering and Service Science*. [S.l.: s.n.], 2014. p. 649–652. ISSN 2327-0594.

Appendix

APPENDIX A – HGRAPH RESULTS - TYPE OF GRAPH

This appendix shows the bar plots (Figures 56 to 61) presented in Chapter 5 subsection 5.2.4 in a bigger size for better visualization of the results.

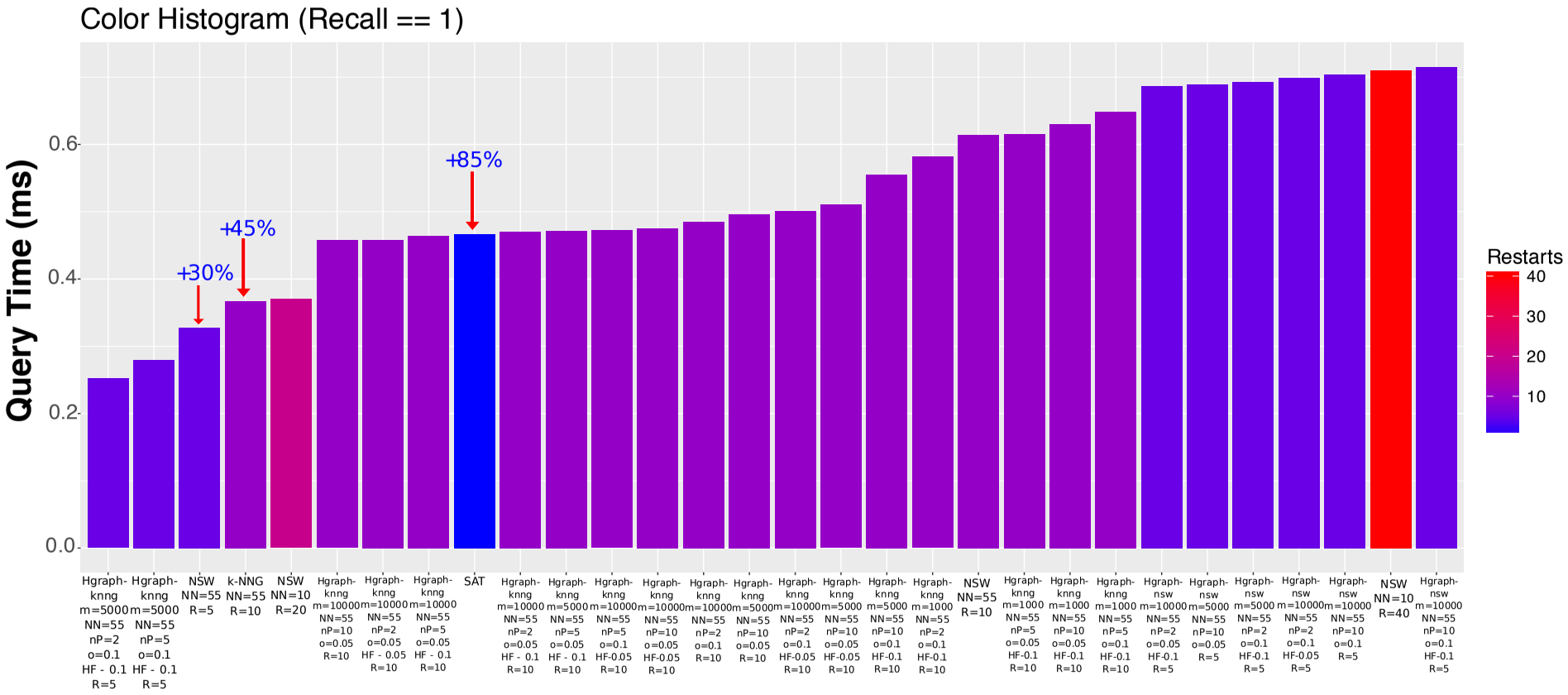

Figure 62 – Query time for Top-30 graph settings (recall = 1) for 1-*NN* queries – Color Histogram dataset (Figure 56).

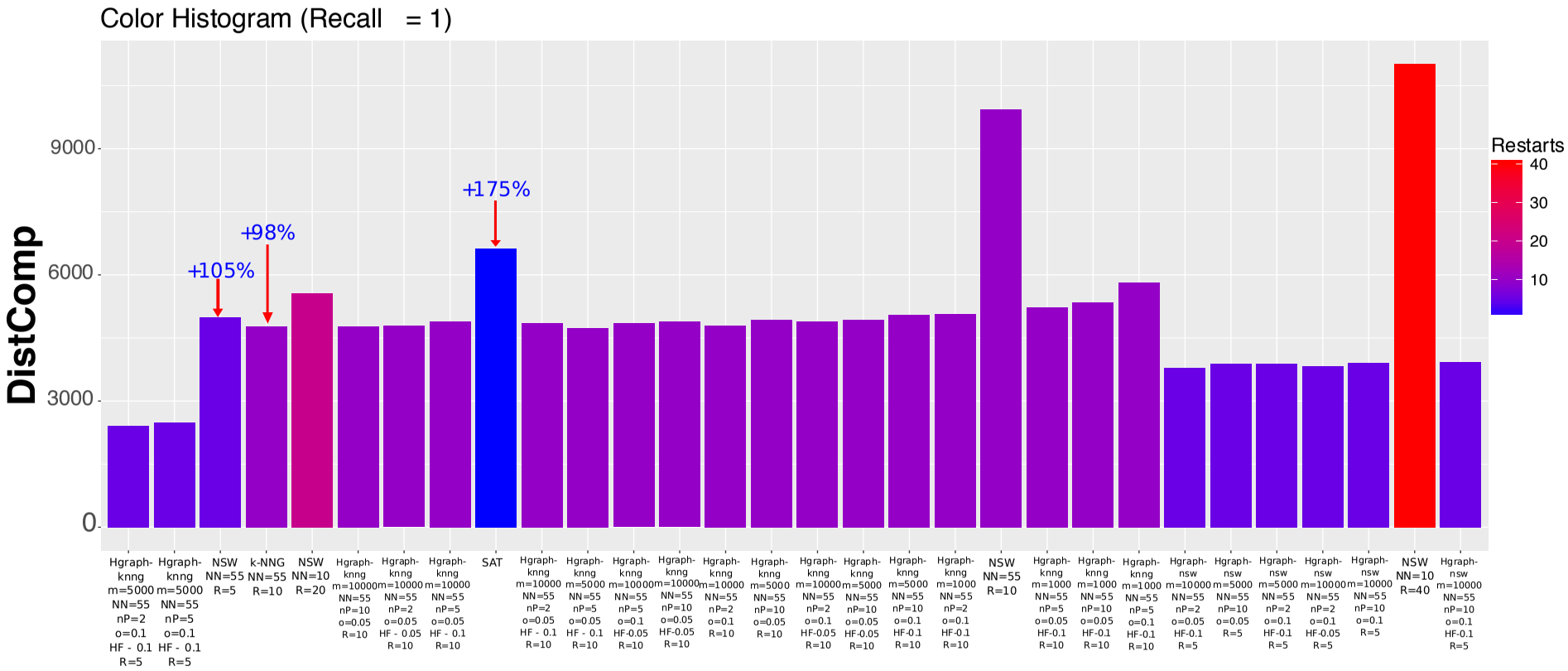

Figure 63 – Distance Computations for Top-30 graph settings (recall = 1) for 1-*NN* queries – Color Histogram dataset (Figure 57).

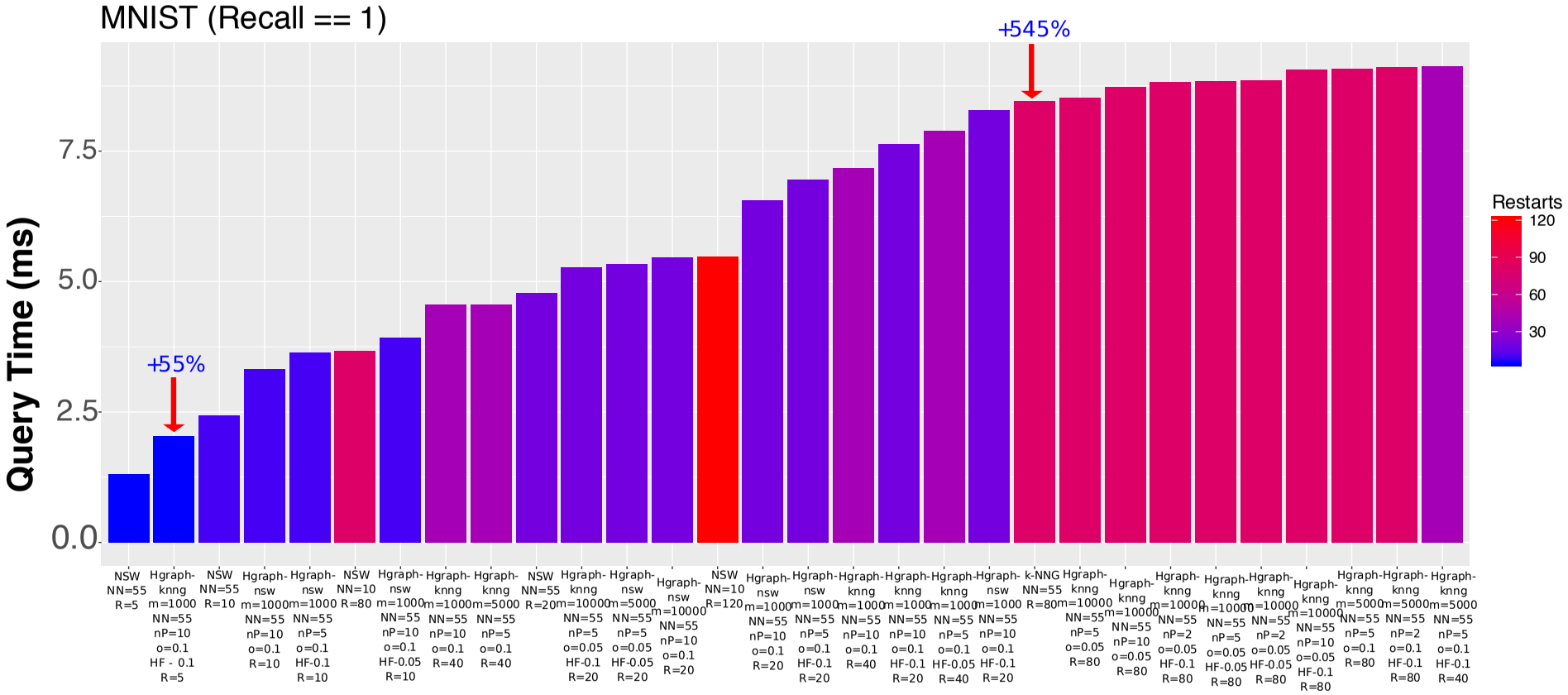

Figure 64 – Query Time for Top-30 graph settings (recall = 1) for 1-*NN* queries – MNIST dataset (Figure 58).

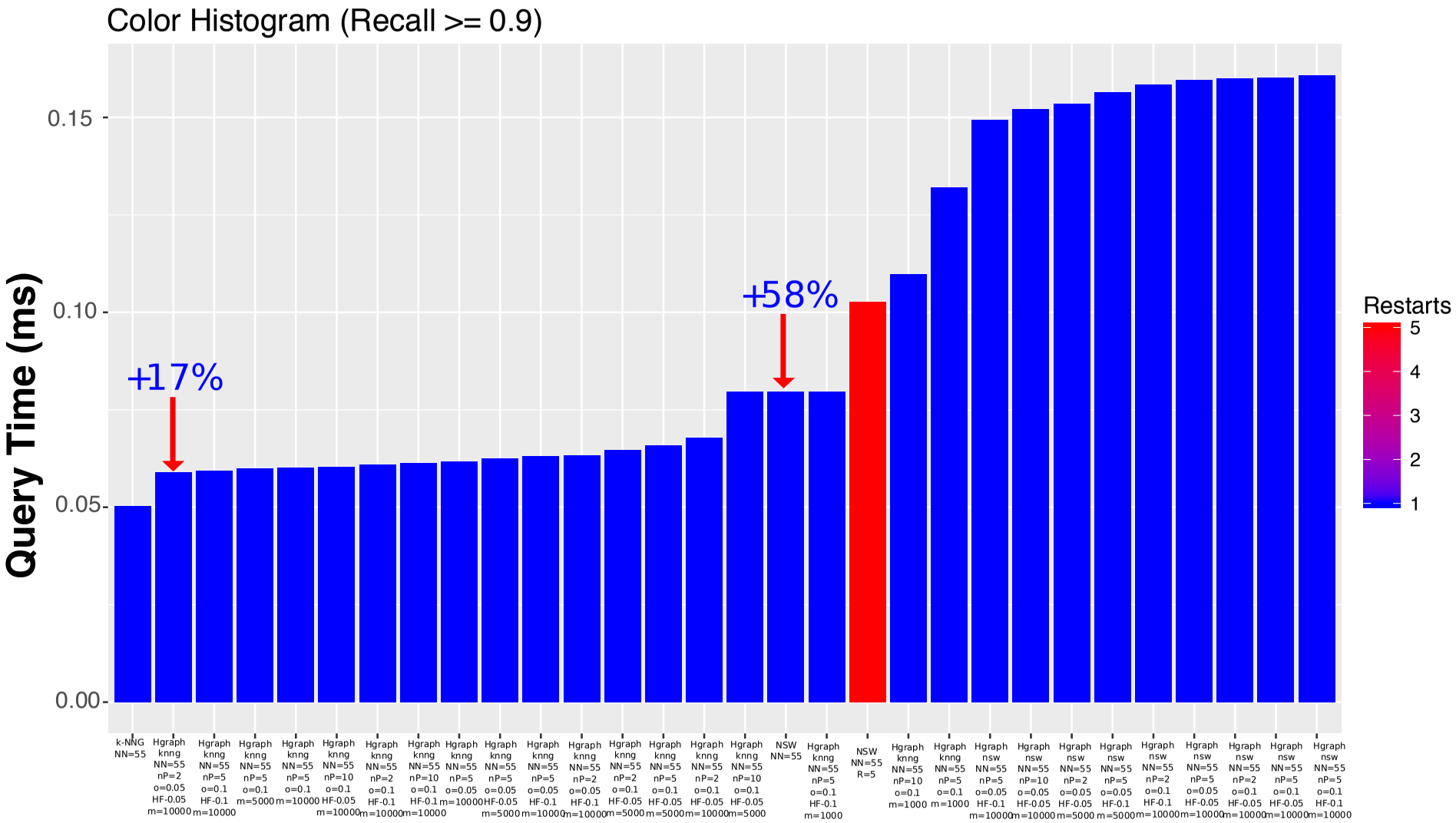

Figure 65 – Query Time for Top-30 graph settings (recall ≥ 0.9) for 1-*NN* queries – Color Histogram dataset (Figure 59).

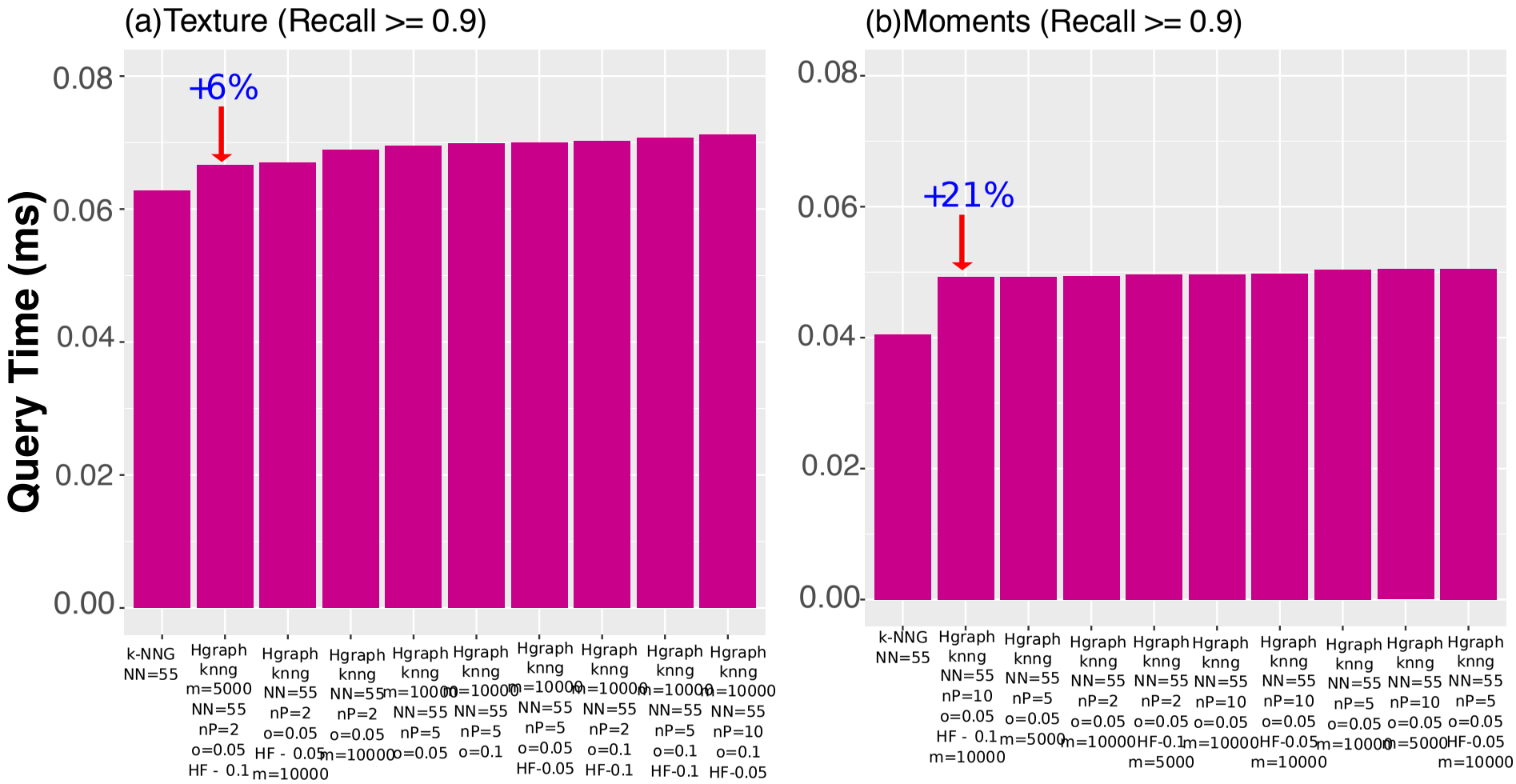

Figure 66 – Query Time for Top-10 graph settings (recall ≥ 0.9) for 1-*NN* queries – Texture and Moments dataset. The number of restarts for all of the selected graph settings in this Figure is 1 (Figure 60).

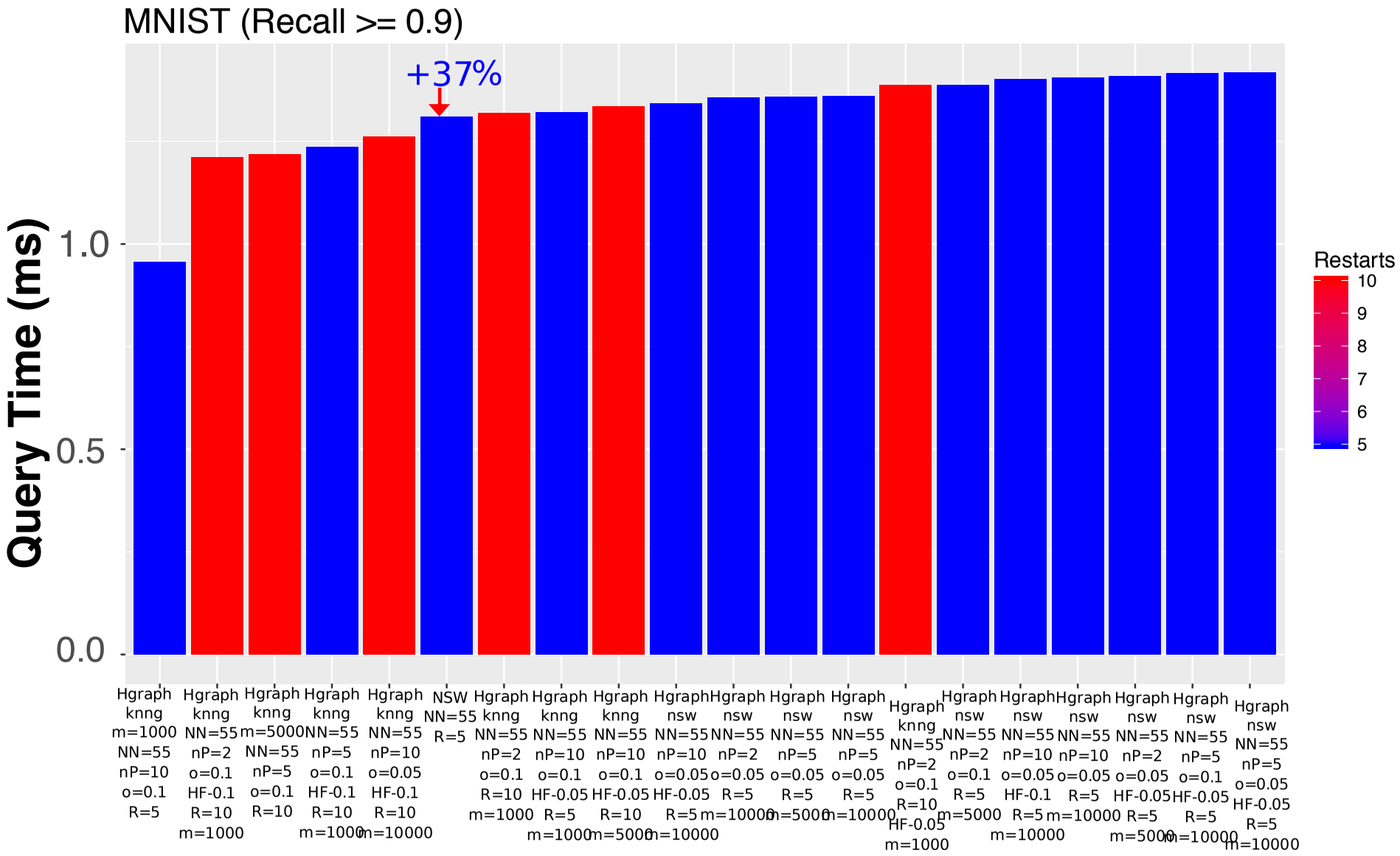

Figure 67 – Query Time for Top-20 graph settings (recall ≥ 0.9) for 1-*NN* queries – MNIST dataset (Figure 61).

PUBLICATIONS

Works published by the author during the Master's Degree:

1. Larissa Capobianco Shimomura, Marcos R. Vieira, Daniel dos Santos Kaster, **Performance Analysis of Graph-based Methods for Exact and Approximate Similarity Search in Metric Spaces**, SISAP 2018 – 11th International Conference on Similarity Search and Applications, October - 2018, Springer Cham, pp. 18-32, ISBN:978-3-030-02223-5, DOI:10.1007/978-3-030-02224-2_2. (Qualis CC 2017, B2)