Dense Sub-Tensor Mining using SQL

Changkai Zhou
Carnegie Mellon University
zchangka@andrew.cmu.edu

Peilun Li
Carnegie Mellon University
peilunl@cs.cmu.edu

April 24, 2017

1 Introduction

Dense subtensor mining is an important method to detect interesting patterns and anomalies from real-world data. Many effective algorithms have been proposed and justified to realize the detection. However, in fact, the real-world data is usually in a fair large size so that it should be stored in relational database and accessed by SQL query instead of being stored in memory. In this paper we We implement D-Cube algorithm using SQL and apply it to real-world datasets. Experiments show that our implementation has an excellent performance.

2 Problem description

In the project, the problem can be divided into these parts to solve below:

- Implemented D-Cube algorithm using SQL
  - Given: (1) multi-aspect data: R, (2) number of dimension attributes: N, (3) number of dense blocks we aim to find: k, (4) density measure: ρ , and (5) a dimension-selection policy.
  - Output: (1) compute the k dense blocks in the given multi-aspect data and store them in a table; and (2) return the densities of the k dense blocks.
- Optimizing D-Cube implementation using indexing methods
- Evaluating the performance of D-cube algorithm using ROC curve
  - Report the the size, mass, and density of the detected blocks in each dataset with $\rho = \rho ari$, $k = 5$, and Maximum density policy.
  - Explain whether the detected blocks indicate specific types of network attacks.
  - Draw a ROC curve and compute AUC for each dataset with $\rho = \rho ari$ and Maximum density policy.
  - Observe how AUC changes depending on density measures and dimension-selection policies.

- Applying D-Cube in real-world data discover interesting patterns and anomalies.
  - Report the size, mass, and density of detected blocks in each dataset with $k = 5$ and Maximum density policy. Regarding density measures, use $k = 5$ for English Wikipedia Revision History and $\rho = \rho ari$ for the other datasets.
  - Provide a justification why you believe the detected blocks do or do not indicate interesting anomalies.

3 Survey

3.1 Papers read by Changkai Zhou

The first paper was the Crossspot paper by Meng Jiang, Alex Beutel, Peng Cui, Bryan Hooi, Shiqiang Yang, and Christos Faloutsos [1]

- *Problem Definition*:

  This paper focus on the detection of suspicious dense blocks in multimodal data with the proposed metric to score the suspiciousness.

- *Main idea*:

  A scalable algorithm, CrossSpot, is proposed to find dense and suspicious blocks in multi-modal data with a novel suspiciousness metric that is based on a principled, probalistic model and the axioms of suspicious behaviour.
  Many effective methods to find suspicious blocks in tensors have been proposed in literature, but no one gives the score and the rank of suspiciousness for the benefit of people to make a reasonable trade-off about cost.
  What's novel in the paper are mainly the proposed suspicious metric and axioms as well as the corresponding scalable algorithm to detect suspiciousness.
  This algorithm performs well on synthetic and real world data compared with other baselines including SVD, High-Order SVD, MAF, AvgDegree.
  However, it's doubtful that no explicit shortcomings or restriction is mentioned in the paper.

- *Use for our project*:

  The algorithm, CrossSpot, will be a effective supplement to our project model.
  It will improve the effectiveness of final result to evaluate and score the suspiciousness of dense blocks with different modes. With such kind of information, the limitation of time and resources will be relieved since people know what to focus on at first according to the score of suspiciousness.

- *Shortcomings*:

  If we want to find a large amount of suspicious blocks, the algorithm may not performance effectively because every time running brings only one most suspicious node and the time cost linear in the number of non-zero entries. In this case, we have to consider other methods.

The second paper was the GBase paper by U Kang, Hanghang Tong, Jimeng Sun, Ching-Yung Lin, Christos Faloutsos [2]

- *Problem Definition*:

  Large data of graphs plays an important role in many applications, so the efficient storation of data and fast response of graph queries attract researchers. The paper propose an applicable analysis platform to fulfill the requirements.

- *Main idea*:

  GBase is an efficient platform designed for graph mining with large data on the top of Hadoop.
  Two main novelities are (1) efficient storage and compression in parallel, distributed settings and (2) well-designed algorithms and query optimization.
  The paper describes the reliable details of the design and implementation.
  As for graph storage optimization, a method named 'compressed block encoding' is used. First, the graph is partitioned into homogeneous blocks. Secondly, the block is compressed using standard algorithms. Finally, the block is placed into different files in a grid form.
  As for algorithm and query optimization, common graph operations can be generalized as matrix-vector multiplication. This kind of operation can be easily transformed into SQL joins query, so it will be easy to be implemented using database systems.
  The experiment's results show that the features of GBase improve the performance from these aspects: space efficiency, indexing time, global query time and targeted query time.

- *Use for our project*:

  The GBase platform will help us to research on graph mining with large data. To

increase the efficiency of data storage and querying, we can use GBase as the main platform.
In addition, we can also use the toolkits accompanying with Graph Database, including Graph Visualizations, Graph Analytics Library, Graph Middleware and Network Science Analytics tools.

- *Shortcomings*:

  In the GBase platform, graph operations are designed based on matrix-vector multiplication. As a result, if we need some other operations not in the form of matrix-vector multiplication, we may not realize them in GBase.

The third paper was the PEGASUS paper by U Kang, Charalampos E., Christos Faloutsos [3]

- *Problem Definition*:

  Graph mining with large data is a great challenge when the problem scales up to TB or even PB level. An optimization of graph mining operations and algorithms are expected to be created and applied into research and numerous applications.

- *Main idea*:

  A graph mining system for Peta-Scale, called PEGASUS, is proposed and created to optimize graph mining algorithms.
  The system is based on Hadoop platform and MapReduce, using distributed and parallel computation.
  One novelity is a highly optimized primitive, GIM-V (Generalized Iterated Matrix-Vector multiplication). The main idea of GIM-V is to iterate three main operations until meeting a specific convergence level. The three operations (*combine2*, *combineAll*, *assign*) fit the SQL operations in database well and improve the speed of queries effectively. It has been proved effective to solve find *finding pagerank* and *connected components* using GIM-V.
  To optimize GIM-V, they propose some faster algorithms for GIM-V than the native ones of Hadoop algorithms. The performance of optimized algorithms has been proved better both in theory and in practice.
  For the experiments on real-world big data, the PEGASUS is proved to give a great performance of finding patterns, outliers and interesting results.
  PEGASUS is expected to contribute a lot in the graph mining in large scale.

- *Use for our project*:

  It will help if we need to solve the graph mining problem for a Pega-Scale level dataset. However, we may lack the access right to super computer to fulfill the demandings of computation.

- *Shortcomings*:

  The PEGASUS is mainly designed for Pega-scale data, thus it needs very high power computation device, which we may not be able to use during the project.

3.2 Papers read by Peilun Li

The first paper was the M-Zoom paper by Kijung Shin, Bryan Hooi, and Christos Faloutsos [4]

- *Problem definition*:

  This paper tries to find dense blocks in large scale dataset with high speed, accuracy and flexibility.

- *Main idea*:

  In order to detect dense sub-tensor blocks with high speed, accuracy and flexibility, this paper proposes a framework called M-Zoom. Compared to other existed methods, M-Zoom supports multiple density measurements such as arithmetic average mass, geometric average mass and suspiciousness, and also has features such as accuracy guarantee, multiple blocks detection and size bounds.
  The general idea of M-Zoom is to find k dense sub-tensor blocks one by one. First M-Zoom will make a copy of relation $\mathcal{R}$ called $\mathcal{R}^{ori}$. Then for each iteration from 1 to k, M-Zoom will find the densest block within size bounds from current relation $\mathcal{R}$, add the corresponding block of original relation $\mathcal{R}^{ori}$ to the result, and remove the block from current relation $\mathcal{R}$. Through this process M-Zoom allows overlapping dense blocks.
  To find the densest block within size bounds for each iteration, M-Zoom uses a greedy approach. It will try to remove each possible attribute value from each dimension to maximize the density measurement. This greedy approach can be time consuming, but if we integrate min-heaps for choosing the attribute with minimum mass, we can speed up the process quite a lot. Actually, after the optimization, the time complexity

of the whole algorithm will be $O(kN|R|\log L)$ and space complexity will be $O(kN|R|)$, where $|R_n| = L\,, \forall n \in [N]$, and $N = \log L$. Also there's a accuracy guarantee that the dense box the algorithm finds (without sizes bounds) is at least $1/N$ the density of the optimum one.
Experiments show that M-Zoom has a good trade off between accuracy and speed. For example, M-Zoom has a 114× speed up compared with other state-of-art methods such as CrossSpot, with similar accuracy, and M-Zoom can also detect more distinct dense blocks. Another advancement of M-Zoom is its scalability. It will scale linearly or even sub-linearly as data size or dimensionality grows. In real world challenges, M-Zoom also detects dense blocks with near perfect accuracy.

- *Use for our project*:

  M-Zoom is extremely related to our project since we have the same aim to detect dense sub-tensor blocks from vertex graph. We may develop our algorithm based on M-Zoom's approach.

- *Shortcomings*:

  M-Zoom is calculated within memory, which is not the case that we want to calculate based on disk storage using SQL. So we need to find a way to apply M-Zoom's approach through SQL.

The second paper was the D-cube paper by Kijung Shin, Bryan Hooi, Jisu Kim, Christos Faloutsos [5]

- *Problem definition*:

  This paper tries to find dense blocks in large scale dataset in a distributed environment and with high speed, accuracy and flexibility.

- *Main idea*:

  All existing methods for dense sub-tensor blocks detecting, including M-Zoom, assume that the data can be fit into the memory, which is not the case in real world applications such as the web and social network, whose data is too large to fit in memory. To overcome this difficulty, this paper proposes D-Cube, a method which allows dense sub-tensor blocks detecting in both out-of-core environment and distributed environment.
  D-Cube can be seen as an improvement on M-Zoom. Like M-Zoom, D-cube also supports multiple density measurements such as arithmetic average mass, geometric average mass and suspiciousness, and it's algorithm's high-level architecture is similar as M-Zoom. First D-Cube will make a copy of relation $\mathcal{R}$ called $\mathcal{R}^{ori}$. Then for each

iteration from 1 to k, D-Cube will find the densest block from current relation $\mathcal{R}$ using $\mathcal{R}$'s mass as a parameter, add the corresponding block of original relation $\mathcal{R}^{ori}$ to the result, and remove the block from current relation $\mathcal{R}$. Through this process D-Cube allows overlapping dense blocks.
However in the details of finding the densest block for each iteration, D-Cube is quite different from M-Zoom. Specifically, it will first select a dimension and calculate the mass for each attribute value in that dimension, then make an order of them based on the sorted mass, through which it can get a densest or nearly densest block. The reason to change the algorithm from M-Zoom is that D-Cube is based on disk storage and M-Zoom's approach will involve a lot of disk seek. In contrast, D-Cube's approach will only scan the disk twice sequentially for each iteration. For the dimension selection in the above process, we can use multiple metrics such as maximum cardinality policy and maximum density policy. Likewise M-Zoom, D-cube also guarantees the accuracy that the dense box the algorithm finds is at least $1/N$ the density of the optimum one. For the complexity, the worst time complexity of D-Cube is $O(kN^2|R|L)$ and space complexity is $O(\sum_{n=1}^{N}|\mathcal{R}_n|)$.
To further speed up D-Cube, we can do some optimization such as combining disk accessing steps among iterations and caching some tensor entries in memory. There's also a MapReduce implementation of D-Cube which change the algorithm to multiple map-only stages and map-reduce stages.
Compared with other in-memory methods, D-Cube is 5 times faster than M-Zoom if caching data in memory, and D-Cube can give much denser results. D-Cube is 1600 times the memory efficient than the second best one and D-Cube can handle 1000 times the size of data than its competitors. With those advancement, D-Cube still scales linearly or even sub-linearly as data grows.

- *Use for our project*:

  D-Cube is extremely related to our project since we have the same aim to detect dense sub-tensor blocks from vertex graph and we want to implement our approach based on D-Cube's algorithms.

- *Shortcomings*:

  D-Cube is implemented by Java, which is not the case that we want our system to run through SQL. So we need to find a way to apply D-Cube's approach through SQL.

The third paper was the Vertica paper by Alekh Jindal, Samuel Madden, Malu Castellanos, Meichun Hsu [6]

- *Problem definition*:

This paper tries to conduct vertex-centric graph analysis through SQL queries efficiently.

- *Main idea*:

  Modern systems for graph analytics are vertex centric, such as Giraph and GraphLab. However, because data is typically stored in relational databases, it would be better if we can do graph analytics using SQL. This approach involves translating logic query plans of vertex centric systems' graph queries into SQL queries, and do some query optimizations.
  Giraph will take a vertex centric query as a map-only job of MapReduce framework, and build a static hard-coded pipeline of query execution, which will then form a logical query plan. But the weakness of Giraph is that the logical query plan behind its pipeline is fixed and is not suitable for many different situations.
  With Vertica we can solve this problem. First we can translate logical query plan to SQL query by eliminating message table and translating vertex compute function, after which the query plan becomes purely relational. Thus we can write corresponding SQL queries. We can also do some query optimizations. For example, we can replace "update" with "replace" in the SQL queries because creating new tables is faster than update existing tables. One weakness of this replacement is that this approach will lose the physical design of the original tables. But this loss is affordable since the replaced tables are usually of small size. Another downside is that "update" may be efficient than "replace" when numbers of update are small. So a more sophisticated way is to apply "update" on first few iterations and then change to "replace" after a threshold has been reached.
  Another optimization is incremental query evaluation. For example, for each iteration, we can only explore a subset of vertices which has been visited or changed in previous iteration, rather than the whole set of vertices. Further more, we can eliminate unnecessary "join" to speed up the query.
  Vertica is also optimized for creating projections, encoding and compressing data, joining over large tables, pipelining query execution, parallel processing, etc. Using Vertica, we can also run vertex function as a table UDF without translating it to relational operator.
  Experiments show that Vertica has comparable performance to GraphLab and has better performance than Giraph regarding query runtime, and Vertica uses much less memory. Vertica also has a better performance on relational and graph operations mixed benchmarks.

- *Use for our project*:

  Vertica is related to our project since we need to find some ways to translate a certain algorithm into SQL queries and optimize its performance.

- *Shortcomings*:

  It only provides some high level advice for translating into SQL and optimizing it. We need to analyze carefully for specific algorithms.

4 Implementation

4.1 Overview

We implement the D-Cube algorithm using SQL. Specifically, given a multi-aspect data relation R, its corresponding dimension attributes and measure attribute, number of dense blocks k, density measurement and dimension selection policy, our algorithm will produce the corresponding k dense blocks and output the corresponding data and its density to tables.

To illustrate our approach, we will first introduce our database schema design, and then our SQL version of D-Cube algorithm.

4.2 Database schema design

Given a multi-aspect data relation R, we will maintain some intermediate global tables, which is shown in Table 1.

Table	Definition
R_block	a block in R
R_card_list	a cardinality list for each dimension attribute in R
R_block_card_list	a cardinality list for each dimension attribute in block
R_tuple_to_remove	data to remove from R
R_res_to_add	data to append to results

Table 1: Intermediate global tables

We also maintain some intermediate dimension attribute tables for each dimension attribute. Specifically, for a dimension attribute *da* in relation R, the maintained intermediate dimension attribute tables are shown in Table 2.

The result of calculated dense boxes will be save into two tables as shown in Table 3.

4.3 Algorithm details

Apart from intermediate tables, we also maintain some parameters in memory, as shown in Table 4.

Table	Definition
R_da_set	a set of different *da* values in current *R*
R_da_block_set	a set of different *da* values in current block
R_da_mass_set	a *da* - mass relation in current block
R_da_order	remove orders for values in *da*
R_da_remove_set	values to remove from *da*
R_da_res_block_set	the *da* value set of resulting dense block

Table 2: Intermediate dimension attribute tables

Table	Definition
R_results	data in dense blocks
R_parameters	parameters for each dense block, such as density and tuple count

Table 3: Result tables

Our SQL D-Cube approach has the same work flow as Algorithm 1 - 4 in [5], in which the relations of symbols are shown in Table 5. However, there are some modifications for the correctness of SQL version:

- Whenever B is updated (line 16 of Algorithm 2 in [5]), we also update *R_da_block_set* and *R_block_card_list*, and recalculate density.
- Whenever R is updated (line 8 of Algorithm 1 in [5]), we also update *R_da_set*.
- The remove order of all values in dimension attributes are initially set to a MAX value, and will be rewritten to current order when removed.

We also provide our source code. You can find a clear SQL work flow from ”DCube_main.py”, just as Algorithm 1 - 4 in [5].

4.4 Optimizations (Task 3)

There are two kinds of optimizations which may be useful for our implementation:

Parameter	Definition
relation_mass	mass of *R*
block_mass	mass of block
cur_density	current density
max_density	max density
r	current order
best_r	best order

Table 4: In memory parameters

Symbols/Tables in our approach	Symbols in [5]
R_block	B
R_card_list	$\{\|R_n\|\}_{n=1}^N$
R_block_card_list	$\{\|B_n\|\}_{n=1}^N$
R_da_set	R_n
R_da_block_set	B_n
R_da_mass_set	$M_{B(a,n)}$
R_da_remove_set	D_i
R_da_res_block_set	$\tilde{B}_n$
relation_mass	M_R
block_mass	M_B
cur_density	ρ
max_density	$\tilde{\rho}$
r	r
best_r	$\tilde{r}$

Table 5: Symbols relationship

- **(T3.1, T3.2)**Use dummy column to prevent time consuming operations on large scale tables. We indicate whether to use this optimization as a boolean value *dummy_column* in our code. If this boolean value is false, we will use **'copy'** method, i.e., we will copy the whole input relation R into block B for each iteration and remove tuples from R after a dense block is found. If this boolean value is true, we will use **'mark'** method, which we also call 'dummy column' method, i.e., add a dummy column '_is_removed' to the input relation R to indicate whether the corresponding tuple is removed. The default value of '_is_removed' is 0, i.e., the corresponding tuple is still valid. Having added the dummy column, we can just take block B as all tuples with '_is_removed=0' in R, and after a dense block is found, we can just set '_is_removed=1' of corresponding tuples in R. Through this approach we won't have to actually remove the tuple from R. This 'mark' operation will be faster than actually removing tuples from R when the dataset size is very large, since the underlying data storage architecture will have to change if we remove tuples from R and it will take more time as data size grows. Experiments and results of this optimization method will also be discussed in Section 5.
- Create index on frequently used tables. A index will make selections and range queries much faster. But we need to trade off between index creation time and index saving time. We indicate whether to use this optimization as a boolean value *use_index* in our code. Details of the indexing optimization will be discussed in Section 5.

5 Experiments

5.1 Overview

We have done some experiments to evaluation our SQL version of D-Cube. Firstly, we conduct a sanity check on unit-tests examples, then we evaluate our implementation on darpa dataset and airforce dataset. We also try different optimization methods and conduct anomaly detection in real-world data.

5.2 Unit-tests

We have done a lot of unit-tests to evaluate the correctness of our implementation, and Table 6 shows a part of them. For each block, the format is "[dimensions], density". The retrieved dense blocks verify the correctness of our implementation: the first retrieved dense block corresponds to the highest density, e.g, [10-13,10-16,10-20] with density 0.99, and the second dense block corresponds to the second highest density, e.g., [0-2,0-5,0-8] with density 0.4, and so on.

We also test our implementation using a sampled darpa dataset with approximate 800 tuples, and the result verifies the correctness.

blocks
[0-50,0-50,0-50],0.001; [30-37,30-34,30-41],0.15; [0-2,0-5,0-8],0.4; [10-13,10-16,10-20],0.99
[0-50,0-50,0-50],0.001; [30-37,30-34,30-41],0.45; [10-13,10-16,10-20],0.99
[0-500,0-500,0-500],0.001; [300-307,300-304,300-311],0.5; [100-103,100-106,100-110],0.90
[0-500,0-500,0-500],0.000; [300-307,300-304,300-311],0.09; [100-103,100-106,100-110],0.99

Table 6: Unit-tests

5.3 Results on bucketized Darpa dataset (Phase 2)

We also evaluate our implementation using darpa dataset (darpa.csv). We first conduct some data preprocessing: bucketize time to hours, and eliminate duplicate tuples, after which some statistics of darpa.csv is shown in Table 7, in which "tuple" means "count(*)", "source" means "count(distinct source ip)" and the same for "dest" and "time".

source	dest	time	tuple
9484	23398	963	233840

Table 7: Statistics of preprocessed darpa dataset

The results for our implementation on preprocessed darpa dataset are shown in Table 8, where ”ari” stands for arithmetic average mass, ”geo” stands for geometric average mass, ”sus” stands for suspiciousness, ”dense” stands for density and ”card” stands for ”cardinality”.

ρ	dimension	source	dest	time	tuple	density
ari	dense	30	32	319	67988	535.33858268
geo	dense	24	24	435	78503	1245.22837068
sus	dense	38	161	961	120427	1064251.43173679
ari	card	80	145	406	106325	505.50713154
geo	card	80	145	442	111553	646.92584449
sus	card	80	145	629	117537	1010478.87524016

Table 8: Results of 1st block in darpa.csv

5.4 Optimization using indexing methods

We optimize our implementation using indexing methods, trying to trade off between index creation time and index saving time. Specifically, we will create index on some frequently used tables, i.e., *R_da_block_set*, *R_card_list*, *R_block_card_list*, and *R_da_order*.

Table 9 shows the running time to retrieve the first dense block in our bucktized darpa dataset for different indexing methods (and without indexing method). Note that these experiments are performed on a local machine, whose speed may not be as fast as a server.

From the results we can see indexing can speed up our algorithm by more than 30%, which is a huge improvement. Moreover, btree is slightly faster than hash in our implementation, so from now on we will use btree as default.

indexing method	running time(s)
btree	145.94244959
hash	146.24519579
no indexing	215.35953347

Table 9: Running time comparison for different indexing methods

5.5 Dummy column (T3.1, T3.2)

Apart from optimization in indexing methods, we also explore possible improvements by using dummy column, i.e., adding a dummy column to the input relation R to markup removed tuples, rather than actually removing tuples from R.

Table 10 shows the running time to retrieve the first dense block in darpa dataset conditioned on whether or not to use dummy column, and whether or not to use bucketized dataset. From the results we can see using dummy column can improve the time efficiency, and actually this improvement will get larger as the size of input relation R gets larger, since it would be much more time consuming for database to actually remove tuples from large scale dataset (e.g., millions of records). In other words, 'mark' method is faster than 'copy' method for large dataset. From now on, we will use dummy column method as default.

use dummy columns	bucketized	running time(s)
true	true	144.54784311
false	true	145.94244959
true	false	513.43721325
false	false	542.93354774

Table 10: Running time comparison for dummy columns

5.6 Results on Darpa Dataset (T4.1)

Table 11 shows the first five dense blocks retrieved by our algorithm with ρ_{ari} and maximum density policy for darpa dataset (without bucketize), in which # means 'number of distinct'. From the given label file we can conclude that all of the first five dense blocks contain only network attack tuples.

block	#source	#destination	#timestamp	tuple	density
1	2	1	39	231388	16527.71428571
2	7	2	115	667555	16137.31707317
3	2	1	39	209392	14887.82926829
4	3	2	24	428299	12621.82758621
5	2	1	15	63206	10534.33333333

Table 11: Results of wiki dataset

5.7 Results on Airforce Dataset (T4.1)

Table 12 shows the first five dense blocks retrieved by our algorithm with ρ_{ari} and maximum density policy for airforce dataset (without bucketize), in which # means 'number of distinct'. From the given label file we can conclude that all of the first five dense blocks contain network attack tuples with a large proportion.

block	#protocol	#service	#flag	#src	#des	#host	#srv	tuple	density
1	1	1	1	2	1	1	1	2263941	1980948.37500000
2	1	1	1	1	1	1	1	263295	263295.00000000
3	3	4	3	8	5	53	32	657792	42634.66666667
4	1	1	2	1	1	88	20	461141	28566.19469027
5	1	1	1	1	1	9	9	55835	16993.26086957

Table 12: Results of wiki dataset

5.8 Evaluation using ROC curve (T4.3)

We evaluated our implementation on AirForce TCP Dump and DARPA TCP Dump dataset with $\rho = \rho_{ari}$ and maximum density policy, using ROC curves and AUC.

As is shown, D-Cube gives a great performance in both datasets with AUC=0.956 in *darpa* (Figure 1) and AUC=0.936 in *airforce* (Figure 2) when we use the first 20 dense blocks detected(k=20). The blue dotted line is the benchmark as random guess.

However, the results may contain some duplicate items according to D-Cube algorithm, which may increase the curve slope to some extent and affect the value of AUC. We manually checked the number of duplicate items and found the number is so small that the duplicates have trivial effect on the value of AUC. Thus, the results and the figures with duplicates are credible.

5.9 Anomaly detection in real-world data (T5.1)

We apply our implementation to real-word multi-aspect datasets such as Amazon Review, Yelp Review and English Wikipedia Revision History, to find possible underlying dense blocks, in other words, anomaly behaviors.

5.9.1 Amazon Dataset

Table 13 shows the first five dense blocks retrieved by our algorithm with ρ_{ari} and maximum density policy for amazon dataset, in which # means 'number of distinct'. We can see from the results that all of the five blocks are indeed dense blocks since all of each user give a rating to all of each app with the same star at the same time. (i.e. $\#user \times \#app \times \#time \times \#star = tuple$). In other words, they are sub-tensors filled with data.

5.9.2 Yelp Dataset

Table 14 shows the first five dense blocks retrieved by our algorithm with ρ_{ari} and maximum density policy for yelp dataset, in which # means 'number of distinct'. We can see from the results that the first four blocks are indeed dense blocks since all of each user give a rating

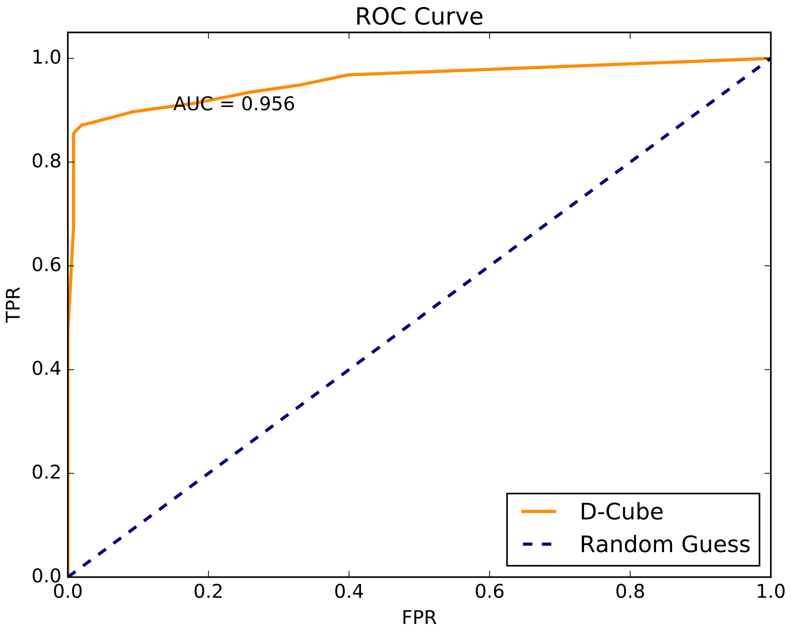

Figure 1: ROC Curve-Darpa

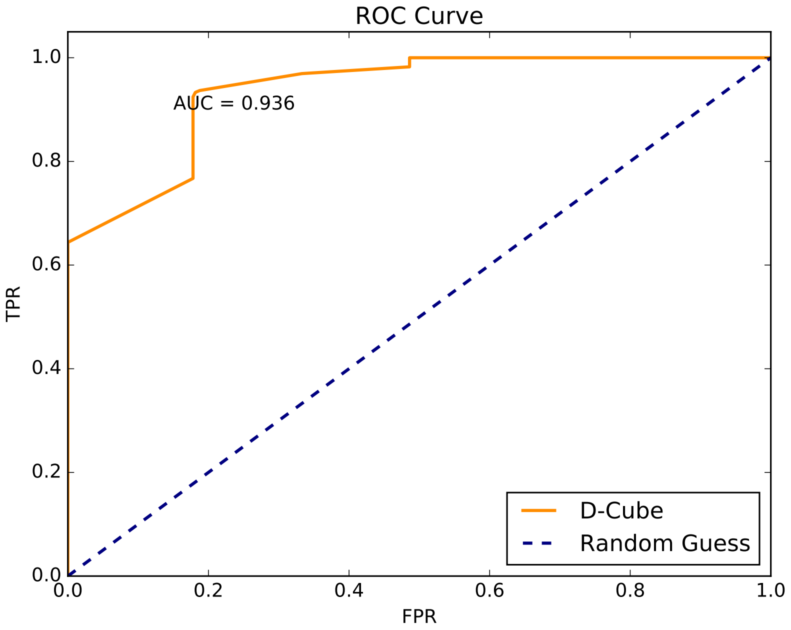

Figure 2: ROC Curve-Airforce

block	#user	#app	#time	#star	tuple	density
1	60	60	1	1	3600	118.03278689
2	55	55	1	1	3025	108.03571429
3	50	50	1	1	2500	98.03921569
4	45	45	1	1	2025	88.04347826
5	40	40	1	1	1600	78.04878049

Table 13: Results of yelp dataset

to all of each business with the same star at the same time. (i.e. $\#user \times \#business \times \#time \times \#star = tuple$). In other words, they are sub-tensors filled with data.

block	#user	#business	#time	#star	tuple	density
1	60	60	1	1	3600	118.03278689
2	55	55	1	1	3025	108.03571429
3	50	50	1	1	2500	98.03921569
4	45	45	1	1	2025	88.04347826
5	5321	4234	2780	5	260832	84.52106286

Table 14: Results of yelp dataset

5.9.3 Wiki Dataset

Table 15 shows the first five dense blocks retrieved by our algorithm with ρ_{geo} and maximum density policy for wiki dataset, in which # means 'number of distinct'. We can see from the results that block 1, 2, 3, 5 are likely to be dense blocks since #page is very small and tuple number is very large. Block 4 is also likely to be dense block since it has extremely large amount of tuples while only a limited number of #page.

block	#user	#page	#time	tuple	density
1	1	1	30	7756	2496.11188900
2	2	2	744	12693	882.44294992
3	2075	1	59	38588	777.13338381
4	329121	14	681	428299	287.90136906
5	6	4	744	18613	571.49574606

Table 15: Results of wiki dataset

6 Conclusions

From the experiments, we can see that our D-Cube implementation can successfully detect dense blocks in multi-aspect data.

We have successfully implemented D-Cube in SQL and optimized it from the algorithm design and with efficient indexing methods. We also have performed experiments on real-world datasets and delivered persuasive results to prove the effectiveness of D-Cube.

Our implementation of D-Cube is proved to be an effective algorithm to detect dense blocks in large-scale data. It has been designed well to minimize the frequency of accessing memory, which enables us to deal with such a large-scale dataset in the use of personal devices with limited computational power.

The optimization methods for D-Cube we implemented play important roles in the time performance. The 'Mark' method significantly improve the efficiency for large datasets, while the suitable indexing methods also lead to the reduction of running time.

The experiments on multiple real-world large datasets have given us confidence that D-Cube has a great performance of generalization in the detection of dense blocks.

Further research could be done to improve the D-Cube's performance of speed to a better level. The running time for a single dataset is still not good enough if you plan to find out many blocks.

References

[1] Meng Jiang, Alex Beutel, Peng Cui, Bryan Hooi, Shiqiang Yang, and Christos Faloutsos. A general suspiciousness metric for dense blocks in multimodal data. In *ICDM*. IEEE, 2015.

[2] U. Kang, Hanghang Tong, Jimeng Sun, Ching-Yung Lin, and Christos Faloutsos. Gbase: a scalable and general graph management system. In *KDD*, pages 1091–1099, 2011.

[3] U. Kang, Charalampos E. Tsourakakis, and Christos Faloutsos. Pegasus: A peta-scale graph mining system. In *ICDM*, pages 229–238, 2009.

[4] Kijung Shin, Bryan Hooi, and Christos Faloutsos. M-zoom: Fast dense-block detection in tensors with quality guarantees. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 264–280, 2016.

[5] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. D-cube: Dense-block detection in terabyte-scale tensors. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining*, WSDM '17. ACM, 2017.

[6] Alekh Jindal, Samuel Madden, Malu Castellanos, and Meichun Hsu. Graph analytics using the vertica relational database, 2014.

A Appendix

A.1 Plan for Activities (Phase 3)

The team performed the following tasks

- Summary papers of CrossSpot, GBase and Pegasus [Changkai Zhou]
- Summary papers of MZoom, DCube and Vertexica [Peilun Li]
- Analyze datasets [all]
- Find ways/guidances to implement D-Cube's overall structure using SQL [Peilun Li]
- Find ways/guidances to implement different density measures and dimension-selection policies using SQL [Changkai Zhou]
- Implement D-Cube algorithm using SQL [all]
- Try to optimize using indexing methods [Peilun Li]
- Try to optimize using view methods [Changkai Zhou]
- Evaluate on real-world datasets and draw ROC curves [all]
- Write reports [all]

A.2 Plan for Activities (Phase 2)

The team performed the following tasks

- Summary papers of CrossSpot, GBase and Pegasus [Changkai Zhou]
- Summary papers of MZoom, DCube and Vertexica [Peilun Li]
- Analyze datasets [all]
- Find ways/guidances to implement D-Cube's overall structure using SQL [Peilun Li]
- Find ways/guidances to implement different density measures and dimension-selection policies using SQL [Changkai Zhou]
- Implement D-Cube algorithm using SQL [all]

The team plans to perform the following tasks in next steps (plans are subject to change)

- Try to optimize using indexing methods [Peilun Li]
- Try to optimize using view methods [Changkai Zhou]
- Evaluate on real-world datasets and draw ROC curves [all]
- Write reports [all]

A.3 Plan for Activities (Phase 1)

The team performed the following tasks

- Summary papers of CrossSpot, GBase and Pegasus [Changkai Zhou]
- Summary papers of MZoom, DCube and Vertexica [Peilun Li]

The team plans to perform the following tasks in next steps (plans are subject to change)

- Analyze datasets [all]
- Find ways/guidances to implement D-Cube's overall structure using SQL [Peilun Li]
- Find ways/guidances to implement different density measures and dimension-selection policies using SQL [Changkai Zhou]
- Implement D-Cube algorithm using SQL [all]
- Try to optimize using indexing methods [Peilun Li]
- Try to optimize using view methods [Changkai Zhou]
- Evaluate on real-world datasets and draw ROC curves [all]
- Write reports [all]

A.4 Full disclosure wrt dissertations/projects

Changkai Zhou: He is not doing any project or dissertation related to this project.

Peilun Li: He is not doing any project or dissertation related to this project.

Contents

1 Introduction 1

2 Problem description 1

3 Survey 2

3.1 Papers read by Changkai Zhou . . . 2

3.2 Papers read by Peilun Li . . . 5

4 Implementation 9

4.1 Overview . . . 9

4.2 Database schema design . . . 9

4.3 Algorithm details . . . 9

4.4 Optimizations (Task 3) . . . 10

5 Experiments 12

5.1 Overview . . . 12

5.2 Unit-tests . . . 12

5.3 Results on bucketized Darpa dataset (Phase 2) . . . 12

5.4 Optimization using indexing methods . . . 13

5.5 Dummy column (T3.1, T3.2) . . . 13

5.6 Results on Darpa Dataset (T4.1) . . . 14

5.7 Results on Airforce Dataset (T4.1) . . . 14

5.8 Evaluation using ROC curve (T4.3) . . . 15

5.9 Anomaly detection in real-world data (T5.1) . . . 15

5.9.1 Amazon Dataset . . . 15

5.9.2 Yelp Dataset . . . 15

5.9.3 Wiki Dataset . . . 17

6 Conclusions 18

A Appendix 19

A.1 Plan for Activities (Phase 3) . . . 19

A.2 Plan for Activities (Phase 2) . . . 19

A.3 Plan for Activities (Phase 1) . . . 19

A.4 Full disclosure wrt dissertations/projects . . . 20