INTERNATIONAL DOCTORAL
SCHOOL OF THE USC

David
Luaces Cachaza

PhD Thesis

EFFICIENT QUERY OVER LARGE DATASETS OF ANALYTICAL CHEMISTRY

Santiago de Compostela, 2023

Doctoral Programme in Information Technology Research

TESE DE DOUTORAMENTO

EFFICIENT QUERY OVER LARGE DATASETS OF ANALYTICAL CHEMISTRY

David Luaces Cachaza

ESCOLA DE DOUTORAMENTO INTERNACIONAL DA UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

PROGRAMA DE DOUTORAMENTO EN INVESTIGACIÓN EN TECNOLOXÍAS DA INFORMACIÓN

SANTIAGO DE COMPOSTELA

2023

DECLARACIÓN DO AUTOR/A DA TESE

D./Dna. **David Luaces Cachaza**

Título da tese: **EFFICIENT QUERY OVER LARGE DATASETS OF ANALYTICAL CHEMISTRY**

Presento a miña tese, seguindo o procedemento axeitado ao Regulamento, e declaro que:

1) A tese abarca os resultados da elaboración do meu traballo.
2) De ser o caso, na tese faise referencia ás colaboracións que tivo este traballo.
3) Confirmo que a tese non incorre en ningún tipo de plaxio doutros autores nin de traballos presentados por min para a obtención doutros títulos.
4) A tese é a versión definitiva presentada para a súa defensa e coincide a versión impresa coa presentada en formato electrónico

E comprométome a presentar o Compromiso Documental de Supervisión no caso de que o orixinal non estea na Escola.

En **Santiago de Compostela**, **20 de Febreiro de 2023**.

Sinatura electrónica

AUTORIZACIÓN DO DIRECTOR/TITOR DA TESE

D./Dna. **José Ramón Ríos Viqueira**

En condición de: **Director/a**

Título da tese: **EFFICIENT QUERY OVER LARGE DATASETS OF ANALYTICAL CHEMISTRY**

INFORMA:

Que a presente tese, correspóndese co traballo realizado por D/Dna David Luaces Cachaza, baixo a miña dirección/titorización, e autorizo a súa presentación, considerando que reúne os requisitos esixidos no Regulamento de Estudos de Doutoramento da USC, e que como director/titor desta non incorre nas causas de abstención establecidas na Lei 40/2015.

En **Santiago de Compostela**, **20 de Febreiro de 2023**

Sinatura electrónica

AUTORIZACIÓN DO DIRECTOR/TITOR DA TESE

D./Dna. **Tomás Fernández Pena**

En condición de: **Director/a**

Título da tese: **EFFICIENT QUERY OVER LARGE DATASETS OF ANALYTICAL CHEMISTRY**

INFORMA:

Que a presente tese, correspóndese co traballo realizado por D/Dna David Luaces Cachaza, baixo a miña dirección/titorización, e autorizo a súa presentación, considerando que reúne os requisitos esixidos no Regulamento de Estudos de Doutoramento da USC, e que como director/titor desta non incorre nas causas de abstención establecidas na Lei 40/2015.

En **Santiago de Compostela**, **20 de Febreiro de 2023**

Sinatura electrónica

Á miña familia

The past is a stepping stone, not a milestone.

Robert Plant

Chega o final dunha longa etapa, e tras moitos anos de traballo gustaríame agradecerlle ás persoas e entidades sen as que este camiño non tería sido posible.

Primeiramente gustaríame darlle as grazas aos meus directores de tese. Tomás, que me axudou moito sobre todo no principio da tese, e especialmente a José Ramón, por ensinarme e guiarme cada día, non só no ámbito académico, e tamén axudarme a dar ese empuxón extra nalgúns deses momentos nos que as cousas non saían.

Tamén me gustaría agradecer a todos os compañeiros do grupo COGRADE, por facerme sentir un máis da familia dende un primeiro momento e polo seu apoio durante estes anos.

Grazas ao CiTIUS por brindarme as ferramentas e os espazos necesarios para realizar este traballo, e a todo o persoal de apoio pola súa axuda tanto en temas técnicos como burocráticos.

Me gustaría agradecer también al Dr. Gonzalo Navarro por supervisar mi estancia en la Universidad de Chile y darme la oportunidad de asistir a sus clases y visitar y conocer un país increíble.

Grazas tamén aos diferentes proxectos que contribuíron á financiación para que este traballo fose posible:

- Nextchrom: Virtual advanced chromatography for an integral management of analytical and molecular data (RTC-2015-3812-2). Agradecemento especial aos compañeiros de Mestrelab pola cesión do seu software aínda despois de ter finalizado o proxecto.
- GEOARPAD: Cultural heritage of the Euroregion Galicia-North of Portugal: valorisation and innovation (Programa INTERREG España – Portugal POCTEP).
- SenseStore1: Efficient storage and publication of remote sensing data (Civil UAVs Initiative: Fase A, Xunta de Galicia - Axencia Galega de Innovación (GAIN), PO FEDER Galicia 2014-2020).
- TRAFAIR: Understanding Traffic Flows to Improve Air quality (CEF-TC-2017-3 – Public Open Data, European Commission - Research & Innovation).

Por último, pero non menos importante, darlle as grazas á miña familia e amigos, por darme o seu apoio incondicional todos estes anos sen cal isto tería sido imposible.

20 de febreiro de 2023

Contents

Resumo **1**

Abstract **11**

1 Introduction **15**
1.1 Background and motivation 15
1.2 Objectives and contributions of the Thesis 17
1.3 Projects and publications 20
1.4 Outline of the Thesis 21

2 Querying Molecular Data: The NEXTCHROM Project **23**
2.1 Technologies Available in Cheminformatics 26
2.2 Text Searching 26
2.3 Peak Searching 28
2.4 Molform Searching 29
2.5 Substructure Searching 31

3 Background and Related Work on Subgraph Searching **33**
3.1 Problem Definition 33
3.2 State of the art 36

4 Subgraph Searching Techniques for Centralised Architectures **47**
4.1 Bitmap GGSX 48
4.2 Column-Wise CT-Index 51
4.3 K-Means CT-Index 53

4.4 Performance Evaluation 59

5 Generic Framework for Subgraph Searching in Distributed Architectures 79
5.1 Index Building 80
5.2 Query Processing 84
5.3 Performance Evaluation 87

6 Approximate Processing for Interactive Searching in Molecular Datasets 101
6.1 Interactive Searching in Molecular Datasets 101
6.2 Approximate Processing Solutions 104
6.3 Experimental Evaluation of Efficacy and Efficiency 110

7 Conclusions and Future Work 121

Bibliography 125

List of Figures 131

Resumo

A xestión eficiente de datos analíticos tales como os xerados pola cromatografía líquida (LC) e gasosa (GC), os espectros de Resonancia Magnética Nuclear (NMR), infravermellos (IR), ultravioleta (UV), espectrometría de masas (MS), ou as estruturas moleculares, é unha das aplicacións máis atractivas e demandadas por sectores como o químico, o biotecnolóxico ou o farmacéutico nos últimos anos. Un dos principais problemas atopados nas investigacións sobre o descubrimento de fármacos baseados en produtos naturais é a identificación e eliminación de réplicas de compostos coñecidos. Esta capacidade de identificar compostos coñecidos nunha etapa temperá da investigación representa un grande aforro de tempo e esforzo. Para que este proceso de identificación sexa efectivo, é necesario dispoñer das técnicas quimioinformáticas oportunas capaces de realizar buscas rápidas de subestruturas, espectros e cromatogramas en grandes volumes de datos experimentais.

Os sistemas de bases de datos especializados en información de experimentos analíticos son claves no traballo rutineiro dun gran número de empresas do sector químico e farmacéutico para facilitar procesos como os de separación, identificación e cuantificación dos ingredientes nunha formulación, entre outros. Por outra banda, o almacenamento e busca de información analítica debe complementarse mediante a utilización de bases de datos de estruturas moleculares, as cales deben posuír as capacidades necesarias para realizar buscas tanto de datos convencionais (alfanuméricos), como de estruturas moleculares, espectros ou cromatogramas. Na actualidade existe unha gran variedade de bibliotecas quimioinformáticas e de solucións para incorporar parte da funcionalidade de estas bibliotecas dentro de sistemas xestores de bases de datos. Algúns exemplos son bibliotecas como RDKit, OpenBabel ou CDK. Neste sentido, a tecnoloxía usada, baseada en bases de datos relacionais e arquitecturas centralizadas, non é óptima en termos de rendemento e escalabilidade e está limitando a posibilidade de cubrir todo o abano de aplicacións demandadas actualmente polo mercado.

Por outra banda, na área de xestión de grandes volumes de datos están a aparecer un conxunto de tecnoloxías que teñen como obxectivo facilitar a xestión e o procesamento analítico escalable de grandes conxuntos de datos de natureza posiblemente heteroxénea (Big Data). Estas tecnoloxías resolven problemas relacionados cos parámetros de volume, velocidade de actualización e variabilidade dos conxuntos de datos. Nas aplicacións de quimioinformática descritas arriba, o volume de datos é enorme e a variabilidade da estrutura dos mesmos tamén, polo que a incorporación de técnicas de Big Data no seu desenvolvemento é algo fundamental.

As bases de datos moleculares almacenan o que se pode definir como entidades moleculares. Unha entidade molecular contén as propiedades dunha molécula, e está composta de diferentes tipos de datos.

- *Propiedades alfanuméricas*: inclúen propiedades como nome molecular, peso molecular ou número de átomos que contén a molécula.
- *Fórmula molecular*: conxunto de pares compostos polo elemento químico e o número de átomos dese elemento presentes na molécula.
- *Estrutura molecular*: un grafo representativo da estrutura molecular, onde os vértices representan os átomos e os arcos representan a unión entre os átomos.
- *Espectro*: estruturas que almacenan os datos necesarios para representar diferentes tipos de espectros, incluíndo resonancia magnética nuclear (NMR), cromatogramas ou espectros de masas.

Tendo en conta os tipos de datos almacenados nas bases de datos moleculares, os tipos de consulta máis demandados son os seguintes.

- *Consultas de texto*: devolve todas as entidades moleculares que conteñan ou teñan exactamente un texto dado nalgunha das súas propiedades.
- *Consultas numéricas*: devolve todas as entidades moleculares que teñan algunha propiedade numérica con valor entre un rango dado.
- *Consultas de picos*: devolve as entidades moleculares que conteñan algún espectro que cumpra as restricións dadas na consulta.
- *Consultas de fórmula molecular*: devolve todas as entidades moleculares que inclúan na súa fórmula molecular os elementos e número de átomos requiridos.

- *Consultas de subestrutura*: devolve todas as entidades moleculares que conteñan dentro da súa estrutura molecular a consulta.

Entre todos os tipos de consulta existentes para as bases de datos moleculares, unha das máis demandadas pola industria e máis desafiantes dende o punto de vista investigador son as buscas por subestrutura, é dicir, recuperar todas as entidades moleculares da base de datos que conteñan na súa estrutura molecular a estrutura molecular dada como consulta. As estruturas moleculares poden codificarse como grafos de pequeno tamaño onde os vértices representan os átomos e os arcos representan os enlaces moleculares entre os átomos. Obter todas as instancias que existen dentro dun grafo dun subgrafo dado como consulta é un problema ben coñecido, chamado *subgraph matching* na literatura. Trátase dun problema NP completo e resólvese habitualmente mediante a aplicación de algoritmos de isomorfismo de subgrafos. Os algoritmos de isomorfismo de subgrafos tenden a ter bos rendementos en bases de datos compostas por unicamente un grafo de gran tamaño, como por exemplo o tipo de base de datos que se pode atopar na web semántica.

Pola contra, as bases de datos moleculares están compostas por norma xeral por un gran número de grafos de pequeno tamaño. Recuperar todos os grafos dunha base de datos que conteñan ao subgrafo dado como consulta é tamén un problema ben coñecido na comunidade, e é chamado *subgraph decision* na literatura. Este problema pode ser resolto dunha maneira directa mediante a aplicación de algoritmos de isomorfismo de grafos, como no caso dos anteriores, aínda que normalmente estes algoritmos sufrirán de severos problemas de rendemento cando a base de datos chega a un tamaño cun número de moléculas significativo. En xeral, unha mellor maneira de resolver o problema de subgraph decision é mediante o uso de técnicas de filtrado, coñecidas como Filter-Then-Verify (FTV) no estado da arte. As técnicas FTV dividen o proceso de consulta en dúas fases. Primeiro, unha fase de filtrado reduce significativamente o número de grafos aproveitando algún tipo de estrutura de datos usada como índice e xerada anteriormente. A segunda fase, coñecida como fase de verificación, aplica un algoritmo de isomorfismo de subgrafos ao conxunto de grafos obtido na fase anterior de filtrado.

Un gran número de propostas baseadas na estratexia FTV foron presentadas pola comunidade para resolver o problema de subgraph decision. Entre todas elas, GraphGrepSX (GGSX) e CT-Index son as que mostraron un maior rendemento para bases de datos cun gran número de grafos, como é o caso das bases de datos moleculares. Tanto GGSX como CT-Index apóian-

se na creación de estruturas de datos que actúan como índices, que se xeran con determinadas características extraídas dos grafos do conxunto de datos.

No caso do método GGSX, xérase unha estrutura de tipo trie con todos os camiños dunha determinada lonxitude máxima contidos en cada grafo do conxunto de datos. Cada nodo do índice trie representa un camiño específico e referencia a todos os grafos da base de datos que o conteñen. Á hora de realizar consultas, na fase de filtrado, GGSX comeza xerando un trie para o grafo usado como consulta, para despois realizar un percorrido en amplitude de ámbolos dous tries, o correspondente ao índice e o correspondente á consulta, para atopar os nodos folla dos camiños da consulta no trie da base de datos. Unha vez se atopan os nodos folla, obtéñense os identificadores dos grafos que conteñen eses camiños, e realízase unha intersección entre todos eles para obter un conxunto de candidatos. Este conxunto de candidatos serán analizados para comprobar si conteñen o grafo consulta na fase de verificación. A verificación de cada grafo candidato faise aplicando o algoritmo de isomorfismo de subgrafos VF2.

Por outra banda, o método CT-Index extrae as características do grafo para xerar fingerprints (bitmaps) que son almacenados nunha lista. Para xerar os fingerprints, primeiro obtén todos os subárbores e ciclos cunha lonxitude máxima dada que hai dentro de cada grafo. Cada subárbore ou ciclo codificase en texto usando unha representación canónica sobre a que despois se aplica unha función hash que obtén un número enteiro. Finalmente, eses números enteiros úsanse para identificar as posición dos bits que se van a activar no fingerprint. Á hora de realizar a consulta, na fase de filtrado, primeiro xérase un fingerprint para o grafo da consulta, e compróbase se está contido en cada un dos fingerprints almacenados no índice, para construír así o conxunto de candidatos. Finalmente, na fase de verificación os grafos candidatos compróbanse usando o algoritmo de isomorfismo de subgrafos VF2, igual que no caso anterior.

Ámbalas dúas técnicas que acabamos de describir foron aplicadas con éxito a conxuntos de datos de decenas de miles de grafos. Sen embargo, o seu rendemento non é axeitado cando as bases de datos utilizadas alcanzan tamaños de varios centos de miles, ou incluso millóns de grafos.

En base a todo o exposto anteriormente, o obxectivo desta tese é o deseño e desenvolvemento dun sistema de almacenamento, indexado e busca de datos moleculares, que permita traballar con todos os tipos de datos existentes nunha entidade molecular, incluíndo datos analíticos e estruturas moleculares. O sistema deberá traballar independentemente da tecno-

loxía de bases de datos subxacente, permitindo a súa implementación tanto nunha arquitectura centralizada como nunha arquitectura distribuída. De entre todos os tipos de consulta, o traballo enfócase principalmente nas consultas por subestrutura molecular. Para este tipo de consulta, a tese propón novas solucións que melloran o estado do arte en arquitecturas centralizadas, e tamén proporciona unha contorna para a implementación de métodos FTV en arquitecturas distribuídas que proporciona a escalabilidade necesaria para conseguir consultar bases de datos de decenas de millóns de grafos con tempos de resposta aceptables. Por último, a tese tamén analiza a posibilidade de usar a fase de filtrado das técnicas FTV, implementada con dous grandes tipos de estruturas de tipo Sketch, como solución para alcanzar os tempos de resposta precisos en interfaces de busca e exploración interactiva.

En resumo, as características do sistema resultante son as seguintes.

- O sistema ten a capacidade de realizar os seguintes tipos de busca en conxuntos de datos moleculares:
  - Consultas de texto realizadas sobre as propiedades textuais das entidades moleculares. Utilizaranse tecnoloxías de indexación e busca existentes no estado do arte.
  - Consultas de picos realizadas en espectros almacenados polas entidades moleculares. Utilizaranse técnicas de indexación multidimensional para resolvelas.
  - Consultas de fórmulas moleculares, que serán resoltas mediante a aplicación de estruturas de datos específicas.
  - Consultas de subestrutura sobre as estruturas moleculares. Utilizaranse técnicas de FTV para a súa resolución.
- O sistema pode ser despregado tanto nunha arquitectura centralizada como nunha arquitectura distribuída. No caso da arquitectura centralizada, os datos son almacenados nunha base de datos relacional. No caso da arquitectura distribuída, utilízanse clústeres distribuídos onde os datos moleculares se almacenan en bases de datos NoSQL baseadas en documentos.
- A implementación utiliza técnicas existentes no estado da arte para resolver as consultas por texto e por picos, así como unha estrutura de datos con datos en memoria para a resolución de buscas por fórmula molecular. No caso das consultas por subestrutura, tivéronse que deseñar e implementar avanzadas estruturas de indexación que permiten

un bo rendemento en bases de datos de gran tamaño (centos de miles ou incluso un millón de moléculas).

- Incorpóranse novas técnicas avanzadas para realizar buscas por subestrutura en sistemas distribuídos de maneira eficiente en bases de datos de gran tamaño, no rango de decenas de millóns de estruturas moleculares.

- Proponse o uso de dúas estruturas de datos de tipo Sketch, os Bloom Filters (BF) e o Count-Min Sketch (CMS), para implementar a fase de filtrado das técnicas FTV, e conseguir así solucións que proporcionan resultados aproximados (con falsos positivos), pero con tempos de resposta moi rápidos (por debaixo do segundo), de maneira que poidan ser utilizadas para implementar interfaces de busca e exploración interactiva sobre grandes cantidades de grafos.

Para resolver as consultas por subestrutura molecular propóñense tres novas técnicas FTV baseadas nas coñecidas GraphGrepSX (GGSX) e CT-Index. Estas tres técnicas son as seguintes.

- Bitmap GGSX (BM-GGSX): baséase no método GGSX xerando unha estrutura trie da mesma maneira, e aproveita o uso de bitmaps comprimidos para modificar o almacenamento dos identificadores das moléculas en cada nodo, obtendo así moito mellores tempos de resposta que o propio GGSX.

- Column-Wise CT-Index (CW-CTI): baséase no CT-Index e adapta o almacenamento dos fingerprints nunha estrutura de almacenamento por columnas, onde a mesma posición dun bit de todos os fingerprints de todos os grafos da base de datos son almacenados nun bitmap comprimido, conseguindo, desta maneira, que na consulta soamente as posicións correspondentes aos bits que están activos no fingerprint da consulta teñan que ser accedidos. Isto permite que se reduzan moito o número de comparacións e que obteña un gran rendemento en consultas de pequeno tamaño, e dicir, aquelas que teñen poucos bits con valor 1 no seu fingerprint.

- K-Means CT-Index (KM-CTI): baséase no CT-Index, e aplica recursivamente o algorimo de clusterización K-Means sobre os fingerprints do CT-Index para construír unha árbore binaria de fingerprints que actúe como índice. Na árbore, os nodos follas almacenan os fingerprints de cada grafo da base de datos, e os nodos pais de cada nodo

almacenan un bitmap resultado da operación binaria OR dos dous fillos. Á hora de facer a consulta, compróbase se o fingerprint da consulta está contido no nodo raíz. Se o está, repítese a comprobación nos dous nodos fillo, e así sucesivamente. Isto permite que no caso de non estar contido nun nodo, xa non se teña que comparar todos os seus descendentes.

Para avaliar estas tres novas propostas, realizáronse probas utilizando bases de datos moleculares de tamaños moito maiores que os existentes en estudos previos no estado da arte. En termos de construción do índice, BM-GGSX é o que ofrece peor resultado, tanto en tempo como en tamaño, sendo o CW-CTI o que obtén mellores resultados seguido de cerca polo CT-Index orixinal. En canto ao rendemento nas consultas, BM-GGSX obtén unha redución do 90 % no tempo de filtrado comparado co GGSX, e, comparado cos outros métodos ofrece uns bos resultados para consultas de tamaño medio e grande. CW-CTI obtén os mellores resultados para consultas de pequeno tamaño. En canto ao método KM-CTI, amosa moi bos resultados para consultas de tamaño medio e grande. En xeral, a avaliación conclúe que unha boa solución para calquera tamaño de consulta sería unha combinación dos métodos CW-CTI e KM-CTI.

O traballo propón tamén o deseño e desenvolvemento dunha contorna xenérica para a implementación de métodos FTV sobre motores de procesamento de datos distribuídas e a gran escala. Para iso, primeiramente, a base de datos divídese en particións, e entón, para cada unha delas constrúese un índice seguindo a técnica escollida. Os índices resultantes codifícanse en arrais de bytes e almacénanse en estruturas distribuídas. Na fase de filtrado realízase unha busca en paralelo en cada partición. Para iso, dependendo da técnica empregada realizarase dunha maneira ou doutra. Unha vez recuperados o conxunto de candidatos de cada partición, faise un reagrupamento para volver a redistribuír os candidatos. Finalmente, a fase de verificación realízase novamente en paralelo para cada unha das particións, reagrupándose os resultados de cada partición reagrúpanse para obter o resultado final.

As técnicas comentadas anteriormente foron adaptadas para poder funcionar neste marco distribuído sobre Apache Spark, e foron novamente avaliadas utilizando neste caso bases de datos de tamaño ordes de magnitude maior que en traballos anteriores existentes no estado da arte (contendo a base de datos de maior tamaño cen millóns de grafos). Comparándoos coas implementacións centralizadas, a avaliación conclúe que a arquitectura distribuída obtén mellores resultados cando as consultas son de tamaño pequeno, ou cando a base de datos ten un tamaño o suficientemente grande. En canto á construción dos índices, a arquitectura dis-

tribuída mostra uns resultados moito mellores que os da arquitectura centralizada. En canto á comparación entre as diferentes técnicas na arquitectura distribuída, CT-Index mostra un bo rendemento cando se utiliza un gran número de executores. KM-CTI amosa mellores resultados en clústers de pequeno tamaño, onde o número de executores ten que ser baixo e consecuentemente o número de grafos en cada partición debe ser grande. En resumo, as implementacións da arquitectura centralizada amosan un bo rendemento con consultas selectivas de tamaño grande en bases de datos de tamaño pequeno. Por outra banda, as implementacións da arquitectura distribuída aproveitan o uso de poderosos clústers que permiten obter bos rendementos tanto para consultas de pequeno tamaño, como en bases de datos de grandes tamaños.

Por último, a tese realiza un traballo de análise de eficacia e a eficiencia de diferentes solucións para resolver consultas de subestrutura de grafos, para avaliar unha tecnoloxía de consultas de procesamento aproximado que se utilizaría nunha ferramenta de busca interactiva traballando con bases de datos de gran tamaño. Para a avaliación utilízanse tres métodos, Count-Min Sketch (CMS), Sachem e KM-CTI como representativo das técnicas propostas anteriormente. Count-Min Sketch é unha estrutura de datos que actúa como unha táboa de frecuencia dos elementos nun conxunto de datos. Sachem é un cartucho quimioinformático para o sistema de bases de datos PostgreSQL, que permite realizar buscas por subestrutura molecular. Para a avaliación téñense en conta os tempos de resposta das consultas para a eficiencia, e a precisión, recall e o F-Score para a eficacia. A avaliación conclúe que o método Sachem se descarta como opción para a interface de busca e exploración interactiva debido a súa baixa eficacia. Unha combinación de CMS e KM-CTI obtén os mellores resultados. Para consultas de tamaño pequeno sería preferible priorizar a precisión sobre o tempo de resposta, posibilitando descartar o maior número de grafos posibles. Os experimentos amosaron que non existe unha gran diferenza entre CMS e KM-CTI en termos de precisión, pero o KM-CTI obtivo mellores resultados en tempos de resposta, polo que sería o método gañador. Para consultas de maior tamaño, o CMS obtén unha mellor eficacia a costa de sacrificar tempo de resposta. Polo tanto, dependendo de si a aplicación da mais prioridade a precisión da resposta ao a eficiencia do sistema, será mellor usar unha estrutura ou a outra.

En resumo, esta tese presenta un sistema que traballa con datos moleculares, independentemente da tecnoloxía de bases de datos utilizada e tanto para arquitectura centralizada como distribuída. Preséntanse tres novas solucións baseadas no paradigma FTV para buscas de subgrafos en bases de datos de gran tamaño. BM-GGSX mellora o GGSX orixinal mediante o uso

de bitmaps comprimidos para a representación das referencias dos grafos, mellorando o seu rendemento. CW-CTI toma proveito do uso de bitmaps comprimidos para almacenar fingerprints en columnas, amosando un gran rendemento en consultas de tamaño pequeno. KM-CTI adapta o algoritmo de clusterización K-Means para funcionar con bitmaps e construír unha árbore binaria que se usa para reducir en gran medida o número de comparacións na fase de filtrado. Unha combinación de KM-CTI e CW-CTI proporciona unha boa solución para o problema. Preséntase tamén un marco xenérico para a implementación de métodos FTV sobre motores de procesamento de datos a grande escala. Isto fíxose serializando as estruturas de datos dos métodos propostos. A avaliación amosou que as implementacións distribuídas reducen significativamente os tempos de xeración dos índices. Tamén se concluíu que as implementacións centralizadas obteñen mellores resultados para consultas de gran tamaño en bases de datos máis pequenas, mentres que as implementacións distribuídas obteñen mellores resultados para consultas pequenas (pouco selectivas) ou bases de datos de gran tamaño. Por último, realizouse unha avaliación de diferentes métodos en termos de eficiencia e eficacia para avaliar unha tecnoloxía de procesamento aproximado no marco dunha posible ferramenta de busca e exploración interactiva de bases de datos de grafos. Os experimentos amosaron que unha combinación dos métodos CMS e KM-CTI sería unha solución que pode cubrir todos os tamaños de consulta obtendo unha boa eficacia e eficiencia.

Abstract

The efficient management of analytical data such as molecular substructures, chromatography or Nuclear Magnetic Resonance Spectra is one of the most demanded technologies by the industry in sectors such as the chemical, biotechnological and pharmaceutical ones. As an example, in drug discovery research, to identify and remove known replicas, it is needed to search molecular datasets.

Currently, there is a wide variety of cheminformatics libraries, some of them used to implement extensions of relational database management systems, which enable the searching of molecular data. However, the technology based on object-relational databases in a centralised architecture is not optimal in terms of performance and scalability, limiting the possibility of covering the full range of applications demanded by the market.

Among all the required types of searches, one of the most important is the substructure searching, i.e., retrieving all the molecules in a dataset that contain in their molecular structure a given query substructure. The molecular structures may be encoded as small graphs where the vertices are used to represent the atoms and the edges represent the bonds between atoms.

Deciding whether a query graph is a subgraph of some other in a very large database of small graphs is a well-known problem. A straightforward solution is the application of a subgraph isomorphism algorithm to each graph of the database. Such a solution suffers from severe efficiency problems, due to the computational complexity of such algorithms and the many times they have to be executed in very large datasets. The approaches used in practice follow a Filter-Then-Verify (FTV) strategy, where an indexing technique is first used in a filtering stage to obtain a set of result candidates and the 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, previous surveys conclude that GraphGrepSX (GGSX) and CT-Index offer the best performance when the datasets are large in size.

In this Thesis, a system that enables the storage and querying of molecular data has been designed and implemented. The system uses cutting edge technologies to solve all the required types of queries. Special attention was paid to one specific type of query, the substructure search over molecular structures. In this case, new methods, beyond the state of the art, were devised, implemented and tested. Those methods provide efficient solutions for the more general subgraph searching problem.

More precisely, three new FTV techniques are presented in this Thesis. First, Bitmap GGSX (BM-GGSX) leverages the use of compressed bitmaps in the GGSX index trie structure to reduce the number of comparisons and achieve performance gains of around 90% in the filtering stage. Second, Column-Wise CT-Index (CW-CTI) exploits a column-wise representation of the fingerprints used by CT-Index to perform a filtering stage based on bitmap intersections, and thus reduce the filtering times around 80% for small size queries. Finally, K-Means CT-Index (KM-CTI) builds a binary tree of CT-Index fingerprints by applying recursively the K-Means algorithm. The use of the binary tree reduces the number of comparisons in the filtering stage, reaching a reduction of around 70% of filtering time for medium queries and 75% for large size queries.

A generic framework for the implementation of FTV techniques on top of large scale distributed data processing engines was developed. The framework enables the application of the above FTV methods on very large graph databases. The evaluation shows the great performance gain in both index building and query execution. In particular, with this framework, it is possible to query databases with sizes in the order of the tens of million graphs using 64 executors and a cluster of 16 machines with a total of 32 processors and with response times that range between 1.5 and 25 seconds, depending on the query size and therefore in the query selectivity.

Finally, two types of data sketches, namely Bloom Filters (BF) and Count-Min Sketches (CMS), were used to implement different solutions for the filtering stage of FTV methods. The use of the approximate results obtained by the filtering stage is proposed as a solution to obtain interactive response times (below one second), which are required to develop interactive searching and exploring interfaces for very large graph databases. The experiments show that CMS-based solutions are good when the result precision is more important than the query response time. On the other hand, the solutions based on fingerprints (Bloom Filters with just one hash function) are better to achieve faster response times, at the cost of sacrificing result precision. A currently available extension of a relational database for molecular querying was

also evaluated, to show that it achieves very fast response times, but at the cost of having a quite poor efficacy when its results are compared with those of a subgraph isomorphism algorithm.

CHAPTER 1

INTRODUCTION

1.1 Background and motivation

The efficient management of analytical data such as molecular structures, chromatography or nuclear magnetic resonance spectra is one of the most demanded technologies in recent years by chemical, biotechnological and pharmaceutical sectors, among others.

One of the main problems found in drug discovery research based on natural products is the identification and removal of replicas of known compounds. This identification process is solved through the application of appropriate cheminformatic techniques able to perform searches on molecular datasets.

Database systems able to perform fast searchers on large datasets of molecular and analytical data are required in the daily work of many companies and researchers in the chemical and pharmaceutical sectors. Molecular and analytical data are widely heterogeneous, including conventional alphanumeric data, molecular structures (graphs) and spectra and chromatograms (multidimensional data).

Nowadays there are many cheminformatic libraries that enable the searching of molecular data, whose functionality has been incorporated in relational database management systems (DBMS). Some examples of the above are OpenBabel [37], RDKit [31] and CDK [43, 50]. However, the technology used, based on object-relational databases and centralised architectures, is not optimal in terms of performance and scalability. As a consequence, the available technologies are limiting the possibility of covering the full range of applications nowadays demanded by the market.

Among all the demanded types of searches, one of the most challenging is the substructure

searching [14, 13], i.e., retrieving all the molecules in a dataset that contain in their molecular structure a given query substructure. Molecular structures have the form of small graphs whose vertices are atoms and whose edges are bonds between atoms. Retrieving all the instances of a given subgraph that occurs inside a graph is a well-known problem, named in the literature as *subgraph matching*. This is an NP-Complete problem that it is solved through the application of *subgraph isomorphism algorithms* [11, 24, 42, 45, 54]. These algorithms tend to perform well in databases consisting of just one very large graph, i.e., the kind of database available in the semantic web for example.

Molecular databases are composed of a very large number of small graphs. Finding all the graphs which contain a query subgraph is also a well-known problem, named *subgraph decision* in the literature. This problem may be solved in a straightforward manner through the application of *subgraph isomorphism algorithms*, although it may suffer from severe performance problems when the database size is large enough. In general, a better approach to solve the *subgraph decision problem* is to use *Filter-Then-Verify* (FTV) techniques, where the query process is divided into two stages. First, a **filtering stage** prunes out a large number of graphs through the use of some indexing structure. Next, a *verification stage* applies a subgraph isomorphism algorithm to the set of candidate graphs obtained in the filtering stage to get the result set of graphs.

Many subgraph search solutions based on a FTV strategy have been proposed over the years [6, 9, 19, 29, 51, 53, 55, 57]. Among them, GGSX [6] and CT-Index [29] have demonstrated better performance than the other when applied to large datasets of small graphs [26], as it is the case of the molecular databases. Both methods rely on building indexing structures with features extracted from the graphs of the dataset. Specifically, GraphGrepSX generates a trie [18] structure with all the paths of each graph whose length is shorter than a given threshold. Each node of the trie represents a specific path and references all the graphs of the database that contains such a path. A breadth-first scan of the query and index trie structures are performed during the filtering stage.

On the other hand, CT-Index represents the set of features of a maximum length extracted from the graph with a fingerprint (bitmap). Extracted features are subtrees and cycles whose length is shorter than a given threshold. A hash function is applied to a canonical representation of each feature to generate an index that is used to activate the relevant bit of the fingerprint. The filtering stage in CT-Index is performed by testing bitmap subset between query and database fingerprints.

The above techniques have been successfully applied to datasets in the range of the tens of thousands of graphs, however, their performance is not adequate when very large datasets of various hundreds of thousands or even millions of graphs are considered.

1.2 Objectives and contributions of the Thesis

Based on the motivation described in the previous section, the main objective of the present Thesis is the design and implementation of a data storage, indexing and searching technology that supports all the types of data contained in molecular and analytical datasets. The system should work independently of the underlying database technology, and it should enable its deployment on both centralised and distributed architectures. Among all the query types, the main effort will be devoted to the molecular substructure searching, aiming at the proposal of new solutions with efficacy and/or efficiency beyond that of the state of the art. New solutions should be scalable to be able to deal with very large datasets, they should be efficient with queries of different sizes and they should enable the interactive searching and exploration of such very large datasets.

The main characteristics of the system developed in this Thesis are summarized below.

- The system provides support for the following types of queries over molecular data:
  - Textual queries performed over text properties of the molecules, such as its name. State of the art text indexing and searching technologies will be applied.
  - Peak queries performed over the spectra recorded for the molecules. Multidimensional indexing techniques will support this kind of queries.
  - Queries over the molecular formulas, for which specific data structures have been developed.
  - Substructure queries performed over the structure of the molecules, supported by advanced indexing structures designed as part of the Thesis.
- The system can be deployed in centralised architectures where the molecular data is stored in relational databases, but also on distributed clusters where molecular data is stored in document-oriented NoSQL databases.
- The implementation uses state of the art techniques to support textual and peak queries. An in-memory straightforward structure enables the evaluation of molecular formula

queries with in memory data. However, new advanced indexing structures had to be developed to enable the application of substructure queries in large databases (in the order of hundreds of thousands of molecules), achieving performance gains in the range of 70%.

- The system incorporates new advanced indexing techniques for substructure searching in a distributed system implementation which enables the efficient searching over very large molecular databases (in the range of tens of millions of structures).

- Sketches have been used as part of the indexing structures which enables the development of solutions that return approximate results with interactive response times (below one second). Such solutions can be used to implement interactive searching and exploration interfaces for very large molecular datasets. Different structures show better results in terms of efficacy and efficiency for different query sizes.

Finally, the main contributions of the Thesis may be resumed as follows.

1. A query system for molecular and analytical data that works on top of different underlying database technologies has been designed and implemented. The system can be deployed in both centralised and distributed architectures, using either relational databases or document-oriented NoSQL databases. The use of advanced data structures enables the improvement of the performance over existing solutions for different types of queries over different types of data.

2. Three new FTV approaches has been proposed and extensively evaluated using real-world databases of size much larger than the one used in previous surveys. The evaluation concludes that a combination of two of them gives a good solution for any query size. Additional details of each of the new approaches are given below.

    a) Bitmap GGSX leverages the use of compressed bitmaps in the GraphGrepSX trie nodes, enabling a drastic reduction of the filtering time (around 90% less than GraphGrepSX), maintaining the index size and building time almost unaltered. Compared with the other two proposals based on CT-Index, it performs much worse in terms of index size and building time. Regarding query response time, it offers good performance for medium and large queries.

b) Column-Wise CT-Index reorganizes the CT-Index fingerprints to store them column-wise. The bits of the same position of the fingerprints of all the graphs of the database are stored together in a compressed bitmap. Index size and index building time remains almost unaltered. However, given that only the bitmaps corresponding to bits with value 1 in the query have to be accessed, for small queries (with few 1s in the fingerprint), the solution reaches very good performance.

c) K-Means CT-Index recursively applies the K-Means clustering algorithm on CT-Index fingerprints to build a binary tree of fingerprints as index. Its index size and index building time are worse than those of the original CT-Index, but they are better than those of Bitmap GGSX. The approach has good performance in terms of query response time when the query is of medium or large size. Notice that small queries are not very selective and do not really take advantage of the fingerprint binary tree.

3. A generic framework for the implementation of FTV methods on top of large scale data processing engines has been designed and developed. Firstly, the database is divided into partitions and each of them is indexed. The resulting indices are encoded into byte arrays and recorded into distributed data structures. The filtering stage performs index searching in parallel in each partition, and the verification stage resdistributes the candidates obtained in the previous stage and parallelizes the subgraph isomorphism tests. The new FTV techniques described above were adapted to the framework and comprehensively evaluated using databases of sizes orders of magnitude larger than the ones used by previous surveys. The evaluation concludes that the distributed architecture obtains better results when either the query is small or the database is large enough. The scalability of the new distributed solutions are specially important for index building, enabling the construction of indexes for database sizes that would be completely impossible using centralised implementations.

4. The problem of the interactive searching of databases of molecular structures has been introduced and illustrated with a user interface example. Two types of data sketches, namely Bloom Filter (BF) and Count-Min Sketch (CMS), were used with different parameters to implement indexes for the filtering stage of FTV solutions. The use of filtering without verification provides approximate results for the queries that are obtained in much shorter response times, enabling this way interactive system responses

(below one second). The evaluation shows that CMS structures provide advantages when the result precision (system efficacy) is more important than the query response time (efficiency), specially for large queries. On the other hand, better efficiency at the cost of worse efficacy is offered by the use of fingerprints (Bloom Filters with just one hash function) in the K-Means CT-Index approach. The evaluation was used also to show that some of the available subgraph search implementations do not give precise and complete results compared to subgraph isomorphism algorithms.

1.3 Projects and publications

The objective of this Thesis arises from the identification of research problems during the execution of the NEXTCHROM project: Virtual advanced chromatography for an integral management of analytical and molecular data [1]. The objectives and main results of this project are described in Chapter 2. The author is this Thesis had active participation and is the main responsible of the results related to data management achived in the project.

The following publications describe research results of the present Thesis, in particular those described in Chapters 4 and 5.

- D. Luaces, J.R. Viqueira, T.F. Pena, J.M. Cotos, Leveraging bitmap indexing for subgraph searching, in: 22nd International Conference on Extending Database Technology (EDBT), OpenProceedings.org, 2019, pp. 49–60. doi:10.5441/002/edbt.2019.06.

- David Luaces, José R.R. Viqueira, José M. Cotos, and Julián C. Flores. Efficient access methods for very large distributed graph databases. Information Sciences, 573:65–81, 2021.

In addition to the work done for the NEXTCHROM project, in the scope of the Thesis, the author was also involved as part of the research team of the following projects, whose objectives were not in the scope of the Thesis.

- GEOARPAD: Cultural heritage of the Euroregion Galicia-North of Portugal: valorisation and innovation [2].

[1] https://citius.gal/research/projects/cromatografia-virtual-avanzada-e-xestion-integral-de-datos-analiticos-e-moleculares

[2] https://citius.gal/research/projects/patrimonio-cultural-da-eurorrexion-galicia-norte-de-portugal-valorizacion-e-innovacion

- SenseStore1: Efficient storage and publication of remote sensing data [3].
- TRAFAIR: Understanding Traffic Flows to Improve Air quality [4].

Also out of the scope of the Thesis is the following publication.

- David Luaces, José R. R. Viqueira, Pablo Gamallo, David Mera, and Julián Flores. Geohbbtv: A framework for the development and evaluation of geographic interactive TV contents. Multimedia Tools and Applications, 77, 11 2018.

1.4 Outline of the Thesis

The structure of this document is organized as follows. Chapter 2 describes the main objectives and results of the NEXTCHROM project. In particular, a description is given of what is considered in this Thesis molecular data and also of what are considered the more relevant types of queries over molecular data.

Chapter 3 provides an in depth description of the background and state of the art relevant for the subgraph searching problem. In particular, the subgraph searching problem is formalized and the most important solutions found in the state of the art related to this problem are described. Special attention is paid and more details are given of the two state-of-the-art graph indexing approaches that are the basis for the new solutions developed in the Thesis, i.e., the GraphGrepSX (GGSX) and CT-Index (CTI) structures.

Three new Filter-Then-Verify(FTV) approaches, called Bitmap-GGSX (BM-GGSX), Colum-Wise CT-Index (CW-CTI) and K-Mean CT-Index (KM-CTI), are described in detail in Chapter 4. A performance evaluation of the three new methods compared with the original GGSX and CT-Index is also introduced and discussed.

Chapter 5 describes a generic framework for subgraph searching in distributed architectures. The framework uses binary encodings of the indexing structures to enable their use in distributed processing engines. An evaluation of the performance of the framework, considering both different cluster sizes (in number of nodes) and different database sizes (reaching tens of millions of graphs), shows its good behaviour in terms of scalability and how it enables dealing with very large datasets.

[3] https://citius.gal/research/projects/almacenamento-e-publicacion-eficientes-de-datos-de-sensorizacion-remota

[4] https://citius.gal/research/projects/understanding-traffic-flows-to-improve-air-quality

Chapter 6 presents approximate solutions for the subgraph searching problem, suitable to be used in interactive user interfaces for search and exploration over very large datasets. Approximate filtering solutions based on two different types of data sketches, namely Bloom Filters and Count-Min Sketch, are described and evaluated. Different alternatives show better behaviour in terms of either efficacy or efficiency for different query sizes.

Finally, the findings of the research undertaken in this Thesis and issues of future work are summarized in Chapter 7.

CHAPTER 2

QUERYING MOLECULAR DATA: THE NEXTCHROM PROJECT

The NEXTCHROM project addresses the development of innovative tools in the field of cheminformatics. One of the two broad objectives of the project includes the efficient querying of huge volumes of molecular and analytical data.

Many real problems that arise during the daily activities undertaken in chemical laboratories require the querying of very large datasets of molecular data. One example is the drug discovery research, where replicas from compounds must be identified and eliminated. The use of currently available technologies from the area of cheminformatics enables the querying of datasets of molecular data that combine conventional alphanumeric data with molecular structures, spectra and chromatograms, among other types of data.

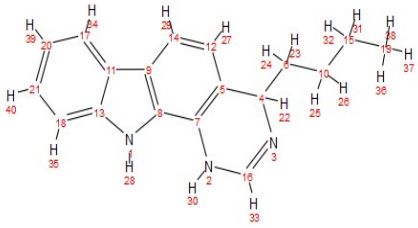
Specifically, each of the entities stored in a molecular database (named *Molecular Entity*) may incorporate different properties of different types:

- *Alphanumeric properties*: Include properties such as the molecule name and molecular formula in textual format, molecular weight, or number of atoms in the molecule.

- *Molecular formula*: Set of pairs(e, n), where e is the chemical element and n is the number of atoms of that element that are present in the molecule.

- *Molecular structure*: Graph representing the molecular structure, where the vertices are the atoms and the edges represent different types of bonds between the atoms.

Name: 4-butyl-4,11-dihydro-1H-pyrimido[4,5-a]carbazole
Heavy_atom_count: 21
Exact_mass: 277.158
Molecular_formula: C18H19N3
Structure:



NMR_Spectrum:

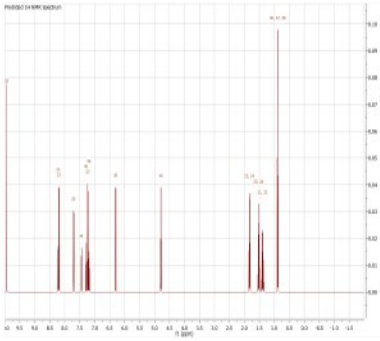


Figure 2.1: Illustration of a molecular entity.

- *Spectra*: Structures that store the data needed to represent different types of spectra, including Nuclear Magnetic Resonance (NRM) in one and two dimensions, Mass Chromatograms and Mass Spectra.

Figure 2.1 shows some of the data elements of a molecular entity. Different types of queries may be issued depending on the type of property that is used to filter the dataset:

- *Text query*: This type of query enables the retrieval of all the molecular entities whose textual properties contain a query set of keywords. This is the type of query supported in information retrieval engines and by full-text search extensions in database technologies.

- *Numeric range query*: It enables the range query over numerical properties of the molecular entities. An example of this type of query is the retrieval of all the molecular entities whose molecular weight lies inside a specific query range.

- *Peak query*: Molecular spectra are internally represented as multidimensional data, which represents the variation of some measure over one or various dimensions. For example, an NRM spectrum may be represented by 2-dimensional points that associate signal intensity with chemical shift. A peak query specifies objective ranges of values for the dimensions and measures to obtain the desired multidimensional points of the spectrum. As an example, obtaining all the peaks of signal intensity and chemical shift of an NRM spectrum that lies inside a given range of chemical shift.

- *Molecular formula query*: This type of query specifies a pattern composed of chemical element symbols accompanied by ranges of the possible numbers of atoms of each element. One example would be a query of the form "C1,4H3,8N1,1", which should retrieve all the molecules with between 1 and 4 atoms of carbon (C), between 3 and 8 atoms of hydrogen (H) and exactly one atom of nitrogen (N).

- *Molecular substructure query*: This is the type of query of maximum interest for the present Thesis. The objective is the retrieval of all the molecules whose molecular structure (graph of atoms and bounds) contains a query molecular substructure. A straightforward strategy for the evaluation of this type of query requires the testing of whether a query subgraph is contained or not in a large collection of graphs. It is noticed that such a test is a hard problem with large computational complexity.

The NEXTCHROM project proposed solutions for all the above types of queries, using state-of-the-art technologies in centralised architectures and proposing new innovative solutions for both centralised and distributed database architectures. Implementations on centralised architectures were developed on top of the PostgreSQL extensible relational DBMS, whereas distributed implementation worked on top of the MongoDB NoSQL document-oriented database. Additional details related to the related available technologies in the field of cheminformatics and to the way each type of query was implemented in the NEXTCHROM project are given in the following subsections.

2.1 Technologies Available in Cheminformatics

In the field o f cheminformatics, several well-known libraries and toolkits have been developed that provide basic functionality that can be used by many applications. One example is the open source Java library Chemistry Development Kit (CDK). The functionality of CDK allows both molecular data representation and processing, which include the substructure testing and the generation of canonical representations of molecular structures.

Another well-known example is the Open Babel collaborative project, whose tools are designed to work with different languages of chemical data. The functionality supported includes molecular searching, format conversion, analysis and data storage.

If we focus the discussion on data storage and querying, most of the current implementations of molecular query systems are based on the use of relational DBMSs, which helps in the specification of the above queries. An example of such a query technology, developed by Mestrelab Research S.L, the company that led the NEXTCHROM project, is the MNova Server (MnServer)[1]. MnServer enables the querying of molecular entities stored in different database technologies, providing support for all the types of queries described above. It is noticed that MnServer works on top of the DBMS, and does not require a specific DBMS technology.

Contrary to the approach adopted by MnServer, some tools enable the incorporation of cheminformatics functionality inside the query engine, using the extension capabilities available in most current DBMS implementations. A broadly used solution of this type is RDKit, which incorporates molecular data querying capabilities inside the PostgreSQL query engine. The functionality includes substructure searching and structure similarity searching. A similar approach is followed by Sachem[30], another PostgreSQL extension that enables the querying of molecular data.

2.2 Text Searching

A straightforward approach to evaluating text search queries is the testing of each text property of each molecule. It is obvious that such an approach cannot lead to an efficient solution and, when the database size increases, the low performance will turn the system to be unusable.

Any information retrieval or full text search solution bases its performance on the existence of indices over the documents or text fields. In the NEXTCHROM project, state of the

[1] https://www.mestrelab.com/downloads/mnova/manuals/MnovaDBServer_2.0.pdf

art technologies were used to implement full text search in both centralised and distributed database architectures.

The Apache Lucene[2] java library was used to create text indices over all the text fields of the molecular database in the centralised architecture. Each text field of each individual molecule was treated as a document in Lucene, identified by the molecule identifier and the field name. The Lucene query engine enables the retrieval of the top K molecules and fields for each text query using the generated index. The molecule identifiers retrieved by Lucene are next used to query the PostgreSQL database to obtain the molecular data, highlighting the field that matched the query keywords.

As it was already stated above, the distributed implementation of the system developed in the NEXTCHROM project was based on the use of the MongoDB NoSQL document-oriented database. All the molecular data was transformed to JSON to be inserted in a MongoDB collection. The full-text search functionality of MongoDB was used to build a text index over the collection of molecular data. The query evaluation engine of MongoDB retrieves directly the top K documents that better match the query keywords. The full-text search module of MongoDB uses stemming, therefore, if the search required the exact match of a specific keyword, then additional processing of the result must be done in the client side.

In general, both approaches above offer similar performance when compared using a single machine. The response time of the distributed solution is not better when only a single query is submitted to the system, since the index already enables direct access to a few molecules. The advantage of the distributed approach is, however, the better support for large numbers of users using the system in concurrence, increasing the throughput of the system in those cases.

Only two solutions were tested in the project, however many other technologies could have been chosen. Thus, for example, one alternative for the centralised architecture would be the direct use of the full-text search capabilities already provided by PostgreSQL. For the distributed architecture, it would be interesting to test specific solutions like the Elastic Search [3]. It has to be noted finally that performing an exhaustive benchmark of full-text search technologies for centralised and distributed database architectures was out of the scope of the NEXTCHROM project.

[2] `https://lucene.apache.org/`
[3] `https://www.elastic.co/es/`

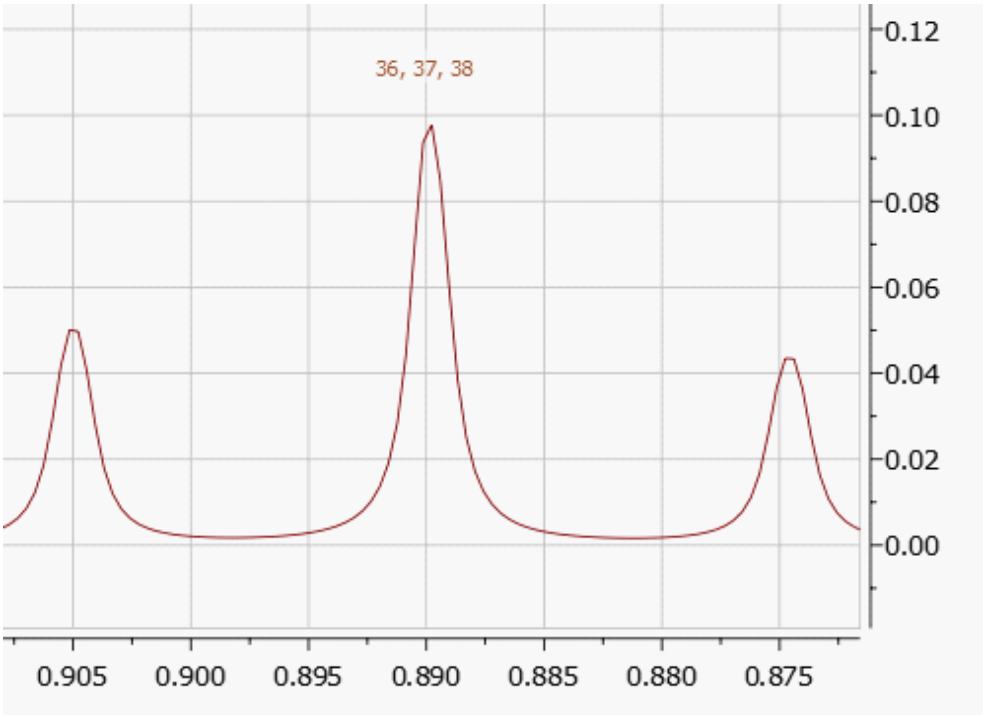


Figure 2.2: Example of an NRM spectrum.

2.3 Peak Searching

Molecular spectra are encoded and stored as a collection of multidimensional points. As an example, Figure 2.2 shows an NRM spectrum that might be represented by three 2-dimensional points, each of them representing one of the peaks of the spectrum shown in the figure. The first dimension defines the position of the peak as a chemical shift. The second dimension provides an intensity value for the peak. A peak query specification consists of a query range for each of the available dimensions of the spectra. In the above example, a peak query will define a range of chemical shifts and a range of signal intensities.

A straightforward approach could represent all the points of each spectrum of each molecule in a single attribute either of a text data type or of some more complex data type that enables the recording of a collection of tuples. The non-existence of any kind of index structure to support peak queries in such a simple approach leads to solutions with such low performance

that makes them useless when the database size rises.

To be able to query single peaks of each spectrum, their representation must be normalized, representing the peaks as individual rows in a separate structure. Two broad types of index structures may be used for the peak dimensions in such a dedicated structure. A first approach is a multikey index using a one-dimensional structure, generally a B-Tree. Therefore, all the keys of a multidimensional point must be transformed into single key by concatenating their values in some order. Range queries over the first dimension in the order are resolved efficiently. However, range queries over the other dimensions require the traversal of larger parts of the index tree, resulting in lower performance. A second approach is a multidimensional index such as the R-Tree. In this case, all the dimensions are treated uniformly in the generated tree and the efficiency of the data structure does not depend on the dimension chosen to filter.

Both multikey indexes and multidimensional indexes were used in the PostgreSQL DBMS. Multikey B-Tree indexes are directly provided by the DBMS whereas multidimensional R-Tree indexes are incorporated with the PostGIS geospatial extension. The spatial distribution of the peak data showed the convenience of using the x dimension (position in the chemical shift frequency) as the first key and leaving the y dimension (intensity of the signal) as the second key. It was noticed that peak data has much more variability in the values of the x dimension than in the values of the y dimension, thus, an index that filters first in the x dimension eliminates more candidates earlier than an index that filters first in the y dimension.

Regarding the distributed implementation based on MongoDB, both multikey and multidimensional indexes were also compared. Multidimensional indexes are provided by the geospatial functionality supported by MongoDB.

In general, the multidimensional index performs better than the multikey index, although there might be some queries for which the multikey index might give a better response time. In any case, the geospatial extensions supported by the two databases used in the project showed their utility also to evaluate peak queries in molecular databases, which is not an application considered during their design.

2.4 Molform Searching

As already mentioned that a molecular formula (molform) represents the elements and the number of atoms of each chemical element that are present in its internal structure. A query

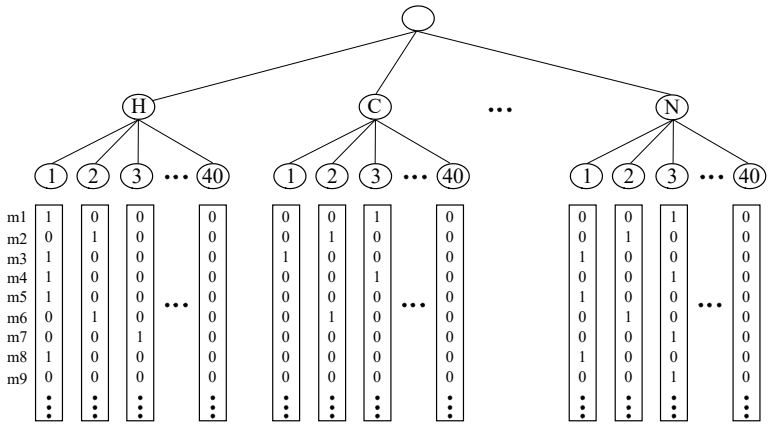


Figure 2.3: Illustration of the molform index developed during the NEXTCHROM project.

over the molecular formula (molform query) specifies a set of chemical elements and a range of possible numbers of atoms for each element. The evaluation of the query must retrieve all the molecular entities whose formula matches the query pattern. A straightforward approach would be to parse a textual representation of the molecular formula of each entity to determine if it matches or not the query. A binary encoding of the formula would slightly reduce the amount of storage and the response time. However, the search will not really improve without a specific indexing structure. Various alternatives were tested in the NEXTCHROM project.

A first approach consisted in normalizing of the molecular formula on a different structure, i.e., a new table in the centralised relational architecture and an embedded array on the distributed architecture based on MongoDB. Conventional indices could be created now on the new fields, i.e., on the chemical element and on the number of elements of each chemical element. This solution with conventional indices led to a first major increase in the performance of the query evaluations.

A second more sophisticated approach involved the development of a specific indexing structure for molform queries. The structure, illustrated in Figure 2.3, has two levels. In the first level, all the chemical elements are represented. The second level contains the different number of atoms that a molecule might have of each chemical element, together with a bitmap that contains a bit for each molecular entity of the database. The bit corresponding to a given molecule in a leaf node of the structure represents that the molecule has exactly the number of elements of the relevant element in its molecular formula. A query must follow the structure

to obtain all the involved bitmaps and combine them with binary *AND* and *OR* operations to obtain a result bitmap with ones in the positions corresponding to the result molecules. A compressed bitmap representation was used to encode those large bitmaps, whose characteristics enable the evaluation of logical operators without the need to decompress. The experiments undertaken in the project with databases of a few millions of molecular entities showed that the new structure provides a significant performance improvement with respect to the conventional structures already provided by the two considered database technologies, i.e., PostgreSQL and MongoDB. It should be noted that the compressed representation for the bitmaps enables the loading of the new data structure completely in memory.

2.5 Substructure Searching

As it was already shown above, the molecular structure of each entity is represented with a graph of chemical elements (nodes) and bonds (edges). A molecular substructure query retrieves all the molecular entities whose molecular structure graph contains a given query structure. Different approaches were implemented in the NEXTCHROM project to solve the above queries, always using state-of-the-art technologies and techniques.

A first straightforward approach consisted in testing the substructure matching between the query and each molecular entity of the database. This approach was implemented on top of both database technologies, respectively for centralised and distributed architectures. The CDK library was used to perform the test, after retrieving the molecular structure in SMILES [49] format from the database. Since the test is not implemented inside the database, the distributed architecture does not bring any advantage to this approach. Response times were high due to the high complexity of the subgraph matching problem.

A more advanced solution involved the implementation of the subgraph testing inside the database. In the NEXTCHROM project, this approach was implemented only in the centralised architecture, using the RDKit extension of the PostgreSQL DBMS. The functions implemented by RDKit and the indexing structures were used to obtain response times near one order of magnitude faster than the straightforward approach implemented with CDK.

The last solution developed in the project was the implementation of a Filter-Then-Verify (FTV) approach based on a specialized graph indexing structure. In particular, the GraphGrepSX (GGSX) [6] data structure was used in a first filtering stage to discard most of the molecular entities of the database. Then, the VF2 algorithm [11] was used to verify each of

the structures retrieved by GGSX to obtain the final result. This approach reaches response times much faster than those based on the use of RDKit. It enables also the implementation of the approach in distributed architectures. It is notice that, in general, the main advantage of a distributed architecture is the support for the parallel verification of many molecules, when the query is not very selective, i.e., when the query consists of a very small graph that is contained in many molecules.

The clear room for improvement in the state-of-the-art techniques used for this complex problem was the main motivation for the start of the present Thesis, which proposes new indexing structures and both centralised and distributed implementations that enable the efficient evaluation of substructure queries over very large databases. Thus, further details about the GGSX structure and also about other state-of-the-art structures will be given in Chapter 3. Discussion about the design, implementation and evaluation of more advanced structures is provided in subsequent chapters.

CHAPTER 3

BACKGROUND AND RELATED WORK ON SUBGRAPH SEARCHING

The molecular substructure queries discussed in the previous chapter, in the context of the NEXTCHROM project, are a particular application of the more general subgraph queries, which may be issued to databases storing graphs in many application domains. This chapter provides the required background and analysis of the state of the art related to the evaluation of subgraph queries. Specifically, the problem of subgraph searching is first described and formalized. Next, state-of-the-art technologies and techniques relevant to this problem are reviewed and discussed. Specific attention will be given to two data structures that can be used in the filtering stage of a Filter-Then-Verify (FTV) approach, namely, the GraphGrepSX (GGSX) [6] and the CT-Index [29] structures. This special attention is due to the fact that those approaches had already been identified in previous surveys as those with better performance in large datasets, and also because the structures proposed in the present Thesis are based on them.

3.1 Problem Definition

Graph databases may be categorized into two main types. A database of the first type consist of a single very large graph. An example of this kind is the semantic web. One of the functionalities most demanded by applications is to identify all the instances in the database of a specific query graph. This is an NP-complete problem, known in the literature as *subgraph*

matching, and it is solved through the application of *subgraph isomorphism* algorithms.

The second type of databases are those that contain a very large number of small graphs. A typical example of this type of databases is a molecular database, where each record stores a molecular compound. Querying molecular databases was one of the main objectives of the NEXTCHROM project, as already summarized in the previous chapter. One of the properties of a molecular compound is its molecular structure, which is represented as a graph where the nodes represent the atoms in the molecule, and the edges represent the bonds that connect those atoms. This type of database is currently reaching very large sizes. As an illustrative example, the PubChem[1] database has an approximate size of one hundred million molecules, as of today. Finding all the records whose graph contains a specific query subgraph is an important functionality for applications, as it was already shown with molecular substructure queries. This is a well-known problem in the literature, named as *subgraph decision* problem. A straightforward solution to this problem is to apply a *subgraph isomorphism* algorithm to each graph in the database. In general, this approach is not efficient, and even not feasible for very large databases. A better approach to reduce the query times is to use a Filter-Then-Verify (FTV) method. FTV methods are divided into two stages. The first stage, named *filtering*, uses an index structure, built from features extracted from the graphs, to discard a large amount of candidates without having to apply the expensive subgraph isomorphism algorithm. Such an algorithm is applied in a second *verification* stage to refine the broad result containing false positives retrieved by the previous stage.

The type of undirected graph with labeled vertices and edges considered in the present work is defined as follows.

Definition 1 *A* graph *is a quadruple* (V, E, vl, el)*, where* V *is a set of vertices,* $E \subseteq [V]^2$ *is a set of undirected edges (represented as sets of two vertices) and* $vl : V \to VL$*,* $el : E \to EL$ *are mappings that associate a label respectively from the set of vertex labels* VL *and from the set of edge labels* EL *to each vertex and edge.*

Figure 3.1 shows a graph that represents a molecular structure. The set of vertex labels VL contains all the symbols of the chemical elements (C, H, N, etc.). Edge labels EL are the types of possible bonds between atoms, i.e., single (s), double (d), etc. Based on the above formalism, the concept of *subgraph isomorphism* and the problem of *subgraph search*, also known as *subgraph decision problem*, are defined below.

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

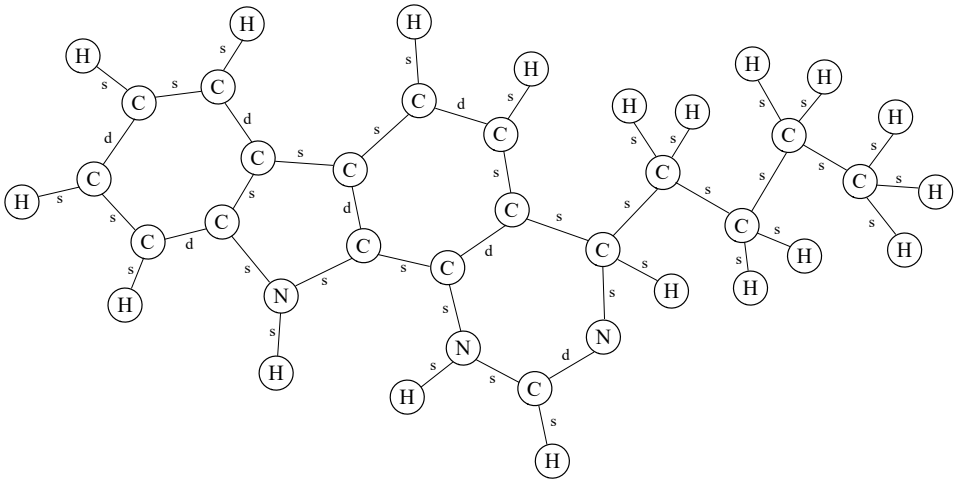


Figure 3.1: Example of an undirected graph representing the structure of a molecule.

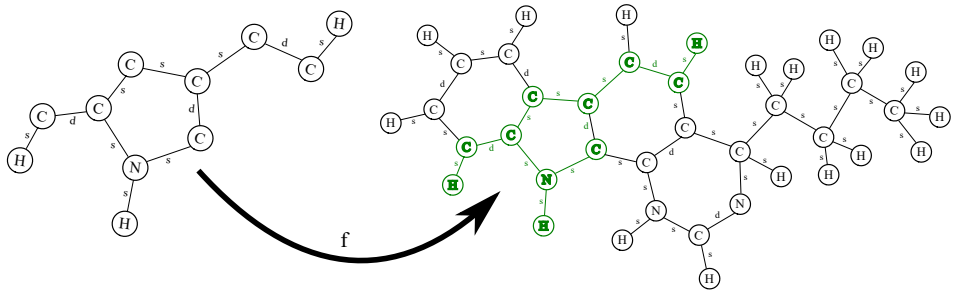


Figure 3.2: Illustration of a subgraph isomporphism.

Definition 2 *If* $g_a = (V_a, E_a, vl_a, el_a)$ *and* $g_b = (V_b, E_b, vl_b, el_b)$ *are two graphs, a* subgraph isomorphism *from* g_a *to* g_b *is an injective function* $f : V_a \cup E_a \rightarrow V_b \cup E_b$*, such that* $\forall v_a \in V_a$*,* $vl_a(v_a) = vl_b(f(v_a))$*, and* $\forall e_a = \{v_{a1}, v_{a2}\} \in E_a$*,* $el_a(e_a) = el_b(f(e_a))$ *and* $\{f(v_{a1}), f(v_{a2})\} \in E_b$*. Graph* g_a *is said to be* subgraph isomorphic *to graph* g_b*.*

Figure 3.2 illustrates a subgraph isomporphism f, which maps each vertex and edge of the first graph to a vertex and edge of the second one with identical label, keeping the same topological structure in both cases.

Definition 3 *If* $D = [g_i]$ *is a graph database (a multiset of graphs), and* q *is a query graph,*

the subgraph searching problem *(also known as* subgraph decision problem*) aims at finding all the graphs* $g_k \in D$*, such that there exists at least one subgraph isomorphism* f *from* q *to* g_k.

Notice that in the case of the *subgraph matching* problem all the *subgraph isomorphisms* from the query graph to the database graph must be found. Contrary to the above, in the *subgraph decision* problem it is enough to check if a *subgraph isomorphism* exists or not from the query graph and each of the graphs stored in the database.

3.2 State of the art

As stated above, the matching problem is solved through the application of *subgraph isomorphism* algorithms. Many approaches have been proposed through the years. Some of the most relevant are shortly explained below.

A first solution to the problem was provided by Ullman's algorithm [45], which has been heavily adapted in posterior solutions. This is a backtracking-based algorithm, designed to find the subgraph-isomorphism mapping from a graph q to a graph g. It uses two vectors: A vector F that records the vertices of g that have been used in some intermediate state of the computation and a vector H that records the mapping from q to g.

Another important algorithm is the VF2 subgraph isomorphism algorithm [11]. This algorithm is especially suited for dealing with small graphs. VF2 is also a backtracking algorithm, and optimizes Ullman's algorithm by choosing the current query vertex u as the one which is connected from the already matched query vertices. VF2 exploits this constraint to prune out candidates from the set of candidate vertices of each query vertex.

QuickSI [42] is another subgraph isomorphism algorithm. QuickSI, in a first indexing phase, precomputes the frequencies of labels and edges and uses them to calculate the *average inner support* of a vertex or an edge. Using this calculation, it gives priority to vertices with infrequent labels and infrequent adjacent edge labels. This is later used in the matching process to assign weights on the edges of the query graph.

The GraphQL [24] subgraph isomorphism algorithm is focused on minimizing the size of the set of candidate vertices. The algorithm, first, indexes the vertex labels along with the labels of neighboring nodes. In a second phase, the algorithm starts by retrieving all possible matches for each node in the pattern. Finally, it applies several rules, which include the use of the indexed labels to prune out unfeasible matches and optimize this way the search order.

Another subgraph isomorphism algorithm is SPath [54]. SPath maintains a neighborhood signature encoding where the shortest paths are organized in a compact indexing structure. Shortest paths are decomposed in a distance-wise structure in order to reduce space. The query is decomposed in shortest paths that are matched into the shortest paths from each graph in the database. The candidates that can cover the query and provide good selectivity are selected from the set of shortest paths.

The performance of the previous algorithms was studied in [32] using four real-world datasets of different sizes and characteristics. Two of those datasets had the form of a collection of multiple small graphs. The other two datasets had the form of a single and relatively large graph. The study concluded that QuickSI had the best performance for both small and large data graphs, even though it was designed for working with small graphs, and GraphQL was the only algorithm to complete all the queries.

Han et al. [22] propose TurboISO, a new subgraph isomorphism algorithm that uses two data structures as its index, an inverse index of vertex labels, that provides easy access to vertices with the same label, and a list of adjacent vertices for every vertex. In the query processing phase, a starting root vertex is chosen based on label frequencies and high node degrees. From this starting vertex, the algorithm identifies candidate regions by performing a Depth-First search, minimizing the intermediate candidate results with the matching sequence.

In [56], SQBC, a subgraph isomorphism algorithm was proposed to work on large graphs. It achieves the reduction of the search space by employing a structure of maximal cliques, where the locality information is taken into account for both clique and vertex code.

BoostIso [39, 48] rewrites the query by merging vertices that share the same label and neighborhoods, and extends this rewriting also to the stored graphs, allowing to dynamically reduce the duplicate computations.

Another approach to the problem is proposed by CFLMatch [3], which postpones Cartesian Products. The algorithm uses a *compact path index* (CPI) to store the potential embeddings of a spanning tree of the query graph.

A more recent approach [21] combines the use of a Direct Acyclic Graph (DAG) with a novel backtracking framework based on DAG-ordering.

Finally, in recent years, new methods that take advantage of distributed architectures with different parallel optimization strategies have been proposed [2, 25, 40].

Regarding the *decision problem*, a straightforward solution is given by the direct appli-

cation of a *subgraph isomorphism* algorithm between the query graph and each graph of the database. As stated above, this solution cannot achieve good results in terms of performance. To improve the efficiency of the above straightforward solution, many Filter-Then-Verify (FTV) solutions were proposed in the literature, which combine some indexing structure for the filtering stage and some subgraph isomorphism algorithm for the verification stage. Some of the most relevant approaches are discussed below.

Yan et al. [51] propose gIndex, an FTV method that performs a frequent data mining algorithm to generate a set of frequent features with at most a previously established maximum size. Additionally, it generates a subset of infrequent subgraphs. In the query processing, gIndex enumerates all the subgraphs whose size is at most the maximum one and checks whether they are in the indices.

A hierarchical tree based on frequent tree features is built in TreePi [53]. In the querying stage, TreePi performs a subgraph isomorphism test for each node of the tree, traversing from the root node to the leaf node.

A frequent data-mining algorithm is performed in FG-Index [9], following an approach similar to that of gIndex, but, in this case, the size of the features extracted from the graph is not important. In particular, all the frequent subgraphs are generated. It also includes all infrequent edges.

The Tree+Δ [55] method extracts trees from the graphs using mining and generates all frequent trees with at most a maximum size. Additionally, it generates features while processing queries by choosing all simple cycles from a query graph and, for each of them, it extends them with one vertex to check whether it is discriminative. In the querying stage, only the subtrees whose size does not exceed the maximum one are checked for matching with the index.

An FTV method that stores a set of graph signatures, each of them containing a hash value, and a list of eigenvalues for all vertices in the index is proposed in gCode [57]. In the filtering stage it performs a two-step strategy, index-level and object-level. In the index-level step, the method finds all the qualified signatures. Next, in the object-level step, for each qualified graph, it uses a list of pairs of the form $(ID, frequency)$ to perform a pruning of the potential candidates.

In [19] is proposed GRAPES, an FTV method that uses a trie structure to index every path extracted from each graph whose length does not exceed a specified maximum. The index records the number of repetitions of each path in each graph and a reference to the starting

node of each path in each graph. The algorithms of this method were designed to work in parallel, including both index building and query processing.

The survey work presented in [23], evaluates the performance of six FTV techniques [9, 42, 51, 53, 55, 57]. The work was extended in [26] with the addition of three more algorithms, namely GraphGrepSX [6], CT-Index [29] and GRAPES, providing a study of their performance and scalability with respect to various dataset features such as the number of nodes per graph and the number of graphs. The conclusions of the work may be summarized as follows. CT-Index and gCode have the smallest index size. GraphGrepSX, CT-Index and GRAPES show the best performance in terms of scalability, although GRAPES fails to build an index for large datasets due to its memory requirements. The algorithms based on mining techniques (gIndex and Tree+Δ) are only competitive for small datasets.

Some proposed solutions [46, 47] use the caching of past queries and their results to improve their overall performance. They do this by taking advantage of the fact that many of the queries of real-world scenarios have subgraph and supergraph relationships among each other.

A recent solution for the decision problem [28] combines GRAPES with several subgraph isomorphism algorithms. The results show a good performance with datasets of a large number of graphs, with the payout of large index building time and index size.

Another recent work [44] proposes a survey comparing three FTV solutions (GraphGrepSX, GRAPES and CT-Index), with the direct use of three advanced subgraph isomorphism solutions (GraphQL [24], CFL [3], and CFQL that combines GraphQL with CFL). It compares the above also with the combination of the indexing structures of GraphGrepSX and GRAPES with the subgraph isomorphism algorithm CFQL. The work concludes that the direct use of the subgraph isomorphism algorithm CFQL outperforms all three FTV algorithms and it also achieves a performance similar to that of its combination with GraphGrepSX and GRAPES, having the advantage of avoiding the use of an index.

Some remarks are important to be able to correctly interpret the results of this work. The subgraph isomorphism algorithms used, GraphQL [24], CFL [3] and CFQL, do not use the edge labels of the graphs during the search process, thus they may not be fairly compared with the structures proposed in the present Thesis. Besides, the largest dataset used in the undertaken experiments was the AIDS database, which contains only 40k graphs. This small dataset was also used by previous surveys. Notice therefore the huge difference in terms of database size with those used in the present Thesis, which reach sizes in the order of the tens

of millions of graphs.

Based on the conclusions of [26], new indexing structures based on GraphGrepSX and CT-Index that can work in both centralised and distributed infrastructures were proposed in the present Thesis. The GRAPES method was discarded due to its problems with large databases. The proposed methods were implemented using the VF2 algorithm for the verification stage. However, they may also be combined with any other subgraph isomorphism algorithm. Additionally, the methods may also be integrated into more complex frameworks, such as those described in [27, 28, 46, 47]. Due to their role as basis for the main structures proposed in this Thesis, a more detailed description of the GraphGGSX and CT-Index structures is provided in the following subsections.

3.2.1 Filtering with GraphGrepSX

GraphGrepSX, GGSX for short henceforth, is an FTV method whose filtering stage builds and uses an index structure based on a trie structure. To achieve this, GGSX extracts from each graph contained in the database all the paths with a length lower or equal to a specified maximum length s. The paths are sequences of vertex labels connected by edges. Paths are generated by starting the traversal of the graph from every vertex. A formalization of the concept of path contained in a graph is given below.

Definition 4 *A* path *p of length $s > 0$ contained in a graph $g = (V, E, vl, el)$, denoted $p^s \subset g$, is a sequence of the form $< v_1, e_1, \ldots, v_s, e_s, v_{s+1} >$, where $\forall i \neq j, 1 \leq i, j \leq s+1, v_i \in V, v_j \in V, v_i \neq v_j$ and $\forall i \neq j, 1 \leq i, j \leq s, e_i \in E, e_j \in E, e_i \neq e_j, e_i = \{v_i, v_{i+1}\}$. Nodes $\{v_1, v_{s+1}\}$ are called the boundaries of p^s.*

Let $D = [g_1, g_2, \ldots, g_n]$ be a graph database of size n. If s denotes a maximum given length, then $P(s, D)$ denotes the set of all paths, whose length is lower or equal than s, which are contained in some graph of D. Formally,

$P(s, D) = \{p^s | s \leq S \wedge p^s \subset g \wedge g \in D\}$

The paths are used in the index building stage to incrementally build an index trie, with a maximum depth of S. Each node of the trie stores a node label and represents a path from the root to that node. The nodes also store a list of key-value pairs (*gid*, *r*), where *gid* is the identifier of a graph stored in the dataset that contains the path represented by the node, and *r* denotes the number of times that the relevant path appears in the stored graph. It is noticed that these lists of pairs increase their size linearly with the database size. On the other hand,

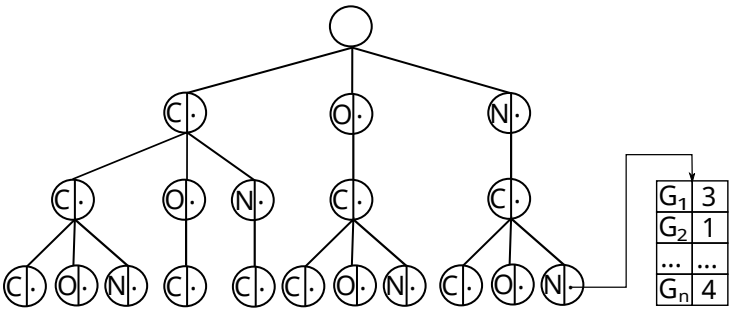


Figure 3.3: Representation of a trie structure used as index in GGSX.

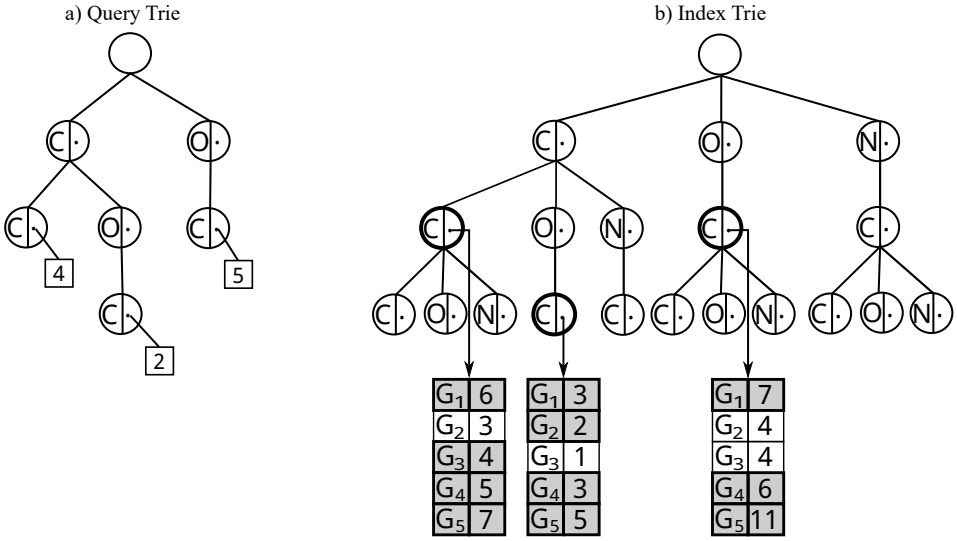


Figure 3.4: Example of the filtering stage in GGSX.

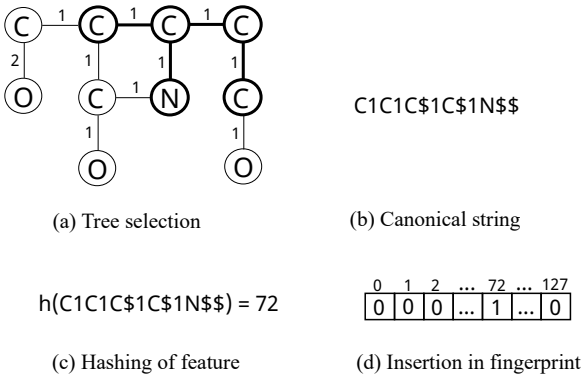
the depth of the trie is completely dependent on the value chosen as maximum path length S. An example of a small trie is represented in Figure 3.3. Each node records a list of key-value pairs, although only one of them is depicted in the figure. Thus, as it is represented in the bottom right of the figure, the path "NCN" is contained 3 times in graph G_1, 1 time in G_2, and 4 times in the graph identified by G_n.

As any other FTV technique, subgraph query processing in GGSX is divided into two stages, namely filtering and verification. In the filtering stage, firstly, all the paths whose

length is lower or equal to the maximum length S are extracted from the query graph and used to build a query trie structure as the one described above. In the case of the query trie, the nodes only record their label, except for the leaf nodes which also record the number of repetitions of the relevant path in the query. The filtering algorithm performs a joint breadth-first traversal of both query and database tries that stops when all the leaf nodes of the query trie are processed. When a query trie leaf node is reached, the number of repetitions r recorded in the node is compared with the list of pairs stored in the corresponding database trie node. The identifiers of the graphs that contain the path at least r times are retrieved and stored in a temporary list. When the joint traversal of both tries finishes, the filtering algorithm performs an intersection between the lists of graph identifiers retrieved from the comparisons of each query leaf node, obtaining this way the final list of candidates.

Figure 3.4 shows an example of the filtering stage for a given query. Figure 3.4(a) represents the query trie and Figure 3.4(b) represents the database trie. During the joint traversal of both tries, when the query trie leaf node corresponding to the path "CC" is reached, the number of repetitions 4 is compared with all the number of repetitions recorded in the relevant node of the database trie. Thus, the list of graphs $<G_1, G_3, G_4, G_5>$ has at least 4 repetitions for path "CC". Similarly, for the path "COC", graphs $<G_1, G_2, G_4, G_5>$ have at least the 2 repetitions present in the query graph. Finally, only graphs $<G_1, G_4, G_5>$ have the 5 repetitions present in the query graph for path "OC". The intersection of the above lists of graph identifiers, $<G_1, G_4, G_5>$, is passed to the subsequent verification stage as the list of potential result candidates to which the VF2 subgraph isomorphism algorithm has to be applied.

Overall, it is expected that the chosen maximum length value S should have an impact on different aspects of the method performance. First, the index size and the index construction time should increase as s increases. It is reminded that S provides a maximum bound for the trie depth. The filtering time should also increase as S increases, since larger paths derive in larger tries with a large number of leaf nodes, and therefore, more comparisons and intersections between identifier lists have to be performed. On the other hand, the verification time should decrease as S increases. This is due to the fact that larger tries perform more intersections and therefore retrieve fewer potential candidates for verification. To the best of our knowledge, there is not, currently, any mathematical model for the determination of the best expected S value for a given dataset, therefore, it must be chosen based on some empirical experimentation.



(a) Tree selection (b) Canonical string

(c) Hashing of feature (d) Insertion in fingerprint

Figure 3.5: CT-Index fingerprint building example.

3.2.2 Filtering with CT-Index

CT-Index is an FTV method that uses a list of bitmaps of a fixed size (2^f bits) as an index for the filtering stage. Each bitmap, called fingerprint, is generated from features (cycles and trees) extracted from a graph contained in the database, thus the number of fingerprints in the index matches the number of graphs in the database. During the query evaluation, the fingerprint generated from the query graph has to be compared with all the fingerprints generated for the database graphs. Before we continue with further explanations, let us formalize the two types of features extracted from the graphs to generate the fingerprints.

Definition 5 *A* cycle *c of length $s > 0$ contained in a graph $g = (V, E, vl, el)$, denoted $c^s \subseteq g$, is a graph $c = (V_c, E_c, vl, el)$, where $V_c \subseteq V$ has the form $\{v_1, v_2, \ldots, v_s\}$, $E_c \subseteq E$ has the form $\{e_1, e_2, \ldots, e_s\}$, $\forall i, 1 \leq i < s, e_i = \{v_i, v_{i+1}\}$ and $e_s = \{v_s, v_1\}$.*

Definition 6 *A* tree *t of length $s > 0$ contained in a graph $g = (V, E, vl, el)$, denoted $t^s \subseteq g$, is a graph $t = (V_t, E_t, vl, el)$, where $\forall (v_i, v_j), i \neq j, v_i, v_j \in V_t$, it exists exactly one path p contained in g whose boundaries are $\{v_i, v_j\}$.*

The generation of a fingerprint from the cycles and trees extracted from a graph is illustrated in Figure 3.5. First, all the cycles and trees contained in the graph are generated, whose size is lower or equal to a specified maximum size S. Figure 3.5 (a) shows an example of a tree contained in a graph. The extracted feature, either tree or cycle has to be encoded with a canonical text representation. Figure 3.5 (b) shows the canonical text representation of the

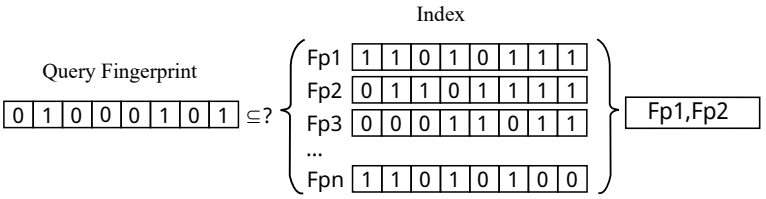


Figure 3.6: Example of the filtering stage in CT-Index.

tree extracted from the graph of Figure 3.5 (a). It is noticed that both node and edge labels are included in the canonical representation, which uses the symbol "$" to represent either the end of a tree branch or the end of the tree. Next, a hash function is applied to the canonical text representation described above to obtain an integer in the range $[0, 2^f)$. Figure 3.5 (c) shows how the hash function yields the number 72 for the canonical representation of the tree given in Figure 3.5 (b). Finally, the above number obtained from the result of the application of the hash function is used to activate (set to 1) a specific bit in the result fingerprint. Thus, in our example, Figure 3.5 (d) shows how bit number 72 of the fingerprint is set to 1. Summarizing, each feature extracted from the graph activates a specific bit in the fingerprint, which is selected using a hash function. Notice that different features might activate exactly the same bit (collision in the hash function) and that the probability of a collision is higher if the size of the fingerprint is smaller. Thus, now the size of the index is affected by the size of the fingerprint, however, the number of potential false positives in the filtering stage is affected by the combination of both the fingerprint size and the maximum feature size. As a final remark, it is also noticed that a fingerprint of a graph is actually a Bloom Filter [5] of all its features of maximum size S, where only a single hash function is used.

CT-Index is also an FTV technique, with relevant filtering and verification stages. In the filtering stage, a query fingerprint F_q is generated for the query graph in the same manner as described above, and with same parameters S and 2^f. F_q is then tested for the bitmap subset with each fingerprint F_g of the index ($F_q \wedge F_g = F_q$). An example of the filtering stage is shown in Figure 3.6, where after testing the query fingerprint for the bitmap subset, only the graphs represented by the fingerprints Fp_1 and Fp_2 result as candidates for the subsequent verification stage. The VF2 subgraph isomorphism algorithm, improved with some heuristics, is used to obtain the query results from the candidates in the verification stage.

It should be noted that with this technique, two parameters can be chosen to construct the index, namely, the maximum feature length S and the fingerprint size 2^f. Contrary to the case

of the GGSX index, now the maximum feature length S has no direct impact either in the index size or in the filtering time. However, large values of S combined with insufficiently large values of 2^f would lead to in many collisions of the hash evaluations during the fingerprint construction, and also in fingerprints with many bits set to 1. Therefore, the list of false positives in the candidate list proposed by the filtering stage would be increased and thus the effort done by the verification stage will also be increased, resulting in an overall poor performance in terms of response time. So, the increase of S would result in better filtering only if 2^f is increased accordingly, and therefore, the index size is also increased. On the other hand, an increase of 2^f has a direct negative impact on both the index size and the filtering response time. However, such a cost increase in filtering does not have a positive impact on verification if the value of S is not increased accordingly, to enable the pruning of more candidates during filtering.

Chapter 4

Subgraph Searching Techniques for Centralised Architectures

This chapter provides a description of the design of three new data structures for the filtering stage of FTV subgraph searching techniques, based on the GGSX and CT-Index approaches described in the previous chapter. The Bitmap GGSX (BM-GGSX), described in SubSection 4.1, replaces the collections of graph identifiers and path repetitions stored in GGSX nodes with more efficient compressed bitmap structures. The use of binary operations to determine the candidate graphs enables achieving performance gains of orders of magnitude with respect to the one achieved by the intersection of collections of integer graph identifiers. The Column-Wise CT-Index (CW-CTI), described in SubSection 4.2, stores the CT-Index fingerprints of all the graphs in column-wise order, i.e., all the values of the same bit position of all the fingerprints are stored together in a single compressed bitmap. This way, the bitmap subset testing between the query fingerprint and each of the database fingerprints is replaced by binary operations among large compressed bitmaps. This results in important performance gains for small queries, i.e., query fingerprints with few bits set to 1. For large queries however, the large number of 1 bit values in the query fingerprint demands too many bitmap operations. Thus,another strategy, based on the use of a tree of fingerprints, is used by the K-Means CT-Index (KM-CTI), which is described in SubSection 4.3. KM-CTI achieves better performance with large and highly selective queries, due to the reduction of the search space achieved by the use of the index tree of fingerprints.

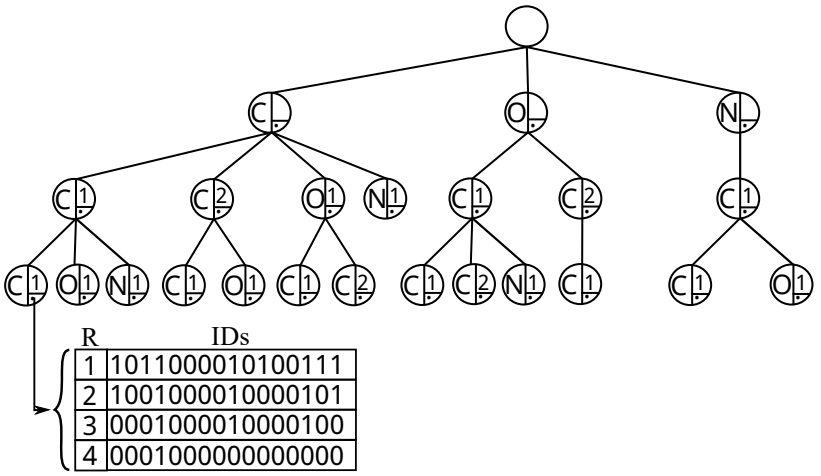


Figure 4.1: Example of a BM-GGSX trie structure.

4.1 Bitmap GGSX

An improved version of the GGSX method, called Bitmap-GGSX (BM-GGSX for short henceforth), is explained in detail in this section. The original GGSX method is modified in two main aspects: i) The edge labels of the paths are now considered and stored in the BM-GGSX trie nodes, giving an additional filtering power to BM-GGSX over GGSX. ii) The list of pairs (*gid*, *r*) of graph identifiers *gid* and repetitions *r*, are represented in BM-GGSX using a collection of very large compressed bitmaps, achieving this way a more compact and efficient structure.

The canonical representation of paths in the original GGSX method considers only the vertex labels. Therefore, if we compare it with the possible additional incorporation of the edge labels, the generated trie structures are smaller. On the one hand, smaller tries derive in shorter index size and faster filtering response times. But, on the other hand, the number of candidates discarded by the filtering stage is also shorter, and as a consequence, the verification effort and time are larger. BM-GGSX incorporates a new field in each trie node to record the edge labels. With this addition, a node in the BM-GGSX trie structure represents a combination of a graph node with the edge that precedes the node in the path. Obviously, given that paths start at nodes and not at edges, the nodes of the first level of the trie have always null values in the edge labels. The resulting trie structure has more nodes than the

(a)

ID	R
0	2
2	1
3	4
8	3
10	1
13	3
14	1
15	2

(b)

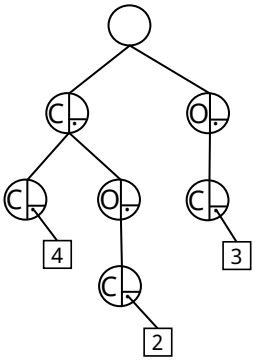
R	IDs
1	1011000010100111
2	1001000010000101
3	0001000010000100
4	0001000000000000

Figure 4.2: Comparison between (a) GGSX and (b) BM-GGSX path repetitions storage.

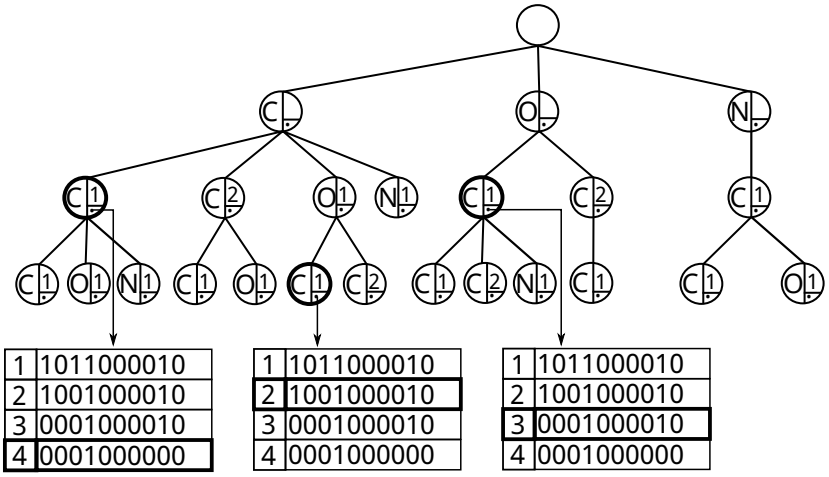
primitive one at each level but it enables identifying paths that, despite being different, the original structure encoded with identical canonical representation. An example of a small BM-GGSX trie structure is depicted in Figure 4.1. Notice that each node below the second level of the trie contains both a node label (C, O, N, etc.) and an edge label (1 for single, 2 for double, etc.).

The original GGSX method stores in each node a list of key-value pairs (*gid*, *r*), where *gid* is an integer identifying the graph that contains the relevant path, and *r* is the number of repetitions of the path in the graph. The size of these key-value lists increases linearly with the size of the database. BM-GGSX replaces these lists with a structure that exploits the use of bitmaps. In particular, each node stores an array of bitmaps. The bitmap recorded at index *r* of the array has a 1 in its position *i* if, and only if, the graph with identifier $gid = i$ contains the relevant path at least *r* times. The bitmaps are compressed in using the Enhanced Word-Aligned Hybrid method [33]. The difference between the two representations used by GGSX and BM-GGSX, respectively, to store the number of times that each path of the trie is repeated in each graph is illustrated in Figure 4.2. Notice that, in BM-GGSX, if the bitmap recorded at the index *r* of the array has a value of 1 at position *i*, the bitmaps recorded in indices lower than *r* must also have that position *i* set to 1. This inserts redundant information in the index structure, but it also reduces query response time, as will be explained below.

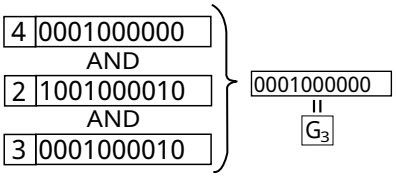
The filtering stage in BM-GGSX exploits the bitmap structure presented above to reduce the number of comparisons. Firstly, the query graph is processed to obtain a trie structure, and as in the original GGSX filtering stage, the BM-GGSX algorithm performs a joint breadth-first traversal of both query and database tries. For each leaf node of the query trie, the bitmap recorded in the position corresponding to the number of repetitions in the query leaf node is retrieved. Finally, the final set of candidates is obtained by performing a binary *AND* operation



(a) Query Trie



(b) Index Trie



(c) Final Candidates

Figure 4.3: BM-GGSX filtering stage.

between all the retrieved compressed bitmaps. This operation results in a new bitmap where the bits set to 1 identify the result graph identifiers. Figure 4.3 shows an example of the BM-GGSX filtering stage. More precisely, Figure 4.3(a) shows a query trie with 3 leaf nodes. Figure 4.3(b) shows the database trie, where the nodes corresponding to paths of the query trie leaf nodes are highlighted. Thus, for the node representing path "CC", the bitmap in position number 4 is retrieved, as 4 is the number of repetitions recorded in the corresponding query trie leaf node. Similarly, for path "COC" the algorithm retrieves the bitmap at position 2 of the array, and for path "OC" bitmap number 3 is retrieved. Finally, the binary *AND* operation is performed, as it is illustrated in Figure 4.3(c), to obtain the final set of identifiers of graph candidates (only graph G_3 is a candidate for verification in this example).

In the verification stage, BM-GGSX performs a parallel execution, in a multi-thread architecture, of the improved version of the VF2 algorithm provided by the CT-Index method.

4.2 Column-Wise CT-Index

Column-Wise CT-Index, CW-CTI for short henceforth, is a modification of the CT-Index method described in SubSection 3.2.2. As described above, the CT-Index structure consists of one fingerprint for each graph of the database. The collection of all fingerprints might be seen as a matrix, where each fingerprint is a row of the matrix and each column is a bit position of the fingerprint. It is noticed that this matrix is recorded row-wise in CT-Index. CW-CTI adapts CT-Index by recording and processing the fingerprints column-wise, leveraging the use of compressed bitmaps. Thus, if 2^f is the fingerprint's length, then the CW-CTI index is a sequence of 2^f bitmaps of length n (database size), where bitmap number k has a 1 at position i if, and only if, graph G_i has a 1 at the position k of its fingerprint.

Algorithm 1 CW-CTI Index Building

```
1: procedure APPENDGRAPH(g, idx)
2:     for all c ∈ extractCycles(g, S) do
3:         idx[identifier(g)][hashCycle(c, f)] ← 1
4:     end for
5:     for all t ∈ extractTrees(g, S) do
6:         idx[identifier(g)][hashTree(t, f)] ← 1
7:     end for
8: end procedure
```

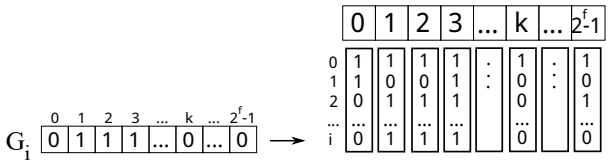


Figure 4.4: Fingerprint insertion in the CW-CTI index.

The index building stage generates fingerprints for each graph in the dataset, in the same manner as that of CT-Index. However, in CW-CTI each bit of the obtained fingerprint is appended to the end of each corresponding column bitmap. Algorithm 1 shows the pseudocode of the algorithm that appends a new graph g to an already existing CW-CTI index idx. First, the extracted cycles from g are processed in lines $2-4$ and next lines $5-7$ do the same with the extracted trees. For each extracted feature (either cycle or tree of size lower or equal to a given one s), a function (either *hashCycle* or *hashTree*) generates first a canonical string representation of the feature using the labels of vertices and edges, and next, applies a hash function to this canonical representation to obtain an integer index k ($0 \leq k \leq 2^f - 1$). The bit of the structure corresponding to the extracted feature is set to 1 using the graph identifier as a row index and k as a column index. Figure 4.4 illustrates an example of the application of this algorithm. Graph G_i has $1's$ at positions 1, 2, and 3, hence, bit number i of the bitmaps corresponding to those three positions is set to 1.

It is noticed that the bitmaps of CW-CTI are much larger than the fingerprints stored in CT-Index, and they are likely to contain larger sequences of bits with identical value, becoming great candidates for the use of bitmap compression. CW-CTI uses Enhanced Word-Align Hybrid compression [33], which enables the reduction of index size compared to CT-Index.

The filtering stage in CW-CTI is completely different from that of the original CT-Index. The process is illustrated in Figure 4.5. First, the query fingerprint is obtained in the same way as it is obtained in the CT-Index method (see Figure 4.5(a)). Next, for each bit set to 1 in the query fingerprint, the compressed bitmap of the relevant position is retrieved from the index structure (see Figure 4.5(b)). In the example, the bits set to 1 in the query are in positions 0, 1, k and $2^f - 1$. Therefore, the compressed bitmaps recorded in those positions are obtained from the data structure. Finally, the binary operation *AND* is applied to all the retrieved compressed bitmaps to obtain a result bitmap whose bits set to 1 refer to candidate graphs for the subsequent verification stage.

The verification stage in CW-CTI performs a subgraph isomorphism test to the final set of

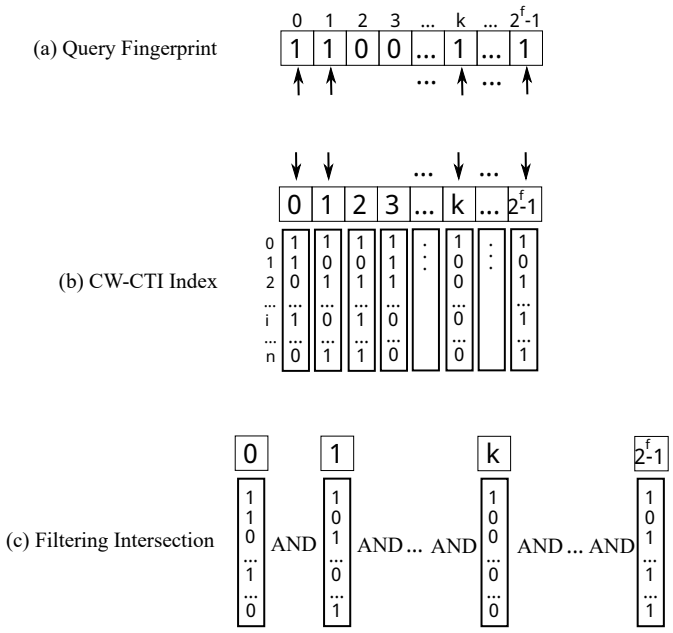


Figure 4.5: CW filtering stage.

candidates with a parallel execution, in a multi-thread architecture, of the improved version of the VF2 algorithm used by the CT-Index method.

4.3 K-Means CT-Index

K-Means CT-Index, KM-CTI for shot henceforth, is another proposal based on the CT-Index method described in SubSection 3.2.2. KM-CTI was constructed with the aim of reducing the number of comparisons performed in the original CT-Index method, which test bitmap containment between the query fingerprint and each one of the fingerprints stored in the index structure. Clearly, the filtering time of CT-Index increases linearly with the database size. KM-CTI uses a binary tree of fingerprints whose leaf nodes are the graph fingerprints. The fingerprints stored in internal nodes are obtained with the binary *OR* of their children fingerprints. Thus, if the bitmap containment test fails for a parent node it would also fail for all its children, and therefore, it is not needed to proceed with further exploration of that branch of the tree. Thus, if the database is large enough and the number of internal nodes has not many

bits set to 1, performing a search in the KM-CTI binary tree would require a lower number of comparisons than the number of database fingerprints, reducing therefore the processing time in the filtering stage. To minimize the number of bits set to 1 in internal nodes, the fingerprints under the same branch must be as much similar as possible. To achieve this, in the index building stage, the K-Means [17, 34] clustering method is recursively applied to the original set of fingerprints.

In general, K-Means is used to distribute a set of elements in K clusters, in a way that a given distance is minimized between the elements of each cluster. Generally speaking, the algorithm works as follows. First, K elements of the set are randomly chosen as centroids of the K clusters. Next, each element is allocated to its nearest centroid, according to a distance function. Once every element is assigned to a cluster, a new centroid is defined for each cluster. This is done by evaluating some kind of "mean" function between the elements of each cluster. These two processes of allocating elements to their nearest neighbor centroid and computing new centroids are iteratively repeated until the elements of each cluster do not change in two consecutive iterations.

The two functions, *distance* and *mean*, that enable the application of the K-Means algorithm to sets of fingerprints are defined below. Before those definitions, some notation must be introduced. If f is a positive integer, then F and F_i are used to denote *fingerprints*, i.e., one dimensional arrays of 2^f bits. Symbols $\wedge, \vee, =, \neg, \oplus$ denote respectively the bitwise operations *and*, *or*, *equal*, *not* and *exclusive or* between a pair of fingerprints. If i is an integer $0 \leq i < 2^f$, then $F[i]$ denotes the boolean element corresponding to bit number i of F. Finally, $int(b)$ denotes the integer element (either 0 or 1) corresponding to a boolean element b. The definition of *distance* between two fingerprints is based on the well-known Hamming Distance [20]. Informally, if either of the two fingerprints is contained in the other, then the distance is defined to be zero, otherwise, the distance is the number of bits set to 1 in the result of the exclusive or of both fingerprints. Formally, function *distance* is defined as follows:

Definition 7 *If F_1 and F_2 are two fingerprints, then distance(F_1, F_2) is defined as the integer*

$$d = \begin{cases} 0 & (F_1 \wedge F_2 = F_1) \vee (F_1 \wedge F_2 = F_2). \\ \sum_{i=0}^{2^f-1} int((F_1 \oplus F_2)[i]) & otherwise. \end{cases}$$

Function *mean* enables K-means algorithm to compute a fingerprint that is representative of a cluster of fingerprints as the centroid. Informally, the bit at position i of the *mean* of a set

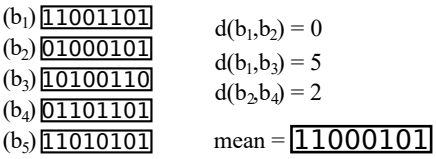


Figure 4.6: Example of distance and mean measures between bitmaps used in KM-CTI.

of fingerprints is set to 0 if at position i there are more fingerprints with a value of 0 than with a value of 1, and it is set to 1 otherwise. Function *mean* is formally defined as follows.

Definition 8 *If $S = F_1, F_2, \ldots, F_n$ is a set of n fingerprints, then $mean(S)$ is defined as the fingerprint M such that*

$$\forall i, 0 \leq i < 2^f, M[i] = \sum_{j=1}^{n} int(F_j[i]) \geq \sum_{j=1}^{n} int(\neg F_j[i])$$

The calculation of distance and mean between bitmaps is illustrated in Figure 4.6 with various examples.

The design of the KM-CTI index building algorithm is illustrated with the pseudocode of Algorithm 2. The input GF is a set of pairs (g, F), where g is a graph identifier and F is its corresponding fingerprint. The K-means clustering method is applied in line 2, using a value of $k = 2$, to classify the input pairs into two subsets, GF_L and GF_R. If either of the above fingerprint subsets is empty, then K-means did not manage to divide further GF and therefore a leaf node of the KM-CTI binary tree has to be created (line 3). The leaf node stores both the fingerprint computed as the binary *OR* of all the fingerprints in GF, and all the pairs of GF. On the other hand, if both GF_L and GF_R have elements, recursive calls to function $KMCTITree$ over both subsets are performed to continue with the creation of left and right branches of the

Algorithm 2 KM-CTI Index Building

```
1: function KMCTITREE(GF)
2:     clusterFingerprints(GF, GF_L, GF_R)
3:     if empty(GF_L) ∨ empty(GF_R) then result ← leafNode(GF)
4:     else
5:         result ← internalNode(KMCTITree(GF_L), KMCTITree(GF_R))
6:     end if
7:     return result
8: end function
```

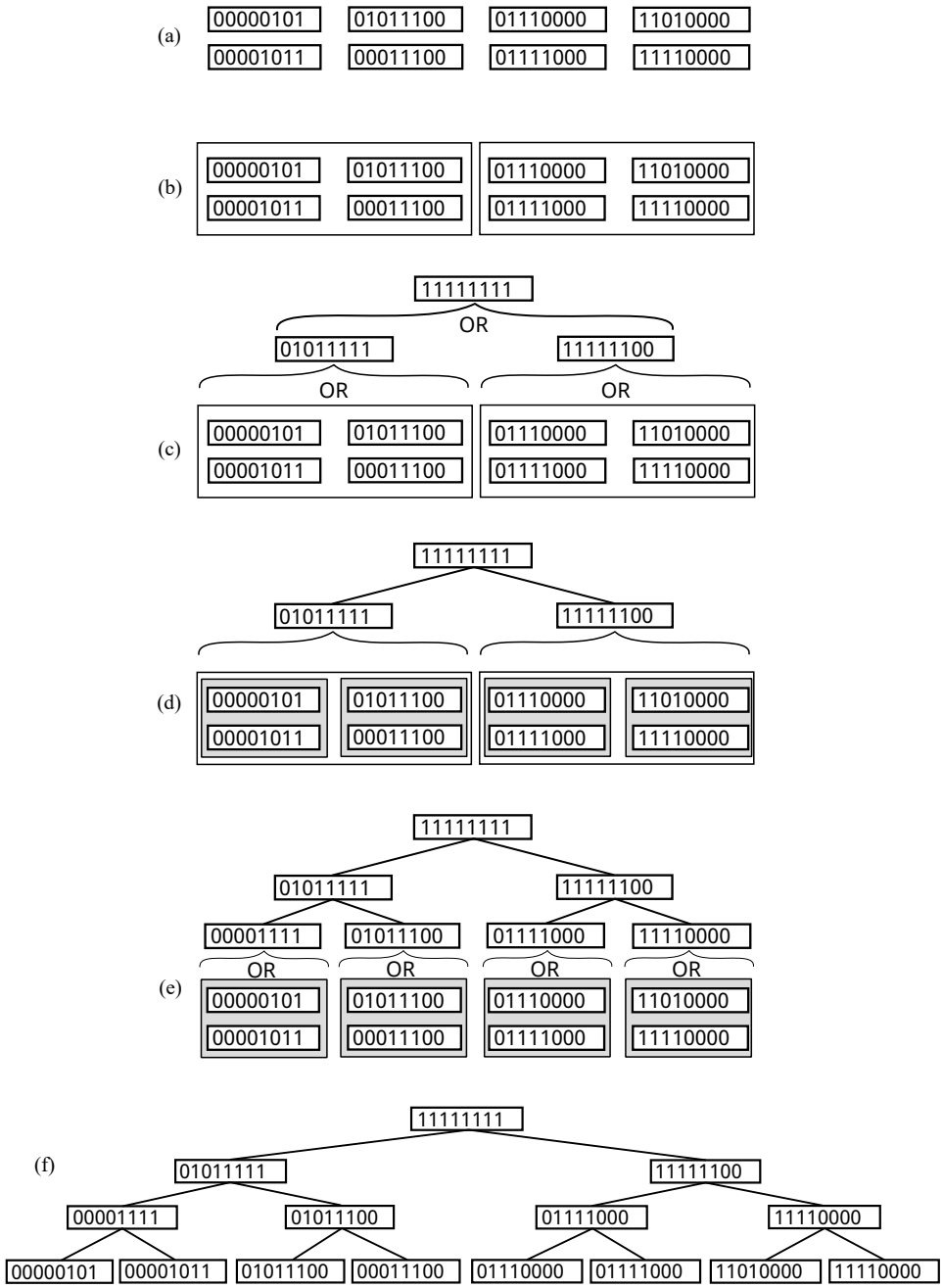


Figure 4.7: Example of the index building stage in KM-CTI.

KM-CTI binary tree. An internal parent node that stores pointers to left and right children and also a fingerprint computed as the binary *OR* of left and right fingerprints is created in line 5. An example of KM-CTI index building for a small set of fingerprints is illustrated in Figure 4.7. The set of eight input fingerprints is shown in Figure 4.7(a). Figure 4.7(b) shows the two subsets obtained by the first application of the K-mean method. The fingerprints chose to head both branches of the tree as shown in Figure 4.7(c), which shows also the fingerprint of the root element. All those fingerprints of internal nodes are obtained using binary *OR* operation over the children fingerprints. Figures 4.7(d) shows the subsequent subdivision of each branch of the tree, and Figure 4.7(e) illustrates the calculation of relevant internal nodes. The final KM-CTI structure is depicted in Figure 4.7(f), where each of the input fingerprints is stored in the index as a leaf node. Notice that it is not always the case that a leaf node contains a single fingerprint, since K-Means might not be able to further subdivide a cluster with two or more fingerprints.

It is remarked that KM-CTI is an in-memory data structure, and therefore a binary tree is expected to offer better results than trees of higher order. However, the K value might be increased in order to obtain a structure more suitable for secondary storage, where node size tends to match disk block size, as is the case of structures like the B+-tree.

In the filtering stage, KM-CTI scans the binary tree in depth-first order, as it is shown in the pseudocode given in Algorithm 3. The current node is explored only if the query

Algorithm 3 KM-CTI Searching

```
1:  function SEARCH(F_q, idx)
2:      if F_q ∧ fingerprint(idx) = F_q then
3:          if isLeaf(idx) then
4:              result ← ∅
5:              for all (g, F) ∈ graphs(idx) do
6:                  if F_q ∧ F = F_q then result ← result ∪ g
7:                  end if
8:              end for
9:              return result
10:         else
11:             return search(F_q, leftChild(idx)) ∪ search(F_q, rightChild(idx))
12:         end if
13:     else return ∅
14:     end if
15: end function
```

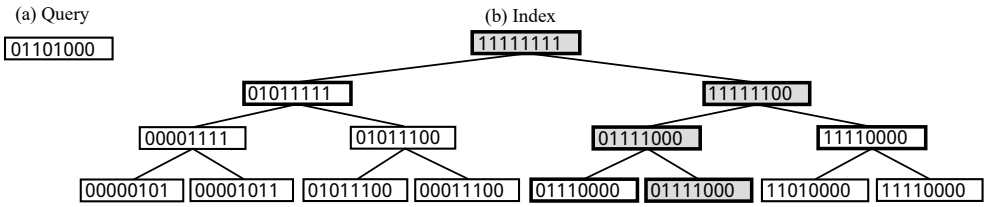


Figure 4.8: Example of the filtering stage in KM-CTI.

fingerprint F_q is contained in the node fingerprint (line 2). Such a test is also performed for each graph fingerprint contained in a leaf node (line 6). An example of the filtering stage is illustrated in Figure 4.8. The query fingerprint, shown in Figure 4.8(a), is contained in the root node of the structure (Figure 4.8(b)), therefore, the search process must continue through left and right branches. The query fingerprint is not contained in the fingerprint of the left node, therefore the left branch of the tree does not need to be explored. On the other hand, the query fingerprint is contained in the fingerprint of the right node. The search process continues therefore through the right branch of the tree structure. The nodes of the tree that result in a positive result for the containment test with the query fingerprint are depicted in the figure with dark background. The result candidate graph is the one corresponding to the leaf fingerprint reached by the searching process. It is noticed that, in such a small example, the tree enabled to obtain the result by performing 7 fingerprint containment tests. However, the CT-Index structure would require one test for each input fingerprint, i.e., 8 tests in this example.

In general, it is expected that KM-CTI will give good results in terms of filtering time when the number of bits set to 1 in the query is not short and when the number of bits set to 1 in database graph fingerprints is not too large. In those cases, it is expected that the search process will not follow too many branches of the tree. Few bits set to 1 in the query happens with small query graphs, i.e., with queries of low selectivity. It is already known, in general, that indexes do not perform well with low selectivity queries. On the other hand, too many bits set to 1 in the database graph fingerprints would be an indication of a fingerprint size too small for the number of features to be represented, i.e., too many collisions of the hash function, and therefore, bad behaviour of the fingerprint to represent the graph features.

In the verification stage, the candidate set of graphs obtained by the filtering stage is tested for subgraph isomorphism using the VF2 algorithm proposed by the original CT-Index

method, which is executed in parallel in a multi-thread architecture.

4.4 Performance Evaluation

This section presents and discusses the experiments performed to evaluate the methods described above. The first subsection describes the system setup, datasets and queries used in the evaluation. Next, a description of the benchmark undertaken to experimentally select the appropriate values for maximum path length and fingerprint size is given. Finally, the last subsection is devoted to the comparison of the performance of the different techniques, using various query and database sizes.

4.4.1 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. All the indexing structures and both filtering and verification stages for all the evaluated methods were implemented in Java language. The GGSX implementation is based on the original C++ implementation provided by the authors of this method. A reverse engineering was done in the case of CT-Index, using as starting point a distribution version provided by the authors. The experiments were executed using a Java Virtual Machine set up to use a maximum of 32 GB of RAM. The parallel verification stage of the proposed BM-GGSX, CW-CTI, and KM-CTI methods was performed using 10 threads.

The databases used in the experiments were constructed using parts of two different well-known datasets. The AIDS[1] dataset has already been used to evaluate other previous works of the state of the art [6, 19, 26, 29, 32]. AIDS is a small molecules dataset that is composed of 42689 graphs, with an average of 45.70 vertices and 47.71 edges per graph. The second dataset, PubChem[2], is a molecular dataset that had around 96 million compounds at the time the experiments were performed, with an average of 41.40 vertices and 42.16 edges per graph. Databases with sizes ranging from 100 k to 1 M graphs were created from the PubChem dataset to evaluate the scalability of the approaches with growing database sizes.

Queries of sizes ranging from 4 to 40 edges (4, 8, 12, 16, 20, 24, 28, 32, 36, 40) were used to perform the experiments. For each query size, a set of 1000 query graphs was constructed.

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

The construction of the queries was performed by extracting subgraphs from the dataset in the same way as in previous works [26, 29]. For each query to be generated of a specific size q_s (number of edges), a graph and a vertex of the graph were randomly selected using a uniform distribution. From such a starting vertex of the graph, a neighbour vertex was recursively and randomly chosen, to complete a tree of q_s edges. Two different sets of queries were constructed, one for the AIDS dataset and another for the PubChem dataset.

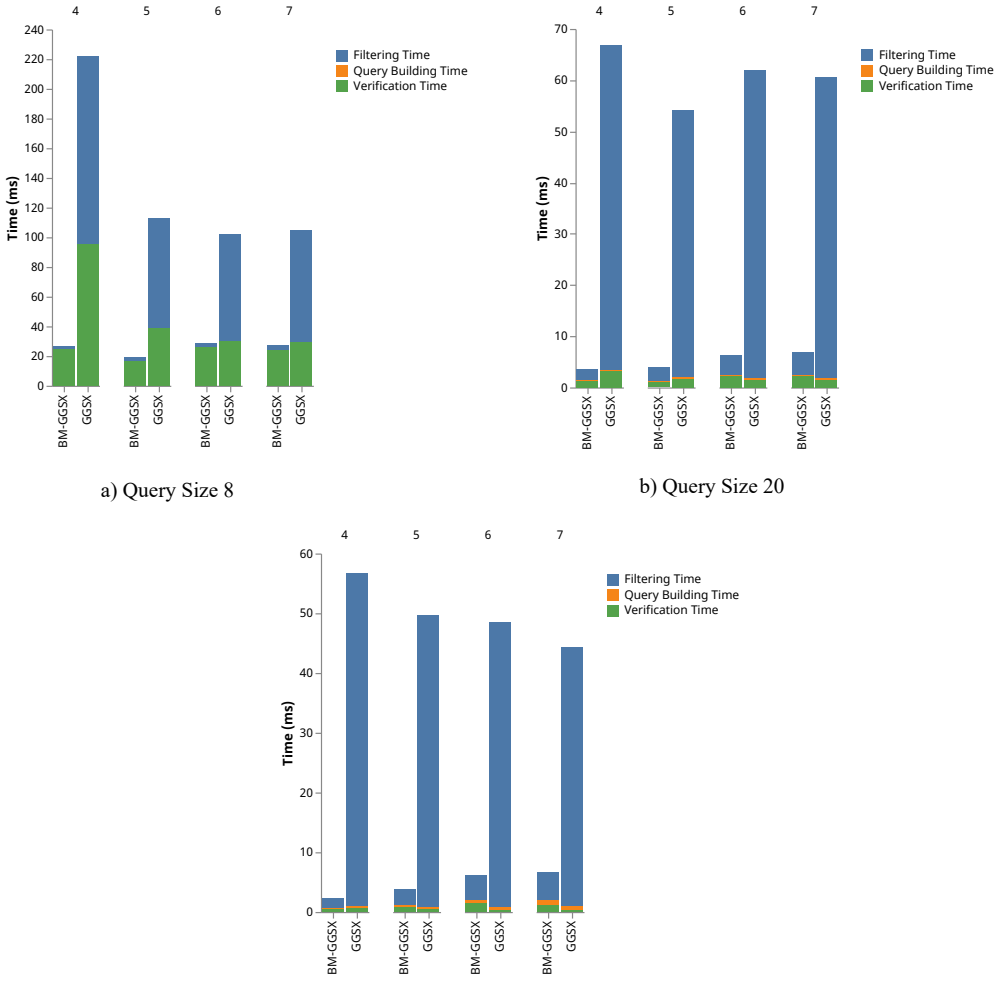
4.4.2 Parameter Selection

This subsection describes a first set of tests that were performed with two main objectives. The first one was to test the behavior of the five techniques: the two methods obtained from the state of the art (GGSX and CT-Index) and the three new proposals described in this chapter (BM-GGSX, CW-CTI and KM-CTI), using two different real-world datasets and different query sizes. The second objective was the evaluation of the different approaches using different values for their configuration parameters. In particular, GGSX and BM-GGSX were tested using different values for the maximum path length. On the other hand, the techniques based on the CT-Index approach were tested using different combinations of maximum feature (either tree or cycle) size and fingerprint size.

The two datasets used for these experiments are the following: i) The complete AIDS dataset, containing 42689 graphs and ii) A set of 200 k graphs obtained from the PubChem dataset, which for sake of referencing, will be called *PubChem200k* henceforth. A set of 1000 queries extracted from each dataset was evaluated five times each query for each search method. The experiments were repeated four times at different moments. The main results obtained are discussed below.

In general, as it was expected, the size and building time of GGSX and BM-GGSX trie structures increase 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. The impact of the database size on both index size and index building time will be further analyzed in a second set of experiments whose results are discussed in the next subsection.

To start with the discussion of the results, Figure 4.9 shows a comparison of query processing response times between the original GGSX and BM-GGSX on the AIDS database. In general, it can be noticed that BM-GGSX outperforms the original GGSX for all query sizes and for any value of the maximum path length (4, 5, 6 and 7). In particular, Figure 4.9(a),



a) Query Size 8

b) Query Size 20

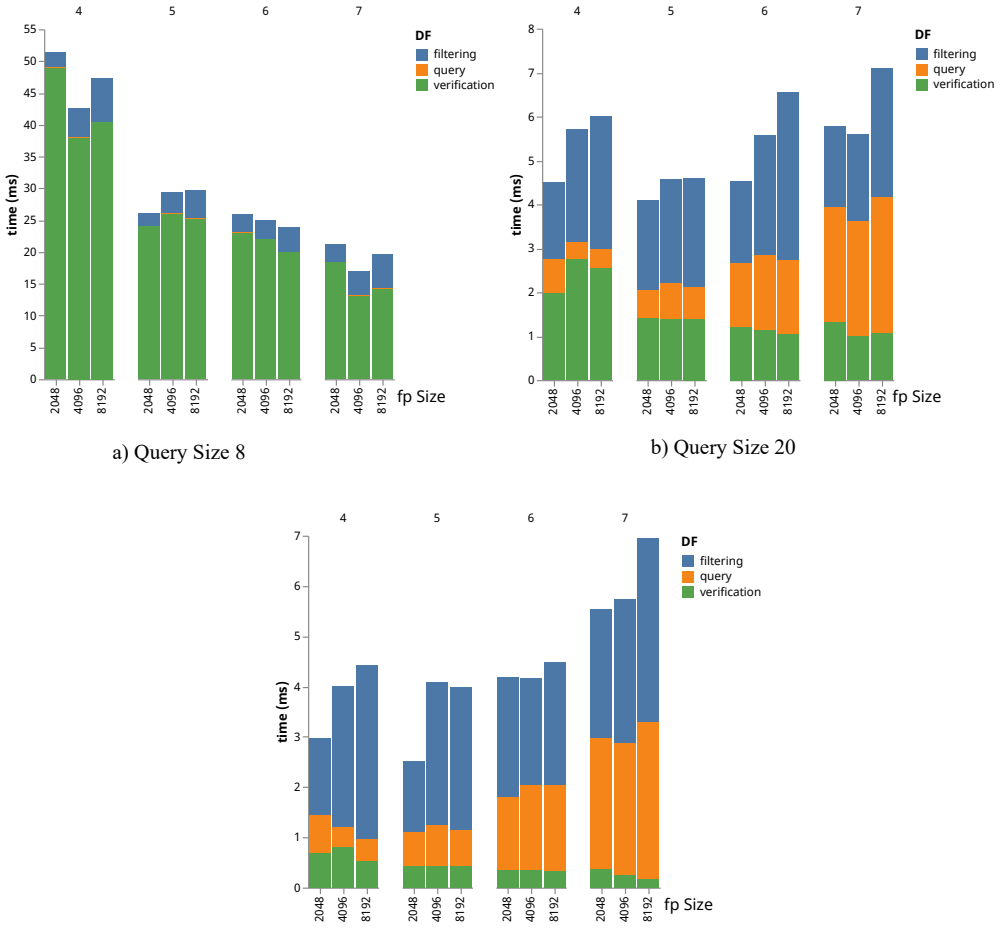
c) Query Size 40

Figure 4.9: Comparison of response times(ms) between GGSX and BM-GGSX on AIDS for different maximum path sizes and query sizes.

shows how, for small queries (8 edges), BM-GGSX achieves much better filtering time for all values of the maximum path length, but also better verification time especially when the value of the maximum path length is short (4 and 5). The reduction of the filtering time is due to the new bitmap based representation of the data of each node of the trie and to the relevant implementation based on bitmap operations. The reduction in the verification time is due to the incorporation of edge labels in the represented paths. It is noticed how such incorporation of edges is really significant when small values of maximum path length are used. In fact, due to this, the best value of such maximum length for BM-GGSX is 5, whereas the best value for GGSX is 6. For queries of medium (20 edges) and large (40 edges) size, again, the reduction of the filtering time achieved by BM-GGSX with respect to the original GGSX is huge, as it may be observed in Figures 4.9(b-c). Regarding verification time, some reduction may also be observed in the case of medium size queries and short path lengths (4 and 5). Thus, it may be concluded that the incorporation of edge labels is only worthwhile when short paths are used in the structure and also the queries are not too large. Regarding the best value for the maximum path length, there are not really significant differences for BM-GGSX in the response times achieved with maximum path lengths of 4, 5, 6 and 7. A short value of 4 provides slightly better response times for medium and large sizes. Short values of the maximum path length have the additional advantage of providing better performance in both index size and index building time.

The comparison of query result times obtained by CT-Index on the AIDS database, using different values of maximum feature size and fingerprint size is shown in Figure 4.10. For small queries (Figure 4.10(a)), increasing the maximum feature size has an important positive impact on the reduction of verification time. This is also true for medium (Figure 4.10(b)) and large (Figure 4.10(c)) queries, however, verification time in those more selective queries is not as important as in small queries with low selectivity. Large fingerprints are only worth with large features sizes, but they incorporate additional overhead on filtering time. Thus, their positive impact on verification time is translated to an overall better performance only for small low selective queries.

The above conclusions obtained for the parameter selection of the CT-Index method, may also be applied to the other two methods based on CT-Index, i.e., CW-CTI and KM-CTI. In particular, the result times obtained by CW-CTI on the AIDS database, using different values of maximum features size and fingerprint size are shown in Figure 4.11, whereas Figure 4.12 shows the results obtained by KM-CTI for all the tested parameter combinations.



a) Query Size 8

b) Query Size 20

c) Query Size 40

Figure 4.10: Comparison of response times(ms) of CT-Index on AIDS for different maximum path sizes, fingerprint sizes and query sizes.

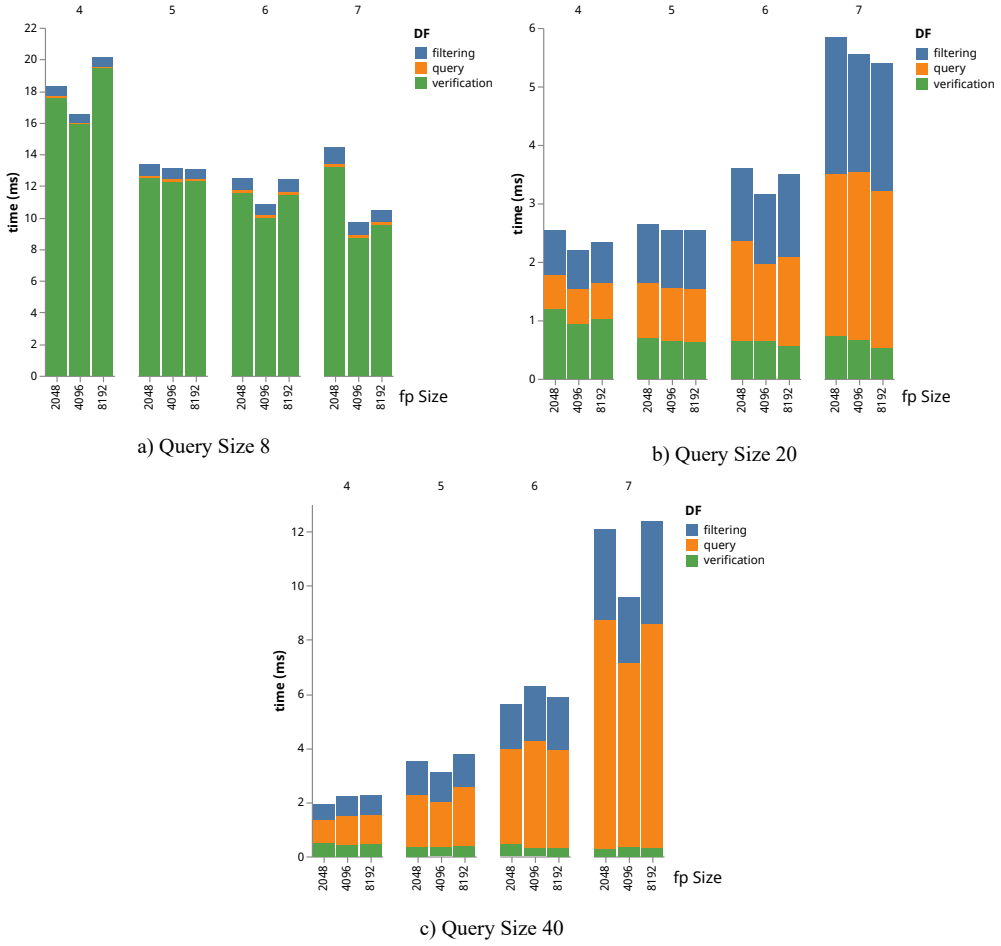


Figure 4.11: Comparison of response times(ms) of CW-CTI on AIDS for different maximum path sizes, fingerprint sizes and query sizes.

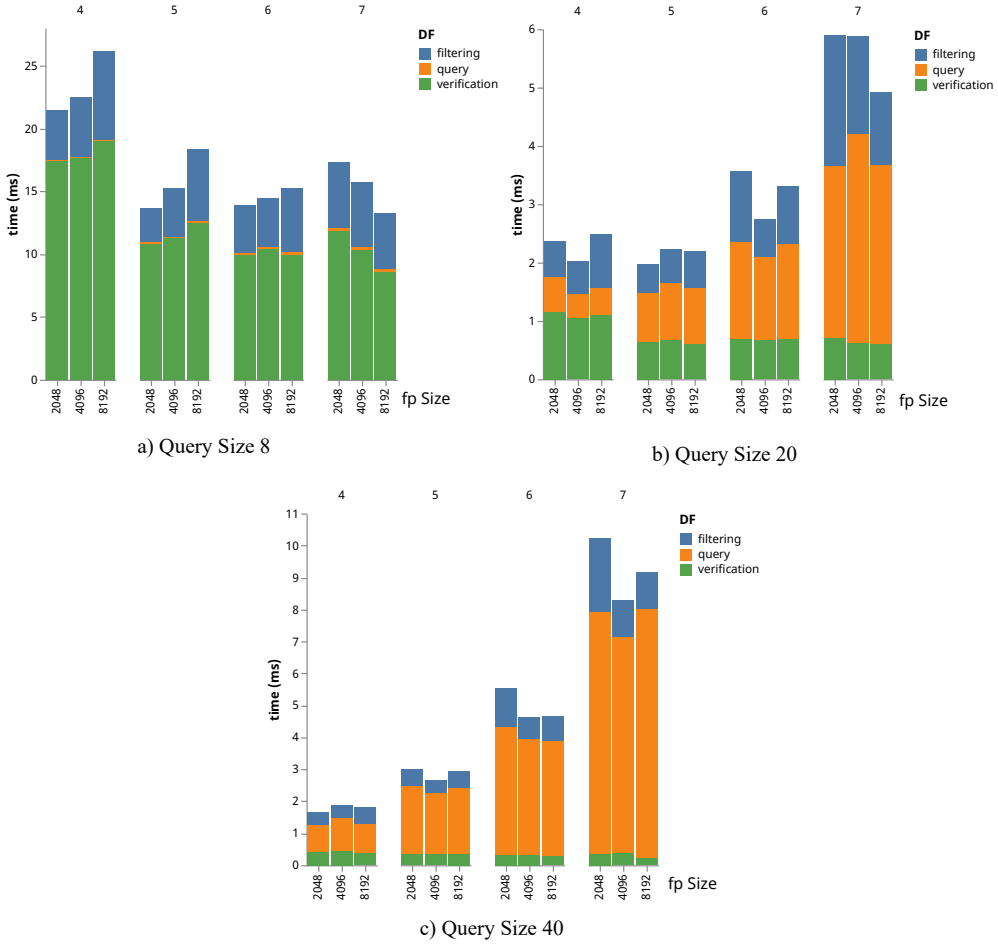


Figure 4.12: Comparison of response times(ms) of KM-CTI on AIDS for different maximum path sizes, fingerprint sizes and query sizes.

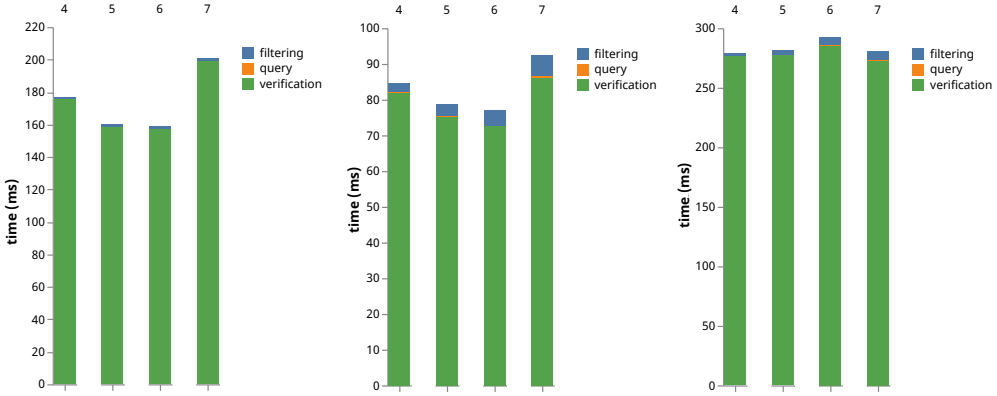


Figure 4.13: Comparison of response times(ms) of BM-GGSX on PubChem200k for different maximum path sizes and query sizes.

As a general conclusion derived from the experiment undertaken with the small AIDS dataset, small feature sizes of 4 or 5 are slightly better options for BM-GGSX than using larger sizes of 6 or 7. In the case of CT-Index based methods, large features sizes (6 or 7) accompanied by larger fingerprint size are better for small queries with low selectivity, where the verification time is dominant. However, when the queries turn to be larger and more selective, the verification time is not very important and reducing filtering time becomes more important, therefore, small feature sizes, which do not demand large fingerprint sizes, become better options.

A similar experiment was performed with the PubChem200k dataset described above. The mean characteristics of the graphs recorded in this new dataset are similar to those of the AIDS dataset, however, the number of graphs recorded is much larger. Therefore, the number of graphs retrieved on average by each query type is also larger, and therefore, the reduction of the verification time becomes more important. As we will see below, this has an impact on the results and thus, the conclusions obtained regarding the best feature and fingerprint size are also different.

Figure 4.13 shows the response times obtained by BM-GGSX for different query and feature sizes. The original GGSX method was not evaluated due to its poor performance with

this large dataset. Contrary to the experiment with the AIDS dataset, now the response time of the verification stage is the most important one for all the query sizes. Feature sizes 5 and 6 are now better for small and medium size queries. For large queries, the size of the extracted features does not seem to have a significant impact on the response time.

The results of the evaluation of CT-Index using PubChem200k are shown in Figure 4.14. Again, contrary to what was observed with the AIDS dataset, the increase of the feature and fingerprint sizes has an important positive overall impact on the response time for small and medium queries. For large queries, however, the impact is not so important. The conclusions are similar also for CW-CTI (Figure 4.15) and KM-CTI (Figure 4.16), except that now, a larger feature size (7) does not bring the best result in any case. In general, it seems that a feature size of 6 and a fingerprint size of 4096 has a good compromise between either the best or almost the best query response times and reasonable index size and building time.

Based on the above experimental results, and also taking into account relevant decisions made in a previous survey [26], in the following experiment all the methods were evaluated using a feature size of 6. In the case of methods based on CT-Index, the chosen fingerprint size was 4096. The objective of this new experiment is the comparison of the performance of all the methods (except the discarded original GGSX), using increasing database and query sizes. The results and relevant discussion are given in the next subsection.

4.4.3 Performance comparison

This subsection discusses the results of the performance comparison of all the methods (except GGSX, which was discarded after the previous experiments). As it was stated above, a maximum path length of 6 and a fingerprint size of 4096 were chosen for building the indices and queries.

A first comparison was done using the AIDS dataset and queries of sizes 8, 20 and 40 edges. The results are shown in Figure 4.17. For small queries (Figure 4.17(a)), CW-CTI and BM-GGSX offer the smallest filtering time, however, the verification time of BM-GGSX is larger than that of CT-Index based methods. The new CW-CTI and KM-CTI methods improve the verification response time with respect to the original CT-Index, due to the parallel evaluation of the graph isomorphism algorithm. The filtering of candidates achieved by BM-GGSX for medium (Figure 4.17(b)) and large (Figure 4.17(c)) queries is very good, reducing verification time and achieving, this way, a very nice total response time. Regarding CW-CTI and KM-CTI, in general, the former yields better filtering time for small queries, as expected,

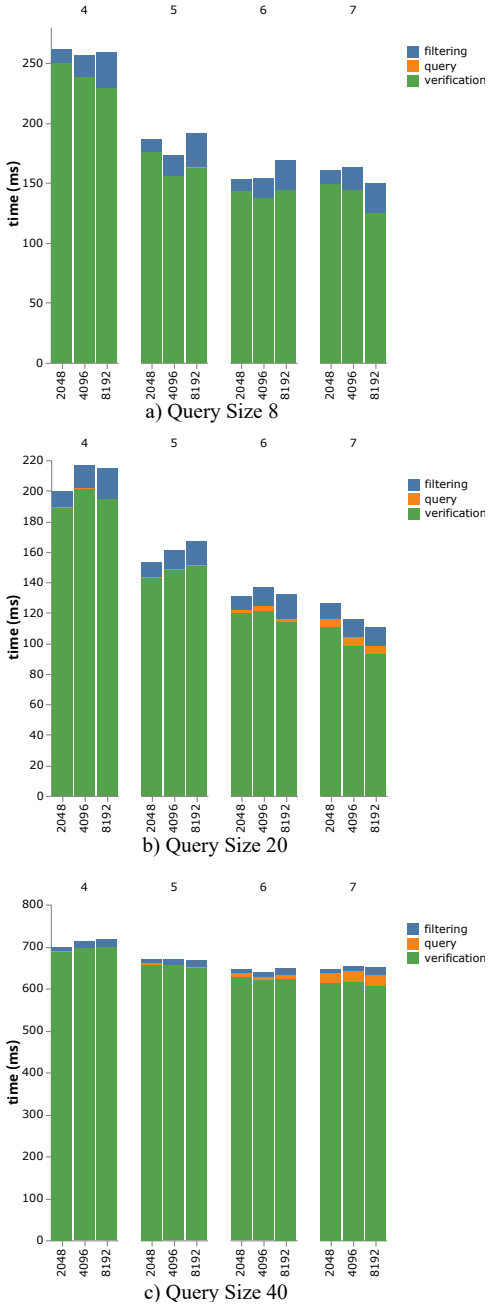
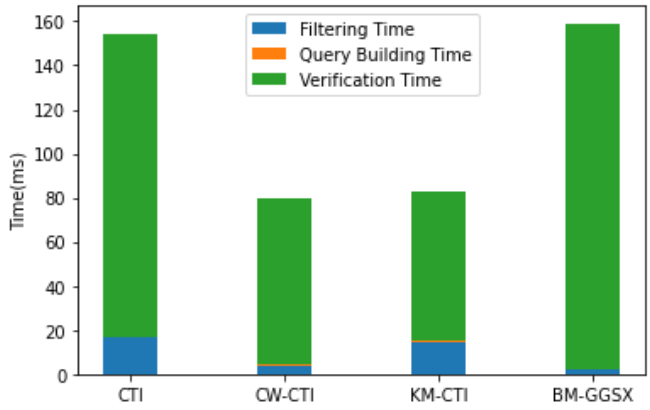


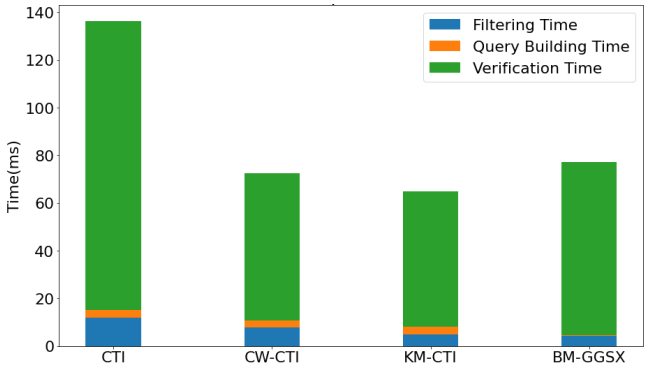
Figure 4.14: Comparison of response times(ms) of CT-Index on PubChem200k for different maximum path sizes, fingerprint sizes and query sizes.

Figure 4.15: Comparison of response times(ms) of CW-CTI on PubChem200k for different maximum path sizes, fingerprint sizes and query sizes.

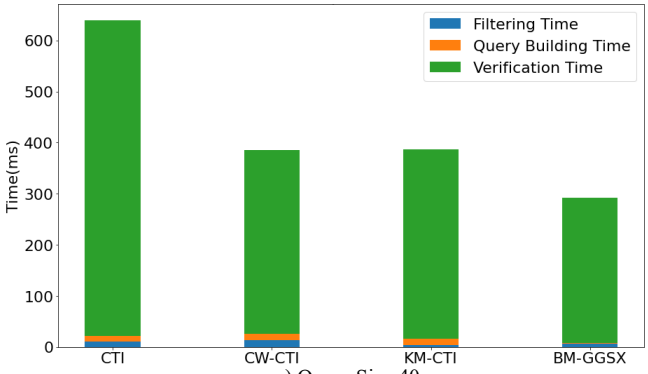
Figure 4.16: Comparison of response times(ms) of KM-CTI on PubChem200k for different maximum path sizes, fingerprint sizes and query sizes.



a) Query Size 8



b) Query Size 20



c) Query Size 40

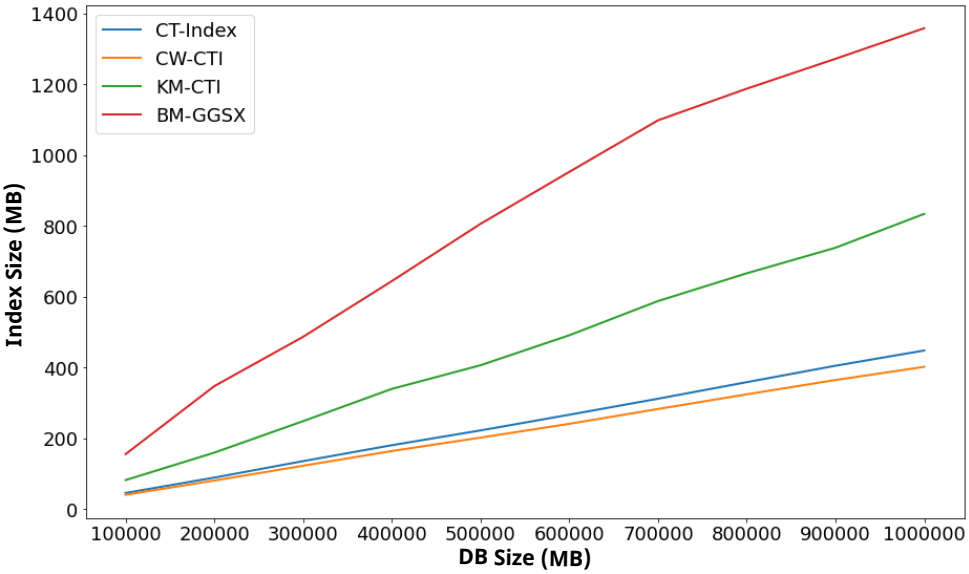
Figure 4.17: Performance comparison of the methods on the AIDS database for different query sizes.

whereas the latter performs better with large queries, as also expected. Remember that the filtering time of CW-CTI is highly affected by the number of bits set to 1 in the query fingerprint since those bits determine the number of bitmap *AND* operations that the method has to perform to obtain the candidate graphs. Small queries with few bits set to 1 are, therefore, evaluated faster with CW-CTI. On the other hand, KM-CTI uses a binary tree. So, in each internal node of the tree, a bitmap subset test between the query fingerprint and the node fingerprint is performed to decide whether the relevant subtree has to be explored or not. A larger number of bits set to 1 in the query increases the chances to have negative tests and, therefore, to avoid the exploration of larger parts of the index tree. These insights, shown by the first experiment, are confirmed with a more complete experiment using databases of increasing size.

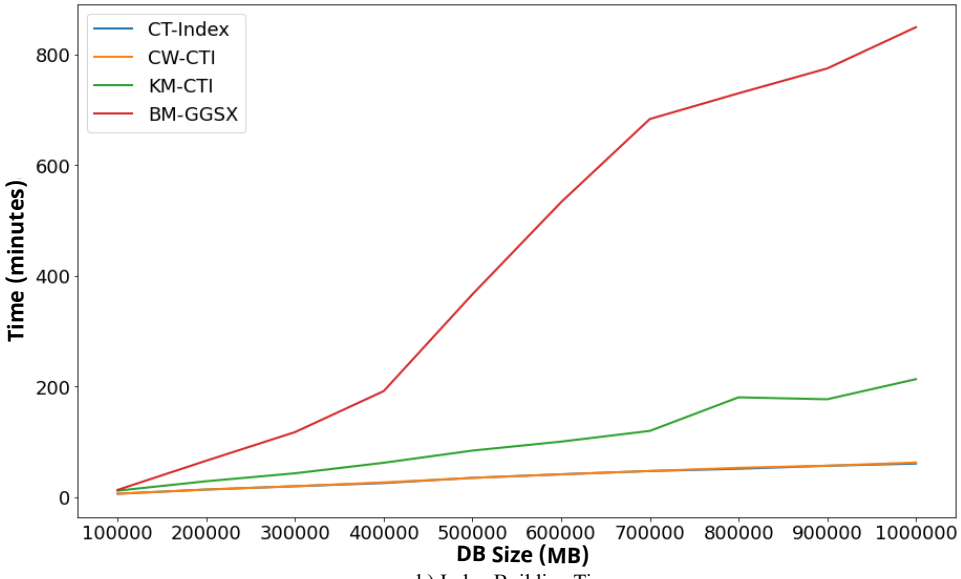
All the methods, except the discarded original GGSX, were evaluated using 10 databases obtained as subsets of PubChem, with sizes ranging from 100 thousand graphs to 1 million graphs. The impact of the database size on the index size of each method is shown in Figure 4.18(a). The best methods in terms of index size are clearly CT-Index and CW-CTI. It is noticed that, despite of the bitmap compression, CW-CTI does not achieve a really significant space reduction with respect to the original CT-Index. The need to record internal nodes in the fingerprint tree of the KM-CTI introduces an important space overhead, which might be a problem with very large databases. Notice that the size of KM-CTI is approximately double the size of CT-Index. Finally, BM-GGSX has a very high memory footprint, compared to the other methods. Notice that around 1.4 GB of RAM is needed to store the trie of BM-GGSX for the database of 1 M graphs.

The above index size results correlate with the index building time results, which are shown in Figure 4.18(b). The time required to generate the fingerprints and the relevant structures of CT-Index and CW-CTI ranges from around 6 minutes (100 k graph database) to around 1 hour (1 M graph database). On the other hand, KM-CTI is approximately 3 times slower in index building than the other two CT-Index based techniques. This is due to the recursive application of the K-Means method to construct the binary tree of fingerprints. Finally, BM-GGSX has the worst performance of all methods, needing more time to index 200k graphs, than the one needed by CT-Index and CW-CTI methods to index 1 M graphs. Its index building time increases drastically with the size of the dataset, and it needed more than 14 hours to index the largest database of one million graphs.

The response time of each method was obtained for the evaluation of the 1000 queries

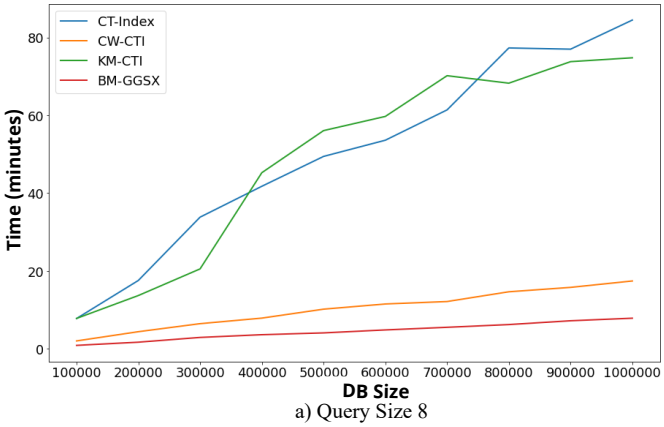


a) Index Size

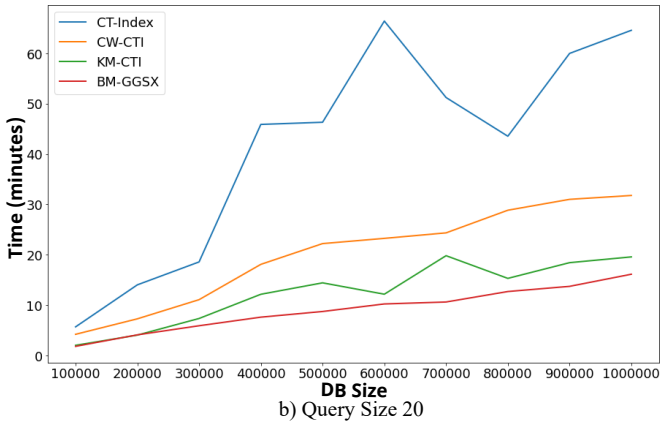


b) Index Building Time

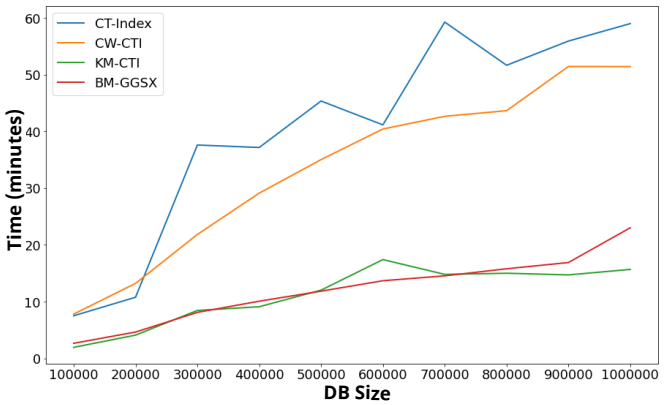
Figure 4.18: Index Size and Building Time for increasing database sizes.



a) Query Size 8



b) Query Size 20



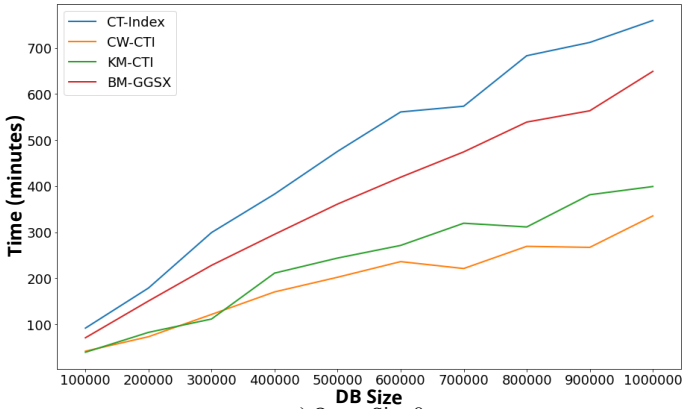
c) Query Size 40

Figure 4.19: Filtering time for increasing database and query sizes.

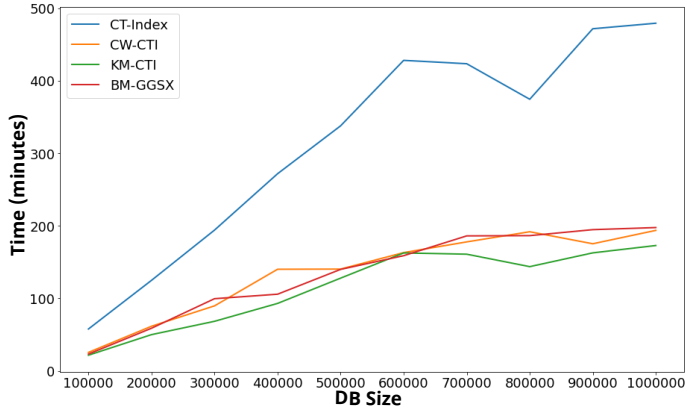
of each size described in SubSection 4.4.1. Each query was executed four times for each database size, and the whole experiment was repeated three times. The results for query sizes of 8 edges (small), 20 edges (medium) and 40 edges (large) are discussed below. Given that the main contribution of this work is in the proposal of new indexing structures for the filtering stage, special attention will be given to the filtering stage response times. Thus, the filtering time of each method for each query size and each database size is shown in Figure 4.19. More precisely, Figure 4.19(a) shows the results for small queries of 8 edges. CT-Index and KM-CTI offer similar results, enabling the filtering of 1 M graphs in around 80 milliseconds. This was expected as it is well-known that, with low selective queries, the use of a tree does not bring performance gains. On the other hand, BM-GGSX and CW-CTI achieve much better performance (around 80% better than CT-Index). This was also expected since, in both structures, the number of bitmap *AND* operations to be performed is directly dependent on the number of features of the query.

As the query size increases, the relative performance of CT-Index and BM-GGSX is maintained, however, the performance of CW-CTI degenerates and, in contrast, KM-CTI improves its performance. Clearly, CT-Index must always perform the bitmap subset test between the query and all the database fingerprints, thus, the query size (number of bits set to 1 in the query) does not have an impact on the performance. On the other hand, BM-GGSX must perform more bitmap *AND* operations with larger queries, however, it is also likely that more bitmaps with fewer bits set to 1 are involved, therefore, those operations on compressed bitmaps are evaluated faster. Regarding CW-CTI, larger query sizes imply more bits set to 1 in the query fingerprint, and therefore, more bitmap *AND* operations to be performed. Finally, KM-CTI has more chances to discard the exploration of larger pieces of its tree when the query fingerprint has many bits set to 1, i.e., when the query is large and very selective. Thus, in the case of large queries (40 edges), KM-CTI achieves response time reductions in the range of 75% with respect to the original CT-Index.

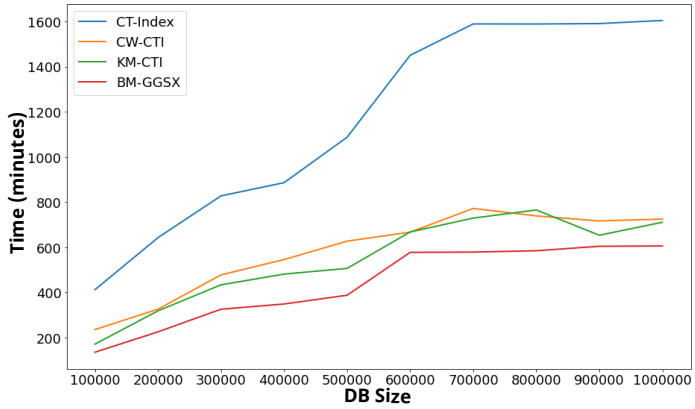
The above filtering time results of the various methods may not be directly extrapolated to total query time, due to three main reasons: i) first, the database sizes are now much larger than those of AIDS, therefore, queries retrieve many results, even if they are large, and therefore, the response time of the verification stage is the dominant one in general. ii) the evaluation of the VF2 subgraph isomorphism algorithm used by the original CT-Index is done in parallel in the new variants (CW-CTI and KM-CTI), reducing much the verification response time that is the dominant one in the total response time, as it was stated above. iii)



a) Query Size 8



b) Query Size 20



c) Query Size 40

Figure 4.20: Total response time for increasing database and query sizes.

BM-GGSX and CT-Index based methods use different types of graph features and relevant representation in their data structures, therefore, different false positive ratios are expected for these two techniques in the results of the filtering stage, which has a big impact on the dominant verification response time. Thus, as it is shown in Figure 4.20, the best performance of BM-GGSX in filtering time is not translated to a better performance in total response time. Besides, the use of a parallel implementation for the verification stage in CW-CTI and KM-CTI provides a great advantage over the original CT-Index method.

Figure 4.20(a) shows that CW-CTI is the best solution for small queries since it combines a very good solution for the filtering stage with the parallel evaluation of the verification stage. Notice that small queries are not very selective and therefore they retrieve many graphs, therefore, the subgraph isomorphism algorithm has to be applied many times. Doing this application in parallel is a key issue to achieve good total response times.

For medium (Figure 4.20(b)) and large (Figure 4.20(c)) queries, BM-GGSX improves the verification stage response times, as its ratio of false positives improves a lot over that of small queries. Besides, KM-CTI improves slightly the performance with respect to CW-CTI, due to the relative reduction of the filtering time. Overall, however, there are small differences between CW-CTI and KM-CTI, since the effectiveness of their filtering stage is identical and they use exactly the same verification stage, whose response time almost determines the total response time.

Taking into account all the evaluated characteristics, it may be concluded that a combination of the CW-CTI method for small queries and the KM-CTI method for medium and large ones is a good choice for the implementation of FTV techniques for subgraph searching over very large datasets. Notice that both structures may be perfectly combined just by storing the database fingerprints column-wise, as in CW-CTI, and adding the KM-CTI fingerprint tree. Leaf nodes of the tree must reference appropriate positions in the CW-CTI bitmaps.

Chapter 5

Generic Framework for Subgraph Searching in Distributed Architectures

This chapter describes the design of a generic framework for the implementation of FTV subgraph searching algorithms in distributed computing platforms. As noted in Chapter 4, index size and building time increases linearly and exponentially, respectively, with the size of the database. When dealing with very large databases (tens of million), working in a centralised architecture might be unfeasible. The designed generic framework allows to deal with very large databases by implementing both filtering and verification algorithms on top of the open-source cluster computing framework Apache Spark [52]. The key components of the framework are those related to the partitioning of both the database and the indexing structures, to enable the parallel filtering. The re-partitioning of the filtered candidates alleviates data skew during the parallel evaluation of the verification stage. The graphs are recorded in the distributed NoSQL document-based database MongoDB. The methods tested on the previous chapter where adapted to work with this framework. The experiments show a great performance gain in both index building and query response times, especially for small size queries, where the framework leverages the use of powerful clusters to reduce significantly the verification times, and for large databases, which are not approachable by a centralised architecture.

5.1 Index Building

The procedure that creates distributed indexes in the platform is generic, and therefore independent of the particular indexing technique chosen. Broadly speaking, the database is first partitioned by the range of the graph identifier. It is assumed that graph identifiers are incrementally generated by the system and that the database is not going to be updated after the partitioning. Thus, partitions are perfectly distributed and data skew is not possible in this scenario. An index is generated in parallel for each partition and stored in a separate structure.

The pseudocode of such a generic index building procedure that allows every FTV method to work in a distributed architecture is given in Algorithm 4. The input data is composed of three elements: i) an index name *iname* that enables its identification, ii) an input graph database D, and iii) a number of partitions p. The number of partitions p should be specified based on the number of data processing nodes available in the cluster. The number of graphs per partition (partition size *pSize*) is automatically obtained by dividing the database size by the number of partitions. This calculation is done in line 2. In line 3, function *createIndex* creates the base data structures for the index, and it also stores both the number of partitions and the partition size. An initial data structure is allocated for each partition of the index. Each iteration of the loop of lines $4-8$ is executed in parallel by a different executor of the distributed processing framework. A range query in the graph database D is performed in line 5 to obtain all the graphs of the current partition i. Graphs should be partitioned and stored in a distributed data management technology, using the same auto-incremental identifier used here to perform the range query, in order to achieve an efficient implementation without data skew. Line 6 creates the index for the current partition i using the graphs retrieved by the above range query. This function must be defined according to the chosen filtering indexing

Algorithm 4 Distributed Index Building

```
1: procedure BUILDINDEX(iname, D, p)
2:     pSize ← ceil(size(D)/p)
3:     idx ← createIndex(iname, p, pSize)
4:     for i ← 0, p − 1 do                          ▷ Each iteration executed in parallel
5:         G ← getGraphsByRange(D, pSize ∗ i, pSize ∗ (i + 1) − 1)
6:         pIdx ← createIndexForPartition(G, pSize, i)
7:         storeIndexPartition(idx, pIdx)
8:     end for
9: end procedure
```

structure. Finally, the generated index partition is stored in the index structure in line 7.

In the implementation of the framework presented in this work, the graphs are incrementally numbered, partitioned and recorded in the document-based NoSQL database MongoDB. Implementation of both the distributed index building and the query processing is done with Apache Spark. Thus, Spark structures and relevant operations are used as follows to implement Algorithm 4. First, line 3 of the algorithm is implemented by creating a Spark *Dataset* structure that contains one row for each partition, which records the partition number i. The dataset is partitioned into p partitions using the partition number, therefore, each partition contains just one row. The loop of lines $4-8$ is implemented by applying Spark *Map* operation to the above *Dataset*. The implementation of line 5 is achieved with a range query over the graph number on MongoDB. Line 6 creates an index for each partition $pIdx$ which is initially encoded in memory using the native index technique structure. However, to be able to embed the index in a row of a Spark *Dataset*, to enable its distributed management, it has to be encoded in a binary format. To achieve this, each underlying indexing technique must be extended with a serialization strategy for its index, that encodes in binary format the whole structure. Line 7 generates a new row in the output *Dataset* of the *Map* operation, which contains both the partition number and the binary serialization of the generated index structure.

Broadly speaking, the availability of a binary serialization of the underlying indexing structure enables its recording into the distributed *Dataset* structures of Apache Spark. Thus, both index building, and index querying (as it will be shown below), may be performed in parallel using *Dataset* level operations of Spark (like the operation *Map*). The following subsections provided details related to the encoding of each of the used indexing structure in binary format.

5.1.1 Index Building in BM-GGSX

To adapt the BM-GGSX method to the generic framework described above, its trie index structure must be encoded and serialized into a byte array. To achieve this, the trie has to be processed in some order and each node of the trie has to be encoded in a byte string. The concatenation of the byte string of each node in the chosen traversal order produces the required trie serialization. The order chosen matches the one used during the query processing, i.e., breath-first order.

The serialization of a trie node into a byte array is illustrated in Figure 5.1. The first two

Figure 5.1: Serialization into a byte array of a trie node in BM-GGSX.

bytes record references to the vertex (V) and edge (E) label fields. The number of children of the present node is recorded in the third byte C. Next, for each child node, four bytes ($[C_iP_1 - C_iP_4]$ for child i) are used to record a pointer to its location in the trie byte array. These pointers enable the implementation of searching processes directly over the binary encoding of the trie. Leaf nodes have no children, therefore, C is set to 0 in these nodes and the children pointers are not recorded. Byte R records the number of rows in the bit array of the node, i.e., the maximum number of repetitions. Each row bitmap is compressed using the Enhanced Word-Aligned Hybrid bitmap compression method [33], therefore each row may have a different size in its serialized representation. Thus, for each row of the bit array, a pointer of four bytes ($[R_iP_1 - R_iP_4]$ for row i) records the location of the serialization of the row in the byte array. Finally, each row bitmap is recorded in ($R_iB_1R_iB_2 \ldots R_iB_{bi}$ for row i).

Notice that nodes are recorded following a specific ordering in the binary serial representation of the trie. However, the recording of references from parent to children nodes in the representation enables the implementation of the searching processes directly over the binary representation, improving the performance of the approach with respect to a straightforward approach based on the complete decoding of the whole structure.

5.1.2 Index Building in CW-CTI

The column-wise representation of the graph fingerprints used by CW-CTI must be serialized into a byte array. The main problem to take into account to devise such a binary format is derived from the variable size nature of the compressed representations of each fingerprint column.

To cope with the variable size of each column, the binary encoding has been divided into two main parts. In the first part of the structure, for each of the fingerprint columns, a pointer to a specific location of the byte array is recorded, where the relevant binary encoding of the

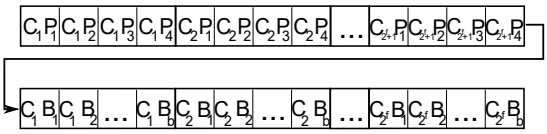


Figure 5.2: Serialization of CW-CTI.

column is stored. Those pointers are represented with four bytes and are illustrated in the first row of Figure 5.2. In particular, bytes denoted by $C_kP1 - C_kP_4$ represent the four bytes of the pointer to column number k. The byte array of the compressed representation of each column is stored at the byte position referenced by the relevant pointer. In the figure, the second row illustrates the serialization of these compressed bitmaps. In particular, $C_kB_1 - C_kB_b$ denotes the b bytes of the compressed representation of column number k.

It should be noted that a more simple representation of the compressed bitmaps could have been chosen, for example, one based on the simple recording of the size preceding the representation of each bitmap, which is very common to represent variable size vectors of data elements. However, such a representation would not provide direct access to a specific column and would demand the processing of all the preceding columns to access a specific one. Contrary to the above, the use of pointers in the serialized structure enables the direct access to the compressed bitmap of each column, which in turn enables a more efficient implementation of the filtering algorithm. It is reminded that such an algorithm performs the binary *AND* operation among the columns that contain a bit 1 in the query fingerprint, thus, only those columns need to be accessed to evaluate a specific query.

5.1.3 Index Building in KM-CTI

As in the case of BM-GGSX, the serialization of the KM-CTI structure requires the serialization of a tree. To achieve this, again, a traversal order has to be chosen and a serialization strategy has to be defined for each of the tree node types. In this case, a depth-first order has been chosen to serialize the tree, as the structure is traversed using this order during query processing. The serialization of each of the two node types is described below.

Each internal node contains a fingerprint and a couple of pointers, referencing the left and right children. Fingerprints are serialized using a fixed number of b bytes. Pointers to nodes in the structure are represented with four bytes. In addition to the above data, the encoding of each node must include a byte that records the type of node, i.e., either internal or leaf. The

Figure 5.3: Serialization in KM-CTI: (a) Internal node and (b) Leaf node.

byte array used to encode an internal node is illustrated in Figure 5.3 (a). The first b bytes, $(V_1 - V_b)$, are used to record the node fingerprint (the disjunction of the fingerprints in its subtree). The next byte (L) is used to record the node type, *internal* node in this case. Finally, the byte array records two sequences of four bytes used as pointers to the left ($C_LP_1 - C_LP_4$) and right ($C_RP_1 - C_RP_4$) children of the node.

Leaf nodes of KM-CTI include the following data: A fingerprint that represents the disjunction of all the fingerprints of all the graphs accessed through the leaf node. One pair (g_i, f_i), for each graph accessed through the node, where g_i is the integer graph identifier and f_i is the fingerprint that represents the features extracted from the graph. The binary encoding contains the b bytes of the node fingerprint, an extra byte that represents the type of node (leaf node in this case), a byte that records the number of graphs accessed through the node and, for each such graph, 4 bytes to represent its identifier and b bytes to represent its fingerprint. The serialization of leaf nodes is illustrated in Figure 5.3 (b). Bytes $(V_1 - V_b)$ record the node fingerprint, byte L represents the node type, byte N records the number of graphs accessed through the node, and, for each pair (g_i, f_i), bytes $[C_iI_1 - C_iI_4]$ represent the graph identifier g_i and bytes $C_iV_1 - C_iV_b$ represent the graph fingerprint f_i.

5.2 Query Processing

Once the indices have been created, partitioned and distributed, they can be used to implement a parallel version of the filtering stage. Broadly speaking, each partition of the index works as a centralised index for the graphs that lie inside that partition, therefore, the searching can be done in parallel in all the partitions. The collection of graph candidates retrieved by each partition must be collected and repartitioned again to enable an efficient parallel implementation of the verification stage, avoiding partition skew. More details of the FTV searching algorithm are given below.

Algorithm 5 provides a pseudocode that shows the design of the FTV distributed subgraph searching function of the proposed framework. The input data of this function is given by a

Algorithm 5 Distributed Subgraph Search

```
1:  function DISTRIBUTEDSUBGRAPHSEARCH(q, D, idx)
2:      p ← getNumPartitions(idx)
3:      candidates ← initializeCollection()
4:      for i ← 0, p − 1 do                          ▷ Filter: Each iteration executed in parallel
5:          pIdx ← getIndexPartition(idx, i)
6:          pCandidates ← filter(q, pIdx)
7:          append(pCandidates, candidates)
8:      end for
9:      repartition(candidates, p)
10:     result ← initializeCollection()
11:     for i ← 0, p − 1 do                          ▷ Verify: Each iteration executed in parallel
12:         pCandidates ← getPartition(candiates, i)
13:         pCandidateRanges ← fold(pCandidates)
14:         for all range ∈ pCandidateRanges do
15:             G ← getGraphsByRange(D, range.start, range.end)
16:             for all g ∈ G do
17:                 if subgraphIsomorphic(q, g) then append(g, result)
18:                 end if
19:             end for
20:         end for
21:     end for
22:     return result
23: end function
```

query graph q, an input graph database D, stored in a distributed data management technology, and a distributed index idx, constructed using the procedure described in the previous section. The algorithm starts by obtaining the number of partitions from the index (line 2) and by initializing a collection of graph candidates for the verification stage (line 3). The filtering stage of the searching process is implemented with the loop of lines $4-8$. Each iteration of this loop performs the searching process in a different partition of the index. The parallel implementation of the filtering stage is achieved by the implementation in parallel of each iteration of this loop.

The filtering stage has three steps: i) line 5 obtains the relevant partition from the index. ii) Line 6 performs the searching of the query graph q in the index partition idx obtained in the previous step. iii) Line 7 appends the graph candidates retrieved by the searching process of the previous step to the result collection of candidates initialized earlier. At this moment, for each partition of the index, there is a partition of the graph candidates obtained at that

partition. Clearly, the distribution of the candidates among the different partitions might not be well balanced, having partitions with many candidates and partitions with few or even without candidates.

Applying the verification stage in parallel to these unbalanced partitions would lead to an inefficient implementation, due to partition skew. To avoid this, line 9 repartitions the result collection of candidates, before it is used as input to the verification stage, which is implemented in the loop of lines $11-21$. The iterations of this loop are executed in parallel. Each loop iteration applies the subgraph isomorphism algorithm to the graphs of one of the candidate partitions. First, the candidate graph identifiers of the current partition are obtained from the relevant collection in line 12. Graph identifiers are ordered and folded in line 13 into collections of ranges to improve the efficiency of the graph retrieval from the database. Thus, for example, for the collection of graph identifiers $\{1,2,3,4,8,9,10,15\}$, the following ranges are generated $\{[1-4],[8-10],[15,15]\}$ and, thus, only three range queries have to be submitted to the distributed database to obtain all the required graphs. Notice that the repartitioning performed in line 9 must use a range approach and not a hash approach. Otherwise, line 13 would general large numbers of ranges and the database access would be too inefficient. For each obtained graph identifier range, line 15 submits a range query to the database to obtain a collection of graphs G. Finally, line 17 evaluates the subgraph isomorphism algorithm VF2 for each graph retrieved from the database, and appends the graph to the result in case of a positive result of VF2.

The current implementation of the framework stores each graph in a JSON document recorded in a MongoDB collection. Apache Spark is used to execute the searching process in parallel in the cluster. Thus, as already described in the previous section, the distributed index structure is stored in a Spark *Dataset*, which includes one row for each index partition. It is reminded that the index is actually recorded in a byte array field of this *Dataset*. The filtering stage (loop of lines $4-8$) is, therefore, implemented with a distributed *Map* operation over the distributed index *Dataset* structure. Graph q is used in each *Map* evaluation to search the serialized encoding of the index stored in the *Dataset*. The output of the above *Map* operation is a *Dataset* of candidate graph identifiers. The *Dataset* with the result candidate graph identifiers is re-partitioned (see line 9) before it is used as input in the verification stage. The verification stage is also implemented with a *Map* operation over the *Dataset* of candidate graph identifiers.

The index building and query processing algorithms described above are generic, in the

sense that any FTV method may be used if a byte array serialization of its data structure is provided. To achieve this, the database partitioning approach followed is independent of the index structure used, as it does not use information related to how graphs are indexed in the structure. As a consequence, all the index partitions have to be searched for each query, even if no candidates for the result are not recorded in a specific partition. The design of more advanced partitioning approaches, adapted to the specific indexing technique implemented, which might discard whole partitions during the searching process, are left as part of future work.

5.3 Performance Evaluation

This section presents and discusses the results of the performance evaluation of all the subgraph searching methods developed in this Thesis, which have been implemented on the distributed framework described in previous sections. As in the case of the relevant implementations on centralised architectures, the evaluation focuses both on index building and query processing. The main difference however is the incorporation of tests oriented to evaluate the horizontal scalability of the system. The sizes of the input datasets reach now amounts of tens of millions of graphs. Summarizing, all the following parameters are taken into account during the evaluation: index size, index building time, filtering response time, total response time, cluster size (number of executors), database size and query size.

5.3.1 Evaluation Framework

The hardware infrastructure used in the experiments consists of 16 Dell EMC PowerEdge R730 Servers, each of them with 2 Intel(R) Xeon(R) CPU E5-2630 v4 (2.20 GHz, 10 cores) processors and 384 GB of RAM. All the implementations were done in Java. Java virtual machines with 32 GB of RAM were used by all the machines.

The databases of different sizes used during the evaluation were generated from the PubChem dataset. As already explained in Section 4.4.1, the PubChem database contains the structure (graph) of around 96 million molecules (by the time these experiments were performed). The dataset graphs have an average of 41.40 vertices and 42.16 edges. To evaluate the performance of the methods using increasing database sizes, databases of 100 k, 250 k, 500 k, 1000 k, 2 M, 4 M, 8 M, 16 M, 32 M, 64 M and 96 M graphs were generated from PubChem.

Three collections of 100 random small (8 edges), medium (20 edges) and large (40 edges) queries were generated from PubChem, following the same random walk approach explained in Section 4.4.1.

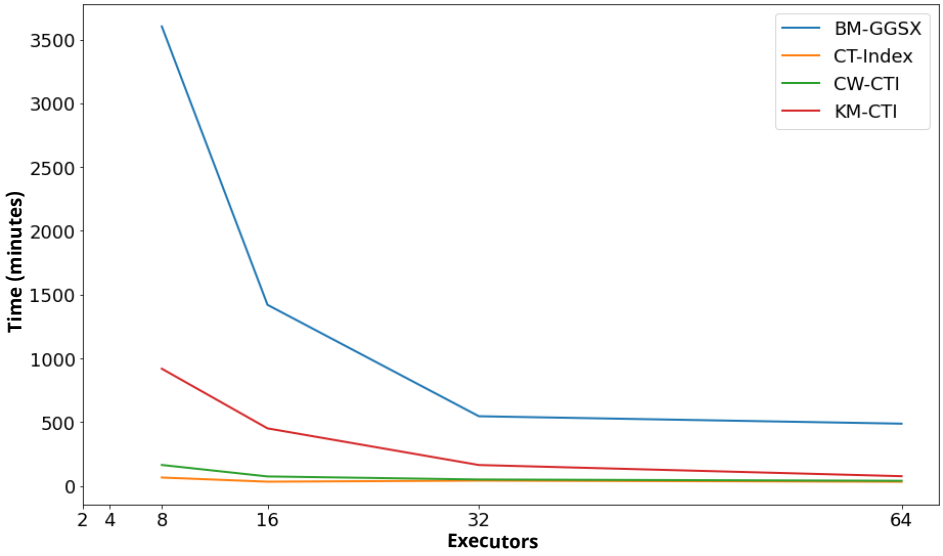
5.3.2 Evaluation of the Distributed Implementation

The databases of the different sizes described in the previous subsection were indexed using the distributed framework using the various methods compared in the evaluation, i.e. BM-GGSX, CT-Index, CW-CTI and KM-CTI. Various indices of each technique were generated for each database size, using $2^1, 2^2, \ldots, 2^6$ partitions. To avoid very large index building times, a restriction was imposed on the number of graphs per partition, avoiding combinations with more than 500k graphs per partition. Thus, for example, for a database of 8 M graphs, the minimum number of partitions used was 16. For the same reason, for databases of more than 32 M graphs, only one index for each method was generated using 64 partitions.

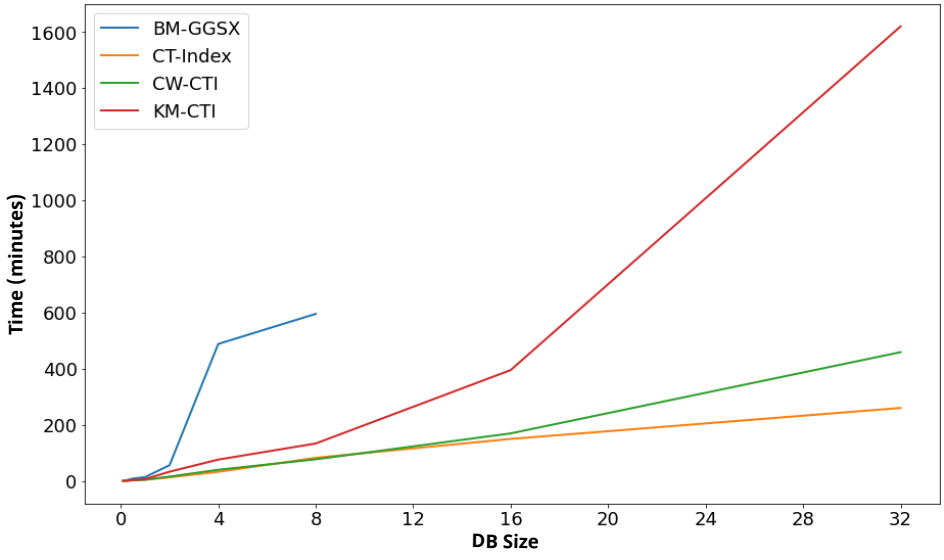
The horizontal scalability achieved by the systems with each indexing approach is shown in Figure 5.4(a). The indices whose building times are shown in the figure were generated for a database of 4M graphs. The same index was constructed using an increasing number of partitions and executors, ranging from 8 to 64 executors (partitions). As in the case of centralised implementation, the BM-GGSX structures require much more time to be constructed, and even with the maximum number of executors, they cannot be generated in less than around 500 minutes. Building KM-CTI is slower than building the other two CT-Index variants.

The scalability of the system with respect to the database size, fixing the number of executors in 64, is shown in Figure 5.4(b). Again, BM-GGSX is clearly the worse approach in terms of index building time. Notice that for databases of more than 8M graphs, even with 64 executors, the index could not be constructed, given that the RAM available for the virtual machine of each node was not sufficient. CT-Index is the approach that behaves better, slightly better than CW-CTI. The building time of KM-CTI becomes too high for very large databases, needing around 26 hours to construct the index for the database of 32M graphs. This is due to the low performance of the recursive evaluation of the K-Means clustering, undertaken during the generation of the KM-CTI fingerprint tree. It is finally noticed that only the distributed version of the original CT-Index could be generated for the largest database size (64M graphs).

Looking at the above index building times, it becomes clear that the implementation of a distributed system enables the generation of indices for very large databases, in the order

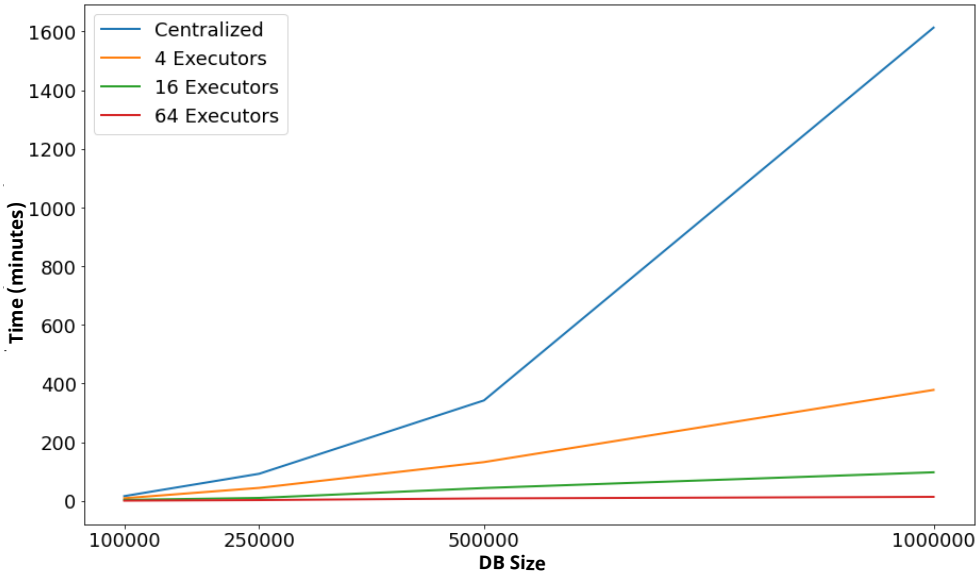


a) 4M DB varying number of executors

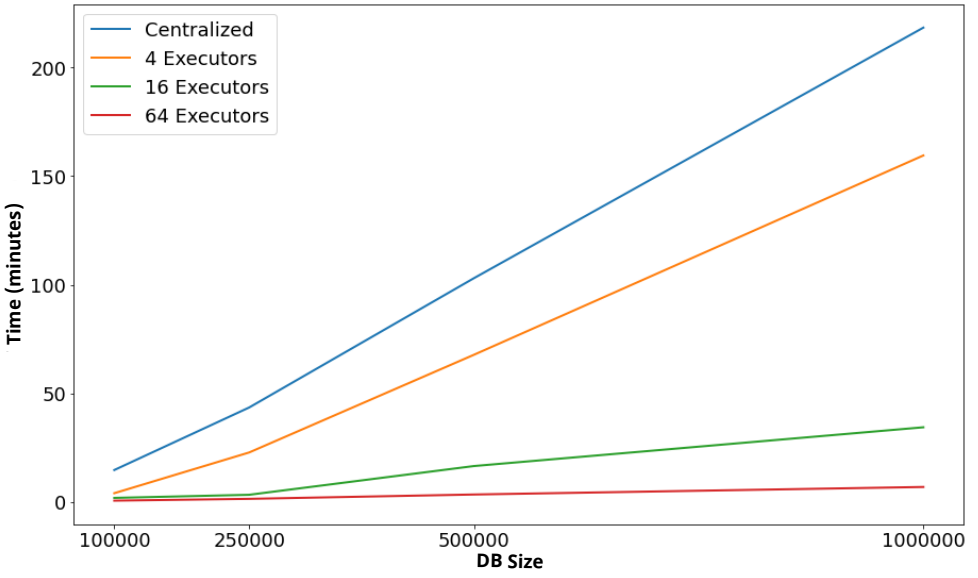


b) 64 executors varying DB size

Figure 5.4: Index Building Time in distributed system varying (a) number of executors and (b) database size.



a) Index Building Time For BM-GGSX



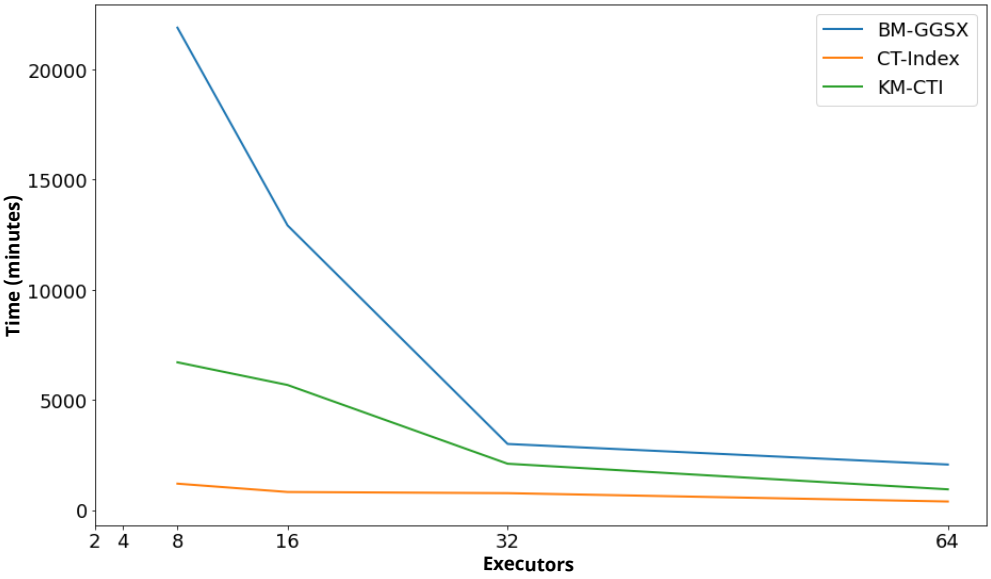
b) Index Building Time for KM-CTI

Figure 5.5: Comparison of index building time between centralised and distributed implementations for (a) BM-GGSX and (b) KM-CTI.

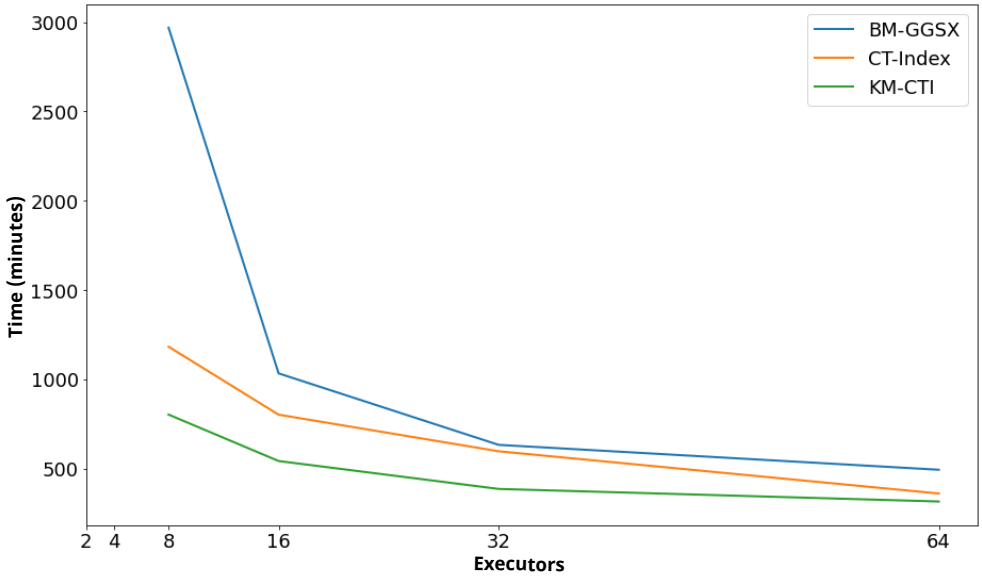
of tens of millions of graphs. A comparison of centralised and distributed implementations, using clusters of different sizes, is shown in Figure 5.5, for the two index approaches that require more resources, i.e., the BM-GGSX and the KM-CTI. In particular, Figure 5.5(a) shows the clear advantage of the distributed systems with respect to the centralised one for the building of BM-GGSX indices. In particular, the centralised implementation of BM-GGSX required more than a whole day to generate the index for 1M graphs, whereas its distributed implementation with 64 executors can do it in only 14 minutes. The comparison of centralised and distributed implementations of the KM-CTI is shown in Figure 5.5(b). The index building time, in this case, was reduced from 3.5 hours to 7 minutes, which is a notable improvement.

The horizontal scalability of the distributed implementation of the filtering stage is illustrated in Figure 5.6. The database used for the experiments had a size of 4 M graphs. The number of partitions and executors was increasing from 8 to 64. The results do not show filtering times achieved by CW-CTI, which are in general around an order of magnitude slower than the other three methods. Due to this, the results of CW-CTI have been eliminated from all the figures and relevant discussions. The distributed implementation of CT-Index achieves the best performance for small queries, as it is shown in Figure 5.6(a). KM-CTI behaves better than BM-GGSX for small queries in distributed implementations. It can be seen that, for centralised implementations, BM-GGSX had the best performance for small queries, whereas CT-Index and KM-CTI had similar results. It might be concluded therefore that CT-Index is the technique that improves more when moving from centralised to distributed implementations. On the other hand, for large queries, KM-CTI achieves better results than those of CT-Index, as it is shown in Figure 5.6(b). The results of KM-CTI are especially better than those of CT-Index when the query is large, i.e., its selectivity is high, and the database partition is also large, i.e., the number of partitions and the number of executors is small. This is coherent with the fact that KM-CTI uses a tree to search fingerprints, which is really an advantage when the query has high selectivity and the database is large enough. In general, it is observed that CT-Index based techniques take more advantage of distributed implementations than GGSX based implementations. It is also noticed that, in general, the response time decreases drastically when the number of executors is increased from 8 to 32. This is reasonable given that the hardware has 16 nodes with 2 processors each, i.e., 32 processors.

Figure 5.7 shows the scalability of the distributed implementation when the database size increases. In this case, all the experiments were performed using the maximum number of partitions and executors, i.e., 64. Again CT-Index shows better behaviour for small queries

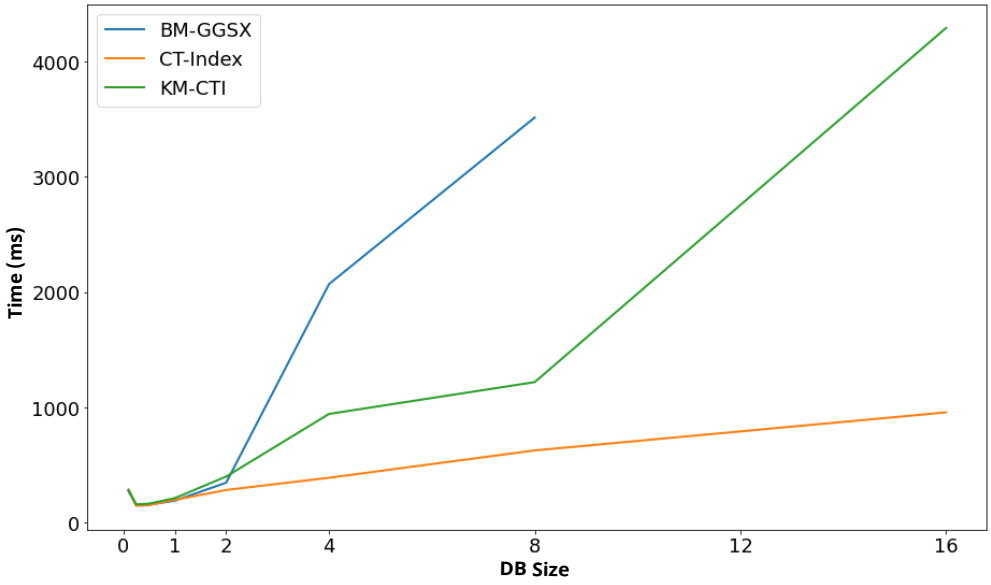


a) Filtering Time on 4M DB for small queries

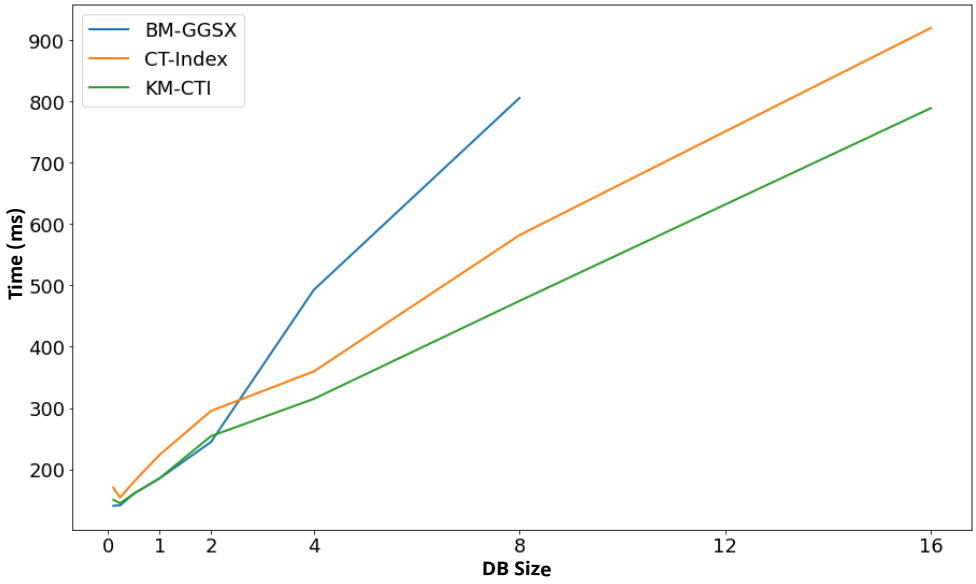


b) Filtering Time on 4M DB for large queries

Figure 5.6: Average filtering time for distributed implementations (4M graph database size).



a) Filtering Time with 64E for small queries



b) Filtering Time with 64E for large queries

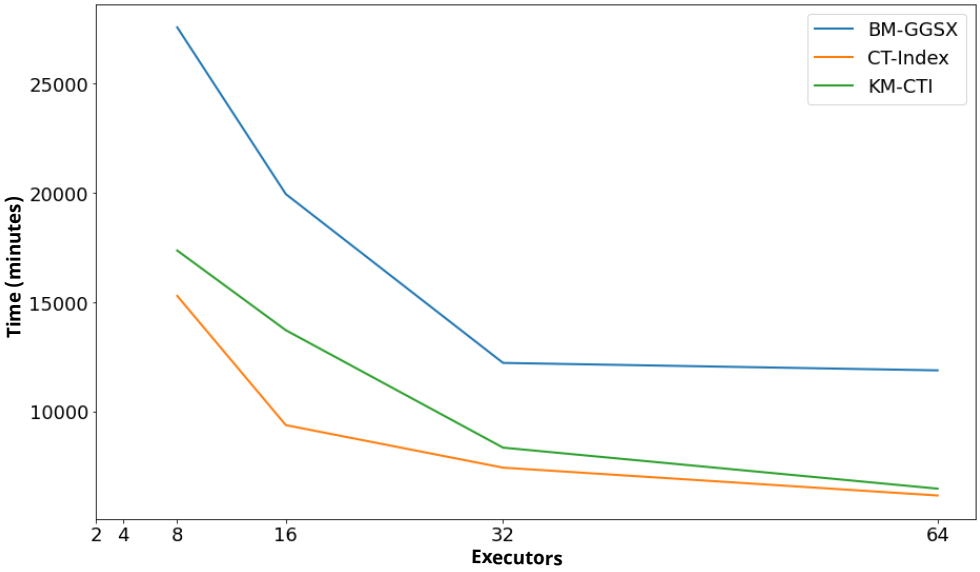
Figure 5.7: Average filtering time for distributed implementation (64 executors).

(Figure 5.7(a)), whereas the best performance for large queries is achieved by KM-CTI (Figure 5.7(b)). It is noticed that the distributed implementation of the system enables the filtering of databases of tens of millions of graphs with response times below one second, both for small and large queries. The best approach would the direct use of the fingerprints (i.e., the use of CT-Index) for small queries and the use of the fingerprint tree (i.e. KM-CTI) for large queries. This reminds also a common optimization decision in conventional databases, which prefer the scan of the whole table for queries of low selectivity and use an index for queries of high selectivity.

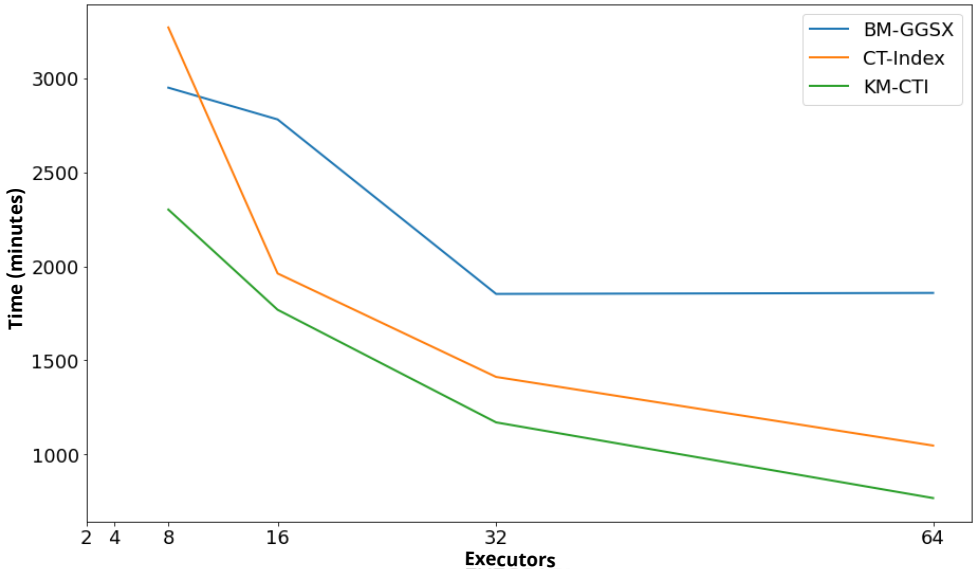
The above conclusions on the response time of the filtering stage may be in general extrapolated to the response time of the whole query, i.e., including both filtering and verification stages. It is noticed that now, contrary to what happened with the centralised implementations, the implementation of the verification stage is identical for all the indexing methods. Figure 5.8 illustrates the horizontal scalability of the distributed implementation of the whole FTV subgraph search. The experiments were executed on the database of 4 M graphs. In general, CT-Index behaves better for small queries and KM-CTI has better performance for large queries. Given that the verification stage has a great influence on the total response time for small queries, the differences between CT-Index and KM-CTI are not really important when the number of executors is large. For large queries, the filtering time gains importance, due to the high selectivity of the query and the fewer candidates to verify. Therefore, KM-CTI keeps its advantage also for total response time in large queries.

The impact of the database size on the total response time of the system is illustrated in Figure 5.9, where all the experiments were executed using the maximum number of executors and partitions (64). Now, it is noticed that for small queries (Figure 5.9(a)), where the verification time is the most important component, both CT-Index and KM-CTI offer similar performance and scalability, being CT-Index slightly better when the database size is very large (16M graphs). On the other hand, it is clear that the use of the fingerprint tree is an advantage for large queries (Figure 5.9(b)), where the filtering time is more important than the verification time.

Once the various indexing approaches have been compared, taking into account both index building time and filtering and total query response time, a couple of questions need still to be resolved. It has been shown that the distributed implementation brings a clear advantage over the centralised one in terms of index building time. However, the filtering and total response time of centralised and distributed implementations have not been compared. Additionally,

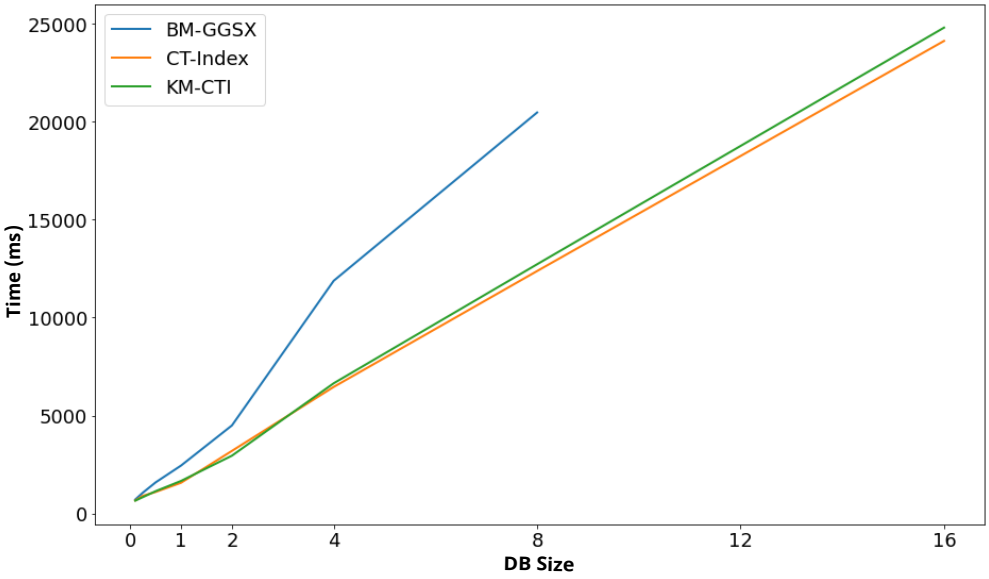


a) Total Time on 4M DB for small queries

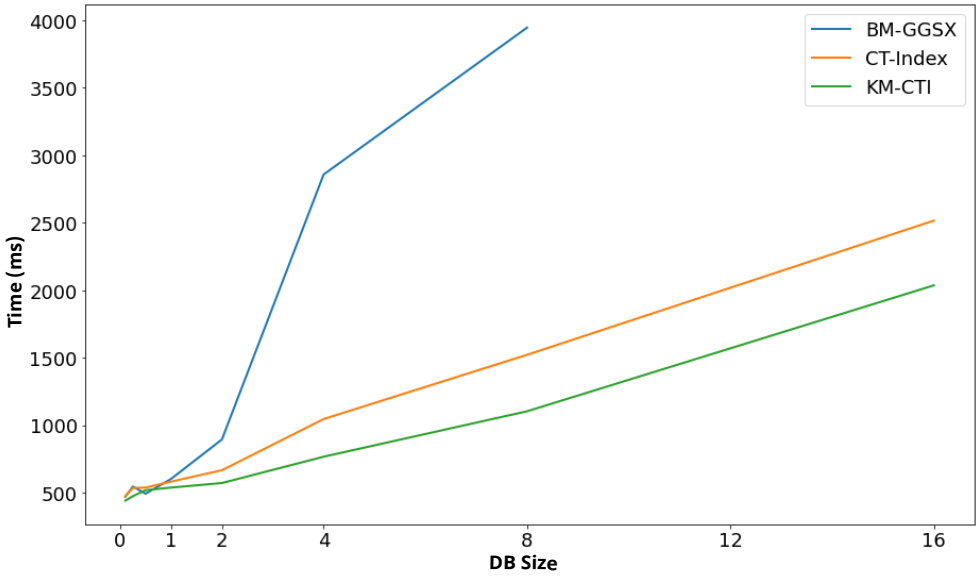


b) Total Time on 4M DB for large queries

Figure 5.8: Average total response time for distributed implementations (4M graph database size).



a) Total Time with 64E for small queries



b) Total Time with 64E for large queries

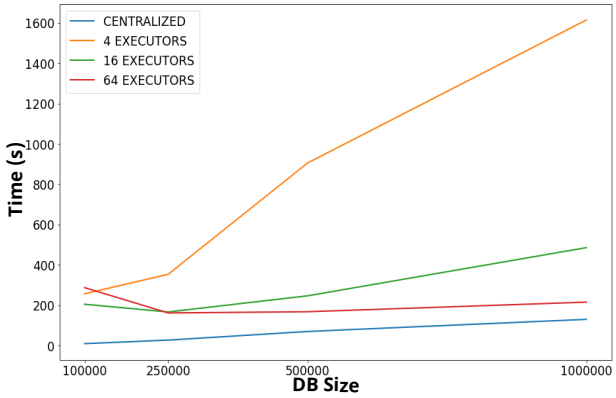
Figure 5.9: Average total response time for distributed implementation (64 executors).

it is not clear whether the parallel application of the VF2 subgraph isomorphism algorithm might bring results competitive with those of the implemented distributed FTV approaches. The discussion below will show some light on these two issues.

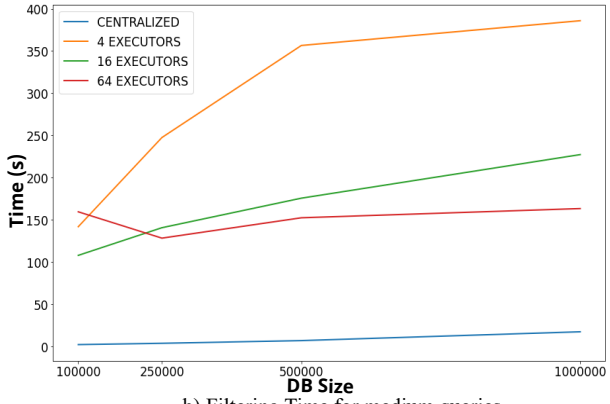
The first question to respond is whether the distributed filtering reaches better performance than the centralised one. Figure 5.10 compares the filtering time obtained by the centralised version of KM-CTI with the filtering time of the relevant distributed implementation, using 4, 16 and 64 executors. For small queries (Figure 5.10(a)), the distributed implementation behaves better when the number of executors is increased. However, the centralised implementation has better performance than the distributed one. It is observed, however, that the difference in performance between the distributed and centralised implementations is reduced when the database size increases. Thus, it is expected that with really large database sizes, the distributed implementation might get better performance than the distributed one. For medium and large query sizes (Figures 5.10(b-c)), the centralised implementation is clearly better than the distributed one.

The comparison of centralised and distributed implementations of KM-CTI in terms of total response time is shown in Figure 5.11. Now, the centralised implementation gets better results only when the query is large and the database is small (below 1M graphs). This is due to the great performance gain achieved by the distributed implementation with respect to the centralised one in the verification stage, when the set of candidates to verify is large, i.e., when the query is small or the database is large enough. As a conclusion, the centralised approach gets better performance in the filtering stage, but the distributed implementation of the verification stage is what enables the efficient querying of very large databases. It is also noticed that, although the centralised approach is faster in filtering, very large datasets may be indexed only with a distributed approach. The fact that the index must fit in main memory is also a limitation of the centralised approach for filtering very large datasets.

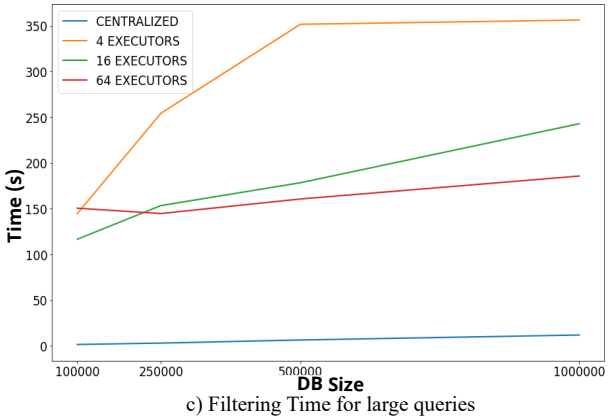
A final experiment tested the performance of the FTV approaches with respect to the distributed evaluation of the VF2 subgraphs isomorphism algorithm over the whole database. The KM-CTI approach was selected to represent the FTV approaches. The results are shown in Figure 5.12. The FTV approach gets performances orders of magnitude better than the direct distributed evaluation of VF2. For this reason the yellow and green lines representing the FTV methods in the Figure seem to be overlapped. The performance improvement of the FTV approach increases linearly with the database size. It is obvious that the direct distributed implementation of VF2 is not an option in practice for very large databases.



a) Filtering Time for small queries

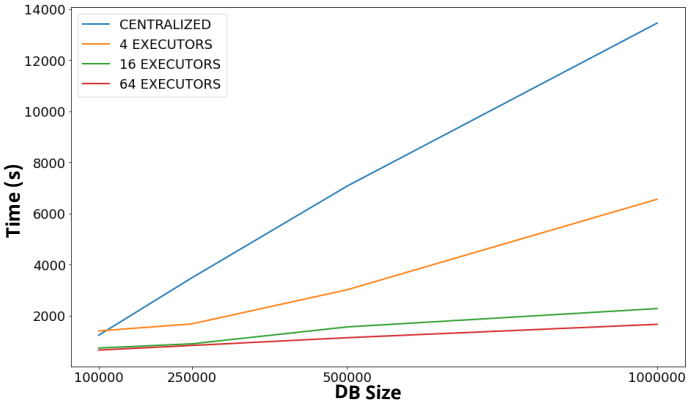


b) Filtering Time for medium queries

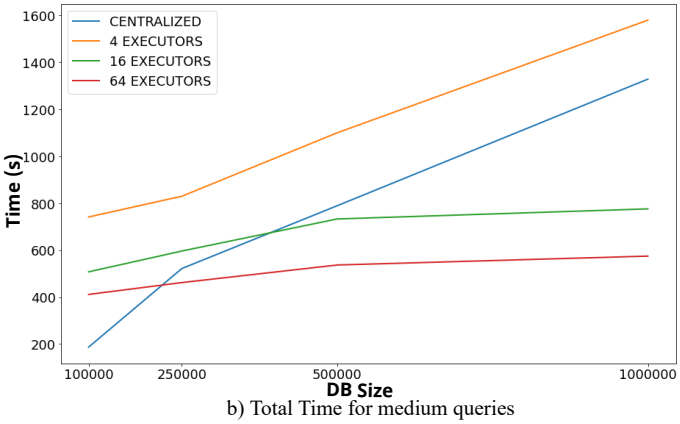


c) Filtering Time for large queries

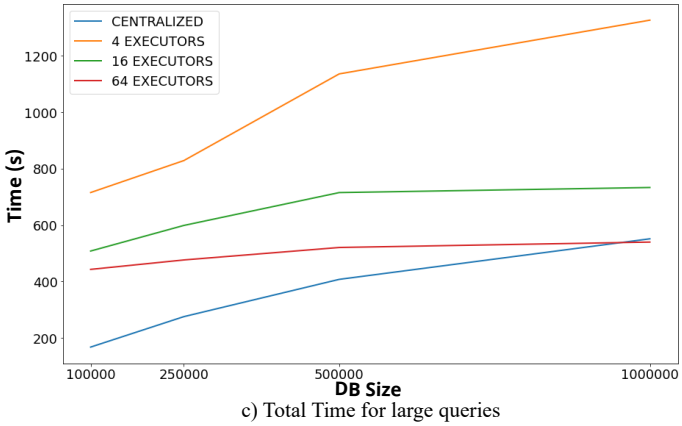
Figure 5.10: Comparison of query filtering time between centralised and distributed implementations for KM-CTI method.



a) Total Time for small queries



b) Total Time for medium queries



c) Total Time for large queries

Figure 5.11: Comparison of query total response time between centralised and distributed implementations for KM-CTI method.

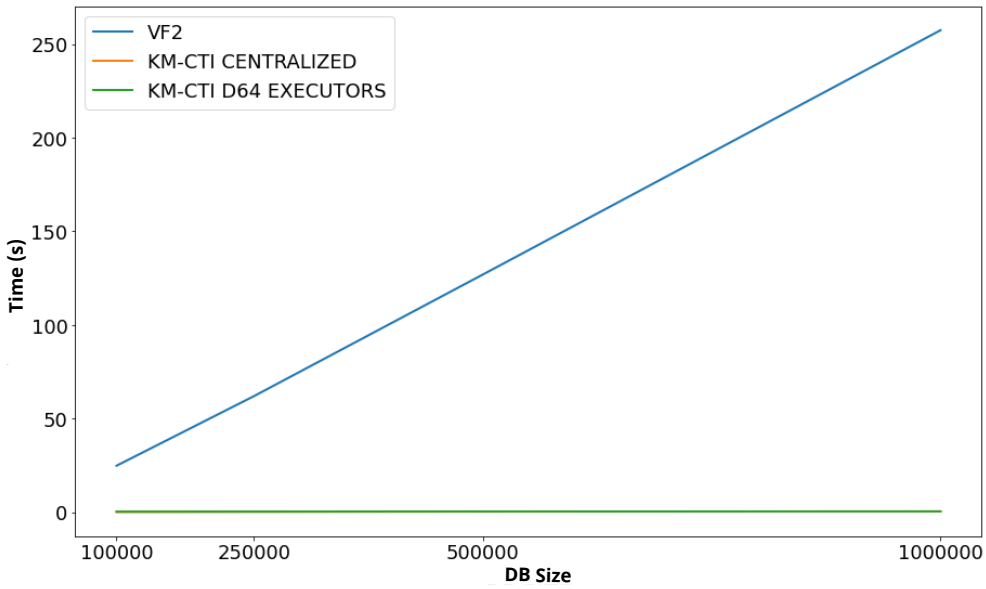


Figure 5.12: Illustration of the significance of the use of indexing with respect to the simple parallel evaluation of a subgraph isomorphism algorithm.

CHAPTER 6

APPROXIMATE PROCESSING FOR INTERACTIVE SEARCHING IN MOLECULAR DATASETS

In previous chapters, it was shown how different indexing techniques used in FTV strategies helped in achieving important performance improvements during the evaluation of subgraph searching queries. However, a very common scenario is the one where the user must perform interactive searching and browsing tasks to get a general overview of the dataset contents. When the size of the dataset is very large, such as the PubChem dataset that contains around 100M graphs, performing searching with interactive response times (subsecond in general) requires the presentation of approximate results. Based on the above, in this chapter, various approximate processing approaches, some of them already used in the filtering stage of FTV techniques, are analyzed and compared to determine if they might be good candidates for the construction of interactive searching solutions.

6.1 Interactive Searching in Molecular Datasets

The need for the elimination of replicas from known compounds is one of the problems, which arises in the chemical, biotechnological and pharmaceutical research areas, that motivate the research effort towards the design of better techniques for substructure searching. However, in many cases, the user does not have a clear and precise idea of what is looking for, and what

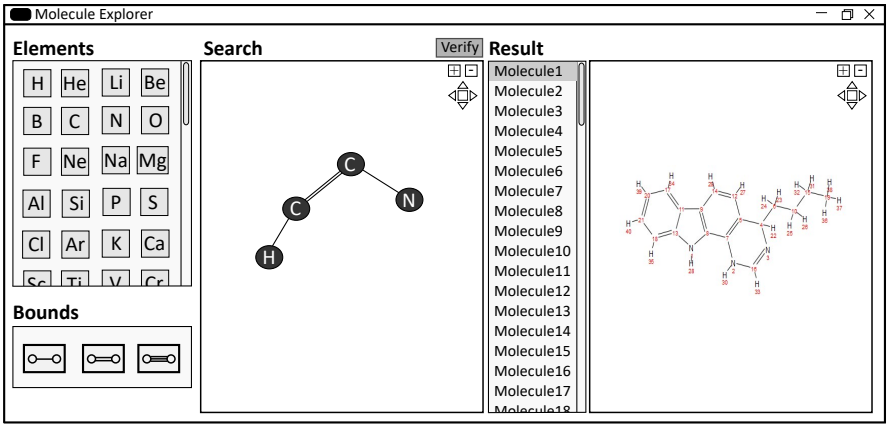


Figure 6.1: Mockup of an interactive searching application.

is really helping is the availability of interactive search interfaces. Such interfaces must have very fast response times [8, 36], but it might be acceptable to get approximate responses, i.e., response sets that might include some molecules whose structure does not really contain the query substructure. One possible such interactive searching interface is described below.

Let us consider the mockup of an interactive searching user interface shown in Figure 6.1. At the left part of the interface, the user may drag chemical elements to the search area to define atoms and, next, link them using the different types of available bounds. As the search structure is being modified by the user, the result list is automatically updated to obtain all the molecules from the database whose structure contains the search substructure. One of the molecules of the list (the first one by default) is shown in the result visualization area. The user may choose any of the molecules of the result list to graphically explore its structure. To achieve a usable interface, the list of result molecules must be updated interactively, i.e., with response times below one second, every time the user does some update in the search structure. Therefore, substructure queries should be solved with response times below one second, independently of the size of the structure drawn by the user and independently of the size of the database. It might be acceptable, however, that some of the molecules of the list

do not really contain the search substructure. Notice that with small search structures, the list is very long, so verifying all the candidates is going to take too much time. Thus, while the search structure is still small, approximate searching might be applied to the result list very fast, accepting false positives in the list. As the search structure gets larger, the list of result molecules gets shorter and, at some moment, the user might click on the *Verify* button to launch the verification stage over the result candidates and eliminate false positives from the list. Applying this verification stage over a reduced set of candidates may also be done with a short response time, keeping the interface interactive. At that moment, it is guaranteed that each molecule of the list contains the objective search substructure. In addition, before that, the user had the opportunity to interactively navigate through a list of molecules with a reasonably low number of false positives and always with very fast response times.

According to [8, 36], 100 milliseconds is the limit for the user to feel that the system is reacting instantaneously. If the system response is between 0.1 seconds and 1 second, then the user will notice some delay but her flow of thought would not be interrupted, thus no special feedback would be necessary for delays shorter than 1 second. Therefore, we shall consider interactive response times below one second.

The filter-then-verify techniques described in previous chapter for both centralised and distributed architectures already provide initial solutions for the searching problem described above. In particular, for centralised architectures, it has been shown that approximate filtering may be done with response times of around 20 milliseconds for datasets of up to 1M graphs. For such a dataset size, in centralised architectures, the total response time, including also the verification, was ranging from 150 milliseconds to 500 milliseconds, depending on the query. Notice that all those figures lay in the range of what we can call interactive response times, although only the filtering stage provides instantaneous responses.

However, if we consider very large datasets (in the order of the tens of millions of graphs), even if we deploy a distributed architecture with 64 executors, the total response times are not interactive anymore, ranging from 1.5 seconds to 25 seconds, depending on the query size. Using only the filtering stage, the response times are reduced to values between 750-1000 milliseconds, which is just on the border of what we can consider interactive response times. However, the results will include false positives.

In the following section, some approximate processing solutions that might be used during the implementation of interactive interfaces like the one above are described. Next, an experimental evaluation shows which techniques obtain the best compromise between fast response

times and a reduced number of false positives in the result, i.e., efficiency versus efficacy.

6.2 Approximate Processing Solutions

This section describes a couple of general purpose data structures that are currently used in the construction of approximate query processing solutions, namely, the Bloom Filters [5] and the Count-Min Sketch[12]. Additionally, the section describes the Sachem [30] extension for the PostgreSQL database management system, which provides a molecular substructure search implementation whose results have both false positives and false negatives when they are compared with the results obtained by the VF2 subgraph isomorphism algorithm.

6.2.1 Bloom Filters

A well-known example of data structure used in approximate processing is the Bloom Filter [5]. A Bloom Filter (BF) is a data structure based on hashing that is used to provide an approximate result to the test of whether an element is a member or not of a given set. The tolerance to the existence of approximate results (false positives in this case), enables the reduction of the required space for the structure and, consequently, also of the query response time.

Let $S = \{s_i\}$ denote a set of n elements from a given universe $U \supseteq S$. A Bloom Filter BF for the approximate representation of S consists of:

- A bitset B of size m, numbered $0, 1, \ldots, m-1$.
- A collection of d hash functions $h_k : U \to \{0, \ldots, m-1\}, k = 1, \ldots, d$.

To insert an element $s_i \in S$ in BF, each hash function h_k is applied to s_i to obtain the position $h_k(s_i)$ of a bit of B that will be set to 1. Thus, for each element to be inserted, d bits of B will be set to 1. To test if an element s_j is a member of S, again each h_k is applied to s_j to get d references $h_k(s_j)$ to bits of B. The test yields true only if all the referenced bits are set to 1. More formally, the result of the query of an element s_j is obtained as follows.

$$BF(s_j) = \bigwedge_{k=0}^{d} B[h_k(s_j)]$$

It should be noted that during the insertion of elements, the bit positions referenced by the hash functions applied to one element may intersect with the bit positions referenced

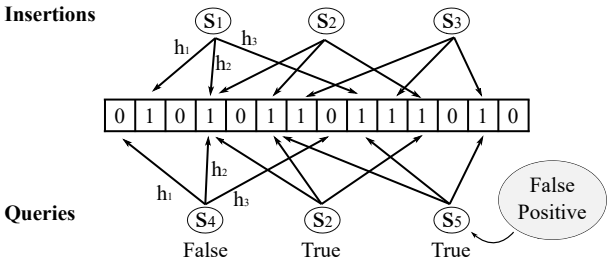


Figure 6.2: Illustration of the insertion and querying in a Bloom Filter.

by the hash functions applied to another element. Therefore, all the bits referenced by the applications of the hash functions to one element might be set to 1, even if the element has not been inserted in the structure, leading this way to a false positive. On the other hand, if an element has been inserted in the structure, it is not possible to get any 0 from some of the bits referenced by the application of the hash functions to it. Therefore, false negatives are not possible as a result of the test.

Figure 6.2 illustrates the insertion and querying of elements in a Bloom Filter. The structure has 14 bits and 3 hash functions (h_1, h_2 and h_3). Three elements have been inserted, namely, s_1, s_2 and s_3, and there was a collision between elements s_1 and s_2 at bit number 3. The testing of element s_4 yields false, since the application of both h_1 and h_3 results in references to bits with a value of 0. On the other hand, the testing of s_2 yields true, since the bits referenced are the same as those activated during its insertion. Finally, the testing of s_5 yields true even though s_5 has not been inserted in the structure. Therefore, s_5 is a false positive.

The theoretical guarantees in terms of false positive probability based on the number of hash functions k, the number of bits m and the number of elements to insert $|S| = n$ have already been studied. In particular, the false positive probability p might be approximated by the following formula:

$$p \approx \left(1 - e^{-dn/m}\right)^d$$

.

Besides, for given values of m and n, the optimal number of hash function k that minimizes the false positive probability is given by the formula.

$$d = \frac{m}{n} \ln 2$$

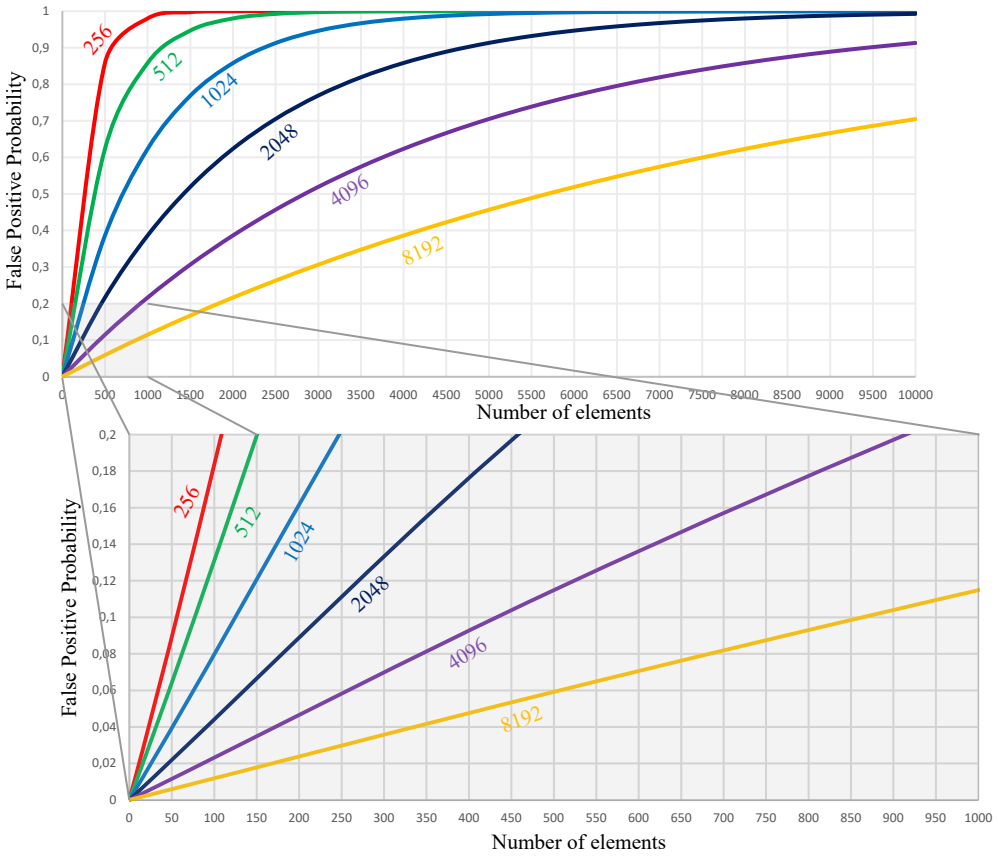


Figure 6.3: False positive probability achieved for increasing number of elements inserted in Bloom Filters of just one hash function with different sizes (256-8192).

Therefore, the number of hash functions d and the number of bits for each inserted element required to guarantee a specific false positive probability may be approximated as follows.

$$m/n \approx 1.44 \log_2(1/p). \quad d = \log_2(1/p).$$

It is noticed that the fingerprints used by any of the indexing structures based on CT-Index are actually Bloom Filters with a specific number of bits and just one hash function. Thus, the probability of having false positives during the querying of the features of each graph in the fingerprints may be approximated using the above theoretical results. To illustrate this,

Figure 6.3 shows the probability of false positives reached by a Bloom Filter of just one hash function, for different values of the number of elements inserted and for different bitset sizes.

Over the years, many extensions and different applications of Bloom Filters have be proposed [1, 7, 10, 12, 15, 16, 38].

6.2.2 Count-Min Sketch

The Count-Min Sketch (CMS) is a data structure designed to approximate the number of times that each element has been repeated in a data stream [12]. To achieve this, the structure uses a collection of pairwise independent hash functions and a 2D matrix of counters. The tolerance to the overestimation of the number of occurrences of elements in queries enables the use of a reduced number of counters, saving memory and reducing the query response time.

Let $S = s_i$ denote a multiset of n, not necessarily distinct, elements of a given universe $U(s_i \in U)$. A Count-Min Sketch CMS that approximates the frequency of each distinct element s_j of S consists of:

- A 2D matrix C of size $d \times w$ of counters of integer type.
- A collection of d hash functions $h_k : U \rightarrow \{0,\ldots,w-1\}, k = 1,\ldots,d$.

To insert an element $s_i \in S$ in CMS, for each hash function $h_k, k = 1,\ldots,d$, the counter at position $C[k,h_k(s_i)]$ is increased in one unit. Therefore, d counters of matrix C (one in each row of the matrix) are increased every time a new element is inserted. To obtain an approximation of the frequency of an element s_i in S (number of times the element has been inserted), again, each hash function h_k is applied to s_i to get the value of a counter $C[k,h_k(s_i)]$. The minimum of the d counters obtained is the approximation of the frequency of the queried element s_i. More formally, the approximated frequency of an element $s_i \in U$ is obtained as follows.

$$CMS(s_i) = MIN_{k=0}^{d} C[k,h_k(s_i)]$$

It is noticed that due to the fact that two or more applications of any of the hash functions to two or more elements might yield exactly the same result (collisions in the hash functions), the value approximated by the structure for the frequency of any element might be greater than the real one, i.e., the approximation of the frequency given by the CMS is an overestimation of the real value of the frequency.

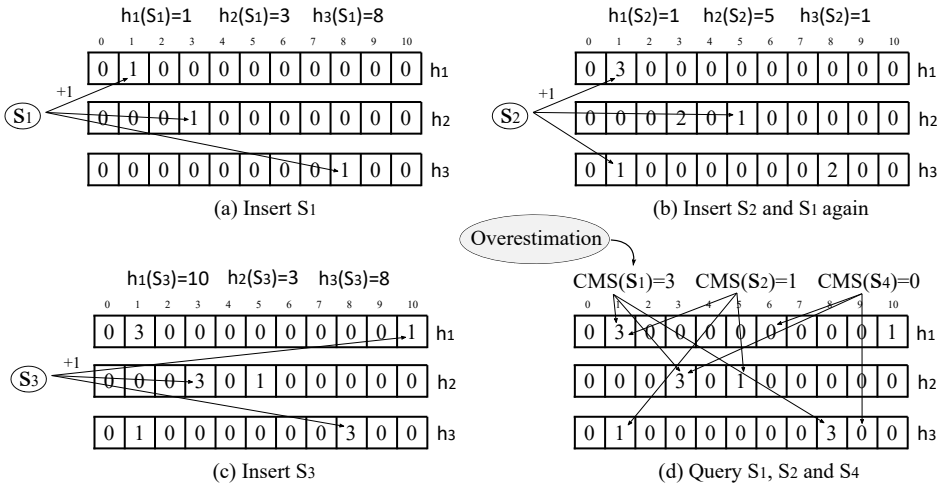
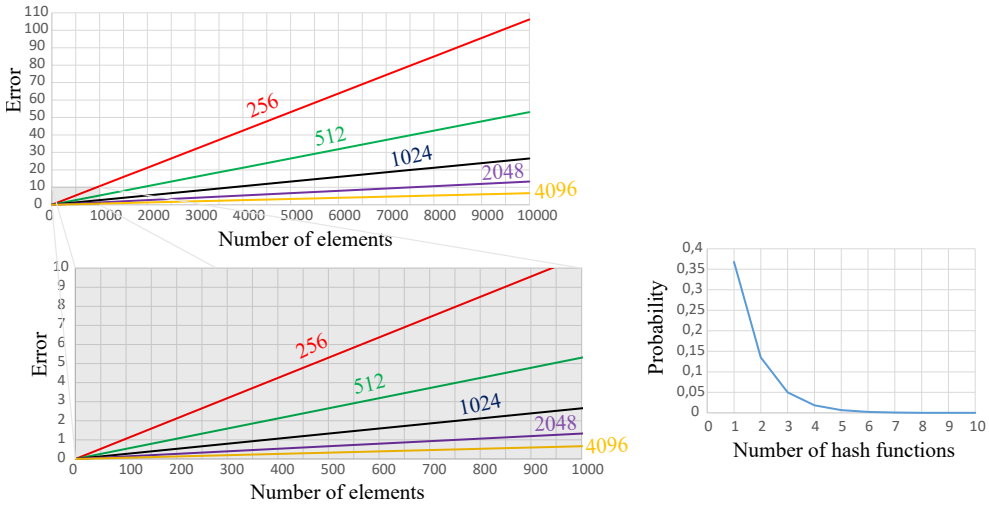


Figure 6.4: Illustration of the insertion and querying in a Count-Min Sketch.

Figure 6.4 illustrates various insertions and queries in a CMS with 3 hash functions and 11 counters for each hash function ($d = 3, w = 11$). In particular, Figure 6.4(a) shows how three counters of the CMS matrix are increased during the insertion of element S_1. Figure 6.4(b) illustrates the insertions of a new element (S_2) and also a second insertion of element S_1. There is a collision in hash function h_1 between elements S_1 and S_2, as for both elements the hash function yields position number 1. Therefore, counter number 1 for hash function h_1 records a frequency that is an overestimation for both S_1 and S_2. In spite of this, the CMS would obtain a correct value for the frequency of any of these two elements, due to the use of the minimum of all the counters. The insertion of element S_3 is illustrated in Figure 6.4(c). In this case, two hash functions h_2 and h_3 yield collisions between S_1 and S_3. Because of this, when element S_1 is queried in Figure 6.4(d), a result value for its frequency of 3 is obtained, which is an overestimation of its real value (S_1 has been inserted only twice). The figure shows also how elements S_2 and S_4 are queried obtaining approximations that match the real values of their frequencies.

Theoretical bounds for the error of the approximation of a CMS have been proposed [12].



(a) Máximum errror for number of elements and number of counters per hash function

(b) Probability of having higher error for number of hash functions

Figure 6.5: Error guaranteed and relevant probability for increasing number of elements inserted in Count-Min Sketch with different number of counters per hash function.

In particular, if $freq(s_i)$ denotes the real frequency of element s_i in S, and $CMS(s_i)$ denotes the approximation of the frequency obtained from the CMS, then the expression

$$CMS(s_i) \leq freq(s_i) + \varepsilon * n, \quad \varepsilon = e/w$$

is guaranteed with a probability of $1 - e^{-d}$. Therefore, given an objective error factor $\varepsilon = error/n$ and a given probability for the guarantee of the error of $1 - \delta$, the required values for d and w may be obtained with the following formulas:

$$w = \left\lceil \frac{e}{\varepsilon} \right\rceil \quad d = \left\lceil \ln \frac{1}{\delta} \right\rceil$$

Figure 6.5(a) shows values of the absolute error for different values of the number of elements inserted n and for the number of counters stored for each hash function w. On the other hand, Figure 6.5(b) shows how the probability of having a higher error decreases by increasing the number of hash functions d used.

6.2.3 Sachem

Sachem [30] is an open-source chemical cartridge for the PostgreSQL database that allows efficient molecular substructure searches in databases of different sizes, including large size databases. Sachem implements two search methods. The first one is a modification of the OrChem[41] cartridge, porting it to PostgreSQL and adding some optimizations, such as replacing the molecule storage format from text to a binary format or optimizing the VF2 algorithm for better performance. The second method is an extension of the previous and employs fixed-size fingerprints, which are adapted to be stored in inverted indices in the Apache Lucy [1] database. Apache Lucy is a port of the well-established Apache Lucene [4]. The indexing works as follows. First, the molecule is decomposed into substructural features, which are encoded into fingerprints. Next, the fingerprint bits are converted into keyword-like descriptors compatible with Apache Lucy text documents. The resulting keywords are concatenated to a string and indexed as documents using a whitespace analyzer. It should be noted that Sachem does not take into account the types of bounds (tags of edges in the graph), and therefore, the edges are used neither for indexing nor for searching. This leads to a faster performance, with the cost of not retrieving exact solutions in some scenarios.

note that if we consider the results obtained by the VF2 algorithm for each query as the correct ones, then both false positives and false negatives can be found in the results obtained by Sachem. It is also reminded that the filtering stage of all the FTV approaches used in this Thesis may yield false positives, but not false negatives. Based on these observations, Sachem will be considered as another approach with approximate results, and thus, it will be compared with the other approaches in terms of both efficiency and efficacy.

6.3 Experimental Evaluation of Efficacy and Efficiency

This section presents and discusses the results of the evaluation performed, in terms of efficacy and efficiency, of subgraph search solutions build with fingerprints (Bloom Filters of just one hash function) and also with Count-Min Sketches. These solutions are also compared with Sachem, to show where they are located with respect to currently available systems.

[1] https://lucy.apache.org/

6.3.1 Evaluation Framework

The experiments were performed in a centralised hardware architecture of a single node with the following configuration. Ubuntu Server release 20.04.2, with a KVM processor of 2.20 GHz and 8 cores, 8 GB of RAM and 40 GB of disk. Various different subgraph search approaches constructed with Bloom Filters (actually fingerprints) and Count-Min Sketches were tested. Specifically:

- *KM-CTI*: Various solutions based on the KM-CTI indexing structure proposed in this Thesis were evaluated. As it was remarked, the fingerprints used by this approach are actually Bloom Filters with a single hash function. Six different fingerprint sizes were considered ranging from 256 bits to 8192 bits. These approaches were evaluated with just the filtering stage (approximate solutions) and with both filtering and verification stages.

- *CMS*: Various solutions based on the use of Count-Min Sketches for the filtering stage of FTV approaches were implemented and evaluated. The CMS was used to replace the fingerprint in an approach based on the CT-Index. Recording the number of repetitions (frequency) for each feature extracted from each graph instead of a single Boolean value provides additional information that is used to discard more graphs during the filtering stage. On the other hand, the resulting indexing structure is larger and the filtering response time is increased. CMS structures with 1, 2 and 3 hash functions and with a number of counters per hash function ranging from 256 to 4096 were considered. Again, these approaches were evaluated with just the filtering stage (approximate solutions) and with both filtering and verification stages.

- *Sachem*: The extension was installed in a PostgreSQL database and its performance was compared with the various combinations of the approaches above.

One database was constructed for the experiments, composed of 200k molecules, extracted from the PubChem dataset. The queries used in the experiments were also obtained from the PubChem dataset following the approach already described in Section 4.4.1. Three sets of 1000 queries were built, one set for small size queries (8 edges), another set for medium size queries (20 edges), and a last set for large size queries (40 edges).

6.3.2 Evaluation of the System Efficacy

To be able to give a measure of the system efficacy, a reference approach that provides maximum efficacy (a gold standard) has to be chosen. In this Thesis, the selected gold standard is the VF2 subgraph isomorphism algorithm [11].

The efficacy measures used in this section are those most commonly used in the field of information retrieval [35]. Let R be the set of graphs retrieved by the system and G the set of graphs contained in the result of the gold standard, i.e., the set of graphs that the system should have retrieved. Then, the following efficacy measures are defined:

- *Precision*: It is defined as the ratio of true positive graphs retrieved by the system with respect to all the graphs retrieved by the system. More formally,

$$precision = \frac{R \cap G}{R}.$$

  The precision gives an inverse measure of the amount of incorrect results retrieved by the system. Thus, a precision of 1 means that all the graphs retrieved by the systems are correct, although perhaps the system may not retrieve all the required graphs.

- *Recall*: It is defined as the ratio of true positive graphs with respect to the result of the gold standard. More formally,

$$recall = \frac{R \cap G}{G}.$$

  The recall gives a measure of how complete the result is with respect to the gold standard. Thus, a recall of 1 means that all the graphs of the gold standard have been retrieved by the system, although there might be some graphs retrieved by the system that should not have been retrieved.

- *F-Score (F1-Score)*: It combines the two measures above into a simple measure, enabling this way the comparison of different approaches with a single aggregated measure. This measure assumes the same importance for both recall and precision. The mathematical expression that obtains this measure from the two previous as follos.

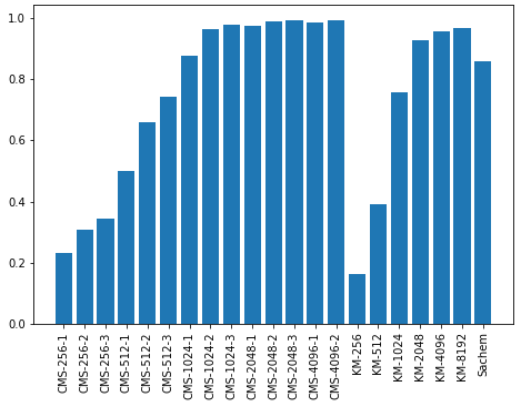
$$F\text{-}Score = 2 * \frac{precision * recall}{precision + recall}.$$

It is noticed that all the subgraph search solutions described in previous chapters reach the gold standard, since their verification stage consists in the evaluation of VF2 to discard false positives. It is reminded, however, that the filtering stages of the FTV approaches described in this Thesis might have false positives, therefore the precision of their results is usually lower than 1. Finally, the Sachem implementation yields both false positives and false negatives, therefore, in general, this system does not achieve a perfect precision nor a perfect recall.

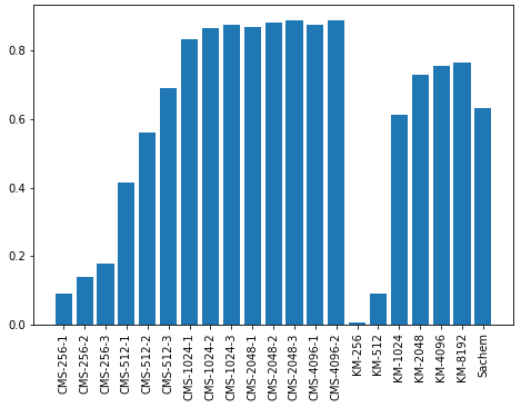
The precision values achieved by each of the approximate search approaches evaluated for different query sizes are illustrated in Figure 6.6. More precisely, 6.6(a) shows the average precision of the 1000 queries of small size (size 8) for various filtering approaches using CMS and KM-CTI; as well as the precision reached by Sachem. The CMS structures reach a very high precision (around 0.9) when the number of counters per hash function is higher or equal to 1024. The use of more than one hash function does not bring a significant improvement in precision for numbers of counters greater than 1024. A similar precision for these small queries is also achieved by KM-CTI using a bitset of at least 2048 bits. The precision of Sachem is lower, slightly above 0.8. For medium and large queries, see Figures 6.6(b) and 6.6(c), respectively, again a CMS with 1024 counters and just one hash function brings a good precision (above 0.8). However, now the KM-CTI does not reach such high precision, even with the highest number of bits. The precision of Sachem is only slightly higher than 0.6.

Both CMS and KM-CTI have recall values of 1, therefore their F-Score is only determined by their precision. This is not the case for Sachem, which has both false positives and false negatives. The values of the F-Score of all the approaches are shown in Figure 6.7. The comparison between CMS and KM-CTI throws the same conclusions as the comparison of their result precision. Again, a CMS of 1024 with just one hash function gives F-Scores that are not significantly improved by higher numbers of counters or hash functions. Again, the KM-CTI has F-Scores similar to those of CMS only for small queries. Now, Sachem is penalised even more, due to the presence of false negatives, and its F-Score is not higher than 0.6 in any case, even for small queries.

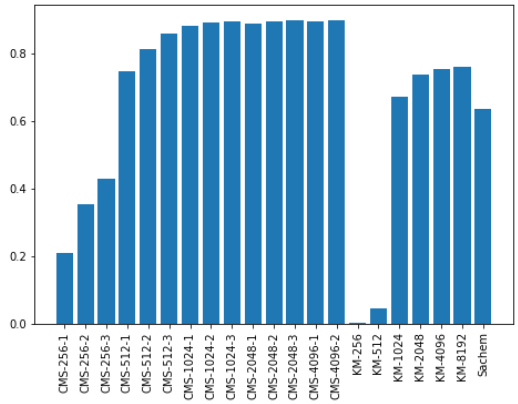
As a conclusion of the above comparison of the efficacy of the different approaches, a solution based on the CMS provides good results for queries of any size. Using a maximum feature size of 6 does not required more than 1024 counters and more than one hash function to achieve good results. KM-CTI should only be used for small queries and Sachem does not give good enough results in any case.



a) Query Size 8

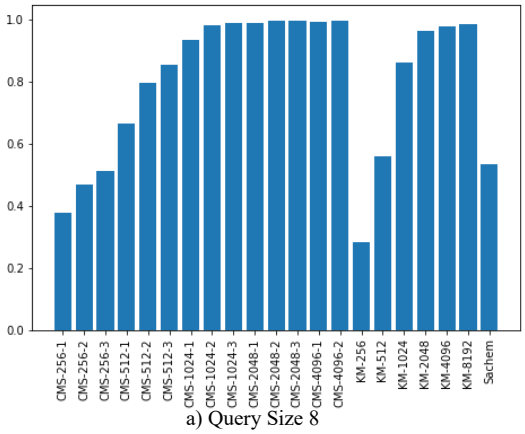


b) Query Size 20

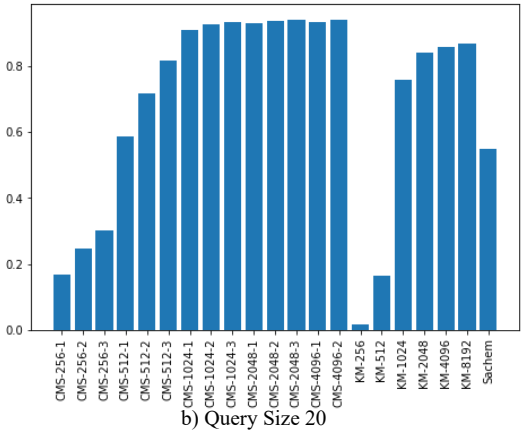


c) Query Size 40

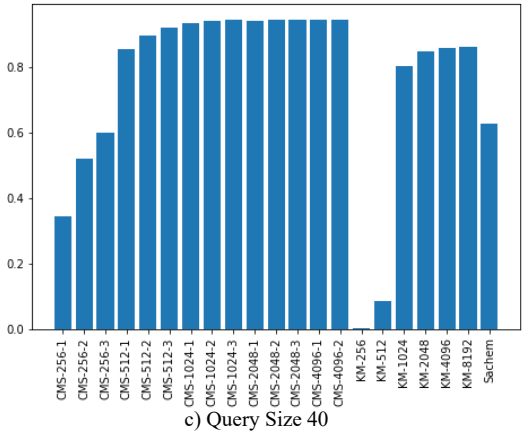
Figure 6.6: Response average precision.



a) Query Size 8



b) Query Size 20



c) Query Size 40

Figure 6.7: Response F-Score

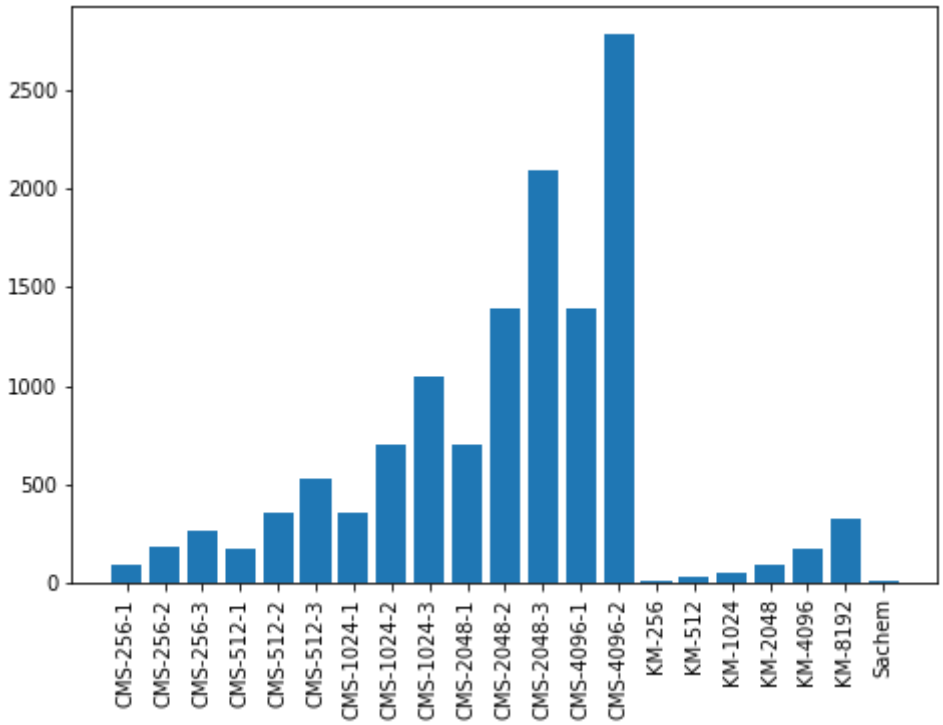


Figure 6.8: Index size (MB).

6.3.3 Evaluation of the System Efficiency

Two different measures were considered to evaluate the efficiency of the different approaches, namely the amount of memory required to record the index structure and the query response time.

The size of the indexing structure used by each approach is shown in Figure 6.8. The size of the index for the approaches based on the CMS structure is larger and increases significantly with the number of counters and hash functions. KM-CTI performs better than CMS in terms of memory usage and Sachem has the smallest index size.

The approaches whose efficacy was compared in the previous section are now evaluated in terms of their query response times. The average response times achieved by each of the approaches with the three query sizes are shown in Figure 6.9. The fastest CMS based

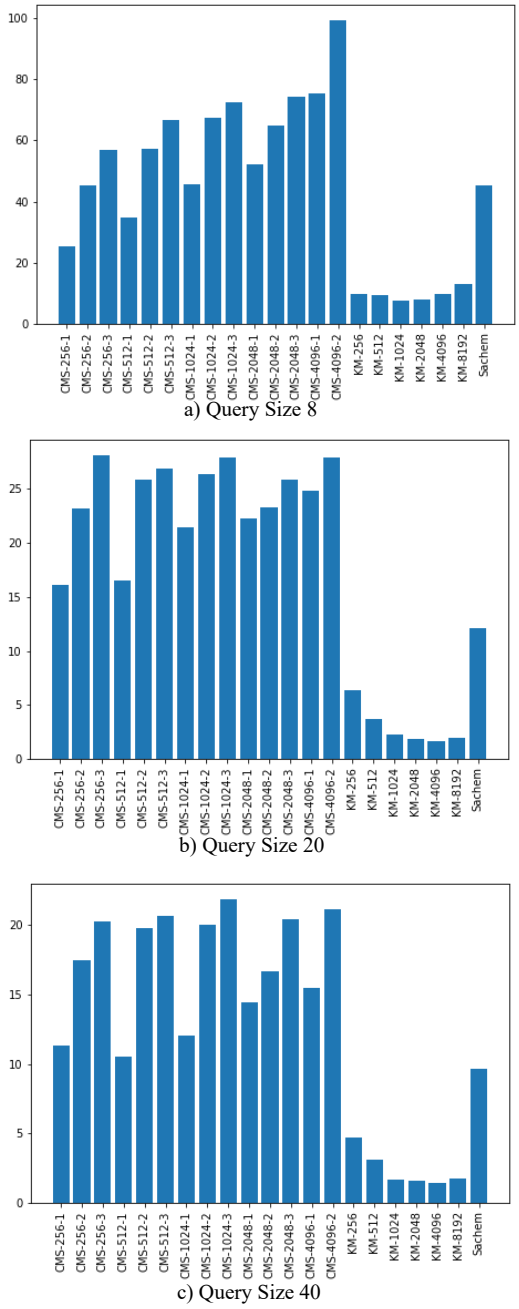
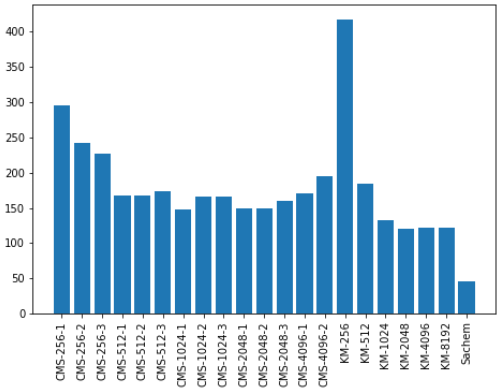


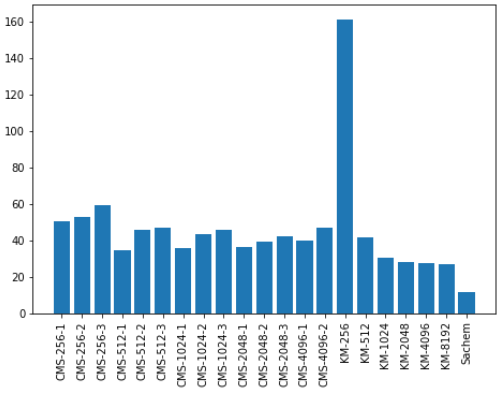
Figure 6.9: Average response time (ms) of the approximate searching approaches.

approach of those identified in the previous section as having good efficacy is the one with 1024 counters and just one hash function. The response time of this CMS based approach is similar to the one achieved by Sachem, but it is reminded that the efficacy of Sachem is too poor. On the other hand, the response times achieved by KM-CTI based solutions are by far much better. Based on the above, we can conclude that CMS and KM-CTI based approaches are better than Sachem because they have better efficacy and also either better or equal efficiency. Regarding whether to use CMS or KM-CTI based approaches, it is clear that KM-CTI is better for small queries. However, for medium and large queries it depends on the application. If efficacy is more important than efficiency, then it is better to use a CMS structure. On the contrary, if the efficiency is the key factor, it is clear that KM-CTI is going to reach much better performance.

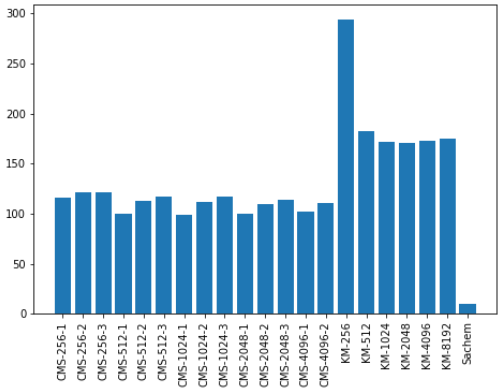
As it was shown above, CMS based approaches have better efficacy and KM-CTI has better response time. One possible way to get an idea of whether a better efficacy compensates for a worse response time is to include the verification stage in the comparison. In particular, we will observe whether the better efficacy produces a gain in verification response time that compensates the worse response time of the filtering stage. The total response times of all the approaches for the three different query sizes are shown in Figure 6.10. Sachem has the best figures in terms of response time, but it is reminded that Sachem does not have a good efficacy, whereas the other approaches have perfect recall and precision. For small queries (see Figure 6.10(a)), as it was expected, approaches based on KM-CTI are slightly better than those based on CMS. It is reminded that KM-CTI has better results also in terms of index size. For queries of medium size (see Figure 6.10(b)), the response times of all the approaches are very similar. Finally, CMS based approaches have better response time than those based on KM-CTI for large queries (see Figure 6.10(c)). It is also noticed that, if we discard the CMS with 256 counters and the KM-CTI with 256 bits, which have clearly worse performance, all the other approaches have similar performance. Thus, if the objective is to have perfect efficacy, it is not worth creating large indexes since 512 counters with just one hash function for CMS and 1024 bits for KM-CTI offer a very good query response time.



a) Query Size 8



b) Query Size 20



c) Query Size 40

Figure 6.10: Average response times (ms) including also the verification stage.

Chapter 7

Conclusions and Future Work

The achievements of the research work carried out in this Thesis are summarised in this section. A first result is the design and implementation of a system for the storage and querying of molecular data, which was developed in the scope of the NEXTCHROM project. As underlying data storage subsystems, the implementation supports both relational databases in centralised architectures and document-based NoSQL databases in distributed architectures. The query capabilities consist of conventional alphanumeric queries, queries over molecular formulas, queries over molecular structures (graphs), and queries over spectra. In general, state-of-the-art technologies have been used to support all of the above types of queries in the first prototype of the system.

The remaining results achieved in the Thesis are related to the development of new solutions, beyond the state of the art, for substructure search, i.e., new methods for solving the subgraph search problem.

Three new FTV solutions for subgraph search on large databases of small graphs have been proposed. The new solutions are based on the existing state of the art methods GGSX and CT-Index. Bitmap GGSX (BM-GGSX) improves GGSX by adding compressed bitmaps for the representation of graph references. BM-GGSX is fast for medium and large size queries, but it suffers from large index size and slow index building. Column-Wise CT-index (CW-CTI) leverages large compressed bitmaps for the column-wise storage of graph fingerprints. It shows great performance for small size queries and good index building time (similar to that of the original CT-Index structure). K-Means CT-Index (KM-CTI) recursively applies the K-Means clustering algorithm to construct a binary tree of fingerprints. This method achieve

a good performance when the queries are of medium or large size. Thus, a combination of CW-CTI and KM-CTI covers all types of queries and could be used to implement subgraph search solutions in centralised architectures.

A generic framework for implementating of FTV methods on top of a large scale distributed data processing engine has been designed and implemented. FTV subgraph search methods can be incorporated into the framework by providing binary encodings for their indexing structures. A comprehensive evaluation has been performed using datasets orders of magnitude larger than those considered in previous studies. The distributed implementations enable a significant reduction in index building time. CT-Index shows good performance in terms of query response times for small queries. On the other hand, KM-CTI is better for large queries. In general, centralised implementations perform well with highly selective large queries over small databases, and on the other hand, distributed implementations leverage the use of powerful clusters to achieve good performance either for small low selective queries or for large database sizes.

Finally, using the filtering stage of FTV subgraph search methods might be a good solution to obtain approximate results with interactive response times (less than one second) in user interfaces developed for interactive searching and exploration of very large graph datasets. Two different data sketches were used to develop filtering solutions, namely Bloom Filters (BF) and Cout-Min Sketch (CMS). The experiments for the evaluation of the efficacy and efficiency of the approaches showed that CMS-based solutions are good to achieve results with high precision (few false positives) but with slower response times. On the other hand, solutions based on fingerprints (BF with just one hash function) are good to obtain very fast response times with not so good result precision. The evaluation also showed that some existing solutions for molecular substructure search do not obtain good efficacy when compared with the results of subgraph isomorphism algorithms.

The outcomes of the research work open up some future research perspectives. An interesting one is the development of hybrid indexing structures for composed queries. As it was shown, a molecular entity has several different types of data. Two of the most common required searches are substructure search and text search. A structure that allows to perform a query with these two different types of data would be very powerful in terms of filtering. A first idea would be some kind of integration of a text index inside the index structure of an FTV approach. Another research issue would face the problem of database partitioning in a distributed architecture. The framework proposed in this Thesis uses a round robin partition-

ing, which achieves a very uniform data distribution, but it does not achieve good co-location of related graphs. More advanced and intelligent partitioning approaches could lead to more efficient distributed search approaches that would increase system throughput and reduce the energy consumption. Another future research line is to address the molecule similarity problem. Some works have been proposed to solve this problem. More advanced approaches could try to incorporate molecule similarity into an index structure of an FTV method.

Bibliography

[1] Michael A. Bender, Martin Farach-Colton, Rob Johnson, Russell Kraner, Bradley C. Kuszmaul, Dzejla Medjedovic, Pablo Montes, Pradeep Shetty, Richard P. Spillane, and Erez Zadok. Don't thrash: How to cache your hash on flash. *Proc. VLDB Endow.*, 5(11):1627–1637, jul 2012.

[2] B. Bhattarai, H. Liu, and H. Howie Huang. Ceci: Compact embedding cluster index for scalable subgraph matching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1447–1462, 2019.

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

[4] A Białecki and G Ingersoll. Lucid imagination. Apache lucene 4. *SIGIR 2012 workshop on open source information retrieval*, page 17, 2012.

[5] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

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

[7] Flavio Bonomi, Michael Mitzenmacher, Rina Panigrahy, Sushil Singh, and George Varghese. An improved construction for counting Bloom filters. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4168 LNCS:684 – 695, 2006.

[8] Stuart K. Card, George G. Robertson, and Jock D. Mackinlay. The information visualizer, an information workspace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, page 181–186, New York, NY, USA, 1991. Association for Computing Machinery.

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

[10] Saar Cohen and Yossi Matias. Spectral bloom filters. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, page 241–252, New York, NY, USA, 2003.

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

[12] Graham Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1):58 – 75, 2005.

[13] H.-C. Ehrlich and M. Rarey. Systematic benchmark of substructure search in molecular graphs - From Ullmann to VF2. *Journal of Cheminformatics*, 4(1), Jul 2012.

[14] H.-C. Ehrlich, A. Volkamer, and M. Rarey. Searching for substructures in fragment spaces. *Journal of Chemical Information and Modeling*, 52(12):3181–3189, 2012. PMID: 23205736.

[15] Bin Fan, David G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. Cuckoo filter: Practically better than bloom. In *CoNEXT 2014 - Proceedings of the 2014 Conference on Emerging Networking Experiments and Technologies*, page 75 – 87, 2014.

[16] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281 – 293, 2000.

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

[18] Edward Fredkin. Trie memory. *Commun. ACM*, 3(9):490–499, sep 1960.

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

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

[21] Myoungji Han, Hyunjoon Kim, Geonmo Gu, Kunsoo Park, and Wook-Shin Han. Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, pages 1429–1446, New York, NY, USA, 2019. ACM.

[22] W.-S. Han, J. Lee, and J.-H. Lee. 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*, pages 337–348. ACM, 2013.

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

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

[25] X. Jin and L. Lai. Mpmatch: A multi-core parallel subgraph matching algorithm. In *Proceedings - 2019 IEEE 35th International Conference on Data Engineering Workshops, ICDEW 2019*, pages 241–248, 2019.

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

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

[28] Foteini Katsarou, Nikos Ntarmos, and Peter Triantafillou. Hybrid algorithms for subgraph pattern queries in graph databases. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 656–665, 2017.

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

[30] Miroslav Kratochvíl, Jiří Vondrášek, and Jakub Galgonek. Sachem: A chemical cartridge for high-performance substructure search. *Journal of Cheminformatics*, 10(1), 2018.

[31] G. Landrum. Rdkit: Open-source cheminformatics software. 2017.

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

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

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

[35] Raghavan P. Schütze H. Manning, C.D. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[36] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann, 1994.

[37] Noel M. O'Boyle, Michael Banck, Craig A. James, Chris Morley, Tim Vandermeersch, and Geoffrey R. Hutchison. Open babel: An open chemical toolbox. 3(10), 2011.

[38] Prashant Pandey, Michael A. Bender, Rob Johnson, and Rob Patro. A general-purpose counting filter: Making every bit count. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, volume Part F127746, page 775 – 787, 2017.

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

[40] Xuguang Ren, Junhu Wang, Wook-Shin Han, and Jeffrey Xu Yu. Fast and robust distributed subgraph enumeration. *Proc. VLDB Endow.*, 12(11):1344–1356, July 2019.

[41] Mark Rijnbeek and Christoph Steinbeck. Orchem - An open source chemistry search engine for Oracle®. *Journal of Cheminformatics*, 1(1), 2009.

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

[43] Christoph Steinbeck, Yongquan Han, Stefan Kuhn, Oliver Horlacher, Edgar Luttmann, and Egon Willighagen. The chemistry development kit (cdk): An open-source java library for chemo- and bioinformatics. 43(2):493 – 500, 2003.

[44] S. Sun and Q. Luo. Scaling up subgraph query processing with efficient subgraph matching. In *Proceedings - International Conference on Data Engineering*, volume 2019-April, pages 220–231, 2019.

[45] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):31–42, 1976.

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

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

[48] J. Wang, X. Ren, S. Anirban, and X.-W. Wu. Correct filtering for subgraph isomorphism search in compressed vertex-labeled graphs. *Information Sciences*, 482:363–373, 2019.

[49] David Weininger. Smiles, a chemical language and information system: 1: Introduction to methodology and encoding rules. 28(1):31 – 36, 1988.

[50] Egon L. Willighagen, John W. Mayfield, Jonathan Alvarsson, Arvid Berg, Lars Carlsson, Nina Jeliazkova, Stefan Kuhn, Tomáš Pluskal, Miquel Rojas-Chertó, Ola Spjuth, Gilleain Torrance, Chris T. Evelo, Rajarshi Guha, and Christoph Steinbeck. The chemistry development kit (cdk) v2.0: atom typing, depiction, molecular formulas, and substructure searching. 9(1), 2017.

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

[52] M. Zaharia, R.S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M.J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica. Apache spark: A unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

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

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

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

[56] W. Zheng, L. Zou, X. Lian, H. Zhang, W. Wang, and D. Zhao. SQBC: An efficient subgraph matching method over large and dense graphs. *Information Sciences*, 261:116–131, 2014.

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

List of Figures

Fig. 2.1 Illustration of a molecular entity. 24
Fig. 2.2 Example of an NRM spectrum. 28
Fig. 2.3 Illustration of the molform index developed during the NEXTCHROM project. 30

Fig. 3.1 Example of an undirected graph representing the structure of a molecule. . . 35
Fig. 3.2 Illustration of a subgraph isomporphism. 35
Fig. 3.3 Representation of a trie structure used as index in GGSX. 41
Fig. 3.4 Example of the filtering stage in GGSX. 41
Fig. 3.5 CT-Index fingerprint building example. 43
Fig. 3.6 Example of the filtering stage in CT-Index. 44

Fig. 4.1 Example of a BM-GGSX trie structure. 48
Fig. 4.2 Comparison between (a) GGSX and (b) BM-GGSX path repetitions storage. 49
Fig. 4.3 BM-GGSX filtering stage. 50
Fig. 4.4 Fingerprint insertion in the CW-CTI index. 52
Fig. 4.5 CW filtering stage. 53
Fig. 4.6 Example of distance and mean measures between bitmaps used in KM-CTI. 55
Fig. 4.7 Example of the index building stage in KM-CTI. 56

Fig. 4.8 Example of the filtering stage in KM-CTI. 58
Fig. 4.9 Comparison of response times(ms) between GGSX and BM-GGSX on AIDS for different maximum path sizes and query sizes. 61
Fig. 4.10 Comparison of response times(ms) of CT-Index on AIDS for different maximum path sizes, fingerprint sizes and query sizes. 63
Fig. 4.11 Comparison of response times(ms) of CW-CTI on AIDS for different maximum path sizes, fingerprint sizes and query sizes. 64
Fig. 4.12 Comparison of response times(ms) of KM-CTI on AIDS for different maximum path sizes, fingerprint sizes and query sizes. 65
Fig. 4.13 Comparison of response times(ms) of BM-GGSX on PubChem200k for different maximum path sizes and query sizes. 66
Fig. 4.14 Comparison of response times(ms) of CT-Index on PubChem200k for different maximum path sizes, fingerprint sizes and query sizes. 68
Fig. 4.15 Comparison of response times(ms) of CW-CTI on PubChem200k for different maximum path sizes, fingerprint sizes and query sizes. 69
Fig. 4.16 Comparison of response times(ms) of KM-CTI on PubChem200k for different maximum path sizes, fingerprint sizes and query sizes. 70
Fig. 4.17 Performance comparison of the methods on the AIDS database for different query sizes. 71
Fig. 4.18 Index Size and Building Time for increasing database sizes. 73
Fig. 4.19 Filtering time for increasing database and query sizes. 74
Fig. 4.20 Total response time for increasing database and query sizes. 76

Fig. 5.1 Serialization into a byte array of a trie node in BM-GGSX. 82
Fig. 5.2 Serialization of CW-CTI. 83
Fig. 5.3 Serialization in KM-CTI: (a) Internal node and (b) Leaf node. 84
Fig. 5.4 Index Building Time in distributed system varying (a) number of executors and (b) database size. 89
Fig. 5.5 Comparison of index building time between centralised and distributed implementations for (a) BM-GGSX and (b) KM-CTI. 90
Fig. 5.6 Average filtering time for distributed implementations (4M graph database size). 92
Fig. 5.7 Average filtering time for distributed implementation (64 executors). 93

Fig. 5.8 Average total response time for distributed implementations (4M graph database size). 95
Fig. 5.9 Average total response time for distributed implementation (64 executors). . 96
Fig. 5.10 Comparison of query filtering time between centralised and distributed implementations for KM-CTI method. 98
Fig. 5.11 Comparison of query total response time between centralised and distributed implementations for KM-CTI method. 99
Fig. 5.12 Illustration of the significance of the use of indexing with respect to the simple parallel evaluation of a subgraph isomorphism algorithm. 100

Fig. 6.1 Mockup of an interactive searching application. 102
Fig. 6.2 Illustration of the insertion and querying in a Bloom Filter. 105
Fig. 6.3 False positive probability achieved for increasing number of elements inserted in Bloom Filters of just one hash function with different sizes (256-8192). 106
Fig. 6.4 Illustration of the insertion and querying in a Count-Min Sketch. 108
Fig. 6.5 Error guaranteed and relevant probability for increasing number of elements inserted in Count-Min Sketch with different number of counters per hash function. 109
Fig. 6.6 Response average precision. 114
Fig. 6.7 Response F-Score 115
Fig. 6.8 Index size (MB). 116
Fig. 6.9 Average response time (ms) of the approximate searching approaches. 117
Fig. 6.10 Average response times (ms) including also the verification stage. 119

The efficient management of molecular data is one of the most demanded technologies by the industry. A very important type of search is the substructure searching. The molecular structures may be encoded as graphs where the vertices and bonds represent the atoms and bonds, respectively. In this Thesis, a cutting edge system that enables the storage and querying of molecular data has been designed and implemented, paying attention to the molecular substructure search,
where new filter-then-verify(FTV) methods, beyond the state-of-the-art, were designed, implemented, and tested,
achieving performance gains over 75% in the filtering stage. A generic framework for the implementation of FTV techniques on a distributed architecture was also developed, enabling the application of the FTV methods on very large graph databases,