Leveraging Bitmap Indexing for Subgraph Searching

David Luaces
Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela (USC)
Santiago de Compostela, Spain
david.luaces@usc.es

José R.R. Viqueira
Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela (USC)
Santiago de Compostela, Spain
jrr.viqueira@usc.es

Tomás F. Pena
Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela (USC)
Santiago de Compostela, Spain
tf.pena@usc.es

José M. Cotos
Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela (USC)
Santiago de Compostela, Spain
manel.cotos@usc.es

ABSTRACT

Deciding whether a query graph is a subgraph of some other in a very large database of small graphs is a problem of major interest in many application domains. As an example, it arises in the searching for specific molecular substructures in currently available molecular databases, whose sizes may reach levels close to one hundred million. State of the art methods to solve this problem follow a *filter-then-verify* (FTV) paradigm, where an indexing technique is first used in a filtering stage to obtain result candidates and a subgraph isomorphism algorithm is next applied to the candidates in a verification stage to obtain the final result. Among all the available techniques of the state of the art, two of them have demonstrated better performance when applied to large datasets, namely, the GraphGrepSX (GGSX) and CT-Index (CTI). In this paper, three new indexing techniques, one based on GGSX and two based on CTI, are proposed. In particular, Bitmap GGSX (BM-GGSX) leverages the use of bitmaps in the trie structure used by GGSX to achieve performance gains of around 90% in the filtering stage. Column-Wise CT-Index (CW-CTI) exploits a column-wise representation of the fingerprints (bitmaps) used by CT-Index to reduce the filtering times around 80% for small queries (8 edges). Finally, K-Means CT-Index (KM-CTI), constructs a binary tree of bitmaps from the CT-Index fingerprints to reach filtering time reductions of around 70% for medium queries (20 edges) and 75% for large queries (40 edges).

1 INTRODUCTION

Graph database research is a topic to which much attention has been paid by the data management community during the last decade. Two major types of graph databases may be found in real problems. In a first type, the database consists of just one very large graph. This is, for example, the kind of database available in the semantic web. Amongst the required query functionality, many applications need to identify all the instances of a specific subgraph that occurs inside the database graph. This NP-complete problem, known in the literature as *subgraph matching*, is solved through the application of *subgraph isomorphism* algorithms [4, 12, 20, 21, 26, 27].

A second type of database contains a very large number of small graphs. Typical examples of this type of databases are molecular databases. For each molecule recorded in such a database it is recorded, among other properties, its structure, i.e., a small graph whose vertices are atoms and whose edges are bonds between atoms. The current size of these databases is already very large, reaching levels close to one hundred million molecules, like in the PubChem dataset[1], and it is increasing. Among other queries of interests, it is important to provide functionality to find all the molecules whose molecular structure (graph) contains a specific query substructure (subgraph) [5, 6].

Finding all the graphs in a database that contain a specific subgraph is known in the literature as the *subgraph decision* problem. A straightforward strategy to solve this problem is a linear search, i.e., the application of a *subgraph isomorphism* algorithm to each graph of the database. This strategy could be the appropriate for not selective queries that return most of the database graphs. Besides, given its simplicity, it is also straightforward to a obtain parallel implementation following this strategy. However, in the general case, a filter-then-verify (FTV) strategy based on indexing will get much better performance. In FTV approaches, query processing is split in tho stages, namely *filtering* and *verification*. In the filtering stage, an index structure is searched to obtain an initial set of candidate graphs. In the second stage, a subgraph isomorphism algorithm is applied to each candidate to refine the final result. Many FTV methods have been proposed in the literature [2, 3, 8, 15, 20, 24, 25, 28, 29].

Among the proposed FTV solutions, GraphGrepSX (GGSX) [2] and CT-Index (CTI) [15] have demonstrated an excellent performance when applied to large datasets of small graphs, as it is the case of molecular databases. Both of them rely on the encoding, in the index structures used during the filtering stage, of features contained in the graph, such as paths, trees, and cycles. In particular, GGSX generates a trie structure with all the paths contained in each graph up to a maximum length. Each node of the trie represents a specific path and references all the graphs of the database that contain such path, storing also the number of times that the path is repeated in each graph. During the filtering stage, the database trie is used to obtain all the graphs that contain all the paths of the query up to a maximum length, at least the number of times that the path is contained in the query.

The indexing technique used by CTI is completely different. CTI generates a fingerprint (bitmap) for each database graph

[1] https://pubchem.ncbi.nlm.nih.gov/

as follows. First, graph features are extracted. CTI may work with either paths or trees, and with cycles. Each feature of the graph up to a maximum length is extracted, and represented in a canonical string format. A hash function is next applied to each extracted string to obtain integer numbers that range from 0 to the fingerprint length. The bits of the fingerprint located at the positions defined by those integers are set to 1. In the filtering stage, bitmap containment is tested between the query fingerprint and the fingerprint of each database graph.

In this paper, three new FTV solutions are proposed for subgraph search in large databases of small graphs. The paper focus on the filtering stage, improving GGSX and CTI structures and algorithms to obtain filtering time reductions between 70% and 80%, whereas the verification stage of the three techniques relies on the parallel execution in a multi-thread architecture of a improved version of the VF2 [4] subgraph isomorphism algorithm already proposed by CTI [15]. In particular, the main contributions of the present work are summarized as follows.

(1) Bitmap GGSX (BM-GGSX) takes advantage of the incorporation of compressed bitmaps in the representation of graph references in the GGSX trie nodes. This enables a drastic reduction of the filtering time (around 90% less than GGSX), maintaining the index size and building time almost unaltered. When compared with the techniques based on CT-Index also proposed in this paper, BM-GGSX offers good response times for medium and large queries, but it is worse for small queries. Besides, BM-GGSX is also much worse in both index size and building time.
(2) Column-Wise CT-Index (CW-CTI) adopts a column-wise storage structure for the CT-Index fingerprints, where the same bit position of all the fingerprints of all the graphs are recorded together in a compressed bitmap. During the filtering stage, only the bit positions with a 1 in the query fingerprint have to be accessed, which makes the filtering time dependent on the query size. Regarding index size and building time, CW-CTI gets the best results.
(3) K-Means CT-Index (KM-CTI) applies recursively the K-Means clustering method on CT-Index fingerprints to construct a bitmap binary tree that is used to discard sets of fingerprints during the filtering stage. KM-CTI index size and building time is worse than CW-CTI, but better than BM-GGSX. Regarding filtering time, it is competitive with BM-GGSX in medium and large queries, thus it is a good complement of CW-CTI.
(4) An extensive evaluation of the performance of all the techniques is undertaken using real databases of molecular graphs of different sizes. A first experiment is conducted to determine the best parameter values to tune the methods. The objective of the second experiment is to test the scalability of the techniques with increasing database sizes. Real databases with sizes ranging from 200000 to one million molecules were built from the PubChem dataset. Notice that those sizes are much larger than even the synthetic datasets used in previous surveys [11, 13].

The remainder of this paper is organized as follows. Section 2 reviews related work, with special attention to the original GGSX and CT-Index methods. The proposed BM-GGSX, CW-CTI and KM-CTI methods are described in detail in Sections 3, 4 and 5, respectively. Section 6 is devoted to the discussion of the evaluation experiments and, finally, Section 7 concludes the paper and outlines some lines of potential future work.

2 RELATED WORK

The current state of the art classifies the subgraph querying problem into two different subproblems. The first one, named *matching problem*, deals with extracting all subgraph isomorphic embeddings of a query graph q in a single graph G. The second one, usually called *decision problem*, is focused in retrieving the ID of every graph g, stored in a given dataset G, which satisfies that the query graph q is a subgraph of g.

The matching problem has been widely studied over the years, and there are several proposals to solve it. As mentioned in the introduction, a subgraph isomorphism algorithm must be applied to ensure that the verified graph contains at least one subgraph that matches exactly the query graph. In [16], the authors provide a performance comparison of six of the most relevant solutions at the time. Among these algorithms, Ullmann [21], VF2 [4], and QuickSI [20] were originally designed for handling small graphs, while GraphQL [12], GADDI [26], and SPath [27] were designed for handling large graphs. The tests were accomplished in four real-world datasets with different size and characteristics, containing two of them multiple and small graphs, and the other two single and relatively large graphs. The study concluded that GraphQL was the only algorithm that completed all the queries, while QuickSI performed the best for both small and large data graphs, even though it was designed for handling small graphs, since the cost of its recursive call is the lowest. The study also concluded that all existing algorithms had problems in their join order selections. Since the survey was published, new algorithms designed for handling large graphs have been proposed [10, 19], showing an improvement in the performance by addressing the issues of matching order selection. Recently, a new approach [1] proposes a new framework to minimize the redundant Cartesian products.

The *decision problem* is typically arising in the scope of applications with big datasets of small graphs. Since this is a NP-complete problem, and datasets contain often a large number of graphs, deciding whether a query graph q is contained in every graph g in the dataset G, by applying one by one a subgraph isomorphism algorithm would be very costly. Due to this, a pruning phase must be undertaken in an early stage of the process in order to select a reasonable number of candidates to be tested with a subgraph isomorphism algorithm. This idea is carried out by the techniques based in the filter-then-verify (FTV) paradigm. FTV techniques rely in building an index of the graph dataset, through decomposing each graph into features (i.e., paths, trees, cycles, etc.), and store them in an appropriate structure (e.g., trie, fingerprints, etc.). The search algorithm of FTV techniques is divided into two different stages. The first stage, called *filtering stage*, aims at obtaining a *candidate set* of graphs. To do this, the query graph q is decomposed into its corresponding features, according to the applied technique, which are used to retrieve from the index a reduced *candidate set* with the graphs that contain all the query features. The *candidate set* is tested finally with a subgraph isomorphism algorithm in the second stage, called *verifying stage*.

A performance comparison of different indexing techniques [3, 20, 24, 25, 28, 29] for subgraph query processing is provided in [11]. The above survey is extended in [13], by including three new algorithms [2, 8, 15] and also by performing an exhaustive performance and scalability study, varying different dataset features such as number of nodes and graphs. According to the conclusions of [13], CT-Index [15] and gCode [29] have the smallest

index size. Regarding query processing time, this study concludes that the approaches that build the index with graph features (Grapes [8], GraphGrepSX [2], and CT-Index [15]) outperform the others, and, among them, those that use simpler features (Grapes and GraphGrepSX) obtain the best results. The techniques that show a better scalability are GraphGrepSX, Grapes, and CT-Index. However, Grapes, fails to build an index for large datasets due its memory requirements. Finally, the survey shows that algorithms based in mining techniques (gIndex [24] and Tree+Δ [28]) are only competitive for small datasets.

Recently, two new methods that employ caching on top of existing FTV techniques have been proposed (iGQ [22] and GraphCache [23]). They take advantage of the fact that, in real-world scenarios, most current queries have subgraph or super-graph relations with the future ones. Therefore, they use a caching system with past queries and their answers, including some novel replacing strategies, to improve the performance of existing methods. Another recent work [14], after analyzing both matching and decision problems, it concludes that there exist algorithm specific straggler queries that are challenging only for specific approaches. Based on the above observation, a novel framework is proposed in [14] that leverages parallel execution and query rewriting to achieve better overall performance.

Based on the conclusions of [13], we select GraphGrepSX and CT-Index as the base of the present work, to develop improved indexing structures to solve the decision problem. Notice that our experiments consider databases whose size is between 10 and 20 times larger than those of [13], and, therefore, we discarded Grapes due to its problems with large databases. Finally, the results of the present work may be incorporated in complex query processing frameworks, as the one proposed in [14], which may also leverage in caching techniques [22, 23].

2.1 GraphGrepSX

GraphGrepSX, GGSX for short henceforth, decomposes each graph in paths up to a maximum length p specified by the user. Repeated paths are taken into account. With these paths, in the index building stage, GGSX incrementally builds a trie with a depth equal to the maximum path length p. Each node of the trie represents a path from the root to that node, and stores a list of key-value pairs, where the key represents the ID of a graph in the dataset that contains the path represented by the node, and the value is the number of times that the path appears in the graph. Figure 1 shows an example of a small trie. Each node of the trie records a large list of key-values pairs, although only two of them are depicted in the figure due to space limitations. Thus, as it is represented at the bottom left node of the trie, path "CCC" is contained 4 times in graph G_1, 5 times in graph G_3 and 3 times in graph G_n. Notice that the size of the main structure of the trie is completely dependent on the chosen maximum path length p, which in practice is a short value. On the other hand, the lists of key-value pairs recorded in each node increase their size linearly with the size of the database.

In GGSX, as in any other FTV technique, subgraph query processing is divided into two stages: filtering and verification. In the filtering stage, a trie is built using all the paths of the query whose length is lower or equal to the chosen maximum length p. The number of repetitions of each path is only recorded now for leaf nodes. The filtering algorithm performs a joint breadth-first traversal of query and database tries. When a query leaf node is reached, the number of repetitions r recorded in the query

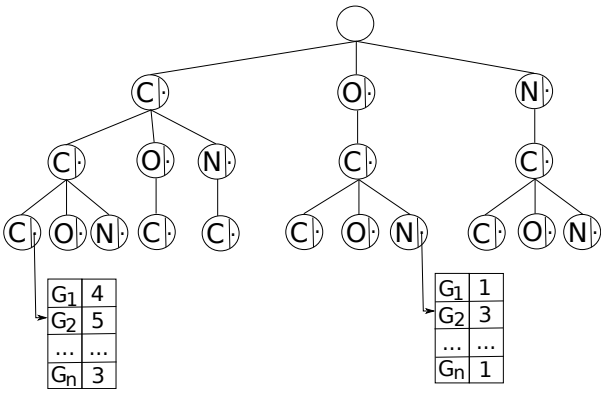


Figure 1: Trie structure used as indexing in GGSX.

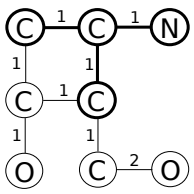
trie node is compared with the list of repetitions recorded in the database trie node, obtaining the set of graphs that contain at least r times the relevant path. Such list of graphs is maintained to be intersected with subsequent lists obtained from the remainder query trie leaf nodes. The final list of graphs resulting from all the intersections is tested, in the verification stage, using the VF2 subgraph isomorphism algorithm.

Overall, it is expected that the chosen p value should have an impact in different aspects: the index size (should increase with p), the index construction time (should increase with p), the filtering time (should increase with p) and the verification time (should decrease with p). To the best of our knowledge, there is not, currently, any mathematical model for the determination of the best expected p value for a given dataset, therefore, it has to be chosen based on some benchmarking.

2.2 CT-Index

CT-Index uses a list of bitmaps of a fixed size f called fingerprints, for indexing purposes in the filtering phase. Roughly speaking, either paths or trees, and cycles of each graph are hashed to integers between 0 and f, which are next used to activate bits in the graph fingerprint. To generate all the possible either paths or subtrees, and cycles, a maximum length p is again considered. Figure 2 illustrates the creation of a fingerprint from a given graph. First, all possible features, either paths or trees, and cycles of a maximum length p are generated from the graph. A specific tree contained in a graph is depicted in Figure 2(a). Next, the extracted features are encoded in a canonical string format (see Figure 2(b)). A hash function is next applied to the generated canonical string to obtain an integer. Integer number 27 is obtained from the string in the example of Figure 2(c). Finally, the relevant bit of the graph fingerprint is set to 1 (see Figure 2(d)). It has to be noticed that the hash function may generate the same integer for different features (collisions). Such collision have a negative effect, decreasing the number of pruned graphs in the filtering stage.

CT-Index is also a FTV technique, with relevant filtering and verification stages. In the filtering stage, a fingerprint is obtained for the query graph as described above, using the same fingerprint size 2^f and the same feature maximum length p. The query fingerprint is then tested for bitmap containment with each fingerprint in the index. Bitmap containment is performed by first splitting both fingerprints (query FQ and database FD) into a sequence of long integers (query LQ and database LD), and then



(a) Tree selection

C1C$1C$1N$$

(b) Canonical string

h(C1C$1C$1N$$) = 27

(c) Hashing of feature

0	1	2	...	27	...	127
0	0	0	...	1	...	0

(d) Insertion in fingerprint

Figure 2: Fingerprint construction process. In (a) a tree feature is selected. In (b) the feature is transformed into a canonical string. It is hashed in (c). In (d) the hashed value is set in the fingerprint.

testing bitmap containment between each pair of (LQ, LD) using bitmap operations ($LQ \wedge LD = LQ$).

CT-Index uses, in the verification stage, a version of the VF2 subgraph isomorphism algorithm, improved with additional heuristics.

It is noticed that with this technique, two parameters may be chosen to construct the index, namely, the maximum feature length p and the fingerprint size 2^f. It is expected that the maximum feature length should not have an impact either in the index size or in the filtering time. However, increasing p should have a positive impact in the verification time (more graphs should be discarded in the filtering phase) and should have a negative impact in the index construction time (more and larger features have to be processed). Regarding the fingerprint size 2^f, on one hand, it has clearly a negative impact in the index size, and therefore in the filtering time (larger fingerprints have to be compared). On the other hand, however, it should have a positive impact in the verification time, by reducing the number of collisions generated by the hash function.

3 BITMAP GGSX

In this section, an evolution of the GGSX method, called Bitmap-GGSX (BM-GGSX), is explained in detail. We modify the original GGSX method in the following three aspects: i) The labels of the graph edges are now considered, improving the performance of the method in graphs such as molecule structures, whose edges contain bound type information (simple bound, double bound, etc.). ii) Very large compressed bitmaps are now exploited to provide a more compact and efficient representation of the list of pairs (graph id, repetitions) recorded in each trie node. iii) The original VF2 subgraph isomorphism algorithm used in the verification stage was replaced by the parallel execution, in a multi-thread architecture, of the improved version of VF2 algorithm provided by the CT-Index implementation.

The original GGSX method discards the edge labels during the construction of the trie structure. On one hand, this reduces the size of the structure, and as a consequence it reduces also the filtering time. But, on the other hand, it increases the number of candidates for the verification stage, which is in general more costly. To incorporate edge labels in the trie, we modify the structure of the trie nodes by adding a new field that records edge labels. Now, the interpretation of a trie node changes from

(a)

ID	R
0	2
3	1
5	2
8	3
10	1
11	2
14	2

(b)

R	IDs
1	100101001011001
2	100001001001001
3	000000001000000

Figure 3: Comparison between (a) GGSX and (b) BM-GGSX graph repetitions storage technique.

interpreting it as a graph node of a path, as it is in the original GGSX, to interpreting it as the combination of a graph node with the edge that precedes it in the path. Obviously, given that paths start at node and not at edges, the nodes of the first level of the trie have always null edges. Clearly, this new trie structure has more nodes in each level, enabling the representation of a larger number of different paths of the same length.

In the original GGSX method each node of the tree records a list of pairs (i, r), where i is an identifier of a graph (integer index) that contains the path represented by the trie node and r is the number of times that such path appears in the graph. Such lists are replaced in BM-GGSX by an structure that exploits the use of bitmaps. In particular, each trie node records now an array of bitmaps. The bitmap recorded at the index r of the array has a 1 in its position i if, and only if, the graph with identifier i contains the path represented by the trie node at least r times. Figure 3 illustrates the difference between the representation of graph references and repetition in both GGSX and BM-GGSX. Bitmaps are compressed in BM-GGSX using the Enhanced Word-Aligned Hybrid method [17].

The algorithm of the filtering stage that exploits the above structure is illustrated in Figure 4. First, as it is shown in the left part of the figure, the query is processed to obtain its trie structure. Remember that, now, each node contains a pair of (node label, edge label) and that the number of repetitions in the query trie is recorded only in the leaf nodes. As in the case of GGSX, a joint breadth-first traversal of both query and database tries is performed, obtaining now, for each leaf node of the query trie, the bitmap recorded in the position of the array whose index matches the number or repetitions recorded in the leaf node. This is illustrated in the central part of the figure. Finally, a binary *AND* operation is performed between all the compressed bitmaps, to obtain the result bitmap, which will contain a 1 in the positions of all the graphs containing, at least the number of times required, all the paths of maximum length of the query.

Notice that, as it is shown in Figure 3, if the bitmap recorded at the index r of the array of a trie node has a 1 in position i, then all the bitmaps recorded in indices lower than r must also have a 1 at position i. This inserts redundant information in the structure, but, as a consequence, it also enables better performance during query processing, since only one of the bitmaps of each candidate node of the trie has to be accessed.

4 COLUMN-WISE CT-INDEX

The index structure used by CT-Index consists of one fingerprint for each graph of the database. Those fingerprints are sequentially generated and recorded during the index construction and sequentially processed during the query evaluation. The collection of all fingerprints might be seen as a matrix, where each

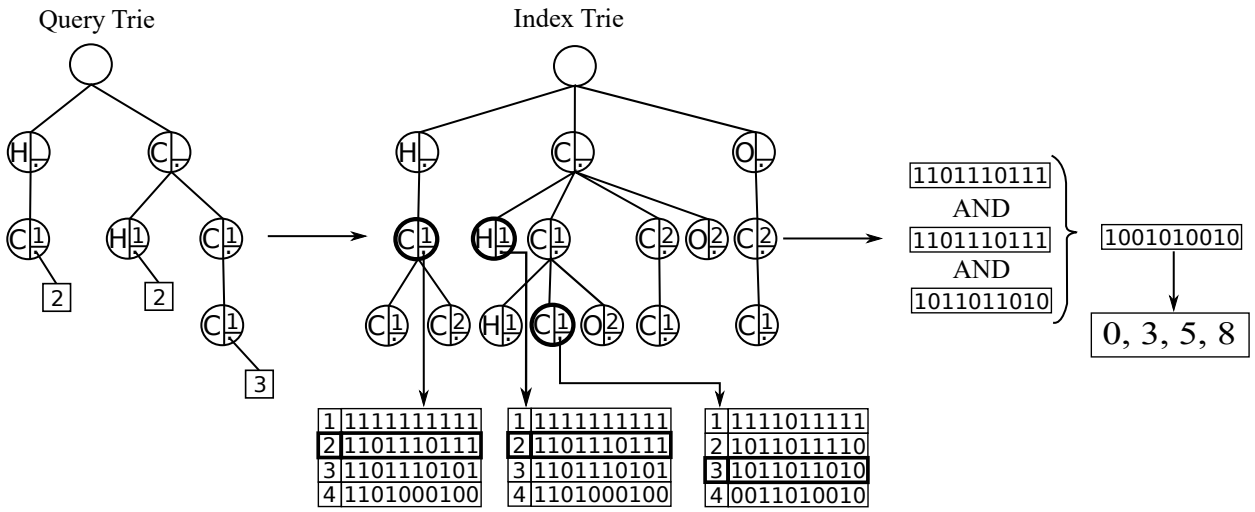


Figure 4: Example of filtering stage in BM-GGSX.

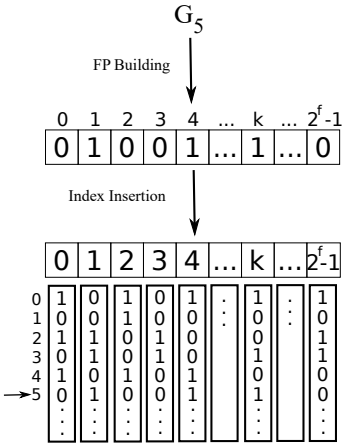


Figure 5: Example of a new graph(ID = 5) insertion in the CW-CTI index.

row is a fingerprint and each column a bit position inside the fingerprint. It is noticed that such matrix is recorded row-wise in CT-Index. The Column-Wise CT-Index, in short CW-CTI, leverages the use of compressed bitmaps by recording and processing the fingerprints column-wise. Thus, if 2^f is the chosen size for the fingerprints, then the index of CW-CTI is a sequence of 2^f bitmaps, such that fingerprint number b has a 1 at position i if, and only if, the fingerprint of molecule number i has a 1 in the position b of its fingerprint. The bitmaps of CW-CTI are much larger than the fingerprints of CT-Index, and they are likely to contain larger sequences of repeated bits, so, they are good candidates for the use of bitmap compression. Enhanced Word-Aligned Hybrid compression [17] is used in the present implementation, which enables the reduction of index size compared to the original CT-Index.

The building stage of CW-CTI is similar to that of CT-Index. Each fingerprint is constructed exactly in the same way, however, now each bit of the obtained fingerprint is appended to the end of each corresponding CW-CTI bitmap. This process is illustrated in Figure 5 for a specific graph G_5.

Figure 6 illustrates the algorithm of the filtering stage of CW-CTI. It is noticed that the algorithm is completely different from the one of CT-Index. First, the fingerprint of the query is obtained in the same way that it is obtained for CT-Index. Next, for each 1 in the query fingerprint, the compressed bitmap recorded at the relevant column of the CW-CTI structure is obtained. Remember that such compressed bitmap contains a 1 for each graph whose fingerprint has a 1 in the same position. Finally, all the obtained bitmaps are intersected to produce the final result of the filtering stage. The result compressed bitmap will contain a 1 for each candidate graph, i.e., each graph whose fingerprint is binary contained in the query fingerprint.

The verification stage in CW-CTI performs a subgraph isomorphism test to the final candidate set of graphs with the improved version of the VF2 used in the CT-Index method, except that now this algorithm is executed in parallel in a multi-thread architecture.

As it was already stated, CW-CTI requires, in general, less storage than the original CTI-Index, due to the use of bitmap compression. The number of candidates obtained from the filtering stage is exactly the same as the one obtained by CTI, however, the verification time should be better, due to the use of a parallel execution of the VF2 algorithm. Regarding the filtering time, it is noticed that the number of binary *AND* operations between compressed bitmaps required is directly determined by the number of 1s in the query, i.e., by the number of different features in the query, which is higher as the query size increases. Therefore, it is expected to behave better with smaller queries.

5 K-MEANS CT-INDEX

In this section K-Means CT-Index (KM-CTI), a new proposal based in the CT-Index method, is described. This new method was constructed with the aim of reducing the number of fingerprint comparisons made in the filtering stage of the CT-Index algorithm. It is reminded that CT-Index performs a comparison between the query fingerprint and each of the fingerprints of the database, therefore, it is expected that the filtering time will increase linearly with the database size. KM-CTI uses a binary tree of bitmaps, where the leaf nodes correspond to the database fingerprints, and parent nodes are constructed by performing

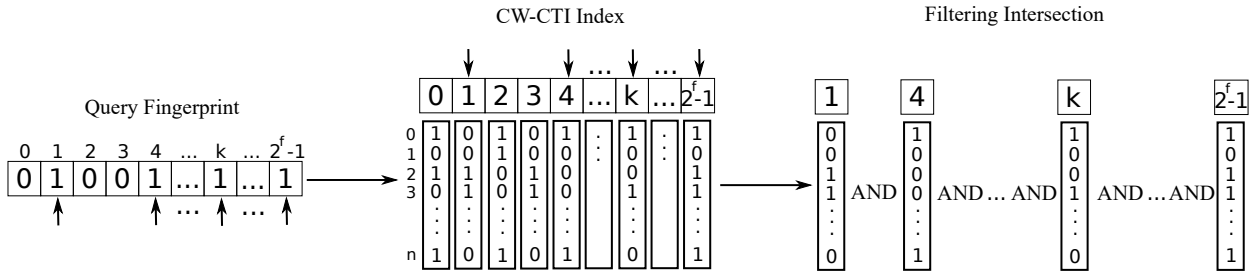


Figure 6: Example of filtering stage in CW-CTI.

the binary *OR* operation between two children. If the comparison between the query fingerprint and a given node of the tree returns false, then the whole branch below the node may be directly discarded. Thus, if the database is large enough, then searching the KM-CTI binary tree will require less comparisons than the number of database fingerprints, reducing the time of the filtering stage.

An example of the use of a KM-CTI binary tree to evaluate the fingerprint of an input query is illustrated in Figure 7. It is noticed that, in the binary tree, all the fingerprints are in the leaf nodes, and each parent node contains the binary *OR* of its children. The query is first compared with the root node. Given that the query is binary contained in the root node, then it has to be compared with each of the two children. The comparison with the left children is positive, and therefore the process has to continue through the left branch. However, the comparison with the right child is negative, therefore the whole branch may be discarded. In the figure, highlighted nodes have been compared with the query and nodes of dark color represent positive results in the comparison. Therefore, in this example, the algorithm performs 7 comparisons (instead of the 8 comparisons that the original CI-Index would require) to select two fingerprint candidates for the verification stage. Again, the chosen algorithm for the verification stage is the VF2 version proposed by the original CT-Index method, and again, this algorithm is executed in parallel in a multi-thread architecture.

The KM-CTI binary tree will offer a good performance if the internal nodes discard large numbers of fingerprints with few comparisons. To achieve this, pairs of bitmaps have to be combined under parent nodes in a way that minimizes as much as possible the number of 1s in those internal nodes. This behavior during the tree construction is achieved in KM-CTI through the application of the K-Means [7, 18] clustering algorithm, recursively to the original set of fingerprints, as it is explained below.

In general, K-Means is used to distribute a set of elements in K clusters, in a way that a given distance measure is minimized between the elements of each cluster. Broadly speaking, the algorithm works as follows. First, K elements of the set are aleatory chosen as centroids of the K clusters. Next, each element is assigned to the cluster corresponding to its nearest centroid. Once the first cluster have been defined, a new centroid is chosen for each cluster by computing some kind of mean between the elements of the cluster. The processes of assigning elements to their nearest neighbor centroid and computing new centroid are iteratively repeated until the elements of each cluster do not change in two consecutive iterations.

To construct the tree, K-Means is first applied with $K = 2$ to the original set of fingerprints, to obtain two clusters. The binary *OR* of the elements of each cluster is computed to obtain the two children of the root node of the tree. This process is recursively repeated inside each cluster until one of the two following conditions hold: i) the elements per cluster reach two (two original fingerprints), ii) the K-Means method cannot subdivide the cluster further. At this stage the tree is completed. The number of fingerprints contained in a leaf node may be greater than two if they are either equal, or so similar that they cannot be subdivided further.

To be able to apply K-Means to sets of bitmaps, both a distance measure and a mean calculation method have to be defined. The number of 1s in common between elements of the same cluster must be maximized, in order to to minimize the 1s in the binary *OR*. If b_1 and b_2 are two bitmaps, then the distance between b_1 and b_2, denoted $d(b_1, b_2)$, is defined in the current proposal as follows.

$$d(b_1, b_2) = \begin{cases} 0, & \text{if } b_1 \subseteq b_2 \vee b_2 \subseteq b_1. \\ count1s(b_1 \oplus b_2), & \text{otherwise.} \end{cases} \quad (1)$$

In the above definition, $b_i \subseteq b_j$ is used to denote that b_i is binary contained in b_j, i.e., $b_i \wedge b_j = b_i$. The *XOR* binary operation between two bitmaps b_i and b_j is denoted by $b_i \oplus b_j$, and $count1s(b)$ denotes the number of 1s of a bitmap b. It is noticed that the above definition of distance between bitmaps is based on the well-known Hamming distance [9], that counts the number of 1s in the *XOR* of the bitmaps.

The mean calculation method enables obtaining a representative centroid bitmap from all the bitmaps of a cluster. If $zeros(i)$ is the number of 0s in position i of all the bitmaps of the cluster and $ones(i)$ is the number of 1s in position i of all the bitmaps of the cluster, then value of the bit at position i of the mean of the bitmaps of the cluster is 0 if $zeros(i) > ones(i)$ and 1 otherwise.

The calculation of distance and mean between bitmaps if illustrated in Figure 8 through various examples. Note that KM-CTI is an in-memory index, therefore a binary tree is expected to offer better results than trees of higher order. The value of K, however, can be increased to obtain an structure suitable for secondary storage, as it is the B-tree.

As it is the case of the tree based indexes of conventional alphanumerical data, KM-CTI is expected to improve the performance of the sequential search adopted by CT-Index, if the query is selective enough. Therefore, it is expected that KM-CTI will behave better with large query graphs than with small ones.

6 EVALUATION

In this section, the experiments performed and their results are described and discussed. The system setup, datasets and queries are described in a first subsection. Next, the benchmark undertaken to select the appropriate values for maximum path length

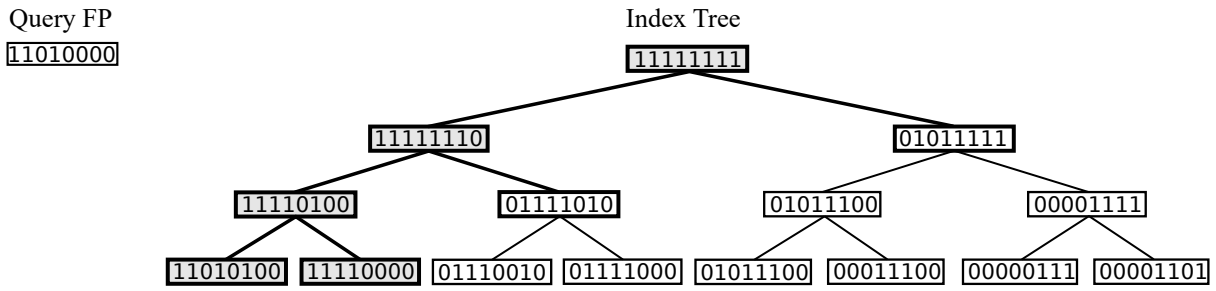


Figure 7: Example of filtering stage in KM-CTI. Highlighted nodes represent the tested ones, and the nodes filled with color are the ones that passed the test.

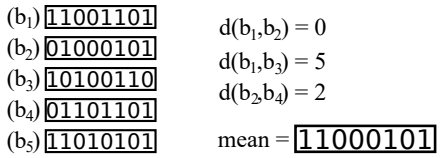


Figure 8: Example of distance and mean measures between bitmaps used in KM-CTI.

and fingerprint size is described. Finally, the last subsection is devoted to the comparison of the performance of the various techniques, with different query and database sizes.

6.1 System Setup, Datasets and Queries

The experiments were performed on a CentOS Linux release 7.4.1708, with 2 processors Intel(R) Xeon(R) CPU E5-2630 v4 (2.20 GHz, 10 cores) and 384 GB RAM. The GGSX implementation in C++ was obtained from the authors, and reimplemented in Java. For the CT-Index method, a jar was obtained from one of the authors website, and then a reverse engineering technique was applied in order to obtain the code. All the new methods were implemented in Java, reusing the above two base implementations. The experiment executions were executed using a JVM with 32GB. The verification stage in BM-GGSX, CW-CTI, and KM-CTI methods was performed using 10 threads.

Two different datasets were used to construct databases of different sizes for the experiments. The AIDS[2] dataset has already been used to evaluate other previous works of the state of the art [2, 8, 13, 15, 16]. It is composed of 42689 graphs, with an average of 45.70 vertices and 47.71 edges per graph. PubChem[3] is a molecular dataset that has currently around 96 million compounds, with an average of 41.40 vertices and 42.16 edges per graph.

In both experiments queries of sizes ranging from 4 to 40 edges (4, 8, 12, 16, 20, 24, 28, 32, 36, 40) were used. For each size, a set of 1000 query graphs was constructed. The construction of the queries was performed in the same way as in previous works [13, 15], by extracting subgraphs from the dataset. For each query, a graph was randomly selected according to a uniform distribution. A vertex of the graph was randomly selected according to a uniform distribution to act as starting point from which to make the query graph grow. From this starting vertex, a random tree was created until the desired size was reached. The size of the query was given by the number of edges of its graph. Two different sets of queries were constructed. One for the AIDS dataset, and the other for the PubChem dataset.

[2]https://cactus.nci.nih.gov/download/nci/
[3]https://pubchem.ncbi.nlm.nih.gov/

6.2 Parameter Selection

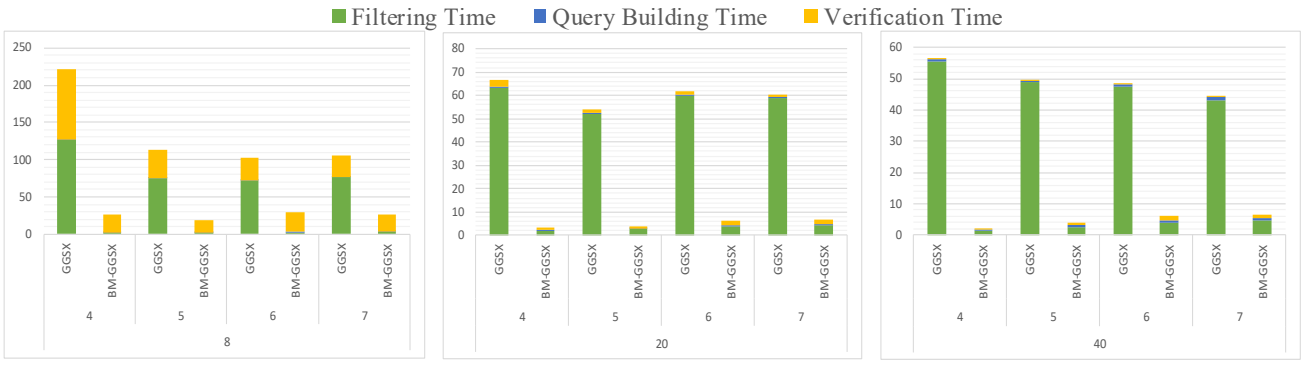
The objective of this first experiment was twofold. The first aim is to test the behavior of the different techniques, including the two methods obtained from the state of the art (GGSX and CT-Index), and the three new proposals, with two different datasets and query sizes. At the same time, the experiment aims at testing the performance of each technique with different values of its parameters, including maximum path length for all the methods and fingerprint size for those based in CT-Index.

The experiment was executed on two databases, AIDS on the one hand and 200k molecular structures obtained from PubChem (PubChem200k) on the other hand. The 1000 queries extracted from each dataset were evaluated 5 times with each method, and the experiment was repeated 4 times at different moments. As it was expected, the size and building time of the BM-GGSX trie structure increases with the maximum path length used. Regarding the CT-Index based methods, the size of the indexes is only affected by the fingerprint size, whereas both fingerprint size and maximum path length have a negative impact on the index building time.

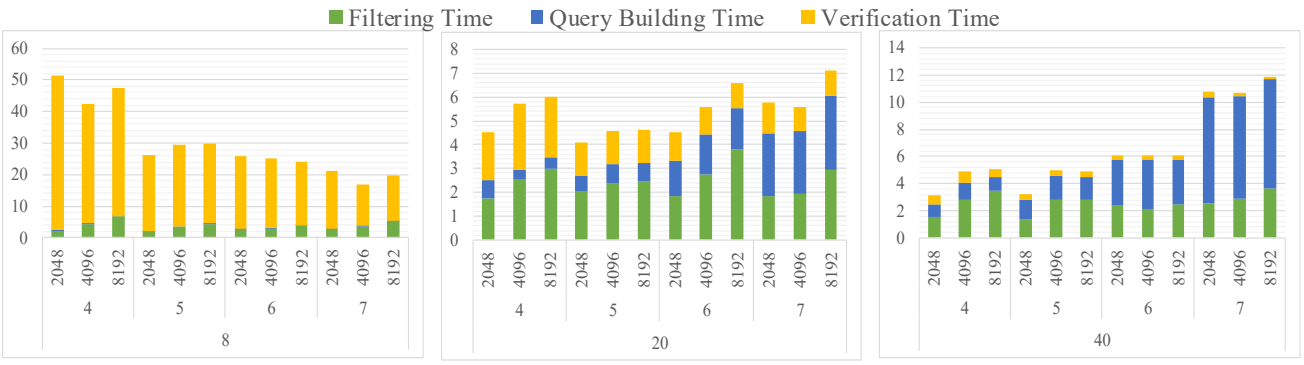
Regarding query response time, Figure 9(a) shows the comparison (subdivided into filtering, query building and verification times) between the original GGSX and BM-GGSX on the AIDS database, using different maximum path lengths and query sizes. As it is shown in the figure, the reduction in filtering time achieved by BM-GGSX is huge (around 90% of reduction in many cases). Due to this large difference, the original GGSX method was not considered for subsequent experiments. As it can be observed in the figure, using different values for the maximum path length (4, 5, 6 and 7) does not show to have a significant impact in the query response time.

Figures 9(b-d) show the query response times obtained with the three CT-Index based methods on the AIDS database, using different maximum path lengths, fingerprint sizes and query sizes. In general, for small queries, where the query building time is not important, a maximum path length higher than 4 offers better results, whereas the fingerprint size does not show a significant impact in the response time. For larger queries, the query building time becomes very significant, due to the few candidates retrieved from such a small database, and therefore, increasing the maximum path length impacts negatively in the response time.

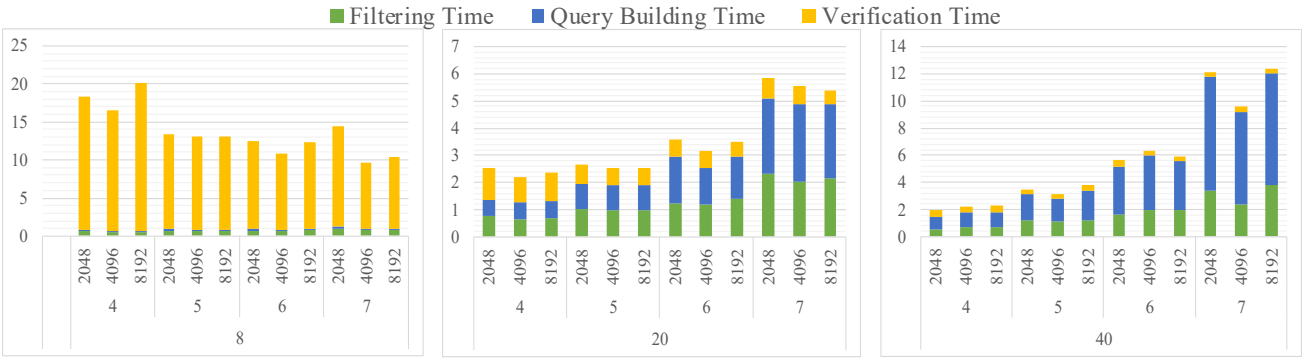
The results of the experiment on the PubChem200k database are illustrated in Figures 10(a-d). In particular, Figure 10(a) shows



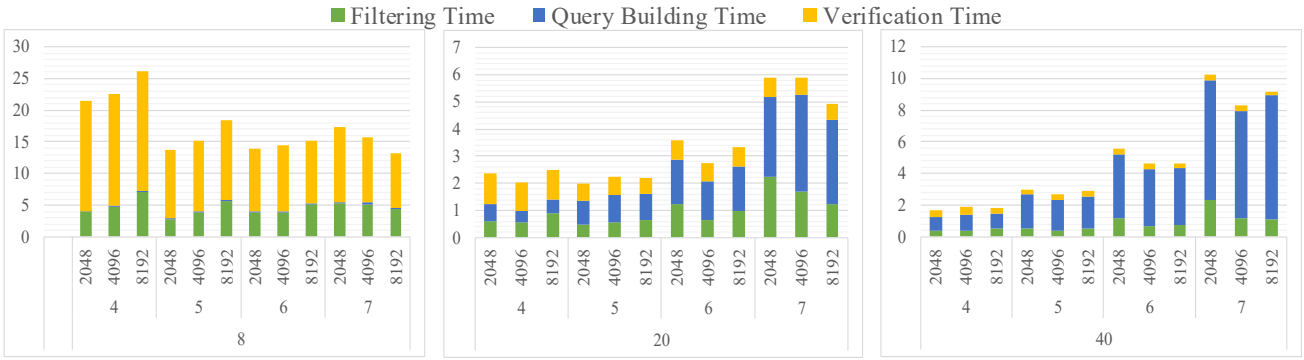
(a) Comparison of response times (ms) between GGSX and BM-GGSX on AIDS for different maximum path lengths and query sizes.



(b) Response times (ms) of CT-Index on AIDS for different fingerprint sizes, maximum tree lengths and query sizes.

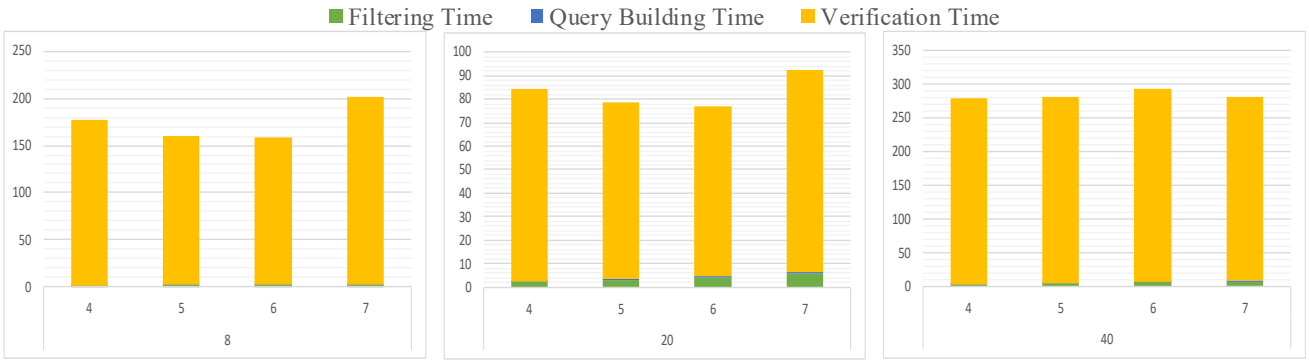


(c) Response times (ms) of CW-CTI on AIDS for different fingerprint sizes, maximum tree lengths and query sizes.

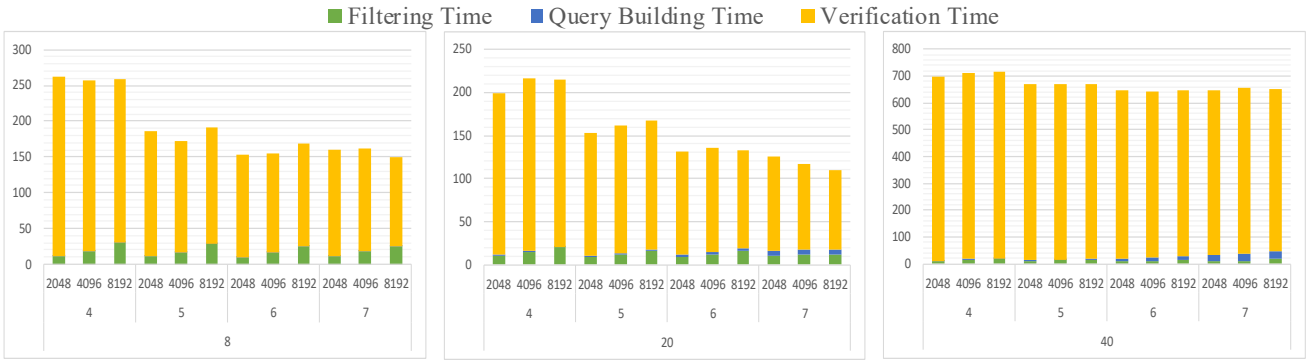


(d) Response times (ms) of KM-CTI on AIDS for different fingerprint sizes, maximum tree lengths and query sizes.

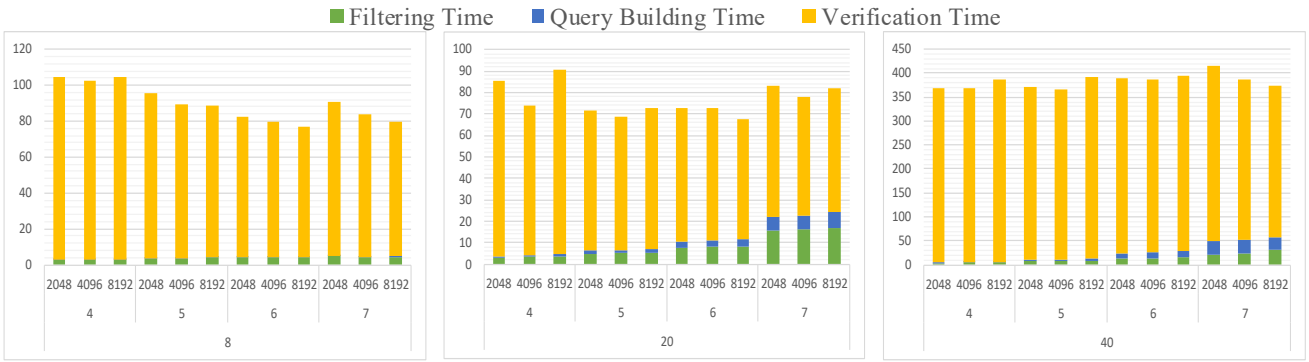
Figure 9: Response times of the methods on the AIDS database using different parameters.



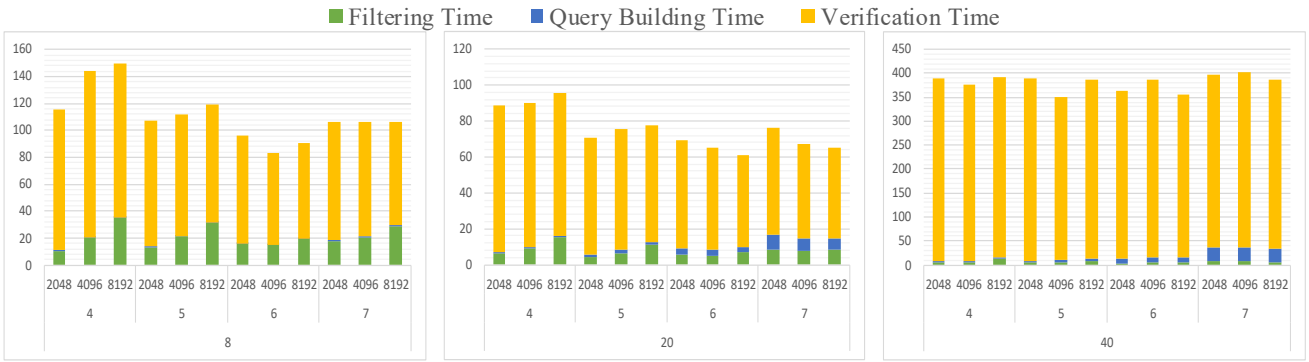
(a) Response times (ms) of BM-GGSX on Pubchem200k for different fingerprint sizes, maximum tree lengths and query sizes.



(b) Response times (ms) of CT-Index on Pubchem200k for different fingerprint sizes, maximum tree lengths and query sizes.



(c) Response times (ms) of CW-CTI on Pubchem200k for different fingerprint sizes, maximum tree lengths and query sizes.



(d) Response times (ms) of KM-CTI on Pubchem200k for different fingerprint sizes, maximum tree lengths and query sizes.

Figure 10: Response times of the methods on the PubChem200k database using different parameters.

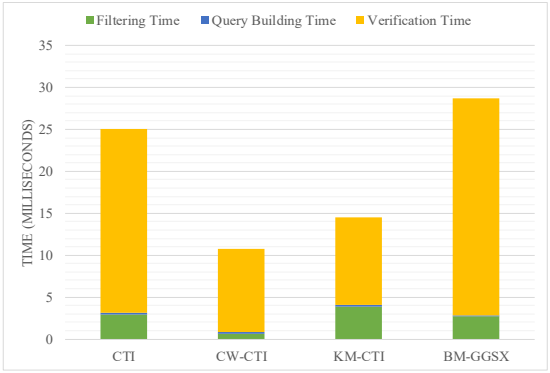
the response times obtained by BM-GGSX, with different maximum path lengths and with different query sizes. As it may be observed, the way the queries are constructed and the the much larger size of the database, compared to AIDS, causes the verification time to be the most significant one in this experiment. Again, the maximum path length does not show a clear impact in the total response time for BM-GGSX. Regarding the methods based on CT-Index (see Figures 10(b-d)), again, significant differences cannot be observed in general between different values of fingerprint size and maximum path length, although maximum path lengths higher than 4 provide slightly better results in most cases.

Based on the above experimental results, and also taking into account relevant decisions made in a previous survey [13], in the subsequent experiment we decided to use the same values of 6 for the maximum path length and of 4096 for the fingerprint size, for all the methods. The objective of such new experiment is the comparison of the performance of all the methods (except the discarded original GGSX), for increasing database and query sizes and the results and relevant discussion are given in the next subsection.

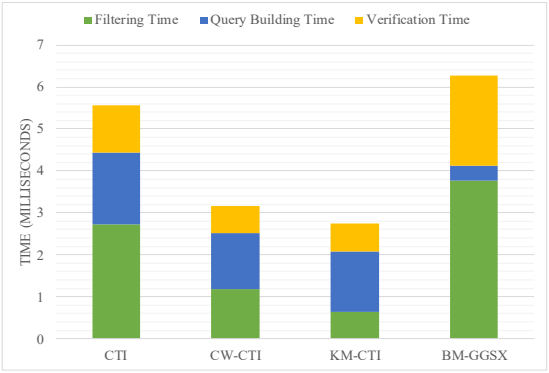
6.3 Performance comparison

In this section, the results of the performance comparison of all the methods (except the discarded GGSX) are discussed. As it was already stated, a maximum path length of 6 and a fingerprint size of 4096 were chosen. First, the results of the experiments described in the previous subsection are used to compare the performance of all the methods on the AIDS database. Figures 11(a-c) show the filtering, query building and verification times of queries of different sizes on the AIDS database. As it may be observed in Figure 11(a), CW-CTI has the best performance for small queries, both in filtering time and also in total response time. This is due to the few intersections between compressed bitmaps that the technique has to perform when few 1s are present in the query fingerprint. However, as the query size increases, the performance of CW-CTI deteriorates, reaching the same performance of CTI for large queries, whereas the performance of KM-CTI improves. The reason is that, with large queries, with many 1s in their fingerprints, many branches of the KM-CTI binary tree of bitmaps will be discarded during the filtering stage.

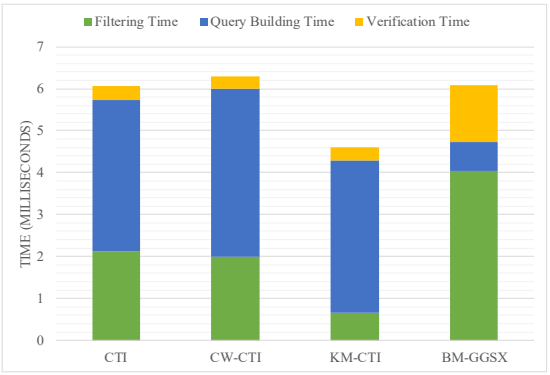
A second experiment was undertaken to compare all the methods, except the discarded GGSX, with increasing database and query sizes. To achieve this, 10 databases with sizes ranging from 100k to one million graphs were extracted from the PubChem dataset. All the index structures were created for each database, and relevant index size and index building time results were obtained. Figure 12 shows the index size and index building time for each of the analyzed methods. Index building time scales well for CT-Index and CW-CTI methods, growing linearly, and ranging from less than 6 minutes to index 100k graphs, to around 1 hour to index the largest dataset of one million graphs. KM-CTI index building time grows linearly as well, but with a higher slope than CT-Index and CW-CTI methods, being more than 3 times slower for the largest dataset. This is due to the recursive application of the K-Means method to construct the binary tree of bitmaps. BM-GGSX has the worst performance of all methods, taking more time to index 200k graphs, than CT-Index and CW-CTI methods to index one million. Its index building time increases drastically



(a) 8 Edges Queries



(b) 20 Edges Queries

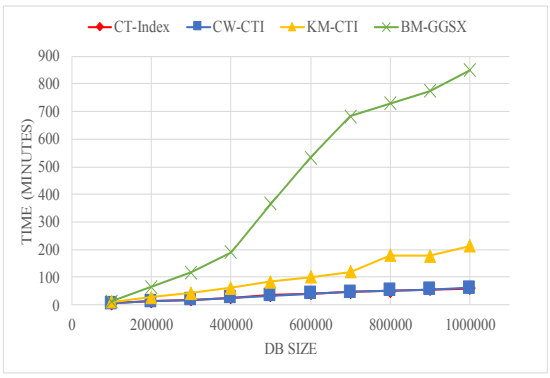


(c) 40 Edges Queries

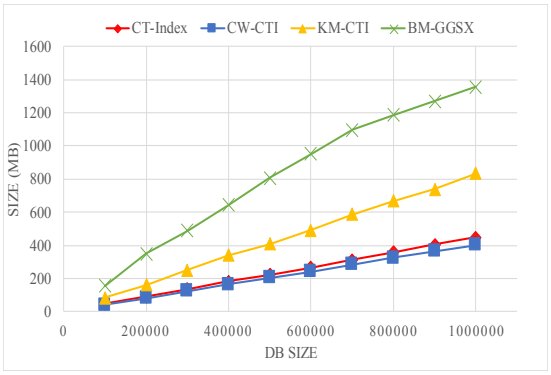
Figure 11: Performance comparison of the methods on the AIDS database.

with the size of the dataset, and it needed more than 14 hours to index the largest database of one million graphs.

Regarding the index size, BM-GGSX performs again the worst among all the proposed methods, showing a big slope in its scalability plot, going from 155MB in the shortest dataset to almost 1.4GB in the largest one. CW-CTI performs the best, scaling slightly better than CT-Index method, due to the use of compressed bitmaps in its storage structure. KM-CTI doubles the space of CW-CTI and CT-Index, due to the additional bitmaps stored in the internal nodes of its binary tree. However, its size is still almost half of that required by BM-GGSX.

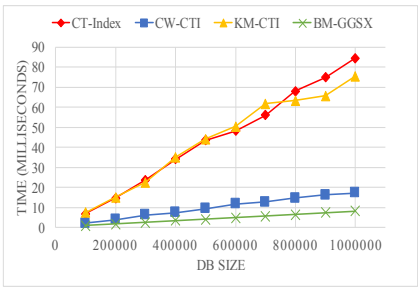


(a) Index Building Time

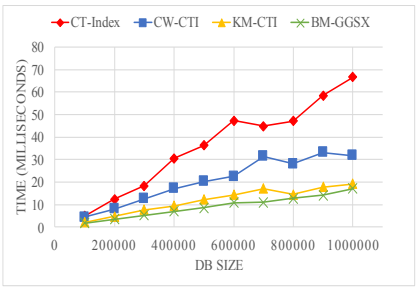


(b) Index Size

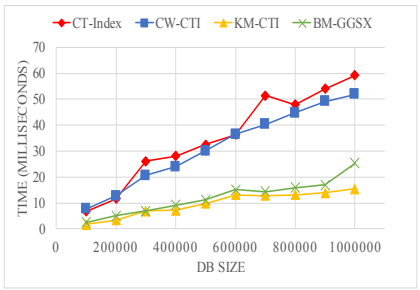
Figure 12: Index Size and Building Time for increasing database sizes.



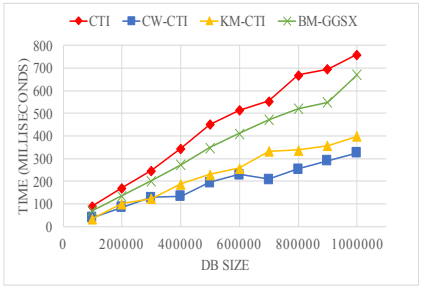
(a) 8 Edges Query Filtering Response Time



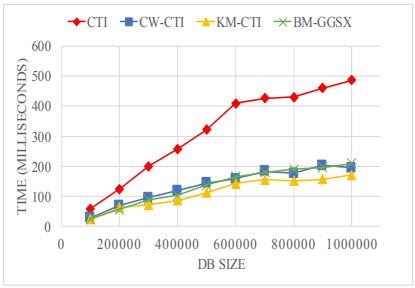
(b) 20 Edges Query Filtering Response Time



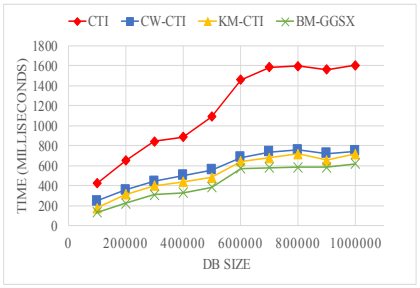
(c) 40 Edges Query Filtering Response Time



(d) 8 Edges Total Query Response Time



(e) 20 Edges Total Query Response Time



(f) 40 Edges Total Query Response Time

Figure 13: Filtering time and total query response time for increasing database and query sizes.

Once the indexes were created, the 1000 queries of each size (from 4 edges to 40 edges), generated for the previous experiment from the 200k PubChem database, were executed four times for each database size, and the whole experiment was repeated three times in different moments. Figures 13(a-c) depict the filtering time of each method. It is reminded that the main contribution of the present work is on the index structures used in the filtering stage of the methods. BM-GGSX and CW-CTI obtain the fastest filtering time for small queries (around 80% of reduction with respect to CTI), as it is shown in Figure 13(a), whereas KM-CTI and CTI offer a similar performance. It is reminded that CW-CTI was expected to behave well with small queries that have few 1s in their fingerprints. On the other hand, small not selective queries, with few 1s in their fingerprints, do not leverage the binary tree of KM-CTI. As the query size increases, CTI and BM-GGSX keep their performance figures, however, CW-CTI and KM-CTI invert their positions. With queries of large sizes, with many 1s in their fingerprints, CW-CTI has to perform many intersections between large compressed bitmaps. On the other hand, KM-CTI behaves better, as many branches of its binary tree are discarded during the evaluation. This situation is clearly shown in Figures 13(b) (around 70% of reduction with respect to CTI) and 13(c) (around 75% of reduction with respect to CTI). These results are completely aligned with those already described for the AIDS database above.

To complete the analysis of the results, the total query response times are shown in Figures 13(d-f). First, looking at the scale of the total response times, it is noticed that, in these databases, contrary to what it was observed with AIDS, the filtering time does not have a great impact on the total response time, since the verification time is the largest one by difference. In spite of this, it is shown how the three methods proposed in the present work outperform the CT-Index method. This is mainly due to the parallel execution in a multi-thread architecture of the VF2

subgraph isomorphism algorithm. It is also noticed that, BM-GGSX, which had the best results in filtering time for all the query sizes, does not show a good total response time for small queries. The reason for this lost of performance is the use of paths as graph features in BM-GGSX, contrary to the use of trees in CT-Index based techniques, which enable the generation of a larger number of different features in small graphs, reducing the number of candidates to verify. The small differences between CW-CTI and KM-CTI are caused by the already mentioned differences in filtering time, since both the effectiveness of their filtering stage and their verification stage are identical.

7 CONCLUSIONS

In this paper, three new FTV methods for subgraph search on large databases of small graphs are proposed. The methods are based on already existing state of the art solutions, namely GGSX and CT-Index, and provide improvements both in filtering and verification response times. BM-GGSX improves GGSX through the incorporation of compressed bitmaps for the representation of graph references, and reaches the best filtering times for all types of queries and the best total response times for medium and large queries. However, it does not improve the bad figures that GGSX already have in both index size and index building time. CW-CTI exploits also compressed bitmap inside a column-wise structure for the storage of the CT-Index fingerprints, obtaining very good performance for small size queries. KM-CTI uses the K-Means clustering method on CT-Index fingerprints to construct a binary tree of bitmaps, which is used during the filtering stage to reduce the required fingerprint comparisons. KM-CTI has very good performance for medium and large queries, with the cost of increasing the index size and building time with respect to the original CT-Index, but keeping the figures much lower than those reached by BM-GGSX. Thus, a combination of CW-CTI and KM-CTI covers all the query sizes with performances either similar of better than those of BM-GGSX, but keeping both index size and building times inside a reasonable range.

Future research work is related to the implementation of the proposed techniques in distributed large scale data processing architectures, to reach reasonable performances for databases of sizes larger than 100 million graphs, as it will be the case of the PubChem dataset in the near future. The results will be incorporated in a query engine prototype that is being implemented in the scope of the NEXTCHROM project, co-funded by the Spanish government and the company Mestrelab Research S.L.

ACKNOWLEDGMENTS

This work has been co-funded by the Ministerio de Economía y Competitividad of the Spanish government, and by Mestrelab Research S.L. through the project NEXTCHROM (RTC-2015-3812-2) of the call Retos-Colaboración of the program Programa Estatal de Investigación, Desarrollo e Innovación Orientada a los Retos de la Sociedad.

REFERENCES

[1] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang. 2016. Efficient subgraph matching by postponing Cartesian products. In *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*. ACM, 1199–1214.

[2] V. Bonnici, A. Ferro, R. Giugno, A. Pulvirenti, and D. Shasha. 2010. Enhancing graph database indexing by suffix tree structure. In *IAPR International Conference on Pattern Recognition in Bioinformatics*. Springer, 195–203.

[3] J. Cheng, Y. Ke, W. Ng, and A. Lu. 2007. FG-Index: Towards verification-free query processing on graph databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 857–872.

[4] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. 2004. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26, 10 (2004), 1367–1372.

[5] H.-C. Ehrlich and M. Rarey. 2012. Systematic benchmark of substructure search in molecular graphs - From Ullmann to VF2. *Journal of Cheminformatics* 4, 1 (31 Jul 2012), 13. DOI: http://dx.doi.org/10.1186/1758-2946-4-13

[6] H.-C. Ehrlich, A. Volkamer, and M. Rarey. 2012. Searching for substructures in fragment spaces. *Journal of Chemical Information and Modeling* 52, 12 (2012), 3181–3189. DOI: http://dx.doi.org/10.1021/ci300283a PMID: 23205736.

[7] E. Forgy. 1965. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics* 21 (1965), 768–780.

[8] R. Giugno, V. Bonnici, N. Bombieri, A. Pulvirenti, A. Ferro, and D. Shasha. 2013. GRAPES: A software for parallel searching on biological graphs targeting multi-more architectures. *PLoS ONE* 8, 10 (2013).

[9] R. W. Hamming. 1950. Error detecting and error correcting codes. *Bell System Technical Journal* 29, 2 (1950), 147–160.

[10] W.-S. Han, J. Lee, and J.-H. Lee. 2013. TurboISO: Towards ultrafast and robust subgraph isomorphism search in large graph databases. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 337–348.

[11] W.-S. Han, J. Lee, M.-D. Pham, and J.X. Yu. 2010. iGraph: A framework for comparisons of disk based graph indexing techniques. *Proceedings of the VLDB Endowment* 3, 1 (2010), 449–459.

[12] H. He and A. K. Singh. 2008. Graphs-at-a-time: Query language and access methods for graph databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 405–417.

[13] F. Katsarou, N. Ntarmos, and P. Triantafillou. 2015. Performance and scalability of indexed subgraph query processing methods. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1566–1577.

[14] F. Katsarou, N. Ntarmos, and P. Triantafillou. 2017. Subgraph querying with parallel use of query rewritings and alternative algorithms. In *Advances in Database Technology - EDBT*, Vol. 2017-March. 25–36.

[15] K. Klein, N. Kriege, and P. Mutzel. 2011. CT-Index: Fingerprint-based graph indexing combining cycles and trees. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. IEEE, 1115–1126.

[16] J. Lee, W.-S. Han, R. Kasperovics, and J.-H. Lee. 2012. An in-depth comparison of subgraph isomorphism algorithms in graph databases. In *Proceedings of the VLDB Endowment*, Vol. 6. 133–144.

[17] D. Lemire, O. Kaser, and K. Aouiche. 2010. Sorting improves word-aligned bitmap indexes. *Data and Knowledge Engineering* 69, 1 (2010), 3–28.

[18] J. MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. University of California Press, Berkeley, 281–297.

[19] X. Ren and J. Wang. 2015. Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs. *Proceedings of the VLDB Endowment* 8, 5 (2015), 617–628.

[20] H. Shang, Y. Zhang, X. Lin, and J. X. Yu. 2008. Taming verification hardness: An efficient algorithm for testing subgraph isomorphism. *Proceedings of the VLDB Endowment* 1, 1 (2008), 364–375.

[21] J. R. Ullmann. 1976. An algorithm for subgraph isomorphism. *J. ACM* 23, 1 (1976), 31–42.

[22] J. Wang, N. Ntarmos, and P. Triantafillou. 2016. Indexing query graphs to speedup graph query processing. In *Advances in Database Technology - EDBT*, Vol. 2016-March. 41–52.

[23] J. Wang, N. Ntarmos, and P. Triantafillou. 2017. GraphCache: A caching system for graph queries. In *Advances in Database Technology - EDBT*, Vol. 2017-March. 13–24.

[24] X. Yan, P.S. Yu, and J. Han. 2004. Graph indexing: A frequent structure-based approach. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 335–346.

[25] S. Zhang, M. Hu, and J. Yang. 2007. TreePi: A novel graph indexing method. In *Proceedings - International Conference on Data Engineering*. 966–975.

[26] S. Zhang, S. Li, and J. Yang. 2009. GADDI: Distance index based subgraph matching in biological networks. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, EDBT'09*. 192–203.

[27] P. Zhao and J. Han. 2010. On graph query optimization in large networks. *Proceedings of the VLDB Endowment* 3, 1 (2010), 340–351.

[28] P. Zhao, J. X. Yu, and P. S. Yu. 2007. Graph indexing: Tree + Delta >= Graph. In *33rd International Conference on Very Large Data Bases, VLDB 2007 - Conference Proceedings*. 938–949.

[29] L. Zou, L. Chen, J. X. Yu, and Y. Lu. 2008. A novel spectral coding in a large graph database. In *Advances in Database Technology - EDBT 2008 - 11th International Conference on Extending Database Technology, Proceedings*. 181–192.