An Approximate Proximity Graph Incremental Construction for Large Image Collections Indexing

Frédéric Rayar, Sabine Barrat, Fatma Bouali, Venturini Gilles

An Approximate Proximity Graph Incremental Construction for Large Image Collections Indexing

Frédéric Rayar (✉)[1], Sabine Barrat[1], Fatma Bouali[2], and Gilles Venturini[1]

[1] Université François Rabelais de Tours, Laboratoire d'Informatique, Tours, France
{frederic.rayar, sabine.barrat, gilles.venturini}@univ-tours.fr
[2] Université Lille 2 - Droit et Santé, IUT, Lille, France
fatma.bouali@univ-lille2.fr

Abstract. This paper addresses the problem of the incremental construction of an indexing structure, namely a proximity graph, for large image collections. To this purpose, a local update strategy is examined. Considering an existing graph G and a new node q, how only a relevant sub-graph of G can be updated following the insertion of q? For a given proximity graph, we study the most recent algorithm of the literature and highlight its limitations. Then, a method that leverages an edge-based neighbourhood local update strategy to yield an approximate graph is proposed. Using real-world and synthetic data, the proposed algorithm is tested to assess the accuracy of the approximate graphs. The scalability is verified with large image collections, up to one million images.

Keywords: image indexing, large image collections, incremental, proximity graphs

1 Introduction

Dealing with large amount of data has become a great challenge. Advances in technology allow to collect data from almost everywhere, everything and everyone in a continuous way. This permanent flow of data can be nearly infinite and occurs in various fields. A perfect illustration of this phenomenon is the exponential growth of images. Thousands of photos are added each minute on online platforms such as Flickr, Instagram or Facebook.

One challenge, along with the storage of this huge amount of images, is the exploration of these image collections. In order to extract relevant information from these images, one needs to have a relevant representation to observe their global topology and search local information. Proximity graphs [6] have the property of extracting the structure of the data they represent. Each piece of data is represented by a vertex, and two vertices are linked by an edge if they are close enough to be considered as neighbours. Such graphs fit perfectly for purposes such as clustering and outlier detection, but also for indexing and retrieval tasks.

Unfortunately, most of these proximity graphs have been studied as static structures, *i.e.* one considers the whole studied collection, and builds a proximity graph for further studies. However, as previously mentioned, as images keep flooding in a continuous way, it becomes mandatory to handle graph structure in a dynamic way: images and their links can be either added, removed or edited if needed. Thus, it becomes essential to have incremental algorithms to build and update proximity graphs, in order to keep organizing images in a dynamic, yet consistent way.

Since one goal is to handle large image collections, one must take into account the following constraints: *(i)* large matrices, namely the adjacency matrix and the distance matrix, must not be stored, *(ii)* parallelism should be leveraged when feasible and *(iii)* heuristics should be considered and assessed to cut the time complexity of the algorithms.

In this paper, we perform a study of the most recent incremental relative neighbourhood graph (RNG) construction algorithm proposed in the literature, to the best of our knowledge, and highlight some drawbacks of this method. Then, an *edge-based neighbourhood* local update strategy is proposed to incrementally build an approximate RNG. Experiments, on both synthetic and real-world datasets, are presented to assess the quality of the proposed strategy and its scalability to handle large image collections.

The rest of the paper is organized as follows: Section 2 introduces the relative neighbourhood graph. Related works on proximity graphs incremental construction and their limits are studied. In Section 3, the edge-based neighbourhood local update strategy is defined and the related insertion algorithm is given. Experiments on several datasets to assess the proposed algorithm are presented in Section 4. Finally, we conclude the paper in Section 5.

2 Relative Neighbourhood Graph

Proximity graphs [6] are weighted graphs with no loops. They aim at extracting the structure of a data point set, where each point is represented by a node. They associate an edge between two points if they are close enough to be considered as neighbours. The notable proximity graphs include k-nearest neighbour graph, relative neighbourhood graph, Gabriel graph and Delaunay graph.

In the present paper, we will focus our attention on the RNG. Indeed, it is the smallest connected proximity graph that embeds local information about vertices neighbourhood. The connectivity property guarantees that each image can be reachable during a content-based exploration.

2.1 Definition

The relative neighbourhood graph has been introduced in the work of Toussaint [5]. The construction of this graph is based on the notion of *relatively close*

neighbours, that defines two points as relative neighbours if they are at least as close to each other as they are to any other points. From this definition, we can define $RNG = (V, E)$ as the graph built from the points of D where distinct points p and q of D are connected by an edge $\overline{pq}$ if and only if they are relative neighbours. Thus,

$$E(RNG) = \{\overline{pq} \mid p, q \in D, p \neq q, \delta(p, q) \leq max(\delta(p, r), \delta(q, r)), \forall r \in D \backslash \{p, q\}\}.$$

where $\delta : D \times D \rightarrow \mathbb{R}$ is a distance function. An illustration of the *relative neighbourhood* of two point $p, q \in \mathbb{R}^2$ is given in Figure 1.

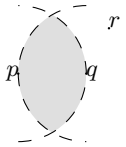

Fig. 1. Relative neighbourhood (grey area) of two points $p, q \in \mathbb{R}^2$. If no other point lays in this neighbourhood, then p and q are relative neighbours.

The main drawback of the RNG is its construction. The classical and brute-force construction has a complexity of $O(n^3)$, where $n = |D|$ is the number of considered data point. A few works in the literature address this complexity for 2D and 3D points. Their key idea is to build a supergraph of the RNG (*e.g.* the Delaunay graph), and adopt a strategy to eliminate some edges to yield the RNG. Thus, one can find in the literature [2] algorithms for 2D and 3D points, whose complexity are $O(nlog(n))$ and $O(n^{\frac{23}{12}}log(n))$, respectively.

2.2 Incremental construction

To the best of our knowledge, only few works have been done in the literature regarding the incremental construction of the RNG. Scuturici et al. [4] explain that they insert the new vertex in the existing RNG graph by verifying the relative neighbourhood criteria specified in Section II. The authors state that the graph is locally updated, but no details are given. They experimented up to 10,000 images and evaluated their work using classification performance metrics, namely precision and recall.

In [1], Hacid et al. propose an algorithm to perform local update of a RNG following the insertion of a new vertex. This algorithm is leveraged to incrementally build the RNG. Given a set of vertices V, the incremental construction of the RNG proposed by Hacid et al. consists in *(i)* randomly selecting 2 vertices of V and creating an edge between them and *(ii)* iteratively inserting the other vertices by locally updating the RNG. The insertion algorithm (Algorithm 1) is detailed below.

Let RNG be the relative neighbourhood graph built from the vertices of V, q be a new vertex to be inserted, and $\epsilon \in \mathbb{R}^+$. First the nearest vertex nn of q is sought in V (line 1). The farthest relative neighbour fn of nn is retrieved in the graph RNG (line 2). A hypersphere SR centred around q is then computed as its neighbourhood. All vertices that lay in that hypersphere are retrieved (lines 6-11). The radius of this hypersphere corresponds to the sum of the distances between q and nn, and the one between nn and fn. Note that this hypersphere radius can be magnified thanks to the parameter ϵ (line 3). The neighbourhood relationships of the hypersphere SR are updated (line 12) with the classical brute-force algorithm.

Algorithm 1 Hacid et al.'s insertion algorithm

Input: $RNG = (V, E)$, q, ϵ
Output: $RNG' = (V', E')$
1: $nn = nearest\_vertex(q, V)$
2: $fn = farthest\_relative\_neighbour(nn, RNG)$
3: $sr = (\delta(q, nn) + \delta(nn, fn)) * (1 + \epsilon)$
4: $V' = V \cup \{q\}$
5: $E' = E$
6: $SR = \emptyset$
7: **for** each $p \in V'$ **do**
8: **if** $\delta(p, q) \leq sr$ **then**
9: $SR = SR \cup \{p\}$
10: **end if**
11: **end for**
12: $E' = Update(SR)$
13: **return** $RNG' = (V', E')$

The complexity of this insertion algorithm is $O(2n + n'^3)$, where $n = |E|$ and $n' = |SR|$. The $2n$ term corresponds to the search of the nearest neighbour and the search of vertices that lay in the hypersphere. The second term is the time for updating the neighbourhood relations between the points within the hypersphere with the classical RNG algorithm. The authors state that the incrementally built RNG corresponds exactly to the RNG built with a brute-force algorithm, using a recall measure and graph correspondence.

We have noticed several drawbacks regarding this insertion algorithm. First, the choice of the parameter ϵ, used by the authors to expand the neighbourhood of q. It is empirically set at $\epsilon = 0.1$ in [1]. However, no proof that relative neighbours of the newly inserted point must lay in this magnified hypersphere is given. This could be the cause of losing relative neighbours as illustrated in Figure 2 (left). Second, due to the spherical definition of the neighbourhood SR, the update step may create false edges. Indeed, the classical RNG algorithm is performed only considering the vertices laying in SR. Figure 2 (right) illustrates such a erroneous edge creation. Thus, the insertion algorithm described above

might not incrementally yield the exact RNG, as stated by the authors, due to the loss of edges or the inclusion of bad ones. This has been observed and reported in Section 4. Third, an assumption is done stating that $n' << n$, *i.e.* the number of vertices in the hypersphere is way less than the number of previously added points. It may not be the case, for instance, if a set of dense points laying in the same part of the space is considered. This has been experimentally observed for a few datasets. Thus the term n'^3 in the complexity might be an issue.

In the next section, we define an edge-based neighbourhood and propose a strategy that leverages it to locally update an RNG, following the insertion of a new vertex. This strategy is then used to incrementally yield an approximate RNG. Thus, we aim at reducing the time complexity of the insertion algorithm while yielding a refined approximation of the exact RNG.

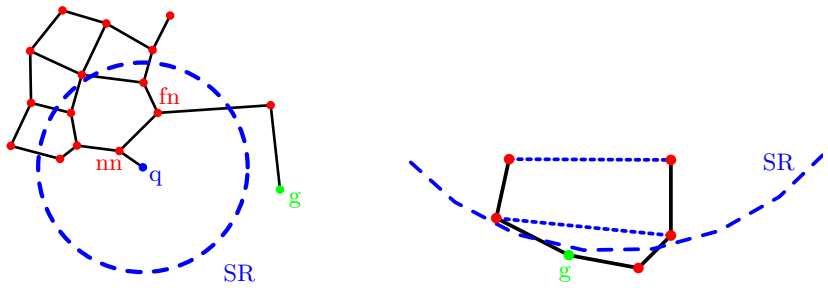

Fig. 2. (Left) Vertex g is a relative neighbour of vertex q. Algorithm 1 fails to retrieve this relationship because g does not lay in the (dashed blue) neighbourhood SR of q, due to ϵ value. **(Right)** False (dotted blue) edges are created. Indeed, as g does not lay in the (dashed blue) neighbourhood SR, it was not considered during the creation of the (dotted blue) edges.

3 Edge-based neighbourhood for local update

3.1 Definition

Let us consider a simple weighted graph $G = (V, E)$, and a vertex $q \in V$. We introduce the notion of *neighbourhood order*, and recursively define the l^{th}*-order vertex neighbours* (Eq. 1) and the l^{th}*-order edge neighbours* (Eq. 2) of a vertex q. Such sets are illustrated in Figure 3.

$$\begin{cases} N^1(q) = \{p \in V \mid \overline{pq} \in E\} \\ N^l(q) = \{p \in V \mid \overline{pr} \in E, r \in N^{l-1}(q)\}, \text{ for } l > 1 \end{cases} \tag{1}$$

$$\begin{cases} N_e^1(q) = \{\overline{pq} \in E \mid p \in N^1(q)\} \\ N_e^l(q) = \{\overline{pr} \in E \mid p \in N^l(q) \text{ and } r \in N^{l-1}(q)\} \end{cases} \tag{2}$$

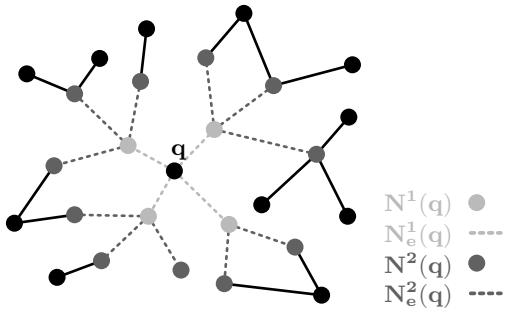

Fig. 3. First and second order *vertex neighbours* (in lightgrey and grey respectively) and *edge neighbours* (in dotted lightgrey and grey respectively) of the vertex q.

Thus, for a given order L, one can define an edge-based neighbourhood of a vertex q given by:

$$N_e^L(q) = \bigcup_{i=1}^{L} N_e^i(q)$$

3.2 Algorithm

We propose an insertion algorithm (Algorithm 2) that relies on our edge-based neighbourhood N_e^L and a local update strategy.

The main steps of Algorithm 2 are as follows. The first steps (lines 1-9) are the same as in Algorithm 1. As explained in the previous section, the hypersphere SR centred around q and its content are computed. Then, we retrieve the relative neighbours of q in SR (lines 12-16). For each vertex p in SR, the pair (p, q) is considered. For each vertex r in SR, we check if r lays in the relative neighbourhood of the pair (p, q). If no vertices lay in this relative neighbourhood, then p and q are relative neighbours, and the edge $\overline{pq}$ is created. This step is carried out in $O(n'^2)$, where $n' = |SR|$. STEP_1 gathers all the edges that belong to the edge-based neighbourhood $N_e^L(q)$ of q, given an order L. This is performed with a recursive algorithm. First, we initialise an empty set of edges A. For each relative neighbour q' of q in RNG, we recursively compute the $(L-1)^{th}$-order edge neighbours of q' and store them in A. At the end of this step, the set A contains the list of edges that belong to $N_e^L(q) \backslash N_e^1(q)$. Finally, the effective update is made in STEP_2: for each edge e in A, we check if e has to be removed due to the apparition of q, *i.e.,* if q lays in the relative neighbourhood of the two endpoints of the considered edge. The overall complexity of the proposed insertion algorithm is $O(2n + n'^2 + deg^L)$, where deg is the average degree of the graph RNG.

Algorithm 2 Edge-based neighbourhood local update strategy

Input: $RNG = (V, E)$, q, ϵ, L
Output: $RNG' = (V', E')$

$nn = nearest\_vertex(q, V)$
$fn = farthest\_relative\_neighbour(nn, RNG)$
$sr = (\delta(q, nn) + \delta(nn, fn)) * (1 + \epsilon)$
$SR = \emptyset$
for each $p \in V$ **do**
  if $\delta(p, q) \leq sr$ **then**
    $SR = SR \cup \{p\}$
  end if
end for
$V' = V \cup \{q\}$
$E' = E$
for each $p \in SR$ **do**
  if $are\_relative\_neighbours(p, q)$ **then**
    $E' = E' \cup \overline{pq}$
  end if
end for
Step_1: $A = compute\_edge\_neighbourhood(q, L, RNG)$
Step_2: $update\_edges\_in\_neighbourhood(q, A, E')$
return $RNG' = (V', E')$

Thus, we propose here an algorithm that reduces the time complexity of the local update strategy. Moreover, the edge-based neighbourhood allows to verify more edges that may be concerned by the apparition of a new vertex. The trade-off between computation time and accuracy that can be achieved will be studied in the experiments.

4 Experiments

4.1 Experimental setup

The algorithms 1 and 2 presented in this paper were implemented in C++. The classical $O(n^3)$ RNG algorithm was also implemented for reference, to assess the graph accuracy. For a fair comparison, the algorithm described by Hacid et al. was implemented under the same constraints as our algorithm (only the local update strategy differs). ϵ value was set to 0.1 as in [1]. In order to speed up some operations (*e.g.* the nearest neighbours search), they were parallelised using OpenMP[1]. In the present experiments, the whole dataset was loaded in memory, and then each piece of data was inserted one by one. The graph was stored as an adjacency list. For runtime experiments, we used an Intel Xeon CPU W3520 (quadcore) at 2.66Ghz, with 8Go of RAM.

[1] `http://www.openmp.org/`

4.2 Datasets

Five datasets were selected (available on the online UCI machine learning repository [2]). They are either artificial or real world multidimensional datasets. Table 1 summarizes the specifications of the datasets. The three first can be considered as *small* datasets, *i.e.* their distance matrices can be stored in the memory. They were used mainly to assess the validity of our algorithm and the accuracy of the resulting graphs. The two last, which are large image collections up to one million images, were used to verify the scalability of the algorithm. For these datasets, the exact computation of the RNG is not tractable (*n.t.*) in reasonable time with the $O(n^3)$ algorithm. Therefore, a CPU/GPU RNG construction method [3], which can handle up to 300.000 entries, was used to generate the exact graph for the Corel68k dataset. Regarding the MIRFLICKR-1M[3] (MF-1M) image collection, its RNG is not tractable at all, thus the number of edges does not appear in Table 1.

Table 1. Datasets used for experiments. The number of vertices, their dimension and the number of edges in the exact RNG are given.

D	Type	| V |	d	| E(RNG) |
Iris	real world	150	4	195
WDBC	real world	569	30	712
Breiman	artificial	5000	40	17,837
Corel68k	real world	68,040	57	190,410
MF-1M	real world	1,000,000	150	*n.t.*

All the five datasets share one common property: their attributes are numerical, hence the euclidean distance was used for data comparison. Note that this work can be applied to the data described by categorical features with an appropriate distance function.

4.3 Accuracy evaluation

First, we evaluate the accuracy of the proposed algorithm and the sensitivity with regards to the parameter L. The exact RNG was computed for the four first datasets. The number of edges of these graphs are used as ground truth and the graph correspondence is computed to evaluate the approximate graphs. Table 2 gives the number of erroneously added edges and removed edges. Since exact RNG could not be produced in reasonable time, this experiment is not reported for the largest dataset, namely *MF-1M*.

One interesting observation in this experiment is that the main difference between the graph produced incrementally and the exact graph is often the

[2] `http://archive.ics.uci.edu/ml`
[3] `http://press.liacs.nl/mirflickr`

Table 2. Number of wrongly added edges and removed edges in the RNGs computed by Algorithms 1 and 2. The symbol == means that the approximate graph corresponds exactly to the exact graph.

	| E(RNG) |	Algorithm 1	Algorithm 2		
			$L = 2$	$L = 3$	$L = 4$
Iris	195	+10/-2	+8/-1	==	==
WDBC	712	+2/-1	+10/-0	+3/-0	==
Breiman	17837	+0/-0	+1161/-0	+299/-0	+26/-0
Corel68k	190410	+20363/-11	+9089/-356	+2165/-388	+637/-397

addition of wrong edges. Actually, it is not the addition of wrong edges, but rather the fact that some edges are not invalidated after an insertion due to the proposed edge-based neighbourhood. Thus, our algorithm leads to create a few number of false similarities between data, which may not be critical in some applications (*e.g.* similar images retrieval or user recommendation systems).

We notice that Hacid et al.'s algorithm does not always incrementally yield the exact RNG as stated in their paper. Furthermore, our proposed algorithm performs at least as well as, if not better, than Hacid et al.'s algorithm in terms of accuracy, considering low edge-based neighbourhood order ($L = 4$).

As expected, the number of the wrongly added or removed edges in the approximate graphs decreases as the edge-based neighbourhood order increases. Indeed, as more edges are checked, less erroneous edges are left, thus improving the accuracy of the approximate RNG. It is possible to build such a graph with less than 1% of wrongly added or removed edges considering low order of edge-based neighbourhood (such as $L = 4$).

4.4 Computation time evaluation

Table 3 presents the computation time of Algorithms 1 and 2. Algorithm 2 results are given over the edge-based neighbourhood order L.

We present the computation times for the small *Breiman* dataset in order to highlight the fact that the computation is high for Algorithm 1. Indeed, this dataset illustrates the incremental worst case scenario. Due to the topology of this peculiar dataset, at almost every iteration, the hypersphere SR contains all the vertices previously added. Thus, this dataset is a perfect counterexample of the assumption $n' << n$, made by Hacid et al., as mentioned in Section III.

For the *Corel68k* dataset, the computation time of Algorithm 1 is five days while our algorithm yields the graph in less than half an hour with $L = 4$. The achieved speed-up ratio is slightly less than 273. For the million images collection, the proposed algorithm succeeds in computing an approximate RNG while Hacid et al.'s algorithm is not tractable in reasonable time. Algorithm 2 computation time might seem quite high, yet this time can be reduced by considering an hybrid approach. First, computing the exact RNG of a subset of images with

Table 3. Comparison of the computation times of Algorithms 1 and 2. Computation times are given in seconds.

	Algorithm 1	Algorithm 2		
		$L = 2$	$L = 3$	$L = 4$
Breiman	7692	16	25	178
Corel68k	122h	889	1371	1604
MF-1M	>> 250h	145h	151h	181h

the embarrassingly parallel $O(n^3)$ algorithm. This subset may contain as much images as possible, provided that the memory can handle its distances matrix. Second, processing the rest of the images using our incremental approach. Indeed, experiments have shown that one can expect an average insertion time of less than 500 ms. This is promising for almost real time updates.

5 Conclusion

In this paper, the problem of the incremental construction of a proximity graph for large image collection indexing is addressed.

Considering constraints such as memory availability, continuous images incoming, and fast computation, we have leveraged a local update strategy to update only a relevant sub-graph of the existing graph, following the insertion of a new image. This strategy allows to incrementally yield an approximate RNG. The proposed algorithm outperforms the existing work in terms of computation time while yielding a refined approximation. Synthetic datasets have assessed the performance of the algorithm. Moreover, scalability is tested on real-world image collections.

Immediate future work will be to implement the hybrid approach in order to reduce computation time for larger image collections. Then, we plan to leverage this incremental proximity graph construction for clustering and large image collection visualisation.

References

1. H. Hacid and T. Yoshida. Incremental neighborhood graphs construction for multidimensional databases indexing. In *Canadian Conference on AI*, 2007.
2. J. W. Jaromczyk, and G. T. Toussaint. Relative Neighborhood Graphs and Their Relatives. *Proceedings of the IEEE*, 80 : 1502–1517, 1992.
3. T. Liu, F. Bouali, and G. Venturini. EXOD: A tool for building and exploring a large graph of open datasets. *Computers & Graphics*, 39:117–130, 2014.
4. M. Scuturici, V.-M. Scuturici, J. Clech, and D. A. Zighed. Navigation dans une base d'images à l'aide de graphes topologiques. In *Inforsid*, 2004.
5. G. T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12:261–268, 1980.
6. G. T. Toussaint. Some unsolved problems on proximity graphs, 1991.