Seeing the Forest for the Trees: Tree-based Uncertain Frequent Pattern Mining

by

Richard Kyle MacKinnon

A thesis submitted to the Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada

November 2015

Thesis advisor

Carson K.-S. Leung

Author

Richard Kyle MacKinnon

Seeing the Forest for the Trees:
Tree-based Uncertain Frequent Pattern Mining

Abstract

Many frequent pattern mining algorithms operate on precise data, where each data point is an exact accounting of a phenomena (e.g., I have exactly two sisters). Alas, reasoning this way is a simplification for many real world observations. Measurements, predictions, environmental factors, human error, &ct. all introduce a degree of uncertainty into the mix. Tree-based frequent pattern mining algorithms such as FP-growth are particularly efficient due to their compact in-memory representations of the input database, but their uncertain extensions can require many more tree nodes. I propose new algorithms with tightened upper bounds to expected support, Tube-S and Tube-P, which mine frequent patterns from uncertain data. Extensive experimentation and analysis on datasets with different probability distributions are undertaken that show the tightness of my bounds in different situations.

Table of Contents

List of Tables

List of Figures

Publications

The following peer reviewed publications by the author contain material used in this thesis.

[1] Leung, C. K.-S. and MacKinnon, R. K., "BLIMP: A compact tree structure for uncertain frequent pattern mining," in *Data Warehousing and Knowledge Discovery*, ser. Lecture Notes in Computer Science, Bellatreche, L. and Mohania, M. K., Eds. Springer International Publishing, 2014, vol. 8646, pp. 115–123. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10160-6_11

[2] Leung, C. K.-S., MacKinnon, R. K., and Tanbeer, S. K., "Fast algorithms for frequent itemset mining from uncertain data," in *2014 IEEE International Conference on Data Mining (ICDM)*, Dec. 2014, pp. 893–898. [Online]. Available: http://dx.doi.org/10.1109/ICDM.2014.146

[3] ——, "Tightening upper bounds to the expected support for uncertain frequent pattern mining," *Procedia Computer Science*, vol. 35, pp. 328–337, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2014.08.113

[4] MacKinnon, R. K., Leung, C. K.-S., and Tanbeer, S. K., "A scalable data analytics algorithm for mining frequent patterns from uncertain data," in *Trends and Applications in Knowledge Discovery and Data Mining*, ser. Lecture Notes in Computer Science, Peng, W.-C., Wang, H., Bailey, J., Tseng, V. S., Ho, T. B., Zhou, Z.-H., and Chen, A. L., Eds. Springer International Publishing, 2014, vol. 8643, pp. 404–416. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-13186-3_37

[5] MacKinnon, R. K., Strauss, T. D., and Leung, C. K.-S., "DISC: Efficient uncertain frequent pattern mining with tightened upper bounds," in *2014 IEEE International Conference on Data Mining Workshop (ICDMW)*, Dec. 2014, pp. 1038–1045. [Online]. Available: http://dx.doi.org/10.1109/ICDMW.2014.129

Acknowledgements

"And chiefly Thou, O Spirit, that dost prefer
Before all Temples th' upright heart and pure,
Instruct me, for Thou know'st; Thou from the first
Wast present, and with mighty wings outspread
Dove-like satst brooding on the vast Abyss
And mad'st it pregnant: what in me is dark
Illumine, what is low raise and support;"

–John Milton, *Paradise Lost, Book I, 17–23*

On teaching...

In his role as advisor, I will be forever grateful to Dr. Carson K.-S. Leung for all he has done for me. He first introduced me to the subject of data mining as an undergraduate student and I have not looked back. Without his reminders and support I would not have had the amazing opportunities that I have had, and my situation would be quite different today. His feedback has been invaluable in the evolution of this work.

I would also like to acknowledge Dr. James E. Young for his help with scholarship applications, his support as a teacher, as well as his continued interest in my progress. They have been most appreciated. His good-natured tease when passing each other in the hallways of: "You're still here?" kept me on schedule and eager to report my progress anew each time we saw each other.

"And as, methinks, shall all,
Both great and small,
That ever lived on earth,
Early or late their birth,
Stranger and foe, one day each other know."

–Henry David Thoreau, *I Knew a Man By Sight, 26–30*

On friends...

My lab mates as well as a small group of graduate students from other labs have been an important part of my life these past two and a half years. Thank-you. I hope I've enriched your lives as much as you have mine.

> *"The best way to find yourself is to lose yourself in the service of others."*
> –Mahatma Gandhi

On my advisory committee...

Drs. Xikui Wang (王熙逵), Yang Wang (王洋), Carson K.-S. Leung and chair Peter C.J. Graham: your time and advice in this service does not go unappreciated. Thank-you for your selfless commitment.

> *"* ... *a grateful mind*
> *by owing owes not, but still pays, at once*
> *indebted and discharged; what burden then?"*
> –John Milton, *Paradise Lost, Book IV, 55–57*

On parents and family...

I owe you everything. Thank-you so much for being there for me throughout this process. Listening for the sake of listening, and understanding when I've been so focused on my work that at times it may have inconvenienced you. I love you.

Richard Kyle MacKinnon
B.C.Sc. (Honours), The University of Manitoba, Canada, 2013

The University of Manitoba
November 2015

—for my parents

Chapter 1

Introduction

People have been using and analyzing data for millennia, however the discipline of data mining as we know it today is a relatively recent one. In 1992, Frawley *et al.* [12] defined the task of *knowledge discovery from data (KDD)*, *data mining*, as "the nontrivial extraction of implicit, previously unknown, and potentially useful information from data". This general definition covers many specific areas of data mining such as clustering, classification, outlier detection and frequent pattern mining. In each of the above tasks the goal is to discover some hidden relationships that exist within the data. Taking the average finishing time among a group of female runners in a five kilometre race seems to fit the above definition at first glance, but fails the requirement of nontriviality. This work concerns itself with frequent pattern mining, specifically, algorithms for tree-based uncertain data frequent pattern mining. As such, frequent pattern mining can be defined as the nontrivial extraction of implicit, previously unknown, and potentially useful information, *in the form of frequent patterns*, from data.

1.1 Motivation of my MSc Thesis

Advances in the area of frequent pattern mining algorithms significantly affect other areas in data mining and machine learning. This is because frequent patterns represent a fundamental quality of a dataset and, as such, are useful for a variety of data processing and analysis tasks. One important motivation of my MSc thesis work comes from the fact that improvements to the basal problem will generate a ripple effect whereby numerous other fields will benefit. Another motivation comes from the ideal of reflecting the real world as accurately as possible in the calculations we use to reason about it. In order to truly model the real world with any data analysis or data mining task, the aspect of uncertainty needs to be considered. Mining of frequent patterns from uncertain data allows us to discover the implicit relationships between data that inform us and drive our decision-making processes. Faster, memory-efficient algorithms advance the state of the art in the data mining and related fields by allowing us to tackle ever-larger datasets with increasing frequency. Further, most real-life data is uncertain.

1.1.1 Applications of Frequent Pattern Mining

Frequent pattern mining is a preliminary step in many data mining and machine learning algorithms [1], but it was originally introduced to solve the problem of association rule mining, which seeks to enumerate rules observed in a data set if their occurrences exceed a confidence threshold. An example rule might be: deli meats, mayonnaise $\Rightarrow$ bread, which indicates that if a person buys deli meats and mayonnaise then they are also likely to buy bread in the same shopping trip. Since association

rules can be determined by simple application of a formula to already calculated values once all the frequent patterns have been discovered [4], finding frequent patterns in the first place is the more interesting and difficult problem.

The Apriori algorithm [4] was the first frequent pattern mining algorithm, introduced in 1994, for the discovery of frequent itemsets and association rules from shoppers' basket data. Some applications in this domain include targeted marketing, product arrangement and inventory management. For example, upon discovering that certain groups of items (e.g., chips and salsa) are frequently purchased together, a manager may decide to: display them side-by-side in their store both as a convenience to the customer and as a subliminal stimulus to drive further purchase; offer a discount on a low margin item to drive sales of a higher; or consolidate restocking orders for the related products to lower overhead costs.

Since then, techniques in many problem domains have been developed that use frequent pattern data as a first step in solving a more complex problem. In the biological domain, biclustering of DNA data is used to study gene regulation under different conditions [7]. By transforming gene microarray data into a transactional database format, initial bicluster seeds can be chosen by application of a frequent pattern mining algorithm [40]. As for clustering in general, it is possible to use frequent patterns as a similarity measure [37] such that the transactions in individual clusters will have many frequent patterns in common without having many frequent patterns in common with those of other clusters. Another algorithm that uses a similar approach but is tailored for website transaction data is GHIC (Greedy Hierarchical Itemset-based Clustering) [39].

Frequent patterns can also be used as input data to train classification models. The CBA (Classification Based on Associations) algorithm [28] uses a modified version of the Apriori algorithm [4] to generate Class Association Rules, from which a subset are used after further processing with training data to form the actual classifier model. Likewise, the CMAR (Classification based on Multiple Association Rules) algorithm [26] is based on the FP-growth [13] frequent pattern mining algorithm. Instead of generating a large set of association rules and then pruning them to select only those ones that contribute most favourably to the final classification model, HARMONY [35] directly mines the highest confidence rules.

In the area of outlier detection, the FP-Outlier algorithm [14] describes a method of identifying outlier transactions by finding those transactions that contain fewer frequent patterns among their items than other transactions. They also develop a method of comparing outlier transactions to frequent patterns based on the number of items in common between the two. If a frequent pattern has very few items in common with a transaction, its "contradictiveness" value will be higher than if a frequent pattern has many items in common with a transaction. This allows the FP-Outlier algorithm to not only discover outlier transactions but also to attempt to provide a reason for why those transactions might be outliers: that they have few items in common with other specific frequent patterns in the dataset.

Finally, aside from these more general areas, there are numerous examples of frequent pattern mining algorithms that have been adapted to more specific problems or types of input data. Mobile users' commerce-based web usage patterns can be mined to provide user specific recommendations for new stores or products [30]. Ra-

dio frequency tag arrays can be employed to monitor users activities using frequent trajectory mining [29]. Association rule mining can be applied to improve production operations processes, as Kamsu-Foguem *et al.* [17] show in a case study with a drilling products supplier. Frequent pattern mining can also be used to analyze social networks and find frequent following patterns [16] or leaders in the network [34].

1.1.2 Uncertain Data

Shoppers' basket data, such as discussed in the association rule mining example in Section 1.1.1, are examples of *precise data* –- the items on a customer's receipt exactly describe what they purchased. In recent years, research into frequent pattern mining has focused on the more general category of *uncertain data* [8, 9, 23, 24, 25, 27]. Uncertain data are everywhere. For example, consider data recorded via a measuring device. By its very nature any measuring device contains some uncertainty with regards to precision, the quality of the measurement and changes in the environment. Uncertainty can also be inherent in mathematical models, predictions and forecasts. Network latencies and delays can add uncertainty due to stale information which may have changed since it was first recorded. Additionally, uncertainty may be artificially introduced as a privacy preserving action [3]. Precise data can also be represented in an uncertain manner by considering it to have a certainty of 100%. In order to properly reason about the world this uncertainty must be considered.

1.2 Challenges

Dealing with uncertainty introduces new challenges not present when solely considering precise data. The amount of data storage required for both the raw data and in-memory data structures necessarily increases to handle the probability values associated with the uncertain data. Additionally, working with probability values requires more complex arithmetic operations, such as multiplication on floating point values, whereas with precise data algorithms integer addition and subtraction are sufficient. These problems consequently lead to runtime increases. Furthermore, the severity of the effect observed depends on the underlying probability distribution of the input data, which is the main difficulty of working with this type of data. Optimizations to uncertain data frequent pattern mining algorithms that bring them more in line with precise data algorithms are therefore desirable features.

The first attempt to handle uncertainty with a frequent pattern mining algorithm came in the form of the UApriori algorithm [9] in 2007. Both the original Apriori algorithm [4] and its uncertain adaptation mine frequent itemsets of increasing length by joining together candidates (potentially frequent itemsets) of smaller sizes before verifying their frequency against the original database. The major difference between the two is in the method of frequency counting. Where the Apriori algorithm only has to increment the frequency of an itemset by one for each occurrence of it in a transaction, the UApriori algorithm increments the frequency (hereafter referred to as expected support when dealing with uncertain data) of an itemset by the product of its items' probability values. When most probability values are low, this can lead to a negligible change in the expected support for a high computational cost. To

counteract this effect, various pruning strategies have been proposed that attempt to discard itemsets early, before the final expected support calculation [8, 9, 15].

Tree-based algorithms have some advantages over the Apriori algorithms discussed previously in both the precise and uncertain cases. Namely, they reduce the number of database scans required by keeping a compact tree-based representation of the database in memory (called the global tree) and they do not waste computation generating candidate itemsets, instead using the information already available in their global trees to directly perform frequency tests. For this reason, the tree-based precise data FP-growth algorithm [13] has been shown to be vastly superior to Apriori style algorithms, especially when certain optimizations are applied, such as in the Top Down FP-growth algorithm [36]. However, extending tree-based algorithms to operate on uncertain data introduces its own set of challenges.

The UF-growth algorithm [23] was proposed as a direct extension of FP-growth, but the necessary inclusion of item uncertainty in the tree structure increased the memory and time requirements over its precise data predecessor. FP-growth can offer a highly compact representation of the input database because it exhibits strong path sharing: if two transactions share a prefix (i.e., they contain the same first few items), instead of creating two separate paths in the global tree with duplicate nodes, FP-growth creates nodes for each shared item only once and increments a counter. When adding probability into the mix, UF-growth can only take advantage of path sharing if both the item and its probability value are the same, drastically reducing the benefit of path sharing observed in precise data tree-based algorithms.

Current algorithms reintroduce compactness in the global tree by storing less infor-

mation at the added cost of using indirect calculations — specifically, upper bounds on the expected support are used, generating some extra frequent patterns which must be pruned in a post-processing step. There is a constant battle between space savings eliminating calculation altogether and overhead from complexities introduced into the remaining calculations cancelling each other out. The biggest challenge in tree-based uncertain data frequent pattern mining algorithms is to match the ideal space and time requirements of FP-growth.

Based on these challenges, some research questions I investigate in this thesis are:

Question 1 Can the upper bound on expected support be further reduced beyond the current state-of-the-art (cf. CUF-growth* [24])?

Question 2 Can the resulting tree still be as compact as CUF-growth*?

Question 3 How would frequent itemsets be mined from such a tree?

Question 4 How would such an algorithm compare with previous uncertain data frequent pattern mining algorithms?

1.3 Thesis Statement

To address the research questions above, I propose tree-based frequent pattern mining algorithms that have the following qualities:

Quality 1 discover all frequent patterns from *uncertain data*,

Quality 2 demonstrate new approaches to calculating upper bounds on expected support,

Quality 3 implement data structures that facilitate mining using the upper bounds, and

Quality 4 offer competitive run-times while keeping memory requirements at a minimum.

1.4 My Contributions

1.4.1 Quality 1: Frequent Patterns

While it may seem redundant, the first quality on the *discovery of all frequent patterns from uncertain data* is explicitly stated in order to distinguish the algorithms proposed here from other uncertain data frequent pattern mining algorithms that do not discover all frequent patterns for a given expected support, for example: closed, maximal, constrained and top-k mining algorithms. These algorithms can be considered extensions of the algorithms proposed here since the types of patterns they mine are either subsets of or condensed representations of the entire set of frequent patterns. As such, trivial implementations of these algorithms can be conceived by first finding all the frequent patterns and then summarizing or trimming the results in some way.

1.4.2 Quality 2: Upper Bounds

One of the most important contributions addresses the second quality on *demonstrating new approaches to calculating upper bounds on expected support*. By its very nature, an upper bound on the expected support of an itemset X, $expSupUB(X)$, is

guaranteed to be greater than or equal to $expSup(X)$. From the perspective of information theory, it is guaranteed that any algorithm (tree-based or otherwise) which calculates $expSupUB$ for all possible combinations of itemsets X in a database $\mathcal{D}$ will contain enough information to describe the exact set of truly frequent patterns and possibly some false positives. Given that the input consists of a list of transactions and their associated existential probabilities, any upper bound on expected support must use those existential probabilities as the starting point for its calculation.

It is useful to consider the larger context of a data mining algorithm when designing an upper bound. For instance, in a database $\mathcal{D}$ of n transactions, the expected supports of two different itemsets X and Y within a particular transaction will contain some identical existential probabilities corresponding to their intersection, $X \cap Y$. An efficient mining algorithm would take advantage of the implicit relationships between different itemsets in a transaction to avoid redundant computation in the upper bound calculation. One such property that can be exploited to reduce redundant computation in the expected support is the downward closure or apriori property (cf. section 2.2), but the same is not necessarily true for any $expSupUB$ calculation.

In order to take advantage of the associated optimizations in a mining algorithm structured around the apriori property, the algorithms presented here exploit those implicit relationships among the itemsets in their upper bound calculations. Two different upper bound calculations are explored, with a mathematical definition given that unifies the two approaches under a common framework that can be reused for describing the behaviour of similar algorithms.

1.4.3 Quality 3: Data Structures

Closely related to the upper bound calculation is the logical model of the algorithm, which describes the data structures and memory layout. Referring to the third quality on *implementing data structures that facilitate mining using the upper bounds*, efficient data structures are important because they eliminate or hide any latencies related to memory accesses, allowing the algorithm to be dominated by the computation of the upper bound. Tree-based frequent pattern mining algorithms use a prefix-tree-based data structure that consists of a root node, along with multiple child nodes containing information from the input database. As transactions are read in, child nodes are added or updated for each item in the transaction.

One of the challenges of designing tree-based frequent pattern mining data structures is deciding what information to store in the nodes. There is a direct trade off between memory usage and runtime (cf. Section 1.2); more existential probabilities and frequencies potentially allow for very tight upper bounds, in which there is little difference between $expSupUB$ and $expSup$. This tightness comes at the cost of increased memory usage. However, tight upper bounds generate fewer false positives, leading to decreased mining and pruning times.

The algorithms presented here attempt to save space by storing less information in each node. A naïve implementation following this approach may lead to increased runtimes despite a tighter upper bound, which would ultimately be counterproductive. Instead, the focus of this work is to tighten the upper bounds as much as possible while considering the limited fidelity of information available in each node. As a result, each algorithm that is described uses only a single extra value in each node

compared to the CUF-growth [24] and PUF-growth [25] algorithms (and matches the number of values per node in the CUF-growth* [25] algorithm) while decreasing overall runtime and space usage.

1.4.4 Quality 4: Performance

The forth and final quality on *offering competitive run-times while keeping memory requirements at a minimum* is demonstrated through the analyses and evaluations conducted on the algorithms. In the literature, uncertain data frequent pattern mining algorithms are evaluated on their runtime, memory usage and scalability. These tests are carried out using both synthetic and real life databases commonly used in the field. To satisfy the competitive aspect of this statement, the proposed algorithms are compared with CUF-growth* [24]. PUF-growth [25] is not considered in the experimental evaluation because of its lack of a compounding value in the upper bound calculation for higher cardinality itemsets.

The structure and content of input databases can vary dramatically, affecting the logical and physical arrangement of data in the tree structures, and ultimately the performance of any algorithm. In the literature, synthetic datasets are often used to measure the performance and scalability of algorithms, however, these datasets are commonly generated using a uniform probability distribution, which does not necessarily yield an accurate representation of real life data.

A major contribution of this work is an extensive battery of factorial tests using synthetic databases with 5 different database sizes, 3 different average transactions lengths and probability values drawn from 15 different parameterizations of the Gaus-

sian distribution. In addition, real life databases are given probability values from the same 15 Gaussian distributions and tested as well. Runtime and memory usage are both evaluated with increasing sizes of input databases to directly measure scalability. The number of false positives generated for some of these tests are further measured and broken down into cardinality levels to help explain the results. I show that different probability distributions affect the results in different ways, highlighting the importance of considering this aspect of the input data.

1.5 Thesis Organization

The overall organization of this thesis is outlined in brief here with a short description of the major points presented in each chapter. In Chapter 2 I cover important background concepts and terminology. For example, frequency or support refers to the number of times a particular item occurs in all the transactions of a precise database, whereas expected support defines a similar measure adapted to uncertain databases. In moving from the simpler precise data concepts to uncertain data, I show the probability, set theory and intermediate mathematical development underpinning the concepts. The section finishes with a formal definition of the problem of frequent pattern mining from uncertain data.

Chapter 3 discusses related work. I organize the literature into candidate-generate-and-test algorithms and pattern growth algorithms, focusing on uncertain data algorithms in general and tree-based uncertain data algorithms specifically. For candidate-generate-and-test algorithms, UApriori [9] is covered with a worked through example. As the focus of my thesis is tree-based algorithms, pattern growth algorithms com-

prise the bulk of this section. UF-growth[23], UFP-growth [2], CUFP-growth [27], CUF-growth [24] and PUF-growth [25] are all discussed. A running example using the same inputs is shown for each of: UF-growth, CUF-growth and PUF-growth.

I propose two new algorithms, Tube-S and Tube-P, in Chapter 4, developing lemmas and proofs for the new upper bounds on expected support used by the two algorithms as well as providing detailed examples of the global tree construction and mining processes. For each algorithm, I show the tree node contents and describe the upper bound calculation for a single transaction; and then over an entire database. At each stage small examples using the calculations are provided, and large examples of the tree construction and mining algorithms follow. All the tree-based frequent pattern mining algorithms in Chapters 3 and 4 use the same uncertain database to allow for an easy side by side comparison between my algorithms and the previous work.

I conduct extensive experiments and analyses in Chapter 5. A theoretical analysis focuses on memory usage and runtime as it relates to the tightness of the upper bounds in Tube-S and Tube-P. In this analysis I also provide a table ranking my algorithms together with all the tree-based algorithms from Chapter 3 on their number of items per node and the tightness of their upper bound. Multiple experimental analyses follow to test the performance and memory usage of my algorithms for different input database characteristics such as: probability distribution; size; and average transaction length and number of domain items (collectively, density). Specifically, there are three sections focusing on runtime, memory usage (as total number of nodes) and number of false positives. The different database characteristics are covered

through synthetically generated data as well as real life databases.

Conclusions and potential areas of future work are discussed in Chapter 6. In my conclusions, each research question and quality enumerated above is touched upon briefly with reference to the specific section the reader can refer to for more detail. For future work, I consider several areas such as: Big data and distributed environments, the alternative probability model associated with probabilistic frequent patterns and different types of itemsets.

Finally, Appendices A and B contain raw data and graphs describing the input database characteristics and supplementary experimental results for the analyses in Chapter 5 that were too long to be included in the main body of the thesis.

Chapter 2

Background

Several important concepts underlie the whole of frequent pattern mining so I give the necessary formal definitions in the contexts of precise and uncertain data (support, expected support, downward closure, transaction and item caps and the possible worlds model), as well as define the task of frequent itemset mining itself.

2.1 Support

Let $\mathcal{I}$ be a set of m domain items and $X = \{x_1, x_2, \ldots, x_k\}$ be a k-itemset, where $X \subseteq \mathcal{I}$ and $1 \leq k \leq m$. Let $\mathcal{D}_p$ be a precise database of n transactions, each transaction $t_j \subseteq \mathcal{I}$. The *support* of an itemset X in a transaction t_j, $sup(X, t_j)$, is equal to 1 if $X \subseteq t_j$ or 0 if $X \not\subseteq t_j$. The *support* of an itemset X in an entire database $\mathcal{D}_p$ is the frequency with which it appears in the database [4]:

$$sup(X) = \sum_{j=1}^{n} sup(X, t_j) \tag{2.1}$$

Consider the transactional database in Table 2.1. The support of item a in t_1,

Table 2.1: A Transactional Database of Precise Data

TID	**Transaction**
1	$\{a, b, c, d, e, f\}$
2	$\{b, c, e, f\}$
3	$\{a, b, d, f\}$
4	$\{c, d, e, f\}$
5	$\{b, c, f\}$

$sup(a, t_1)$, is equal to 1 because $a \subseteq t_1$; likewise, $sup(\{c, e\}, t_2) = 1$. On the other hand, the support of itemset $\{b, c, f\}$ in t_3 is equal to 0 because $\{b, c, f\} \not\subseteq t_3$; likewise, $sup(d, t_5) = 0$. The support of each of the domain items over the entire database is: $(a, 2)$, $(b, 4)$, $(c, 4)$, $(d, 3)$, $(e, 3)$, $(f, 5)$.

2.2 Downward Closure

Sometimes referred to as the *apriori property*, downward closure is a fundamental concept in frequent pattern mining which states that *all proper subsets of a frequent pattern must themselves be frequent patterns.* This is based on the fact that any smaller pattern contained within a larger one must appear at least as many times (if not more) in an input database than the original, larger, pattern. The contrapositive: *all proper supersets of an infrequent pattern must themselves be infrequent patterns* also holds. The downward closure property is used in frequent pattern mining to prune infrequent patterns during the course of an algorithm, saving unnecessary calculations for patterns that are ultimately infrequent. If instead the frequency of every single pattern is calculated prior to eliminating any infrequent ones, the run-time of such an algorithm becomes exponential in the number of domain items.

2.3 Possible Worlds Probability Model

Moving to the realm of uncertain data, the concept of support as seen in precise databases is no longer sufficient. When we consider an attribute-based uncertainty model — where each item (or attribute) in a transaction is associated with a probability value — simply counting whether an item is there or not assumes the probability will always be 1, which is almost never the case (cf. the uncertain database in Table 2.2).

More formally, let $\mathcal{I}$ be a set of m domain items and $X = \{x_1, x_2, \ldots, x_k\}$ be a k-itemset, where $X \subseteq \mathcal{I}$ and $1 \leq k \leq m$. Let $\mathcal{D}_u$ be a database of n transactions, each transaction $t_j \subseteq \mathcal{I}$. Each item y_i in $t_j = \{y_1, y_2, \ldots, y_h\}$ in an uncertain database is associated with an *existential probability* $P(y_i, t_j)$ with value $0 < P(y_i, t_j) \leq 1$, where $P(y_i, t_j)$ represents the likelihood of the presence of y_i in t_j [19].

One can view the existential probabilities as delimiting multiple possible worlds. Consider a particular event, for example, a coin toss; there is a 50% chance of observing a heads outcome and a 50% chance of observing a tails outcome. If we asked a friend to flip a coin but to keep the result a secret, without knowing the result of the coin toss we can consider two possible worlds: one in which the toss *is* heads (with 100% probability) and one in which the toss *is* tails (again, with 100% probability). Only one of these possible worlds can actually be true (the knowledge of which only our friend is currently privy), but by considering them both separately we remove any uncertainty from the data and can use the concept of support from precise data to help us find frequent patterns. What we have done is effectively shifted the existential probability from describing the event itself to instead describe the probability that a

particular possible world *is* the true world.

Extending the example of a single coin flip to an entire uncertain database as in Table 2.2 requires only a slightly farther jump in abstraction. If an event (or transaction) involves more than one random variable (or attribute), then (assuming the existential probabilities are independent), every single combination of choices of $P(x_i, t_j)$ or $1 - P(x_i, t_j)$ for each x_i in each t_j denotes a possible world, yielding 2^q possible worlds, $q = \sum_{j=1}^{n} |t_j|$ for an uncertain database [19]. The product of each chosen $P(x_i, t_j)$ or $1 - P(x_i, t_j)$ gives the likelihood of that world being the true one.

2.4 Expected Support

Using the possible worlds probability model, it is now possible to individually apply the precise data concept of support to each possible world, taken together with the likelihood of each world being considered to be the true world, to come up with an *expected support*. The *expected support* of an itemset in an uncertain database is just the sum of products of supports for that itemset in each world and the likelihoods of each possible world to be the true world.

The *expected support* of an itemset X in an entire database $\mathcal{D}_u$ with W possible worlds is given by:

$$expSup(X) = \sum_{w \in W} \left(\sum_{j=1}^{n} sup(X, t_j) \times P(w \text{ is the true world}) \right) \tag{2.2}$$

This can be simplified, when the items within X are independent, to [19]:

$$
\begin{aligned}
&= \sum_{w \in W} \left(\sum_{j=1}^{n} sup(X, t_j) \times \prod_{j=1}^{n} \left(\prod_{x \in t_j \text{ in } w} P(x, t_j) \times \prod_{y \notin t_j \text{ in } w} (1 - P(y, t_j)) \right) \right) \\
&= \sum_{w \in W} \left(\sum_{j=1}^{n} sup(X, t_j) \times \ldots \right. \\
&\quad \left. \cdots \prod_{j=1}^{n} \left(\prod_{x \in (t_j \cap X) \text{ in } w} P(x, t_j) \times \left(\prod_{x \in (t_j - X) \text{ in } w} P(x, t_j) \times \prod_{y \notin t_j \text{ in } w} (1 - P(y, t_j)) \right) \right) \right) \\
&= \sum_{j=1}^{n} \prod_{x \in X} P(x, t_j)
\end{aligned}
$$

The terms in the inner parentheses can be eliminated in line 3 because, when considering all possible worlds, all combinations of probabilities for items not in X in a transaction are enumerated, cancelling each other out. If we define the expected support of an itemset in a single transaction as $\prod_{x \in X} P(x, t_j)$ then we can rewrite the expected support of an itemset X as a single summation:

$$expSup(X) = \sum_{j=1}^{n} expSup(X, t_j) \tag{2.3}$$

Consider the transactional database in Table 2.2. The support of item a in t_1,

Table 2.2: A Transactional Database of Uncertain Data

TID	**Transaction**
1	$\{a : 0.4, b : 0.6, c : 0.3, d : 0.2, e : 0.5, f : 0.9\}$
2	$\{b : 0.6, c : 0.2, e : 0.3, f : 0.2\}$
3	$\{a : 0.4, b : 1.0, d : 0.8, f : 0.1\}$
4	$\{c : 0.3, d : 0.6, e : 0.4, f : 0.9\}$
5	$\{b : 0.7, c : 0.5, f : 0.2\}$

$expSup(a, t_1)$, is equal to 0.4 because $P(a, t_1) = 0.4$; likewise, $expSup(\{c, e\}, t_2) = P(c, t_2) \times P(e, t_2) = 0.2 \times 0.3 = 0.06$. On the other hand, the expected support of

itemset $\{b, c, f\}$ in t_3 is equal to 0 because $\{b, c, f\} \not\subseteq t_3$; likewise, $expSup(d, t_5) = 0$. The expected support of each of the domain items over the entire database is: $(a, 0.8)$, $(b, 2.9)$, $(c, 1.3)$, $(d, 1.6)$, $(e, 1.2)$, $(f, 2.3)$.

2.5 Transaction and Item Caps

Important concepts used by both the CUF-growth algorithm [24] and the PUF-growth algorithm [25] to estimate expected support are those of *transaction caps* and *item caps* respectively. These caps give upper bounds on the expected support of an itemset.

CUF-growth defines the *transaction cap* of an transaction $t_j = \{y_1, \ldots, y_h\}$ as $T^{Cap}(t_j)$ [24]:

$$T^{Cap}(t_j) = \begin{cases} M_1(t_j) & \text{if } h = 1, \\ M_1(t_j) \times M_2(t_j) & \text{if } h \geq 2 \end{cases} \tag{2.4}$$

where $M_1(t_j) = \max_{q \in [1,h]} P(y_q, t_j)$ and $M_2(t_j) = \max_{r \in [1,h], r \neq q} P(y_r, t_j)$, the highest and second highest existential probability values respectively in t_j. This cap is used to estimate the support of any itemset X in t_j, regardless of whether or not the items used to calculate the *transaction cap* are actually in X. PUF-growth improves upon this in two ways: it requires that at least one of the items in X be used to calculate its *item cap* (which may now change depending on X), and instead of selecting the maximum existential probability values from the entire transaction, it now requires a global ordering of items so that valid items for the cap calculation can be restricted to a prefix of t_j.

PUF-growth defines the *item cap* of an itemset X in an "ordered" transaction $t_j =$

$\{y_1, \ldots, y_{r-1}, y_r, \ldots, y_h\}$, where $X = \{x_1, \ldots, x_k\} \subseteq t_j$ and $x_k = y_r$ as $I^{Cap}(X, t_j)$ [25]:

$$I^{Cap}(X, t_j) = \begin{cases} P(y_1, t_j) & \text{if } h = 1, \\ P(y_r, t_j) \times M_1(y_r, t_j) & \text{if } h \geq 2. \end{cases} \tag{2.5}$$

where $M_1(y_r, t_j) = \max_{1 \leq q < r} P(y_q, t_j)$ is the highest existential probability value among all $r - 1$ items in the proper "prefix" $\{y_1, \ldots, y_{r-1}\} \subset t_j$.

Consider the uncertain database with n=5 transactions and m=6 domain items as shown in Table 2.2. For a 4-itemset $X = \{a, b, c, d\}$ and $t_1 = \{a : 0.4, b : 0.6, c : 0.3, d : 0.2, e : 0.5, f : 0.9\}$, $expSup(X, t_1) = 0.4 \times 0.6 \times 0.3 \times 0.2 = 0.0144$, which is bounded above by its ***item cap***, $I^{Cap}(X, t_1) = 0.2 \times 0.6 = 0.12$, which is itself bounded above by its ***transaction cap***, $T^{Cap}(t_1) = 0.9 \times 0.6 = 0.54$.

2.6 Frequent Itemset Mining

In order to distinguish between frequent and non-frequent itemsets, we can either use an absolute, user defined, minimum support threshold, *minsup*, or attempt to separate itemsets based on some relative measure calculated from the input data. A user defined *minsup* allows more flexibility to adjust the mining algorithm to different input data; it is also easy for the user to understand the impact of a change to the *minsup* without first having to analyze the results (i.e., if the *minsup* goes up, itemsets must appear more frequently in the input database before they are deemed to exceed the threshold and considered "frequent", and vice-versa).

An itemset X is considered frequent in a precise database $\mathcal{D}_p$ if $sup(X) \geq minsup$. An itemset X is considered frequent in an uncertain database $\mathcal{D}_u$ if $expSup(X) \geq minsup$. Given a database $\mathcal{D}$ and a *minsup*, the research problem of frequent itemset

mining is to discover from the database a complete set of frequent itemsets having support or expected support $\geq$ *minsup*.

2.7 Summary

In order to talk about frequent patterns we must have some notion of frequency. Support and expected support capture this concept for precise and uncertain data respectively. For uncertain data, the possible worlds probability model allows us to transfer our existing precise data knowledge of support to uncertain data by counting the support of an itemset over all possible worlds suggested by the existential probabilities of all the items in the database. In terms of mining algorithms, downward closure is a property that can be exploited during the mining process to safely prune itemsets that are guaranteed to be infrequent without knowing their expected support. Upper bounds to expected support, such as item caps and transaction caps, satisfy the downward closure property in specific situations and can be used to prune transactions during a mining algorithm. CUF-growth [24] uses a transaction cap to bound the maximum possible expected support of any itemset in a transaction by multiplying the two highest existential probability values in the transaction. PUF-growth [25] uses an item cap instead of a transaction cap; the difference is that item caps exploit the fact that different itemsets may have different expected supports within the same transaction. When considering an itemset with a suffix item that has a low existential probability (i.e., it is not the highest or second highest in the transaction), PUF-growth uses that low probability to tighten the upper bound further than CUF-growth.

Chapter 3

Related Work

Frequent pattern mining algorithms can be broadly classified into *candidate-generate-and-test* and *pattern growth* algorithms. Pattern growth algorithms are more efficient since they directly generate candidate patterns whereas candidate-generate-and-test algorithms generate candidate patterns as well as some extra patterns that need to be discarded. The earliest frequent pattern mining algorithms are in the former category while the more recent ones are in the latter. The algorithms and examples explored in this chapter all handle uncertain data; some of the algorithms have a precise data equivalent, but the focus will be on their uncertain implementations.

3.1 Candidate-generate-and-test

The Apriori algorithm [4] was the first example of the *candidate-generate-and-test* approach as a means of discovering frequent patterns from precise data. Using the downward closure property (cf. Section 2.2), the Apriori algorithm mines fre-

quent itemsets of increasing length from precise data by taking multiple scans of an input database. More recent improvements to this algorithm include representing the transaction database as a coloured, undirected graph, which allows symmetries to be discovered and dynamically pruned from the search space [15]. UApriori [9], an uncertain extension of Apriori [4], is an example of a candidate-generate-and-test algorithm.

3.1.1 UApriori

The UApriori algorithm was developed by Chui *et al.* [9] from the basis of the Apriori algorithm [4] in order to handle uncertain data. During a first scan of the input database the expected support of every single domain item is counted, with the frequent 1-itemsets (i.e., those single items that appear frequently) forming the set L_1. Each subsequent scan is separated into two stages, a join stage and a pruning stage:

1. In the join stage the set C_k of candidate k-itemsets is formed by joining the items in L_{k-1}. From the apriori property we know that C_k is a superset of the frequent k-itemsets, L_k. As such, any k-itemset created by joining two itemsets in L_{k-1} for which not all of its subsets exists in L_{k-1} is immediately pruned.

2. In the pruning stage, the set of frequent k-itemsets, L_k, is formed by computing the expected support of every itemset in C_k during one scan of the input database and removing all infrequent itemsets.

This process continues until there are no itemsets in C_k. Candidate itemsets are stored in a hash tree to efficiently enumerate subsets in transactions while scanning

the database in the pruning stage. Consider Table 3.2 at the end of this section where the UApriori algorithm is used to find all the frequent itemsets that appear with a *minsup* of 0.7 in the uncertain database given in Table 3.1.

Because uncertain data deals with existential probabilities instead of frequencies, items with low existential probabilities contribute to minuscule increments in expected support during the pruning stage. These calculations take up processing time without significantly affecting the results. To remedy the situation, Chui *et al.* [9] proposed a data trimming framework and local trimming strategy that exclude those items with low existential probabilities from the input database. This has the benefit of reducing the amount of computation in the pruning stage. Moreover, the I/O cost is also reduced as the algorithm is now reading in a trimmed version of the database for each scan.

On the other hand, such a strategy incurs the added cost of keeping extra statistics on the maximum error in expected support due to the absence of the trimmed items, as well as separately maintaining a list of known frequent itemsets and potentially frequent itemsets. The potentially frequent itemsets need to be verified by one final database scan at the end of the algorithm. It should also be noted that the performance gain realized by the local trimming strategy is directly tied to choosing an optimal threshold value to cull low probability items. As the probability distribution of the input database tends towards uniformity, the optimal value of the trimming threshold becomes unclear.

In a later work, Chui and Kao [8] proposed a decremental pruning technique for their UApriori algorithm to address some of the issues with their local trimming

strategy. In particular, they integrated prefix decremental counters into the hash tree structure. These counters store upper bounds on the expected supports of prefix itemsets by initially assuming that the existential probabilities of their items are 100% in each transaction and then progressively lowering them as the algorithm scans the database and reads in the actual values. Once the expected support of a prefix itemset given by the prefix decremental counter drops below the minimum support threshold, that itemset along with all of its supersets are pruned and no longer considered in any further steps of the algorithm. The decremental pruning technique was shown to outperform the local trimming strategy when the percentage of items with low existential probabilities in the input database was lower than 50%. Additionally, the two approaches can be combined to achieve an overall best performance.

Table 3.1: A Transactional Database of Uncertain Data

TID	**Transaction**
1	$\{a : 0.57, b : 0.96, d : 0.97, f : 0.18\}$
2	$\{a : 0.96, c : 0.68, e : 0.04, f : 0.86\}$
3	$\{b : 0.04, c : 0.31, d : 0.06, e : 0.12\}$
4	$\{a : 0.78, b : 0.81, c : 0.21, e : 0.52, f : 0.48\}$
5	$\{a : 0.14, b : 0.53, d : 0.96, e : 0.37\}$
6	$\{b : 0.96, d : 0.57, e : 0.16, f : 0.17\}$
7	$\{b : 0.38, d : 0.22, f : 0.28\}$
8	$\{c : 0.32, d : 0.77, e : 0.77, f : 0.41\}$
9	$\{a : 0.59, d : 0.50, e : 0.01, f : 0.37\}$
10	$\{a : 0.29, b : 0.67, d : 0.71, e : 0.76\}$

Table 3.2: UApriori Example, $minsup = 0.7$

(a) C_1

Itemset	**Expected Support**
$\{a\}$	3.33
$\{b\}$	4.35
$\{c\}$	1.52
$\{d\}$	4.76
$\{e\}$	2.75
$\{f\}$	2.75

(b) L_1

Itemset	**Expected Support**
$\{a\}$	3.33
$\{b\}$	4.35
$\{c\}$	1.52
$\{d\}$	4.76
$\{e\}$	2.75
$\{f\}$	2.75

(c) C_2

Itemset	**Expected Support**
$\{a, b\}$	1.4475
$\{a, c\}$	0.8166
$\{a, d\}$	1.0702
$\{a, e\}$	0.7221
$\{a, f\}$	1.5209
$\{b, c\}$	0.1825
$\{b, d\}$	2.5489
$\{b, e\}$	1.2849
$\{b, f\}$	0.8312
$\{c, d\}$	0.2650
$\{c, e\}$	0.4200
$\{c, f\}$	0.8168
$\{d, e\}$	1.5911
$\{d, f\}$	0.8338
$\{e, f\}$	0.6306

(d) L_2

Itemset	**Expected Support**
$\{a, b\}$	1.4475
$\{a, c\}$	0.8166
$\{a, d\}$	1.0702
$\{a, e\}$	0.7221
$\{a, f\}$	1.5209
$\{b, d\}$	2.5489
$\{b, e\}$	1.2849
$\{b, f\}$	0.8312
$\{c, f\}$	0.8168
$\{d, e\}$	1.5911
$\{d, f\}$	0.8338

(e) C_3

Itemset	**Expected Support**
$\{a, b, d\}$	0.739969
$\{a, b, e\}$	0.503658
$\{a, b, f\}$	0.401760
$\{a, c, f\}$	0.640032
$\{a, d, e\}$	0.209162
$\{a, d, f\}$	0.208672
$\{b, d, e\}$	0.637628
$\{b, d, f\}$	0.226752

(f) L_3

Itemset	**Expected Support**
$\{a, b, d\}$	0.739969

3.2 Pattern-growth Algorithms

One of the major deficiencies of candidate-generate-and-test algorithms is the necessity of multiple scans of the entire input database each time higher level candidate itemsets are generated. A *pattern growth* approach, such as that used in tree-based algorithms, overcomes this limitation by reducing the number of database scans to two or three. Han *et al.* [13] introduced the first tree-based algorithm to combat this problem for precise data, called FP-growth. In addition, due to its tree-based data structures, FP-growth provides a high degree of compression over the input data. Non-tree-based pattern-growth algorithms have been developed as well. Pei *et al.* [32] proposed a hyperlinked structure based algorithm, H-mine, which also mines frequent patterns using a pattern growth approach. Since neither H-mine nor its uncertain extension UH-mine [2] perform any compression on the input data, their implementation is not discussed further. Instead, I focus on tree-based uncertain data extensions of FP-growth.

3.2.1 UF-growth

UF-growth was proposed by Leung *et al.* [23] as an adaptation of FP-growth [13] to the realm of uncertain data. A UF-tree is constructed in two scans of the input database in the same fashion as the FP-tree. However, each node of a UF-tree contains: an item ID, an existential probability, an occurrence count and a pointer to a sibling node, whereas an FP-tree does not contain an existential probability.

Nodes in a UF-tree are shared when multiple transactions contain the same pairs of items and existential probabilities for a particular item and its ancestors in its prefix

path. Once the global UF-tree is built, all of the frequent patterns can be mined directly from the tree in the same fashion as FP-growth, however, the existential probability values stored in the nodes are multiplied at each stage in the recursion. The recursion stops when the existential probability values become lower than the minimum support threshold or the tree contains only a single node. Tree based algorithms are also called pattern growth algorithms as the frequent patterns grow in size at each level of the recursion.

Due to the increased memory requirements of the UF-tree over the FP-tree imposed by stricter path sharing conditions, Leung *et al.* [23] also proposed two optimizations: discretizing and rounding of the expected support values up to k decimal places, and limiting the maximum depth of recursion to two. In the latter case, instead of recursively growing the patterns, all patterns in L_3 or higher are obtained by traversing the tree and enumerating all subsets of the path extensions. UF-growth was shown to outperform the original UApriori, however, it was not initially compared to UApriori augmented with the decremental pruning technique as both papers were published simultaneously.

In a later work, Tong *et al.* [33] preformed a comprehensive comparison of many uncertain frequent pattern mining algorithms. They showed that UApriori with the decremental pruning technique outperformed UF-growth in a variety of situations, with the main reason being that UF-growth spent too much time on redundant computation recursively constructing sub-trees. It is not clear from their work whether they employed the optimizations mentioned in the original UF-growth paper during their comparisons, in particular, the one limiting the maximum recursion depth.

This optimization would seem to directly address their concerns, making its lack of mention questionable.

UF-growth Example

For an example of UF-tree construction, consider the following diagrams (Figure 3.1 and Figure 3.2) and tables (Table 3.3, 3.4 and 3.5). Given the initial table, a single scan through the database is required to calculate the expected support of each singleton itemset and remove infrequent items. These expected supports are shown in Table 3.4. After removing these items from consideration, Table 3.5 shows the remaining items in the uncertain database from which the global UF-tree can be constructed.

Table 3.3: A Transactional Database of Uncertain Data

TID	**Transaction**
1	$\{a:0.44, b:0.38, c:0.05, d:0.43, e:0.61, f:0.47\}$
2	$\{a:0.44, b:0.68, c:0.69, d:0.81, e:0.54, f:0.23\}$
3	$\{a:0.26, b:0.70, c:0.50, d:0.68, e:0.81, f:0.37\}$
4	$\{a:0.65, b:0.44, c:0.57, e:0.68, f:0.54\}$
5	$\{b:0.22, c:0.48, d:0.84, e:0.89, f:0.81, g:0.35\}$
6	$\{a:0.50, c:0.79, d:0.67, e:0.68\}$
7	$\{b:0.98, c:0.71, d:0.82, e:0.96, f:0.93\}$
8	$\{a:0.64, b:0.64, c:0.94, d:0.81, e:0.93\}$
9	$\{b:0.79, c:0.56, d:0.79, e:0.35, f:0.09, g:0.21\}$
10	$\{b:0.57, c:0.42, d:0.43, e:0.93, f:0.26\}$

Table 3.4: Items and their Expected Supports ($minsup = 2$)

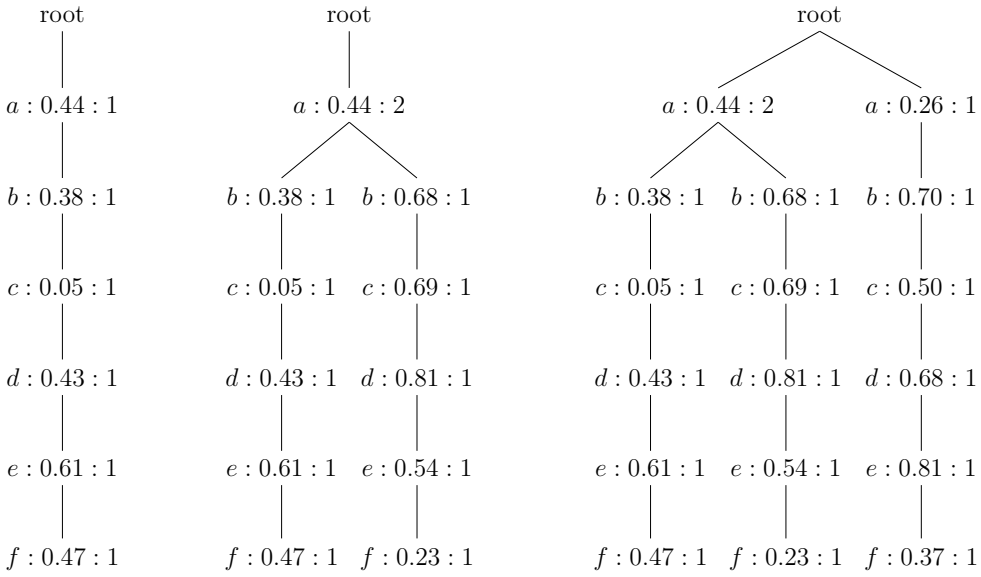
Item	**Expected Support**
a	2.93
b	5.40
c	5.71
d	6.28
e	7.38
f	3.70
g	0.56

Table 3.5: Table 3.4 with Infrequent Items Removed ($minsup = 2$)

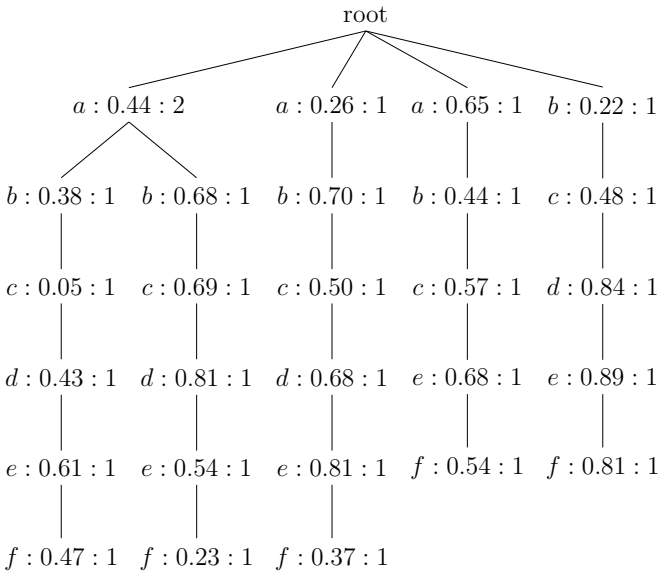
TID	**Transaction**
1	$\{a : 0.44, b : 0.38, c : 0.05, d : 0.43, e : 0.61, f : 0.47\}$
2	$\{a : 0.44, b : 0.68, c : 0.69, d : 0.81, e : 0.54, f : 0.23\}$
3	$\{a : 0.26, b : 0.70, c : 0.50, d : 0.68, e : 0.81, f : 0.37\}$
4	$\{a : 0.65, b : 0.44, c : 0.57, e : 0.68, f : 0.54\}$
5	$\{b : 0.22, c : 0.48, d : 0.84, e : 0.89, f : 0.81\}$
6	$\{a : 0.50, c : 0.79, d : 0.67, e : 0.68\}$
7	$\{b : 0.98, c : 0.71, d : 0.82, e : 0.96, f : 0.93\}$
8	$\{a : 0.64, b : 0.64, c : 0.94, d : 0.81, e : 0.93\}$
9	$\{b : 0.79, c : 0.56, d : 0.79, e : 0.35, f : 0.09\}$
10	$\{b : 0.57, c : 0.42, d : 0.43, e : 0.93, f : 0.26\}$

Adding the first transaction (cf. Figure 3.1(a)) creates a new node for each item in the transaction as a child of the previous node. Adding the second transaction (cf. Figure 3.1(b)) uses the same a node since both t_1 and t_2 have $a : 0.44$ as their first item, but increments its frequency counter by one. After that, new child nodes are created for the rest of t_2 since the existential probability values for those items are not the same as those in t_1. Adding t_3 (cf. Figure 3.1(c)) creates a new path from the root for each node as its first item, $a : 0.26$, has a different existential probability from the first item in the other path ($a : 0.44$). Every other transaction in Table 3.5 receives

a new path as well since their prefixes differ in either item, existential probability or both. The partial UF-tree after adding t_5 is shown in Figure 3.1(d). The final UF-tree is shown in Figure 3.2. Note that, as in FP-growth [13], horizontal node traversal pointers are maintained between nodes with the same item and are accessible through a header table. They are not shown in the figures for simplicity.



(a) Adding t_1 (b) Adding t_2 (c) Adding t_3



(d) Adding t_4 and t_5

Figure 3.1: UF-tree Construction, $minsup = 2$

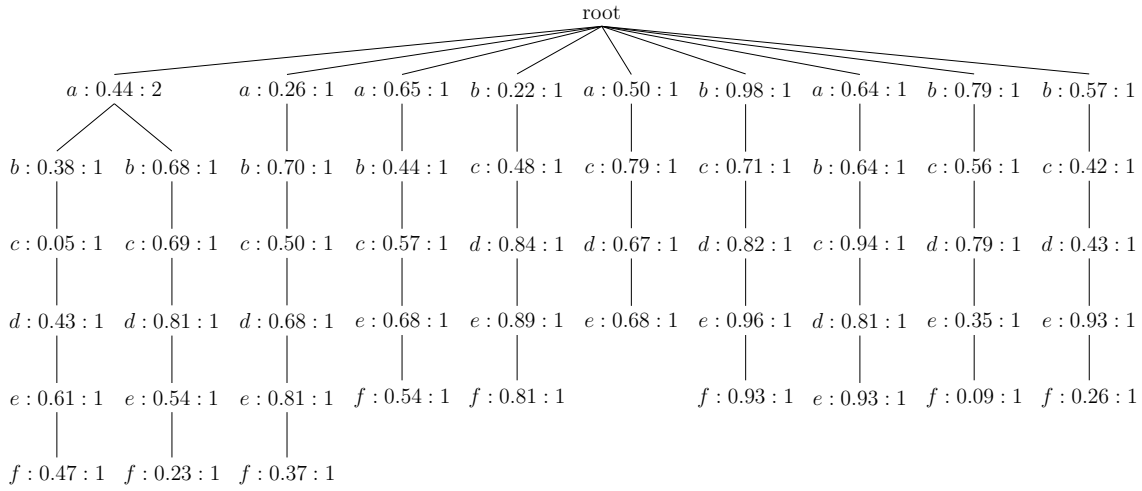


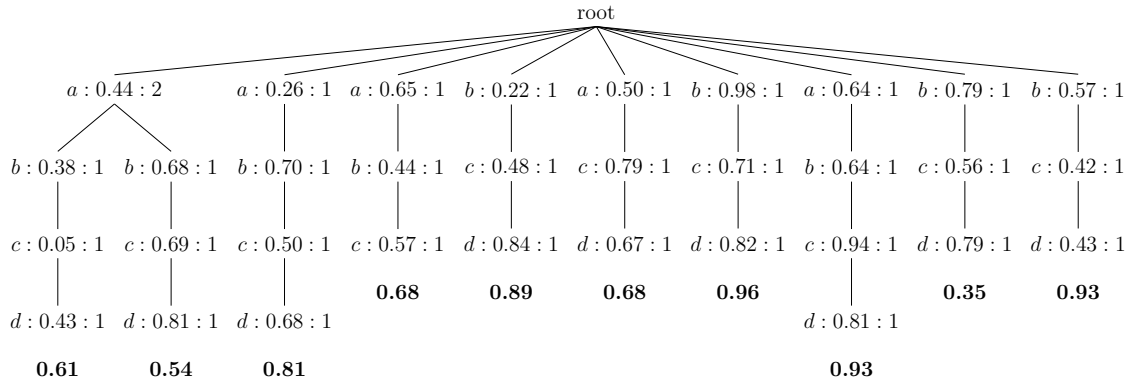
Figure 3.2: The Full UF-tree, $minsup = 2$

In order to mine frequent patterns from a UF-tree, UF-growth extracts tree paths using a recursive algorithm, extending patterns upwards to the root while keeping track of the expected support. Starting from the global tree in Figure 3.2 UF-growth calculates conditional pattern bases for each item. The $\{e\}$-conditional pattern bases are illustrated in the tree shown in Figure 3.3 by following all the branches with an e in them up to the root while keeping track of the expected support of $\{e\}$. As it does this, UF-growth checks the expected support of all two itemsets containing e and some other item in the tree by multiplying the existential probability of e for each branch with the frequency and existential probability of the other item, adding this to a running total for each branch. For example, to calculate the expected support of $\{e, d\}$, the first branch contributes $1 \times 0.61 \times 0.43 = 0.2623$; the second branch contributes $1 \times 0.54 \times 0.81 = 0.4374$; the third branch contributes $1 \times 0.81 \times 0.68 = 0.5508$; the forth branch is skipped because d does not appear; the fifth

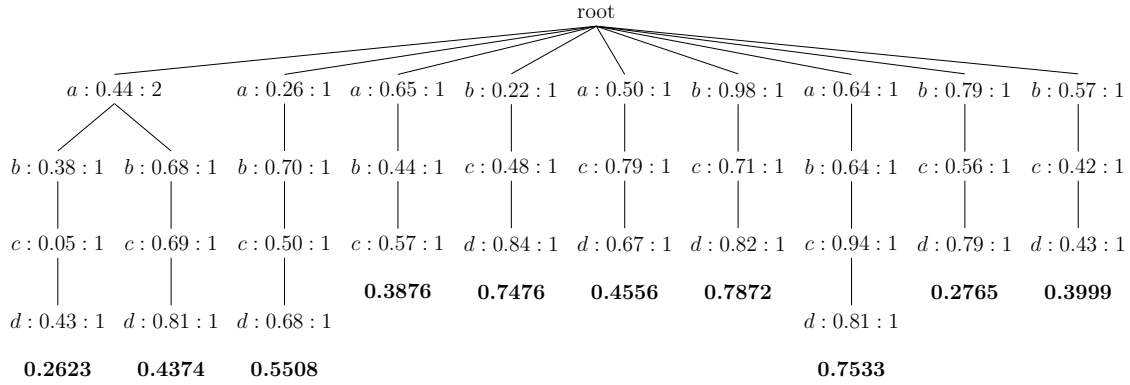
branch contributes $1 \times 0.89 \times 0.84 = 0.7476$; and so on until the total expected support of $expSup(\{e, d\}) = 4.6706$. The expected supports of the other 2-itemsets calculated from the $\{e\}$-conditional pattern bases are: $\{e, c\} : 4.3025$, $\{e, b\} : 4.0036$, $\{e, a\} : 2.0938$. Since every 2-itemset containing $\{e\}$ is frequent, no nodes need to be removed from the conditional pattern bases and the final $\{e\}$-projected tree is as shown in Figure 3.3 (note that expected supports are only shown for the leaf nodes here but they have in fact been calculated for each node while the pattern bases were scanned).

The same process is recursively applied to calculate the $\{e, d\}$-projected UF-tree. The $\{e, d\}$-conditional pattern bases are given in Figure 3.4(a). From the pattern bases, the expected supports of all 3-itemsets containing $\{e, d\}$ are: $\{e, d, c\} : 2.89891$, $\{e, d, b\} : 2.64708$ and $\{e, d, a\} : 1.160988$. Since $expSup(\{e, d, a\}) < minsup$, a is removed from the $\{e, d\}$ conditional pattern bases and does not appear in the $\{e, d\}$-projected tree. The $\{e, d\}$-projected tree is shown in Figure 3.4(b).

The $\{e, d, c\}$-conditional pattern bases are given in Figure 3.4(c). From the pattern bases, the expected support of all 4-itemsets containing $\{e, d, c\}$, namely $\{e, d, c, b\}$, is $1.70091704 < minsup = 2$. Since no frequent four itemsets can be found in the conditional pattern bases and since the conditional pattern bases consist of only a single level, the $\{e, d, c\}$-projected UF-tree is never built. The recursion halts here and the algorithm continues by examining the $\{e, c\}$-conditional pattern bases. Eventually, all conditional pattern bases for each item or set of items are examined and projected trees created as necessary. The final set of frequent itemsets and expected supports is as in Table 3.6.

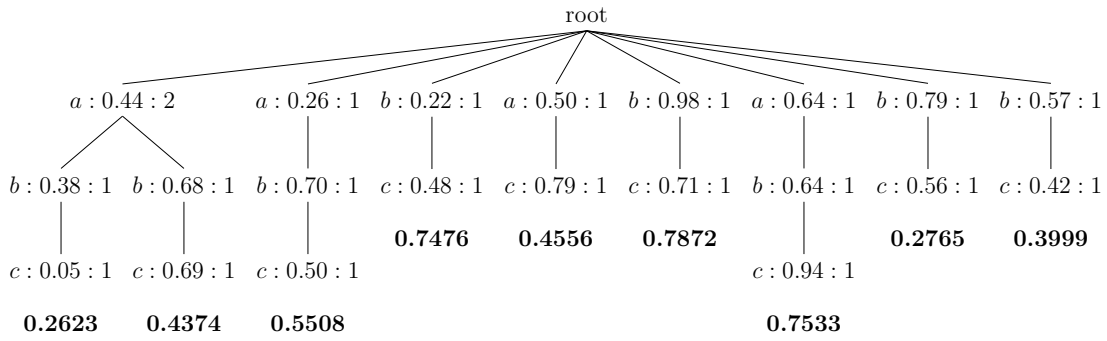


(a) The $\{e\}$-conditional Pattern Bases of the UF-tree

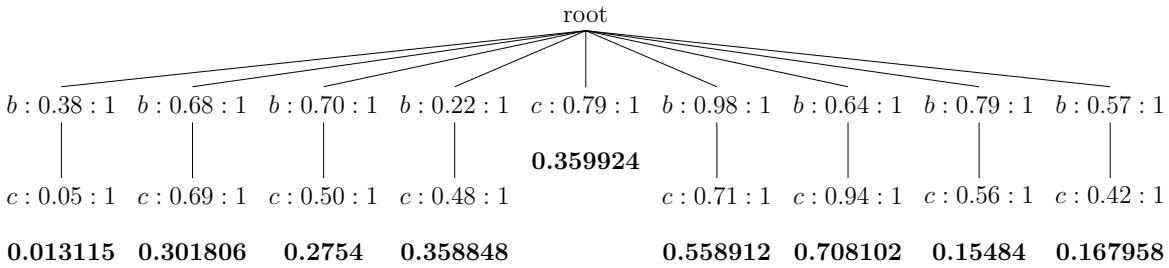


(b) The $\{e\}$-projected UF-tree

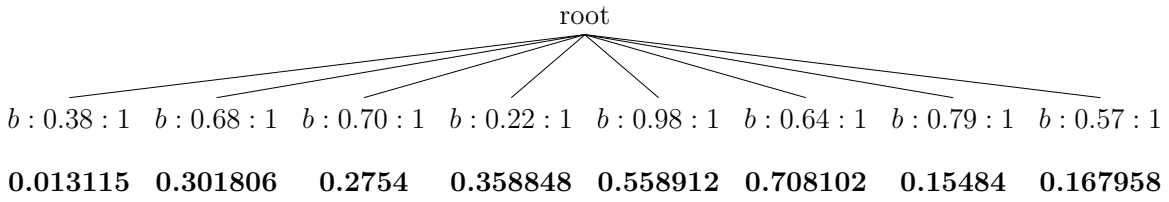
Figure 3.3: Creation of the $\{e\}$-projected UF-tree, $minsup = 2$



(a) The $\{e, d\}$-conditional Pattern Bases of the UF-tree



(b) The $\{e, d\}$-projected UF-tree



(c) The $\{e, d, c\}$-conditional Pattern Bases of the UF-tree

Figure 3.4: Creation of the $\{e, d\}$-projected UF-tree and $\{e, d, c\}$-conditional Pattern Bases, $minsup = 2$

Table 3.6: Frequent itemsets and their Expected Supports

Frequent Itemset	**Expected Support**
$\{a\}$	2.93
$\{a, e\}$	2.0938
$\{b\}$	5.4
$\{b, c\}$	3.1738
$\{b, c, d\}$	2.22522
$\{b, c, e\}$	2.41792
$\{b, d\}$	3.5662
$\{b, d, e\}$	2.64708
$\{b, e\}$	4.0036
$\{b, f\}$	2.1405
$\{c\}$	5.71
$\{c, d\}$	3.8195
$\{c, d, e\}$	2.89891
$\{c, e\}$	4.3025
$\{d\}$	6.28
$\{d, e\}$	4.6706
$\{d, f\}$	2.2659
$\{e\}$	7.38
$\{e, f\}$	2.9648
$\{f\}$	3.7

3.2.2 UFP-growth

The UFP-growth algorithm proposed by Aggarwal *et al.* [2] attempts to address the memory issue of UF-growth by using the same path sharing condition as the original FP-tree (paths are shared if the item is the same regardless of existential probability). UFP-growth uses a modified version of the k-means clustering algorithm to store pairs of ⟨maximum existential probability in the cluster, number of items in the cluster⟩. Using these maximum probabilities, an upper bound on the expected support can be calculated, with all potentially frequent itemsets generated in this manner verified by a third database scan at the end of the algorithm. More specifically,

the upper bound on the expected support of an itemset X in UFP-growth is calculated by multiplying n values for each tree node, where n is the number of probabilities captured in the last node of the tree path for the itemset X. The n values for each node are chosen in groups giving precedence to the highest probability values and the minimum cluster size for that node at a time. If X appears in multiple tree branches than the contribution from each branch is summed.

The upper bound becomes tighter as the size of the UFP-tree nodes increase, and can be adjusted based on the clustering parameter k. Choosing an appropriate value for k needs to be done carefully to obtain the best benefits of compression but not incur too high a cost in false positives. In the worst case, this algorithm reduces to UF-growth (where each cluster only contains a single probability value). On the other hand, the upper bound favours higher probability values in parent nodes, so if the cluster sizes are large, the upper bound may not be so tight either. Experimental results showed that UFP-growth was slower than UApriori on multiple datasets due to extra time spent eliminating false positives (itemsets pruned during the third scan). Although Aggarwal *et al.* cited the UF-growth paper written by Leung *et al.*, they did not compare the two algorithms.

3.2.3 CUFP-growth

Lin and Hong [27] presented the CUFP-growth algorithm in another attempt to tackle the memory issue of large UF-trees. CUFP-trees use the same path sharing condition of FP-growth without calculating an upper bound on expected support. In order to achieve this, each node in a CUFP-tree stores not only the expected support

of the singleton itemset, but also the partial expected supports of every superset formed from the current node and the nodes in its prefix. Thus, each node at depth d stores an expected support and an array of size $2^d - 1$ of expected supports for the supersets of the item in that node. Due to the exact solution nature of this algorithm and the smaller number of nodes compared to UF-growth it was reported to outperform UF-growth in terms of execution time and number of nodes, however, no work was done to compare the actual memory usage. This is a significant oversight considering the size of the superset array in each node grows exponentially with the depth of the tree.

3.2.4 CUF-growth

Further work on reducing the memory requirements for tree based uncertain frequent pattern mining algorithms was put forward by Leung and Tanbeer [24] with the CUF-growth algorithm. CUF-growth achieves the same number of nodes as FP-growth by relaxing the path sharing conditions found in UF-growth. Similar to UFP-growth, it also calculates an upper bound on expected support, denoted $expSup^{Cap}$. The $expSup^{Cap}$ of an itemset is calculated as the sum of all the *transaction caps* in which that itemset occurs, and the transaction cap is calculated as the product of the two highest existential probabilities of a given transaction. This transaction cap is the only other value other than the item ID stored in each node of a CUF-tree. Therefore, not only is the number of nodes the same between the CUF-tree and the FP-tree, but the number of items *within* each node is also equal (cf. UF-growth [23], which stores the item ID, existential probability and a frequency count). CUF-growth mines

frequent itemsets in the same fashion as FP-growth, but due to the upper bound it requires a third database scan to eliminate any false positives. For a detailed example see below. Experimental results reported that CUF-growth was faster than both UFP-growth and UH-mine, as well as having less than half the false positives of UFP-growth.

CUF-growth Example

For an example of CUF-tree construction, consider the following diagrams: Figures 3.5 and 3.6; and Tables 3.3, 3.4 and 3.7. Given the initial table, a single scan through the database is required to calculate the expected support of each singleton itemset and remove infrequent items. These expected supports are shown in Table 3.4. After removing these items from consideration, Table 3.5 shows the remaining items in the uncertain database from which the global CUF-tree can be constructed.

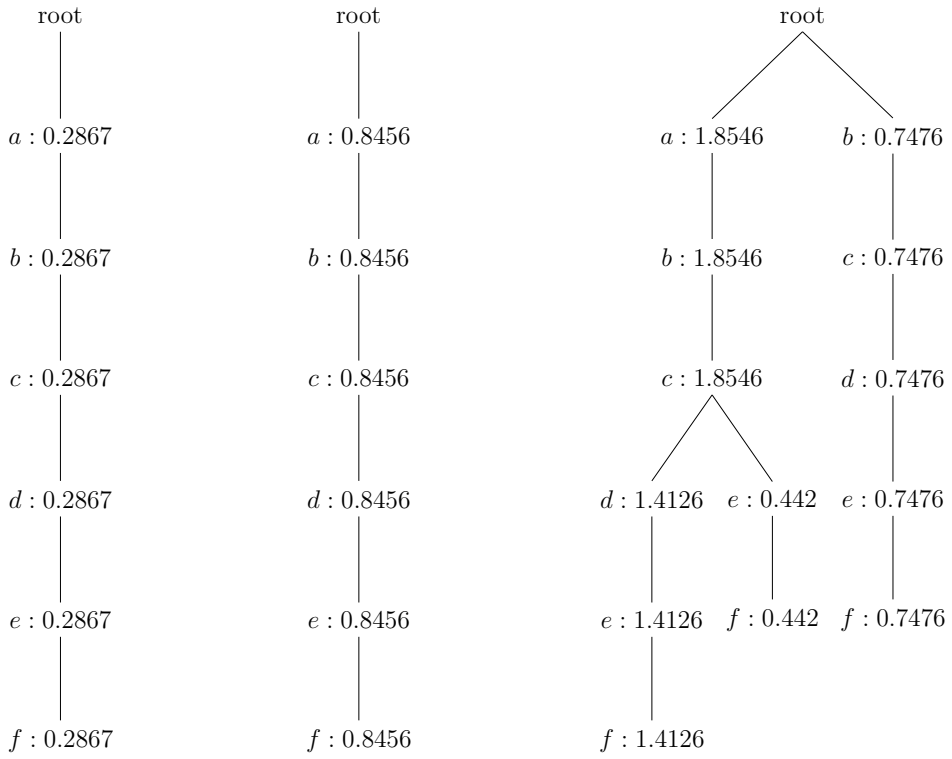
As transactions are being inserted into the tree, transaction caps are calculated based on the remaining frequent items in each transaction. For a transaction t_j, its transaction cap is calculated by multiplying the two highest existential probability values in that transaction. For example, consider t_2; its T^{Cap} is calculated by multiplying the two highest existential probability values in that transaction, $P(c, t_2) \times P(d, t_2) = 0.69 \times 0.81 = 0.5589$. These transaction caps are computed on the fly during the second database scan but are summarized in Table 3.7.

Table 3.7: Transaction Cap Calculations for Table 3.3 ($minsup = 2$)

TID	**Transaction**	T^{Cap} Calculation
1	$\{a : 0.44, b : 0.38, c : 0.05, d : 0.43, e : 0.61, f : 0.47\}$	$0.61 \times 0.47 = 0.2867$
2	$\{a : 0.44, b : 0.68, c : 0.69, d : 0.81, e : 0.54, f : 0.23\}$	$0.81 \times 0.69 = 0.5589$
3	$\{a : 0.26, b : 0.70, c : 0.50, d : 0.68, e : 0.81, f : 0.37\}$	$0.81 \times 0.70 = 0.567$
4	$\{a : 0.65, b : 0.44, c : 0.57, e : 0.68, f : 0.54\}$	$0.68 \times 0.65 = 0.442$
5	$\{b : 0.22, c : 0.48, d : 0.84, e : 0.89, f : 0.81\}$	$0.89 \times 0.84 = 0.7476$
6	$\{a : 0.50, c : 0.79, d : 0.67, e : 0.68\}$	$0.79 \times 0.68 = 0.5372$
7	$\{b : 0.98, c : 0.71, d : 0.82, e : 0.96, f : 0.93\}$	$0.98 \times 0.96 = 0.9408$
8	$\{a : 0.64, b : 0.64, c : 0.94, d : 0.81, e : 0.93\}$	$0.94 \times 0.93 = 0.8742$
9	$\{b : 0.79, c : 0.56, d : 0.79, e : 0.35, f : 0.09\}$	$0.79 \times 0.79 = 0.6241$
10	$\{b : 0.57, c : 0.42, d : 0.43, e : 0.93, f : 0.26\}$	$0.93 \times 0.57 = 0.5301$

Adding the first transaction (cf. Figure 3.5(a)) creates a new node for each item in the transaction as a child of the previous node. However, unlike in UF-growth, the transaction cap is stored in place of the existential support and frequency. Adding the second transaction (cf. Figure 3.5(b)) doesn't create any new nodes since the items in t_2 are exactly the same as the items in t_1. Instead, the transaction caps in each node are updated by summing together the transaction cap of t_2 with their previous value. This summation imparts frequency information, obviating the need to store this separately as in UF-growth. When t_4 and t_5 are added to the tree, some items in those transactions require new nodes to be created as they do not share common prefix items (regardless of their existential probability) with the previously added transactions. New branches are added for $\{e, f\}$ in t_4 and all of t_5 as shown in Figure 3.5(c). The final CUF-tree is shown in Figure 3.6(b). Note that, as in FP-growth [13], horizontal node traversal pointers are maintained between nodes with the same item and are accessible through a header table. Additionally, in CUF-growth the header table keeps track of the total transaction cap value for a given item across

all tree branches. The header table for the global CUF-tree in this example is given in Figure 3.6(a).

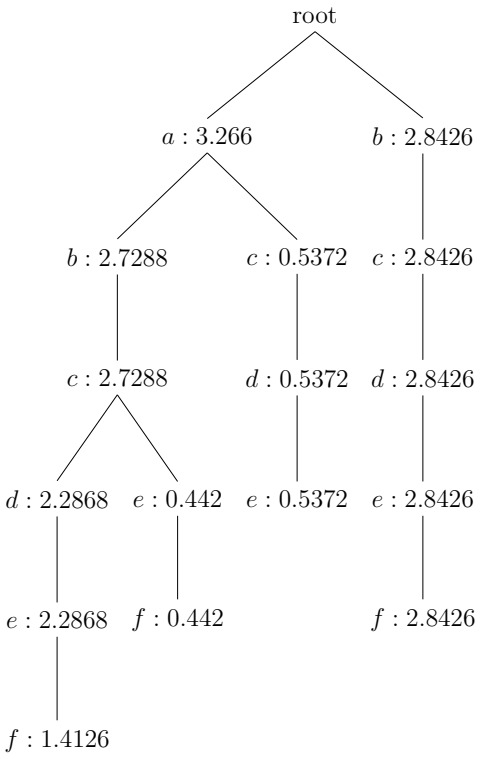


(a) Adding t_1 (b) Adding t_2 (c) Adding t_3, t_4 and t_5

Figure 3.5: CUF-tree Construction, $minsup = 2$

Item	**Σ Transaction Cap**	**Pointer to CUF-tree Node, e.g.**
a	3.266	0x100400270
b	5.5714	0x1004004a0
c	6.1086	0x1004005d0
d	5.6666	0x100400580
e	6.1086	0x100400600
f	4.6972	0x100400540

(a) CUF-tree Header Table



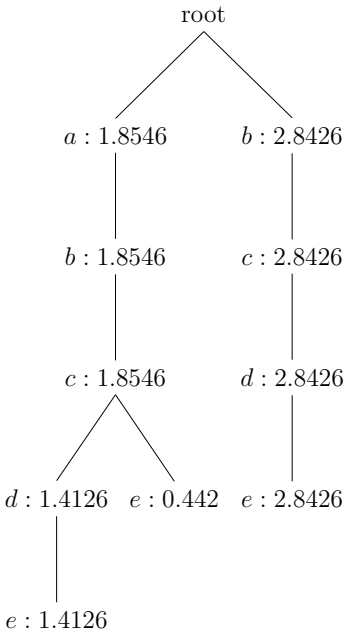
(b) The Full CUF-tree

Figure 3.6: The CUF-tree and Header Table, $minsup = 2$

In order to mine frequent patterns from a CUF-tree, CUF-growth extracts tree paths using a recursive algorithm, extending patterns upwards to the root while keeping track of the *transaction caps*. Let us consider $\{f\}$-conditional pattern bases in this example. First we check that the total transaction cap for f in the tree is greater than *minsup* in the header table (cf. Figure 3.6(a)). Since it is, we can proceed to directly generate the $\{f\}$-projected tree without taking an extra step to calculate the expected support as we had to do in UF-growth (as the sum of transaction caps is an upper bound to expected support). Starting from the global tree in Figure 3.6(b), the $\{f\}$-projected tree is created by as shown in Figure 3.7(b) by following all the branches with an f in them up to the root while replacing the transaction caps of those nodes that are ancestors of an f node with the sum of the transaction caps of all their children f nodes. The $\{f\}$-projected tree also has an associated $\{f\}$-projected header table as shown in Figure 3.7(a).

Item	**Σ Transaction Cap**
a	1.8546
b	4.6972
c	4.6972
d	4.2552
e	4.6972

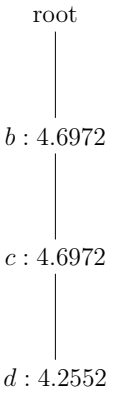
(a) $\{f\}$-projected Header Table



(b) The $\{f\}$-projected CUF-tree

Item	**Σ Transaction Cap**
b	4.6972
c	4.6972
d	4.2552

(c) $\{f, e\}$-projected Header Table



(d) The $\{f, e\}$-projected CUF-tree

Figure 3.7: CUF-growth Mining, $minsup = 2$

From the header table, we can see that all the sums of transaction caps are greater than *minsup* except for a. This means that every 2-itemset created from f and one of the items in the header table, except for $\{f, a\}$, is potentially frequent — we know that $\{f, a\}$ is not frequent since the sum of transaction caps is an upper bound to the expected support; if the upper bound is less than *minsup* than the expected support must also be less than *minsup*, therefore the node for a will be pruned in any deeper level of recursion. We cannot say at this stage whether or not the potentially frequent itemsets are truly frequent, as the sum of transaction caps is only an upper bound to the expected support, so we set them aside for now as *candidate frequent patterns* and the recursion continues. Figures 3.7(c) and 3.7(d) show the $\{f, e\}$-projected header table and CUF-tree. Since $\{f, a\}$ was not frequent in the $\{f\}$-projected tree, a does not appear in the $\{f, e\}$-projected tree.

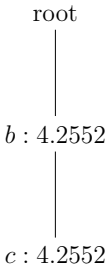
Every entry in the $\{f, e\}$-projected header table is greater than *minsup*, so all 3-itemsets created from $\{f, e\}$ and one of the items in the header table are set aside as candidates. Similarly, the $\{f, e, d\}$-projected header table and CUF-tree are shown in Figures 3.8(a) and 3.8(b). At this stage, every transaction cap sum in the header table is still above the *minsup* of 2, so $\{f, e, d, c\}$ and $\{f, e, d, b\}$ are set aside as candidate frequent patterns. Continuing, the $\{f, e, d, c\}$-projected tree has only a single node containing b as in Figures 3.8(c) and 3.8(d). At this point, pattern $\{f, e, d, c, b\}$ is set aside as a candidate frequent pattern since its sum of transaction caps is greater than *minsup* and the recursion unwinds.

All of the candidate frequent patterns generated during the mining are then checked against the input database one final time to calculate their true expected

support. The final list of frequent patterns and expected support values is the same as in UF-growth (cf. Table 3.6). However, unlike UF-growth, CUF-growth will find a total of 47 candidate itemsets during the mining process and determine that 20 of them are truly frequent. CUF-growth sacrifices a single extra database scan for a tree that has only 16 nodes compared to UF-growth's 51 on this small example. On full size databases, this leads to a significant memory and run-time savings, a compactness from around 1% to 13% of the size of the corresponding UF-tree was reported for several different datasets and *minsup* values [24].

Item	**Σ Transaction Cap**
b	4.2552
c	4.2552

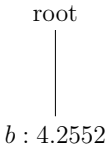
(a) $\{f, e, d\}$-projected Header Table



(b) The $\{f, e, d\}$-projected CUF-tree

Item	**Σ Transaction Cap**
b	4.2552

(c) $\{f, e, d, c\}$-projected Header Table



(d) The $\{f, e, d, c\}$-projected CUF-tree

Figure 3.8: CUF-growth Mining (Continued), $minsup = 2$

CUF-growth*: an Optimization

An optimization to the regular CUF-growth algorithm, called CUF-growth*, was briefly presented in the original CUF-growth paper [24]. It involves storing an additional "bronze value" in each tree node which is the third highest existential probability value in a transaction. CUF-growth* uses this bronze value to tighten the upper bound provided by the transaction caps in each node by multiplying it with the transaction cap $k-2$ times, once for each cardinality $k \geq 3$ during the mining process. Because of this compounding multiplicative effect, CUF-growth* consistently outperforms CUF-growth in returning fewer high cardinality candidate itemsets. Using the same example database and *minsup*, CUF-growth* finds a total of 41 candidate itemsets during the mining process – six less than the regular CUF-growth algorithm – and determines that 20 of them are truly frequent.

3.2.5 PUF-growth

Improving on their design for CUF-growth, Leung and Tanbeer [25] proposed the PUF-growth algorithm. Similar to CUF-growth, PUF-growth uses a prefix-tree structure with caps of expected support — the difference is in the calculation of $expSup^{Cap}$. While usage of the transaction cap allowed the compactness of CUF-trees to match that of FP-trees, the PUF-growth algorithm refined the cap calculation to take advantage of the tree structure itself to tighten the upper bound. In place of transaction caps, PUF-growth uses prefixed item caps, which are calculated as the product of the current item and the highest existential probability of any item in its prefix. For this reason, PUF-growth achieved faster runtimes than UFP-growth and

UH-mine. It also had less than a third the number of false positives as UFP-growth.

PUF-growth Example

For an example of PUF-tree construction, consider the following diagrams (Figure 3.9 and Figure 3.10) and tables (Table 3.3, 3.4 and 3.8). Given the initial table, a single scan through the database is required to calculate the expected support of each singleton itemset and remove infrequent items. These expected supports are shown in Table 3.4. After removing these items from consideration, Table 3.5 shows the remaining items in the uncertain database from which the global PUF-tree can be constructed.

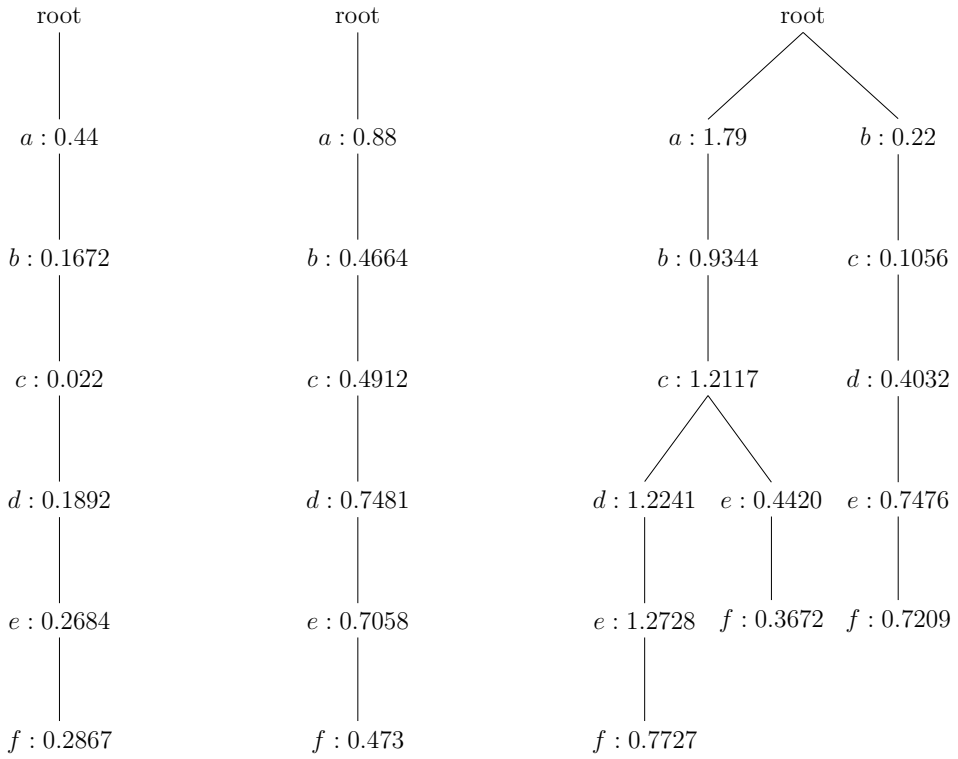
As transactions are being inserted into the tree, prefixed item caps (I^{Cap}s) are calculated for each item based on the remaining frequent items in each transaction. These I^{Cap}s are computed on the fly during the second database scan but are summarized in Table 3.8. For example, consider t_3; each frequent singleton item will have its own I^{Cap} value. $I^{Cap}(f, t_3)$ is calculated by multiplying the existential probability of f in t_3, $P(f, t_3)$, with the highest existential probability value in the prefix of f in that transaction, $P(f, t_3) \times P(e, t_3) = 0.37 \times 0.81 = 0.2997$. Likewise, $I^{Cap}(e, t_3) = P(e, t_3) \times P(b, t_3) = 0.81 \times 0.70 = 0.567$, and $I^{Cap}(b, t_3) = P(b, t_3) \times P(a, t_3) = 0.70 \times 0.26 = 0.182$. When calculating the I^{Cap} of a value that has no prefix item in a transaction there is no multiplication performed and the I^{Cap} is just the existential probability of that item; $I^{Cap}(a, t_3) = P(a, t_3) = 0.26$.

Table 3.8: Prefixed Item Caps for Table 3.3 ($minsup = 2$)

TID	**Prefixed Item Caps**
1	$\{a : 0.44, b : 0.1672, c : 0.0220, d : 0.1892, e : 0.2684, f : 0.2867\}$
2	$\{a : 0.44, b : 0.2992, c : 0.4692, d : 0.5589, e : 0.4374, f : 0.1863\}$
3	$\{a : 0.26, b : 0.1820, c : 0.3500, d : 0.4760, e : 0.5670, f : 0.2997\}$
4	$\{a : 0.65, b : 0.2860, c : 0.3705, e : 0.4420, f : 0.3672\}$
5	$\{b : 0.22, c : 0.1056, d : 0.4032, e : 0.7476, f : 0.7209\}$
6	$\{a : 0.50, c : 0.3950, d : 0.5293, e : 0.5372\}$
7	$\{b : 0.98, c : 0.6958, d : 0.8036, e : 0.9408, f : 0.9114\}$
8	$\{a : 0.64, b : 0.4096, c : 0.6016, d : 0.7614, e : 0.8742\}$
9	$\{b : 0.79, c : 0.4424, d : 0.6241, e : 0.2765, f : 0.0711\}$
10	$\{b : 0.57, c : 0.2394, d : 0.2451, e : 0.5301, f : 0.2418\}$

Adding the first transaction (cf. Figure 3.9(a)) creates a new node for each item in the transaction as a child of the previous node. However, unlike in UF-growth, the prefixed item cap is stored in place of the existential support and frequency. As with CUF-growth, adding the second transaction (cf. Figure 3.9(b)) does not create any new nodes since the items in t_2 share a common prefix (regardless of the existential probability) with an already existing branch in the tree. Since the criteria for creating new nodes in a PUF-tree and CUF-tree are the same, the global tree of PUF-growth and CUF-growth will also be almost exactly the same; they will have the nodes for the same items in the same places but those nodes will have different cap values (prefixed item caps instead of transaction caps). The remaining tree construction steps are shown in Figure 3.9(c) and the final PUF-tree is shown in Figure 3.10(b). Note that, as in FP-growth [13], horizontal node traversal pointers are maintained between nodes with the same item and are accessible through a header table. Additionally, in PUF-growth the header table keeps track of the total item cap value for a given item across all tree branches. The header table for the global PUF-tree in this example is

given in Figure 3.10(a).

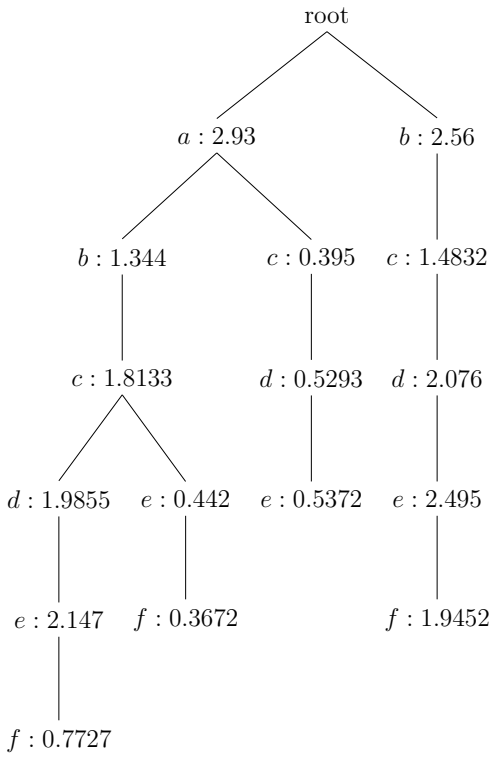


(a) Adding t_1 (b) Adding t_2 (c) Adding t_3, t_4 and t_5

Figure 3.9: PUF-tree Construction, $minsup = 2$

Item	**Σ Prefixed Item Cap**	**Pointer to PUF-tree Node, e.g.**
a	2.93	0x1005004a0
b	3.904	0x1005006a0
c	3.6915	0x1005003d0
d	4.5908	0x1005007e0
e	5.6212	0x100500810
f	3.0851	0x100500760

(a) PUF-tree Header Table



(b) The Full PUF-tree

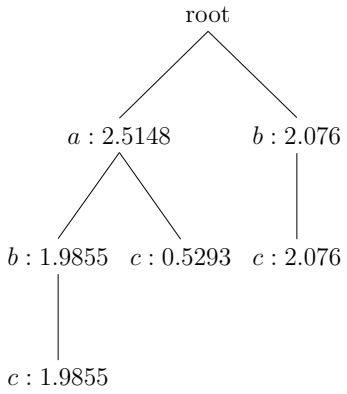
Figure 3.10: The Full PUF-tree and Header Table, $minsup = 2$

To mine frequent patterns, PUF-growth extracts tree paths using a recursive algorithm, extending patterns upwards to the root while keeping track of the *prefixed item caps*. Let us consider $\{d\}$-conditional pattern bases in this example. First we check that the total prefixed item cap for d in the tree is greater than *minsup* in the header table (cf. Figure 3.10(a)). Since it is, we can proceed to directly generate the $\{d\}$-projected tree without taking an extra step to calculate the expected support as we had to do in UF-growth. Starting from the global tree in Figure 3.10(b), the $\{d\}$-projected tree is created as shown in Figure 3.11(b) by following all the branches with a d in them up to the root while replacing the prefixed item caps of those nodes that are ancestors of a d node with the sum of the item caps of all their children d nodes. The $\{d\}$-projected tree also has an associated $\{d\}$-projected header table as shown in Figure 3.11(a).

From the header table, we can see that all the sums of prefixed item caps are still greater than *minsup*. This means that every 2-itemset created from d and one of the items in the header table is potentially frequent. We cannot say at this stage whether or not they are truly frequent, as the sum of prefixed item caps is an upper bound to the expected support, so we set them aside for now as *candidate frequent patterns* and the recursion continues. Figures 3.11(c) and 3.11(d) show the $\{d, c\}$-projected header table and PUF-tree. Again, every entry in the header table is greater than *minsup*, so all 3-itemsets created from $\{d, c\}$ and one of the items in the header table are set aside as candidates. At this point, notice that there is no b node in the $\{d, c\}$ projected tree that has a sum of prefixed item cap value greater than *minsup* that also has a parent node. This indicates that there will be no frequent extensions of the pattern $\{d, c, b\}$,

Item	**Σ Prefixed Item Cap**
a	2.5148
b	4.0615
c	4.5908

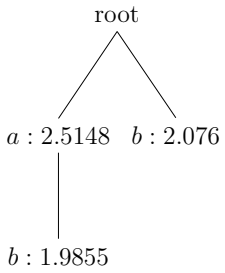
(a) $\{d\}$-projected Header Table



(b) The $\{d\}$-projected PUF-tree

Item	**Σ Prefixed Item Cap**
a	2.5148
b	4.0615

(c) $\{d, c\}$-projected Header Table



(d) The $\{d, c\}$-projected PUF-tree

Figure 3.11: PUF-growth Mining, $minsup = 2$

therefore the recursion stops here and patterns $\{d, c, b\}$ and $\{d, c, a\}$ are set aside as candidate frequent patterns before the recursion unwinds. The $\{d, b\}$-projected tree is also not created for the same reason. Now, all the patterns containing d and some other items in d's prefix have been checked. The PUF-growth algorithm next proceeds to create other single item projected trees by reading paths off of the global tree.

All of the candidate frequent patterns generated during the mining are then checked against the input database one final time to calculate their true expected

support. The final list of frequent patterns and expected support values is the same as in UF-growth (cf. Table 3.6). However, unlike UF-growth and CUF-growth, PUF-growth will find a total of 43 candidate itemsets during the mining process and determine that 20 of them are truly frequent. This is a total of 4 fewer candidates than CUF-growth finds for the same database and *minsup*. The reason for this improvement is that the item cap provided by PUF-growth is always less than or equal to the transaction cap provided by CUF-growth. On full size databases this difference becomes magnified, and PUF-growth is the obvious winner over CUF-growth. When comparing PUF-growth with the optimized version of CUF-growth, CUF-growth*, PUF-growth finds 2 extra candidates since it does not use any compounding value.

3.3 Summary

Frequent pattern mining algorithms can be broadly classified into candidate-generate-and-test algorithms (e.g., Apriori [4] and UApriori [9]) and pattern growth algorithms (e.g. tree based algorithms such as FP-growth [13] and UF-growth [23]). Candidate-generate-and-test algorithms require multiple scans of the input database, and their join operation may produce extra patterns that must be pruned if not all of their subset patterns are frequent in a lower cardinality. Pattern growth algorithms do not have this problem because they directly generate frequent or candidate frequent patterns. However, when uncertain data are used, memory usage can still become a problem due to the challenge of working with multiple existential probability values for the same item. UF-growth [23], as a first attempt to work with uncertain data in the tree-based pattern growth framework, directly adapts the principals from

the precise data FP-growth algorithm but requires new tree nodes for every unique combination of item value and existential probability in the input database. Newer algorithms attempt to reduce the number of tree nodes for the same item by using upper bounds. UFP-growth [2] uses a modified k-means clustering to combine multiple existential probability values for the same item together in the same node, storing the maximum existential probability value and the size for each cluster. The upper bound calculated from the cluster data favours higher probability values in parent nodes, so if the cluster sizes are large (for a maximal reduction in memory usage), the upper bound may not be so tight. CUFP-growth [27] uses the same path sharing condition of FP-growth without calculating an upper bound on expected support (thus providing an exact solution). While the total number of nodes is drastically reduced, the size of the superset array in each node grows exponentially with the depth of the tree, thus the actual memory usage can still be quite high. CUF-growth [24] and PUF-growth [25] also use the same path sharing condition of FP-growth, but calculate upper bounds using transaction and item caps (cf. Section 2.5) respectively.

Chapter 4

Tree-based Uncertain Data Mining Algorithms

I now present two new algorithms, (1) Tube-S and (2) Tube-P, for tree-based uncertain data mining. These algorithms share a general mathematical and data structure design with the previous PUF-growth [25] and, to a lesser extent, the original FP-growth [13], by borrowing the idea of item caps as the basis for an upper bound calculation and using prefix trees, but they take different approaches to achieve tighter upper bounds on the expected support than the previous works. The data structures and mining processes used in the algorithms are given formal definitions and described in detail below, including worked through examples for comparison with those from Section 3 (cf. Sections 3.2.1, 3.2.4 and 3.2.5). Recall that CUF-growth* stored a "bronze value", which was the third highest existential probability in the entire transaction, in addition to the transaction cap in each node (cf. Section 3.2.4). With Tube-S, I further improve on this concept by storing an even tighter M_2 value in

each tree node, allowing the upper bound on expected support to be progressively tightened at each stage of the recursion. In Tube-P, I show an alternative data layout for the nodes in the tree structure that leads to tighter upper bounds than Tube-S in some situations, which is supported by my experimental evaluation. Furthermore, I give formal definitions for the upper bounds used in both algorithms using a common framework that considers the actual tree paths involved.

4.1 Tube-S

Recall from Example 3.2.5 that PUF-growth used the item cap $I^{Cap}(x, t_j)$ as the basis of its $expSup^{Cap}$ calculation for a k-itemset X. The upper bound based on this value is quite close to the actual expected support for any 2-itemset, but as the cardinality of k becomes greater than two this is no longer the case. The reason is that the item cap only considers the existential probability of two items regardless of the cardinality of X. To solve the aforementioned problem, I extend the concept of item caps to higher cardinalities and aim to obtain tighter upper bounds on expected support. My first idea is to keep track of the second highest existential probability in the proper "prefix" of an "ordered" transaction t_j and use it every time a frequent extension ($k > 2$) is added to the suffix item (i.e., when a frequent item is added to the currently frequent k-itemset to form a $(k + 1)$-itemset). For tree-based mining, I build a tree structure such that each tree path represents some transactions of uncertain data. In a tree path representing a single transaction, each tree node keeps:

- an item y_i in $t_j = \{y_1, \ldots, y_i, \ldots, y_h\}$,

- its item cap $I^{Cap}(y_i, t_j)$, and
- the second highest existential probability $M_2(y_i, t_j)$ in the proper "prefix" $\{y_1, \ldots, y_{i-1}\} \subset t_j$.

With this information, we get a ***tightened upper bound to expected support*** based on a ***second highest*** existential probability (***tube-S***). For $X = \{x_1, \ldots, x_k\} \subseteq t_j$ where $x_k = y_r$:

$$tube\text{-}S(X, t_j) = \begin{cases} I^{Cap}(x_k, t_j) & \text{if } k \leq 2, \\ I^{Cap}(x_k, t_j) \times \prod_{i=1}^{k-2} M_2(y_r, t_j) & \text{if } k \geq 3 \end{cases} \tag{4.1}$$

where $M_2(y_r, t_j)$ is the second highest existential probability value among all $r-1$ items in the proper "prefix" $\{y_1, \ldots, y_{r-1} \subset t_j\}$. Unlike CUF-growth*, whose bronze value considers existential probabilities over the entire transaction, restricting my M_2 value to the proper "prefix" of the transaction not only has a compounding effect on tightening the upper bound for higher cardinality candidates, but it has a further effect of eliminating high existential probabilities from spuriously inflating the upper bound when they are associated with items in the transaction but guaranteed never to be in the itemsets under consideration by the mining process.

Example 4.1. For the 4-itemset $X = \{b, c, d, e\}$ with $expSup(X, t_1) = 0.0049837$ in $t_1 = \{a : 0.44, b : 0.38, c : 0.05, d : 0.43, e : 0.61, f : 0.47\}$ from Table 3.5, $tube\text{-}S(X, t_1) = (0.61 \times 0.44) \times 0.43^2 = 0.04962716$ is tighter than that provided by $I^{Cap}(X, t_1) = 0.2684$. For a 3-itemset $Y = \{b, d, f\}$, $tube\text{-}S(Y, t_1) = (0.47 \times 0.61) \times 0.44 = 0.126148$, which is also the tube value for 3-itemsets $\{a, b, f\}$, $\{a, c, f\}$, $\{a, d, f\}$, $\{a, e, f\}$, $\{b, c, f\}$, $\{b, e, f\}$, $\{c, d, f\}$, $\{c, e, f\}$ and $\{d, e, f\}$. ■

Lemma 4.1. As $expSup(X, t_j) \leq tube\text{-}S(X, t_j) \leq I^{Cap}(X, t_j)$, tube-S serves as a tighter upper bound to the expected support of X in t_j than the item cap.

Proof. For any $X = \{x_1, \ldots, x_k\} \subseteq \{y_1, \ldots, y_r\} = t_j$ where $x_k = y_r$,

$$expSup(X, t_j) = P(x_k, t_j) \times \prod_{i=1}^{k-1} P(x_i, t_j)$$

$$\leq tube\text{-}S(X, t_j) = P(x_k, t_j) \times M_1(x_k, t_j) \times [M_2(x_k, t_j)]^{k-2}$$

$$\leq I^{Cap}(X, t_j) = P(x_k, t_j) \times M_1(x_k, t_j)$$

because: (a) $M_1(x_k, t_j)$ and $M_2(x_k, t_j)$ are the two highest existential probabilities among all $P(y_q, t_j)$ for $1 \leq q \leq r-1$, including all $P(x_i, t_j)$ for $1 \leq i \leq k-1$ and (b) $0 \leq P(x, t_j) \leq 1$. ∎

Recall that the UF-tree shares a tree path only if tree nodes on the path have the same item and the same existential probability. Here, to make my proposed tree structure compact and practical for mining, I merge tree paths when items are the same, regardless of their existential probability values. So, given two tree nodes $x : I^{Cap}(x, t_1) : M_2(x, t_1)$ and $x : I^{Cap}(x, t_2) : M_2(x, t_2)$ that share the same path p, I (i) sum their I^{Cap} values and (ii) take the maximum between the two M_2 values. This results in a tree node $x : I^{Cap}(x, t_1) + I^{Cap}(x, t_2) : \max\{M_2(x, t_1), M_2(x, t_2)\}$. Then, a tube-S for $X = \{x_1, \ldots, x_k\}$ in p can be computed as:

$$tube\text{-}S(X, p) = \begin{cases} I^{Cap}(x_k, p) & \text{if } k \leq 2, \\ I^{Cap}(x_k, p) \times [M_2(x_k, p)]^{k-2} & \text{if } k \geq 3 \end{cases} \tag{4.2}$$

where (i) $I^{Cap}(X, p) = \sum_{t_j \in p} I^{Cap}(X, t_j)$ and (ii) $M_2(x_k, p) = \max_{t_j \in p} M_2(x_k, t_j)$. Consequently, the tube-S of X in the entire database is the sum of tube-S in every

path p in the tree capturing the uncertain data:

$$\textit{tube-S}(X) = \sum_{p} \textit{tube-S}(X, p) \tag{4.3}$$

Lemma 4.2. As $\textit{expSup}(X) \leq \textit{tube-S}(X) \leq I^{Cap}(X)$, tube-S serves as a tighter upper bound to the expected support of X (in the entire database) than the item cap.

Proof. It follows from Lemma 4.1 that summing over every tree branch containing X in the manner of Equations 4.2 and 4.3 does not change the inequality.

$$\begin{aligned} \textit{expSup}(X) &= \sum_{p} \sum_{t_j \in p} \left[P(x_k, t_j) \times \prod_{i=1}^{k-1} P(x_i, t_j) \right] \\ \leq \sum_{p} \textit{tube-S}(X, p) &= \sum_{p} \left[\sum_{t_j \in p} I^{Cap}(X, t_j) \times \left(\max_{t_j \in p} M_2(x_k, p) \right)^{k-2} \right] \\ \leq \sum_{p} I^{Cap}(X, p) &= \sum_{p} \sum_{t_j \in p} I^{Cap}(X, t_j) \end{aligned}$$

■

Example 4.2. Continue with Example 4.1. For the 4-itemset $X = \{b, c, d, e\}$ appearing in $t_1 = \{a : 0.44, b : 0.38, c : 0.05, d : 0.43, e : 0.61, f : 0.47\}$, $t_2 = \{a : 0.44, b : 0.68, c : 0.69, d : 0.81, e : 0.54, f : 0.23\}$, $t_3 = \{a : 0.26, b : 0.70, c : 0.50, d : 0.68, e : 0.81, f : 0.37\}$, $t_5 = \{b : 0.22, c : 0.48, d : 0.84, e : 0.89, f : 0.81\}$, $t_7 = \{b : 0.98, c : 0.71, d : 0.82, e : 0.96, f : 0.93\}$, $t_8 = \{a : 0.64, b : 0.64, c : 0.94, d : 0.81, e : 0.93\}$, $t_9 = \{b : 0.79, c : 0.56, d : 0.79, e : 0.35, f : 0.09\}$ and $t_{10} = \{b : 0.57, c : 0.42, d : 0.43, e : 0.93, f : 0.26\}$ in Table 3.5, $\textit{expSup}(X)$ is $0.0049837 + 0.20522808 + 0.19278 + 0.07894656 + 0.54773376 + 0.45318528 + 0.1223236 + 0.09573606 = 1.70091704$. As t_1, t_2, t_3 and t_8 share the same path p_1, the node for e in p_1 captures (i) a sum of I^{Cap} value of $(0.61 \times 0.44) + (0.54 \times 0.81) + (0.81 \times 0.70) + (0.93 \times 0.94) = 2.147$

and (ii) a maximum M_2 value of $\max\{0.43, 0.69, 0.68, 0.81\} = 0.81$. Likewise, as t_5, t_7, t_9 and t_{10} share the same path p_2, the node for e in p_2 captures (i) a sum of I^{Cap} value of $(0.89 \times 0.84) + (0.96 \times 0.98) + (0.35 \times 0.79) + (0.93 \times 0.57) = 2.495$ and (ii) a maximum M_2 value of $\max\{0.48, 0.82, 0.79, 0.43\} = 0.82$. The *tube-S*$(X)$ in the entire database is then the sum of products over each path $\sum_p$ *tube-S*$(X, p) = (2.147 \times 0.81^2) + (2.495 \times 0.82^2) = 3.0862847$, which is much tighter than the bound provided by the sum of item caps $\sum_p I^{Cap}(X, p) = 2.147 + 2.495 = 4.642$. ■

4.1.1 Tube-S Tree Construction

Consider the following diagrams (Figure 4.1 and Figure 4.2) and tables (Table 3.3, 3.4, 3.8 and 4.1) for an example of Tube-S tree construction. Given the initial table, a single scan through the database is required to calculate the expected support of each singleton itemset and remove infrequent items (cf. the downward closure property, Section 2.2). These expected supports are shown in Table 3.4. After removing these items from consideration, Table 3.5 shows the remaining items in the uncertain database from which the global Tube-S tree can be constructed. So far, the process is exactly the same as in PUF-growth.

As transactions are being inserted into the tree, prefixed item caps (I^{Cap}s) are calculated for each item based on the remaining frequent items in each transaction as in PUF-growth. In the Tube-S algorithm, M_2 values are additionally calculated for each item. Both the I^{Cap}s and M_2 values are computed on the fly during the second database scan but are summarized in Table 4.1. Note that M_2 values for nodes within two edges from the root are not calculated as they will never be used

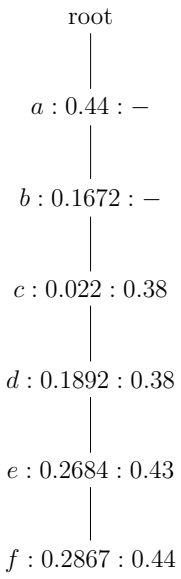
during the mining. For example, consider t_{10}; each frequent singleton item (except the first two) will have its own M_2 value. Since the M_2 value of an item is the second highest existential probability value in the proper prefix of that item, $M_2(f, t_{10}) = P(e, t_{10}) = 0.87$ since $P(e, t_{10}) \leq P(c, t_{10})$, which is the highest existential probability value in the prefix of f in t_{10}. Likewise, $M_2(e, t_{10}) = 0.67$, $M_2(d, t_{10}) = 0.67$ as well and $M_2(c, t_{10}) = 0.55$. $M_2(b, t_{10})$ and $M_2(a, t_{10})$ are not calculated and are represented as $-$ in the table.

Table 4.1: M_2 Values for Table 3.3 ($minsup = 2$)

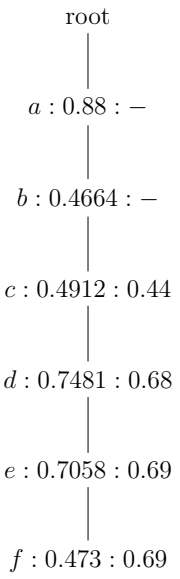
TID	M_2 **Values**
1	$\{a:-, b:-, c:0.38, d:0.38, e:0.43, f:0.44\}$
2	$\{a:-, b:-, c:0.44, d:0.68, e:0.69, f:0.69\}$
3	$\{a:-, b:-, c:0.26, d:0.50, e:0.68, f:0.70\}$
4	$\{a:-, b:-, c:0.44, e:0.57, f:0.65\}$
5	$\{b:-, c:-, d:0.22, e:0.48, f:0.84\}$
6	$\{a:-, c:-, d:0.50, e:0.67\}$
7	$\{b:-, c:-, d:0.71, e:0.82, f:0.96\}$
8	$\{a:-, b:-, c:0.64, d:0.64, e:0.81\}$
9	$\{b:-, c:-, d:0.56, e:0.79, f:0.79\}$
10	$\{b:-, c:-, d:0.42, e:0.43, f:0.57\}$

Adding the first transaction (cf. Figure 4.1(a)) creates a new node for each item in the transaction as a child of the previous node. However, unlike in UF-growth, the prefixed item cap and M_2 value are stored in place of the unique existential probability and frequency. As with CUF-growth and PUF-growth, adding the second transaction (cf. Figure 4.1(b)) shares the same path, however, each node in Tube-S stores only the maximum M_2 value for its item in the transactions that pass through that node. Since all of t_2's M_2 values are higher than t_1's, only the M_2 values for t_2 end up getting stored. Since the item cap calculation is not changed from PUF-

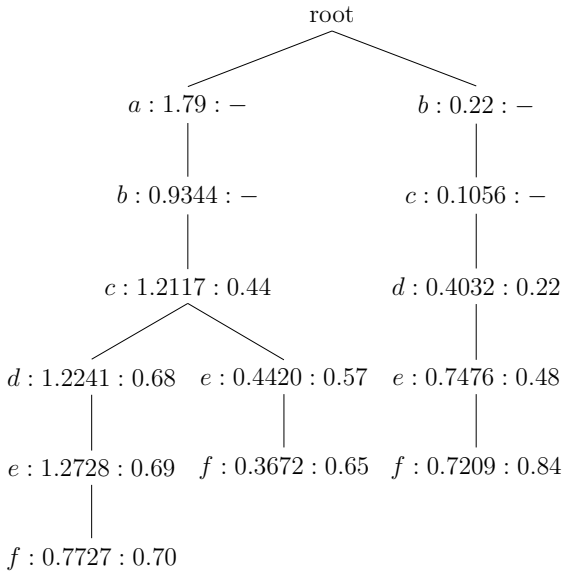
growth, the tree will be the same except for the new addition of M_2 values. The remaining tree construction steps are shown in Figure 4.1(c). Notice that adding new paths happens only at the first point where the item values in any existing path no longer match up with those in the prefix of the current transaction (regardless of the I^{Cap} or M_2 values). The final Tube-S tree is shown in Figure 4.2(b). As in FP-growth [13], horizontal node traversal pointers are maintained between nodes with the same item and are accessible through a header table. The header table for a Tube-S tree maintains the same values as for a PUF-tree. The header table for the global Tube-S tree in this example is given in Figure 4.2(a).



(a) Adding t_1



(b) Adding t_2

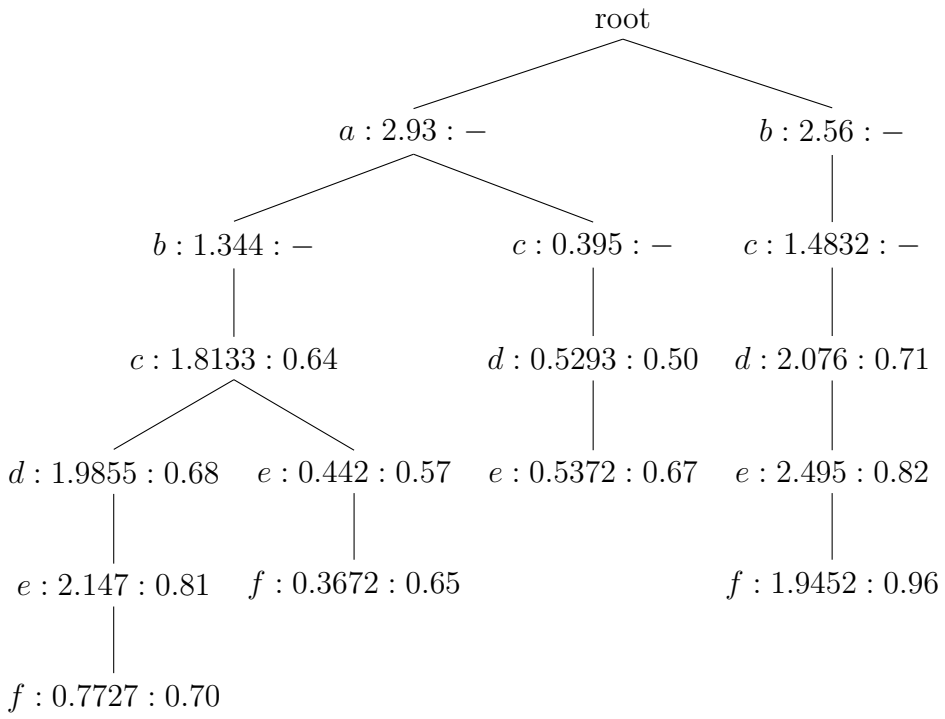


(c) Adding t_3, t_4 and t_5

Figure 4.1: Tube-S Tree Construction, $minsup = 2$

Item	**Σ Prefixed Item Cap**	**Pointer to Tube-S Tree Node, e.g.**
a	2.93	0x1003051e0
b	3.904	0x1004003c0
c	3.6915	0x1004007e0
d	4.5908	0x100400820
e	5.6212	0x100400860
f	3.0851	0x100400760

(a) Tube-S Tree Header Table



(b) The Full Tube-S Tree

Figure 4.2: The Tube-S Tree and Header Table, $minsup = 2$

4.1.2 Mining with Tube-S

To mine frequent patterns, Tube-S extracts tree paths using a recursive algorithm, extending patterns upwards to the root while keeping track of the prefixed item caps and M_2 values. Let us consider $\{f\}$-conditional pattern bases in this example. First we check that the total prefixed item cap for f in the tree is greater than *minsup* in the header table (cf. Figure 4.2(a)). Since it is, we can proceed to directly generate the $\{f\}$-projected tree without taking an extra step to calculate the expected support as we had to do in UF-growth. Starting from the global tree in Figure 4.2(b), the $\{f\}$-projected tree is created by as shown in Figure 4.3(b) by following all the branches with an f in them up to the root while replacing the prefixed item caps of those nodes that are ancestors of an f node with the sum of the item caps of all their children f nodes; and replacing the M_2 values of those nodes that are ancestors of an f node with the maximum M_2 value of all their children f nodes. When we do this, we find that the sum of prefixed item caps for $a = 1.1399 < minsup = 2$, therefore a can be pruned from the projected tree. The pruned $\{f\}$-projected tree and header table are shown in Figure 4.3(c) and Figure 4.3(d). This is shown as two separate steps for clarity, but in actuality it is not necessary to create a tree and then prune it. While reading the $\{f\}$-projected pattern bases all the necessary information is available to create the fully pruned $\{f\}$-projected tree in a single step. A quick glance at the header table shows that the sum of prefixed item caps for each item is greater than *minsup* so every 2-itemset created from f and one of the items in the $\{f\}$-projected header table becomes a candidate.

For each higher level of recursion, the M_2 value is used to tighten the upper bound

and reduce the number of candidate frequent patterns. For the $\{f, e\}$-projected tree, we first check if the sum of the prefixed item cap is greater than *minsup* in our $\{f\}$-projected tree. This is the case in our example, but before we build the $\{f, e\}$-projected tree, while looking at the $\{f, e\}$-conditional pattern bases, we multiply the value of the item cap in each e node with the M_2 value in the same node to determine the new sum of prefixed item cap for any 3-itemset in the $\{f, e\}$-projected tree. In this case, multiplication with the M_2 value for each e node yields a sum of products of $(0.3672 \times 0.65) + (2.7179 \times 0.96) = 2.847864 > minsup = 2$. Since we now know the sum of prefixed item caps will be greater than *minsup*, we can proceed to generate the $\{f, e\}$-projected tree and header table as shown in Figures 4.4(a) and 4.4(b). We replace the prefixed item cap in each node that is an ancestor of an e node with the *sum of products* of the item cap and M_2 values of each of its children e nodes; and replace the M_2 values of each node that is an ancestor of an e node with the maximum M_2 value among its children e nodes. Because the sum of prefixed item cap for each item in the $\{f, e\}$-projected header table is greater than *minsup*, every 3-itemset created from f, e and one of the items in the $\{f, e\}$-projected header table becomes a candidate.

Proceeding deeper into the recursion, we look at the $\{f, e, d\}$-conditional pattern bases. We multiply the value of the item cap in each d node with the M_2 value in the same node to determine the new sum of prefixed item cap for any 4-itemset in the $\{f, e, d\}$-projected tree. Multiplication with the M_2 value for each d node yields: $(2.60918 \times 0.96) = 2.5048128 > minsup = 2$, therefore we proceed to generate the $\{f, e, d\}$-projected tree in the same manner as the $\{f, e\}$-projected tree as shown in

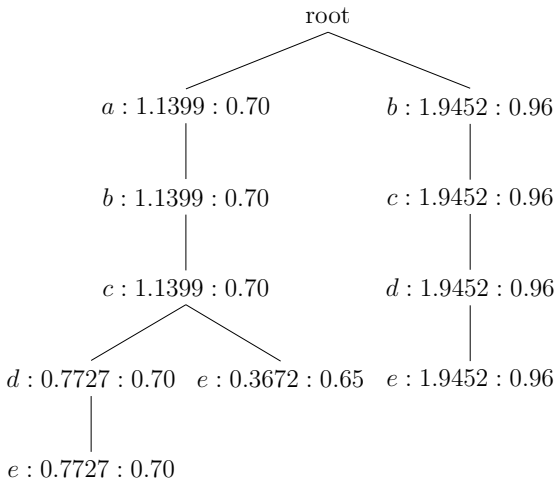
Figures 4.4(c) and 4.4(d). Because the sum of prefixed item cap for each item in the $\{f, e, d\}$-projected header table is greater than *minsup*, every 4-itemset created from f, e, d and one of the items in the $\{f, e, d\}$-projected header table becomes a candidate.

The $\{f, e, d, c\}$-projected tree and header table (cf. Figures 4.4(e) and 4.4(f)) are created in a similar manner, after which pattern $\{f, e, d, c, b\}$ is set aside as a candidate frequent pattern. At this stage, since the tree consists of only a single level, the recursion unwinds and the Tube-S algorithm generates $\{f, e, c\}$-, $\{f, d\}$-, $\{f, d, c\}$- and $\{f, c\}$-projected trees before unwinding further and creating other top level projected trees from the remaining items in the global tree header table.

All of the candidate frequent patterns generated during the mining are then checked against the input database one final time to calculate their true expected support. The final list of frequent patterns and expected support values is the same as in UF-growth (cf. Table 3.6). However, unlike PUF-growth, Tube-S will find a total of 38 candidate itemsets (cf. 39 in PUF-growth) during the mining process and determine that 20 of them are truly frequent. While an improvement in candidate pattern reduction of only a single pattern may seem underwhelming, recall that the database used for this example had only ten transactions. With many times this number of transactions being used in practice, the result is significant. Also recall that this algorithm is *guaranteed* to find *at most* as many frequent patterns as PUF-growth in the worst case, and performs much better the vast majority of the time.

Item	**Σ Prefixed Item Cap**
a	1.1399
b	3.0851
c	3.0851
d	2.7179
e	3.0851

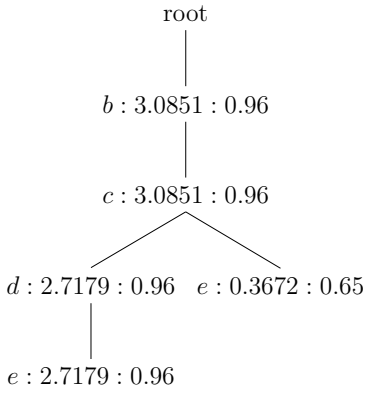
(a) $\{f\}$-projected Header Table



(b) The $\{f\}$-projected Tube-S Tree

Item	**Σ Prefixed Item Cap**
b	3.0851
c	3.0851
d	2.7179
e	3.0851

(c) $\{f\}$-projected Header Table (After Pruning)

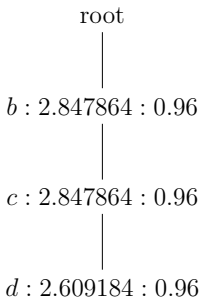


(d) The $\{f\}$-projected Tube-S Tree (After Pruning)

Figure 4.3: Tube-S Mining, $minsup = 2$

Item	**Σ Prefixed Item Cap**
b	2.847864
c	2.847864
d	2.609184

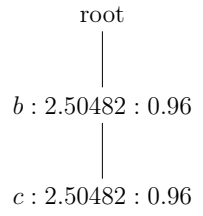
(a) $\{f, e\}$-projected Header Table



(b) The $\{f, e\}$-projected Tube-S Tree

Item	**Σ Prefixed Item Cap**
b	2.50482
c	2.50482

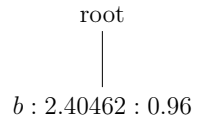
(c) $\{f, e, d\}$-projected Header Table



(d) The $\{f, e, d\}$-projected Tube-S Tree

Item	**Σ Prefixed Item Cap**
b	2.40462

(e) $\{f, e, d, c\}$-projected Header Table



(f) The $\{f, e, d, c\}$-projected Tube-S Tree

Figure 4.4: Tube-S Mining (cont.), $minsup = 2$

4.2 Tube-P

My second idea is to keep track of the existential probabilities of specific items in the same nodes as those items. As opposed to keeping track of the maximum second highest value (i.e., M_2 value) in the prefix of an item for those transactions that share a tree path (regardless of what that item is), I instead restrict the domain of interested values to a single item for each node. For an "ordered" transaction t_j, I use one of these existential probabilities every time a frequent extension ($k > 2$) is added to the suffix item (i.e., when a frequent item is added to the currently frequent k-itemset to form a $(k+1)$-itemset). For tree-based mining, I build a tree structure such that each tree path represents some transactions of uncertain data. In a tree path representing a single transaction, each tree node keeps:

- an item y_i in $t_j = \{y_1, \ldots, y_i, \ldots, y_h\}$,
- its item cap $I^{Cap}(y_i, t_j)$, and
- its existential probability $P(y_i, t_j)$.

With this information, we get a ***tightened upper bound to expected support*** based on the existential probabilities of ***prefix items*** (***tube-P***). For $X = \{x_1, \ldots, x_k\} \subseteq t_j$ where $x_k = y_r$:

$$tube\text{-}P(X, t_j) = \begin{cases} I^{Cap}(x_k, t_j) & \text{if } k \leq 2, \\ I^{Cap}(x_k, t_j) \times \prod_{i=1}^{k-2} P(x_i, t_j) & \text{if } k \geq 3 \end{cases} \tag{4.4}$$

where $P(x_i, t_j)$ is the existential probability of $x_i \in t_j$.

Example 4.3. For the 4-itemset $X = \{b, c, d, e\}$ with $expSup(X, t_1) = 0.0049837$ in $t_1 = \{a : 0.44, b : 0.38, c : 0.05, d : 0.43, e : 0.61, f : 0.47\}$ from Table 3.5,

tube-P$(X, t_1) = (0.61 \times 0.44) \times 0.05 \times 0.38 = 0.0050996$ is tighter than that provided by $I^{Cap}(X, t_1) = 0.2684$ and *tube-S*$(X, t_1) = 0.04962716$ (cf. Example 4.1). For a 3-itemset $Y = \{b, d, f\}$, *tube-P*$(Y, t_1) = (0.47 \times 0.61) \times 0.38 = 0.108946$, which is tighter than that provided by $I^{Cap}(Y, t_1) = 0.2867$ and *tube-S*$(Y, t_1) = 0.126148$. ■

Lemma 4.3. As $expSup(X, t_j) \leq$ *tube-P*$(X, t_j) \leq I^{Cap}(X, t_j)$, tube-P serves as a tighter upper bound to the expected support of X in t_j than the item cap.

Proof. For any $X = \{x_1, \ldots, x_k\} \subseteq \{y_1, \ldots, y_r\} = t_j$ where $x_k = y_r$,

$$expSup(X, t_j) = P(x_k, t_j) \times P(x_{k-1}, t_j) \times \prod_{i=1}^{k-2} P(x_i, t_j)$$

$$\leq \textit{tube-P}(X, t_j) = P(x_k, t_j) \times M_1(x_k, t_j) \times \prod_{i=1}^{k-2} P(x_i, t_j)$$

$$\leq I^{Cap}(X, t_j) = P(x_k, t_j) \times M_1(x_k, t_j)$$

because: (a) $M_1(x_k, t_j)$ is the highest existential probability among all $P(y_q, t_j)$ for $1 \leq q \leq r - 1$, including $P(x_{k-1}, t_j)$ and (b) $0 \leq P(x, t_j) \leq 1$. ■

Again, to make my proposed tree structure compact and practical for mining, I merge tree paths when items are the same, regardless of their existential probability values. So, given two tree nodes $x : I^{Cap}(x, t_1) : P(x, t_1)$ and $x : I^{Cap}(x, t_2) : P(x, t_2)$ that share the same path p, I (i) sum their I^{Cap} values and (ii) take the maximum between the two existential probabilities for x. This results in a tree node $x : I^{Cap}(x, t_1) + I^{Cap}(x, t_2) : \max\{P(x, t_1), P(x, t_2)\}$. Then, a tube-P for $X = \{x_1, \ldots, x_k\}$ in p can be computed as:

$$\textit{tube-P}(X, p) = \begin{cases} I^{Cap}(x_k, p) & \text{if } k \leq 2, \\ I^{Cap}(x_k, p) \times \prod_{i=1}^{k-2} P(x_i, p) & \text{if } k \geq 3 \end{cases} \tag{4.5}$$

where (i) $I^{Cap}(X,p) = \sum_{t_j \in p} I^{Cap}(X,t_j)$ and (ii) $P(x_k,p) = \max_{t_j \in p} P(x_k,t_j)$. Consequently, the tube-P of X in the entire database is the sum of tube-P in every path p in the tree capturing the uncertain data:

$$tube\text{-}P(X) = \sum_{p} tube\text{-}P(X,p) \tag{4.6}$$

Lemma 4.4. As $expSup(X) \leq tube\text{-}P(X) \leq I^{Cap}(X)$, tube-P serves as a tighter upper bound to the expected support of X (in the entire database) than the item cap.

Proof. It follows from Lemma 4.3 that summing over every tree branch containing X in the manner of Equations 4.5 and 4.6 does not change the inequality.

$$\begin{aligned} expSup(X) &= \sum_{p} \sum_{t_j \in p} \left[P(x_k,t_j) \times P(x_{k-1},t_j) \times \prod_{i=1}^{k-2} P(x_i,t_j) \right] \\ \leq \sum_{p} tube\text{-}P(X,p) &= \sum_{p} \left[\sum_{t_j \in p} I^{Cap}(X,t_j) \times \prod_{i=1}^{k-2} P(x_i,p) \right] \\ \leq \sum_{p} I^{Cap}(X,p) &= \sum_{p} \sum_{t_j \in p} I^{Cap}(X,t_j) \end{aligned}$$

■

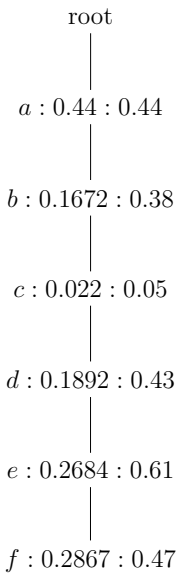
Example 4.4. Continue with Example 4.3. For the 4-itemset $X = \{b,c,d,e\}$ appearing in $t_1 = \{a:0.44, b:0.38, c:0.05, d:0.43, e:0.61, f:0.47\}$, $t_2 = \{a:0.44, b:0.68, c:0.69, d:0.81, e:0.54, f:0.23\}$, $t_3 = \{a:0.26, b:0.70, c:0.50, d:0.68, e:0.81, f:0.37\}$, $t_5 = \{b:0.22, c:0.48, d:0.84, e:0.89, f:0.81\}$, $t_7 = \{b:0.98, c:0.71, d:0.82, e:0.96, f:0.93\}$, $t_8 = \{a:0.64, b:0.64, c:0.94, d:0.81, e:0.93\}$, $t_9 = \{b:0.79, c:0.56, d:0.79, e:0.35, f:0.09\}$ and $t_{10} = \{b:0.57, c:0.42, d:0.43, e:0.93, f:0.26\}$ in Table 3.5, $expSup(X)$ is $0.0049837 + 0.20522808 + 0.19278 + 0.07894656 + 0.54773376 + 0.45318528 + 0.1223236 + 0.09573606 = 1.70091704$. As t_1,

t_2, t_3 and t_8 share the same path p_1, the node for e in p_1 captures (i) a sum of I^{Cap} value of $(0.61 \times 0.44) + (0.54 \times 0.81) + (0.81 \times 0.70) + (0.93 \times 0.94) = 2.147$ and (ii) the nodes for b and c capture maximum existential probabilities of $\max\{0.38, 0.68, 0.70, 0.64\} = 0.70$ and $\max\{0.05, 0.69, 0.50, 0.94\} = 0.94$ respectively. Likewise, as t_5, t_7, t_9 and t_{10} share the same path p_2, the node for e in p_2 captures (i) a sum of I^{Cap} value of $(0.89 \times 0.84) + (0.96 \times 0.98) + (0.35 \times 0.79) + (0.93 \times 0.57) = 2.495$ and (ii) the nodes for b and c capture maximum existential probabilities of $\max\{0.22, 0.98, 0.79, 0.57\} = 0.98$ and $\max\{0.48, 0.71, 0.56, 0.42\} = 0.71$ respectively. The *tube-P*(X) in the entire database is then the sum of products over each path $\sum_p$ *tube-P*$(X, p) = (2.147 \times 0.94 \times 0.70) + (2.495 \times 0.71 \times 0.98) = 3.148747$, which is much tighter than the bound provided by the sum of item caps $\sum_p I^{Cap}(X, p) = 2.147 + 2.495 = 4.642$. ■

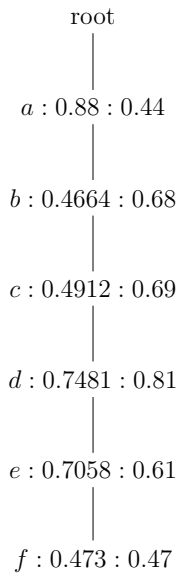
4.2.1 Tube-P Tree Construction

Consider the following diagrams (Figure 4.5 and Figure 4.6) and tables (Table 3.3, 3.4 and 3.8) for an example of Tube-P tree construction. Given the initial table, a single scan through the database is required to calculate the expected support of each singleton itemset and remove infrequent items (cf. the downward closure property, Section 2.2). These expected supports are shown in Table 3.4. After removing these items from consideration, Table 3.5 shows the remaining items in the uncertain database from which the global Tube-P tree can be constructed. As transactions are being inserted into the tree, prefixed item caps (I^{Cap}s) are calculated for each item based on the remaining frequent items in each transaction as in PUF-growth. In the Tube-P algorithm, existential probability values are additionally stored for each item.

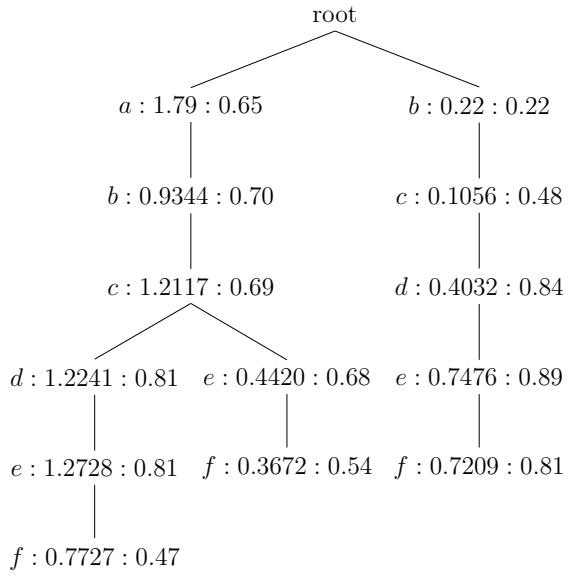
Adding the first transaction (cf. Figure 4.5(a)) creates a new node for each item in the transaction as a child of the previous node. However, unlike in UF-growth, the sum of prefixed item caps and maximum existential probability value are stored in place of the unique existential probability and frequency. As with CUF-growth and PUF-growth, adding the second transaction (cf. Figure 4.5(b)) shares the same path, however, each node in Tube-P stores only the maximum existential probability value for its item in the transactions that pass through that node. Since all of t_2's existential probability values are higher than t_1's except for e and f, t_2's existential probability values of for a, b, c and d and t_1's existential probability values for e and f end up getting stored. Since the item cap calculation is not changed from PUF-growth, the tree will be the same except for the new addition of maximum existential probability values. The remaining tree construction steps are shown in Figure 4.5(c). Notice that adding new paths happens only at the first point where the item values in any existing path no longer match up with those in the prefix of the current transaction (regardless of the I^{Cap} or existential probability values). The final Tube-P tree is shown in Figure 4.6(b). As in FP-growth [13], horizontal node traversal pointers are maintained between nodes with the same item and are accessible through a header table. The header table for a Tube-P tree maintains the same values as for a PUF-tree. The header table for the global Tube-P tree in this example is given in Figure 4.6(a).



(a) Adding t_1



(b) Adding t_2

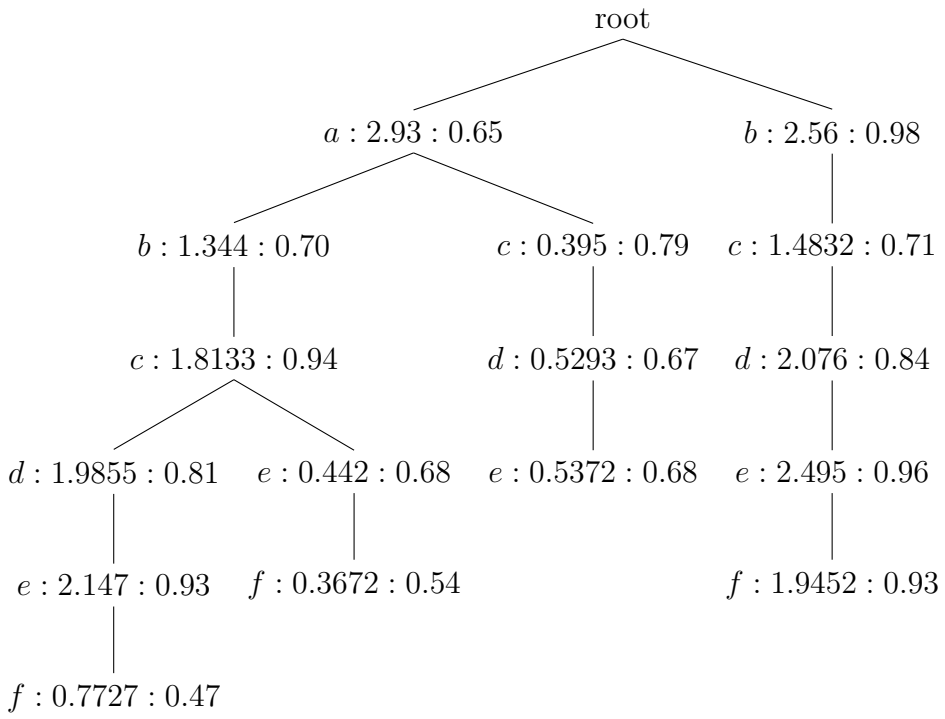


(c) Adding t_3, t_4 and t_5

Figure 4.5: Tube-P Tree Construction, $minsup = 2$

Item	**Σ Prefixed Item Cap**	**Pointer to Tube-P Tree Node, e.g.**
a	2.93	0x100305260
b	3.904	0x100305160
c	3.6915	0x100305660
d	4.5908	0x1003051b0
e	5.6212	0x1003055c0
f	3.0851	0x100305620

(a) Tube-P Tree Header Table



(b) The Full Tube-P Tree

Figure 4.6: The Tube-P Tree and Header Table, $minsup = 2$

4.2.2 Mining with Tube-P

To mine frequent patterns, Tube-P extracts tree paths using a recursive algorithm, extending patterns upwards to the root while keeping track of the prefixed item caps and maximum existential probability values. Let us consider $\{e\}$-conditional pattern bases in this example. First we check that the total prefixed item cap for e in the tree is greater than *minsup* in the header table (cf. Figure 4.6(a)). Since it is, we can proceed to directly generate the $\{e\}$-projected tree without taking an extra step to calculate the expected support as we had to do in UF-growth. Starting from the global tree in Figure 4.6(b), the $\{e\}$-projected tree is created by as shown in Figure 4.7(b) by following all the branches with an e in them up to the root while replacing the prefixed item caps of those nodes that are ancestors of an e node with the sum of the item caps of all their children e nodes; and leaving the existential probability values of those nodes unchanged. A quick glance at the header table shows that the sum of prefixed item caps for each item is greater than *minsup* so every 2-itemset created from e and one of the items in the $\{e\}$-projected header table becomes a candidate.

For each higher level of recursion, the maximum existential probability value in each node is used to tighten the upper bound and reduce the number of candidate frequent patterns. For the $\{e, d\}$-projected tree, we first check if the sum of the prefixed item cap is greater than *minsup* in our $\{e\}$-projected tree. This is the case in our example, so we can proceed to generate the $\{e, d\}$-projected tree as shown in Figures 4.7(c) and 4.7(d). We replace the prefixed item cap in each node that is an ancestor of a d node with the *sum of products* of its maximum existential probability value and the item caps of its children d nodes; and leave the existential

probability values of those nodes unchanged. When we do this, we find that the sum of prefixed item caps for $a = 1.74473 < minsup = 2$, therefore a can be pruned from the projected tree. The pruned $\{e, d\}$-projected tree and header table are shown in Figure 4.8(a) and Figure 4.8(b). This is shown as two separate steps for clarity, but in actuality it is not necessary to create a tree and then prune it. While reading the $\{e, d\}$-projected pattern bases all the necessary information is available to create the fully pruned $\{e, d\}$-projected tree in a single step. A quick glance at the header table shows that the sum of prefixed item caps for each item is greater than *minsup* so every 3-itemset created from e, d and one of the items in the $\{e, d\}$-projected header table becomes a candidate.

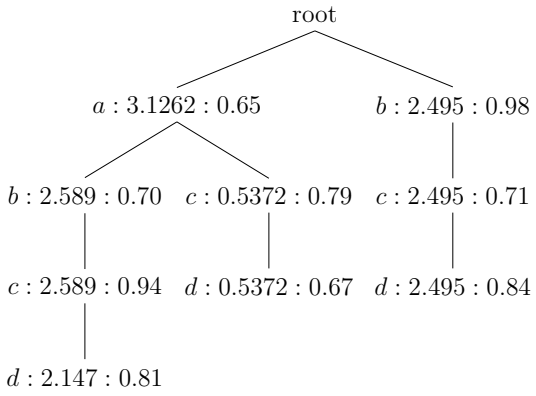
Proceeding deeper into the recursion, we look at the $\{e, d, c\}$-conditional pattern bases and proceed to generate the $\{e, d, c\}$-projected tree in the same manner as the $\{e, d\}$-projected tree as shown in Figures 4.8(c) and 4.8(d). Because the sum of prefixed item cap for each item in the $\{e, d, c\}$-projected header table is greater than *minsup*, every 4-itemset created from e, d, c and one of the items in the $\{e, d, c\}$-projected header table becomes a candidate (i.e., pattern $\{e, d, c, b\}$ is recorded as a candidate frequent pattern). At this stage, since the tree consists of only a single level, the recursion unwinds and the Tube-P algorithm generates other top level projected trees from the remaining items in the global tree header table.

All of the candidate frequent patterns generated during the mining are then checked against the input database one final time to calculate their true expected support. The final list of frequent patterns and expected support values is the same as in UF-growth (cf. Table 3.6). However, unlike PUF-growth, Tube-P will find

a total of 36 candidate itemsets (cf. 38 in Tube-S) during the mining process and determine that 20 of them are truly frequent.

Item	**Σ Prefixed Item Cap**
a	3.1262
b	5.0840
c	5.6212
d	5.1792

(a) $\{e\}$-projected Header Table

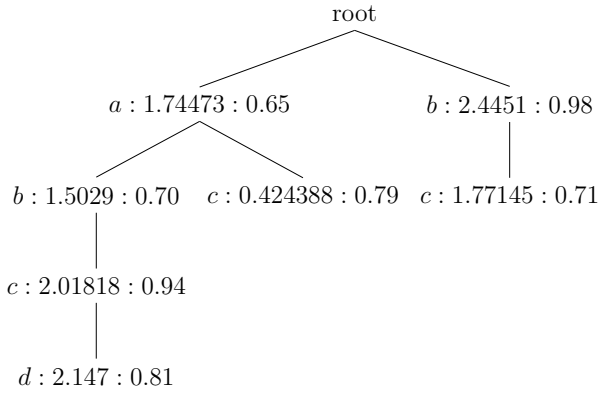


(b) The $\{e\}$-projected Tube-P Tree

Item	**Σ Prefixed Item Cap**
a	1.74473
b	3.948
c	4.214018

(c) $\{e, d\}$-projected Header Table

(Before Pruning)



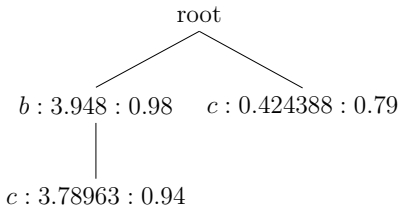
(d) The $\{e, d\}$-projected Tube-P Tree

(Before Pruning)

Figure 4.7: Tube-P Mining, $minsup = 2$

Item	Σ Prefixed Item Cap
b	3.948
c	4.214018

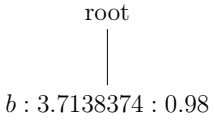
(a) $\{e, d\}$-projected Header Table (After Pruning)



(b) The $\{e, d\}$-projected Tube-P Tree (After Pruning)

Item	Σ Prefixed Item Cap
b	3.7138374

(c) $\{e, d, c\}$-projected Header Table



(d) The $\{e, d, c\}$-projected Tube-P Tree

Figure 4.8: Tube-P Mining (cont.), $minsup = 2$

4.3 Summary

In this chapter, I introduced Tube-S and Tube-P as two new algorithms for tree-based uncertain data frequent pattern mining. Tube-S stores three values in each node, an item, an item cap and an M_2 value, which is the second highest existential probability in the prefix of the item (for the node under consideration) in each transaction passing though a node. Paths are shared in the Tube-S global tree so long as the items in a transaction have the same prefix as an existing path. Tube-S mines frequent patterns in a fashion similar to PUF-growth [25], however for each candidate k-itemset, $k \geq 3$, the upper bound to expected support, $expSup^{Cap}$, is compounded by multiplying the item cap with its M_2 value. On the other hand, Tube-P also stores three values in each node, but instead of an M_2 value it stores the maximum exis-

tential probability of that node's item among all the transactions passing through it. The path sharing conditions are the same for Tube-P as for Tube-S, but the mining is quite different. Instead of propagating suffix items' max existential probabilities upwards in projected trees, only the item caps are propagated thusly. For each candidate k-itemset, $k \geq 3$, $expSup^{Cap}$ multiplies the item cap with the max existential probability value already in the node. Tube-S and Tube-P both have tighter upper bounds than PUF-growth, but as they are still upper bounds, a final database scan after the mining is still required to filter out any remaining false positives.

Chapter 5

Evaluation

First, I perform a theoretical analysis of my Tube-S and Tube-P algorithms and compare the results with existing algorithms. Next, I perform an extensive experimental analysis using multiple different input databases to understand the runtime and memory usage behaviour of my algorithms under different configurations of the input parameters.

5.1 Theoretical Analysis

In terms of memory space, Tube-S and Tube-P store the same amount of information in each node of their trees: the item value, the prefixed item cap, the tube value (either M_2 or existential probability, respectively) and node pointers. Based on this, to definitively call one algorithm the winner I must consider the total number of nodes when comparing memory usage between the two. In the examples in the previous sections, nodes were ordered alphabetically based on the item value, but that

is not a requirement of the algorithms. As long as a global ordering of any kind is adhered to, the correctness of results is guaranteed according to Lemmas 4.2 and 4.4. In fact, other orderings, such as the descending order of expected support of singleton itemsets, lead to lower node counts than a straightforward alphabetical or numerical ordering on item IDs. This is the case because descending order of expected support places the most frequent items closer to the root of the tree, leading to more instances of path sharing between these items. For my comparisons, I assume the same ordering between all the algorithms I compare against. In terms of the number of nodes in the global trees, Tube-S and Tube-P are identical. It is only when considering the nodes in the projected trees that there is a difference.

The number of nodes in the projected trees is directly related to characteristics of the input database and the *minsup* parameter. A trivial case would be to set *minsup* to zero to find every pattern in the entire database; no nodes would ever be pruned! On the other hand, I could set *minsup* so high that no frequent itemsets of cardinality two or higher are ever found, thus no projected trees would be created. The number of projected trees, and thus to a large extent the total number of nodes used in any tree-based frequent pattern mining algorithm depends directly on the tightness of the upper bounds used. Therefore, I use the amount of information in each node and the tightness of upper bounds on expected support as metrics to compare these algorithms.

Unfortunately, for Tube-S and Tube-P, it is not possible to give a blanket statement that one has a tighter bound than another. For some input databases Tube-S is the tighter bound, and for others Tube-P is the tighter bound. Consider a database

with only two transactions, $t_1 = \{a : 0.1, b : 0.8, c : 0.5\}$ and $t_2 = \{a : 0.9, b : 0.2, c : 0.5\}$. In this situation, to calculate the upper bound on expected support for the itemset $\{a, b, c\}$, Tube-S multiplies the I^{Cap} value of c by its M_2 value, 0.2, whereas Tube-P multiplies the I^{Cap} value of c by the maximum existential probability of a, 0.9. Clearly Tube-S is the winner here. In fact, for transactions of length three, the bound provided by Tube-S is guaranteed to be tighter than or equal to the bound provided by Tube-P. This is because, for transactions with very few items, the maximum existential probability of a single item is more likely to be higher than the maximum of all the second highest existential probabilities (regardless of item). As the length of transactions increases, it is much more likely that the maximum of any single item is lower than the maximum of all the second highest probabilities (regardless of item). Recall the example from Section 4.2.2 where this was the case. Of course, probability distribution comes into play here, which is why experiments with different probability distributions are so important.

Table 5.1 below shows the rankings for each algorithm based on these metrics. The algorithms with tighter bounds appear above those with looser bounds in the table. It follows that with tighter upper bounds, fewer nodes will be created due to more pruning of infrequent itemsets, therefore those algorithms with tighter upper bounds not only have lower memory usage but also lower runtimes.

Table 5.1: Memory Usage/Runtime Ranking of Tree-based Algorithms

Algorithm	**# items/node**	**Tightness of upper bound**
UF-growth [23]	4	exact bound
CUFP-growth [27]	2^d	exact bound
Tube-P	3	prefixed item cap and max probability
Tube-S	3	prefixed item cap and M_2
CUF-growth* [24]	3	transaction cap and bronze value
PUF-growth [25]	2	prefixed item cap
CUF-growth [24]	2	transaction cap
UFP-growth [2]	$1 + 2k$	favours higher max probabilities near root

where d is the depth of the current node when the root node has depth $d = 0$, and k is number of clusters of item probabilities in each node.

5.2 Experimental Analysis

In my experimental analysis, I first describe some characteristics of the input databases I used and describe the methodology applied to create my synthetic data. Then, I run three separate experiments, measuring the effect of *minsup* on run-time, on memory usage (measured as number of nodes) and on false positives broken down into cardinality level, for different sizes of input databases across three algorithms. The algorithms I tested are Tube-S, Tube-P and CUF-growth* [24]. CUF-growth* is an optimization of CUF-growth [24] given in the original paper that keeps an extra "bronze value" in each node that is the maximum of all the third highest existential probability values in those transactions passing through a node. I do not test directly against PUF-growth [25] since it does not make use of any extra compounding values in its nodes. Previously published experiments have already

shown that the memory savings of storing one less value in each node are completely negated by excessive extra nodes created in subtrees due to the much looser bound on expected support [21, 22, 24].

As graphs for all the combinations of different parameters tested (close to 500) would take up too much space in the body of this thesis, a selection of graphs summarizing the results are presented here and the remaining graphs are available in Appendix B. As the same trends were observed for all sizes of synthetic data tested over different minsup values, I narrow my focus in this section to a single *minsup* and avoid showing parameterizations of the input databases with similar densities to deliver a more succinct analysis.

5.2.1 Dataset Characteristics

The datasets I used consisted of both real and synthetic data. I generated my own synthetic data using several parameterizations of the Normal distribution, with existential probabilities ranging from narrow, $\sigma = \sqrt{0.03}$, to almost uniform, $\sigma = 1$, centred at 0.25 spaced intervals from zero to one and truncated on the interval [0, 1]. For a visualization of the probability density functions used see Figure 5.1. The densities were chosen to cover a wide range from very sparse to very dense. The properties I manipulated to control density were transaction lengths and number of domain items. For each transaction, transaction lengths were drawn from a separate truncated and scaled Normal distribution ($\mu = 0.5$, $\sigma = \sqrt{0.01}$) and item values were chosen from a scaled uniform distribution and filtered to remove repeats. There were one million total transactions in each dataset. Smaller sized datasets were tested by

reading only the first 10000, 250000, 500000 and 750000 transactions from each of the previous files.

I also used real life databases from the Frequent Itemset Mining Dataset Repository (`http://fimi.ua.ac.be/data/`), specifically: mushroom, kosarak and retail. For each of these datasets I added existential probability values for each item from the same 15 parameterizations of the Normal distribution used for my synthetic data but otherwise left them unchanged in order not to remove any effects to due the distribution of the items themselves. For a full list of the minimum, maximum and average number of items per transaction; and database density for each input database used, see Tables A.2 and A.3 in Appendix A.

5.2.2 Experimental Setup

The experiments were conducted on Amazon Web Services r3.xlarge instances, each with an Intel® Xeon® E5-2670 v2 2.50GHz CPU, 30.5GiB memory and 80GB SSD storage. A default, up to date, Ubuntu Server 14.04 LTS image was used. The programs were compiled with g++ using the -std=c++0x and -O3 optimization flags.

5.2.3 Correctness Analysis

To assure that CUF-growth*, Tube-S and Tube-P returned the same set of frequent patterns as the precise data algorithm FP-growth and the uncertain data algorithm UF-growth, I ran multiple datasets with all algorithms and confirmed that the output frequent patterns and expected supports were identical within a small epsilon to account for rounding errors in the floating point calculations. To accommodate for

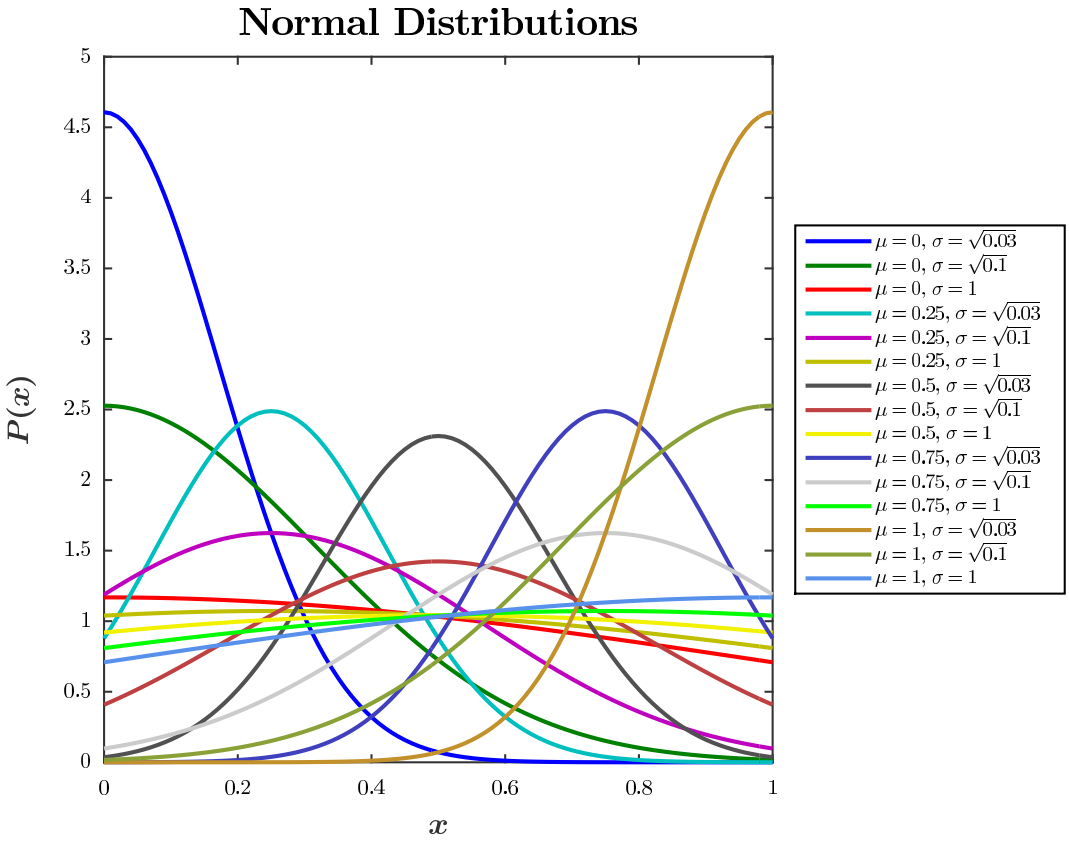


Figure 5.1: Normal Distribution Parameterizations

the precise data aspect of FP-growth, all existential probabilities were set to 100%.

5.2.4 Runtime Analysis

To measure the runtime of the different algorithms, I tested multiple synthetic and real databases of different sizes at different levels of *minsup*. The graphs below show the runtime of synthetic datasets with different probability distributions and densities for each of the three algorithms.

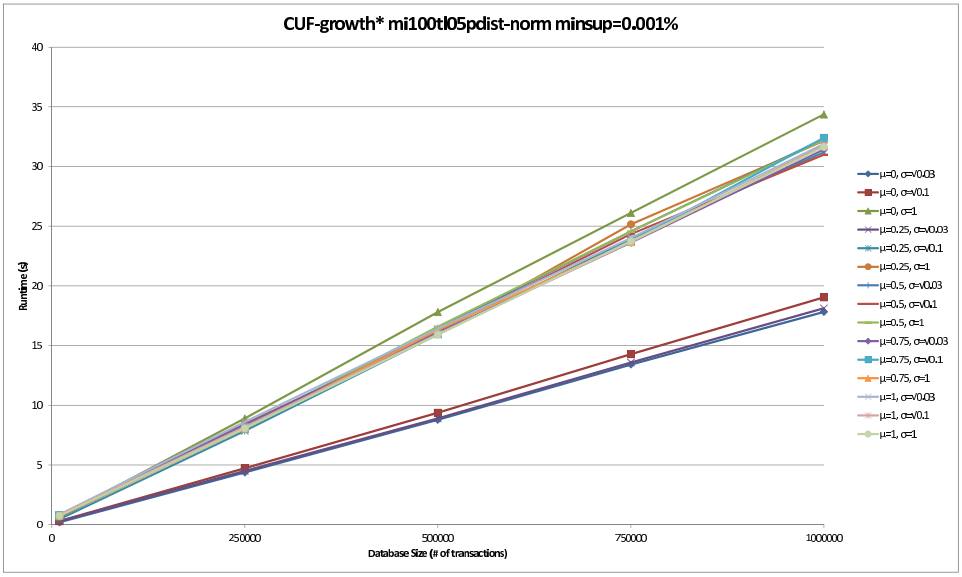
Transaction Length 5

Consider the graphs in Figures 5.2(a), 5.2(b) and 5.3(a). There is a readily observable relationship that an increase in database size leads to an increase in runtime. For this set of synthetic data, the transaction length is quite low, an average of about five, and the number of domain items is 100, leading to a density of about 0.05. CUF-growth* actually runs faster than Tube-S and Tube-P, with Tube-P taking a close second place. This is not what I would initially expect, but when considering the extremely small number of domain items, the advantages gained by a tighter upper bound in Tube-S and Tube-P when considering only portions of a transaction are easily lost. This is weighed against a slightly more expensive global tree setup cost due to an optimization in my code that removes items in the global tree with no frequent extensions [31]. In this case, with no extra advantage provided by a tighter upper bound, it would have been more beneficial to forgo the optimization for databases with few domain items, as the reduction in number of nodes and false positives does not make up for the extra cost, however, I wanted to keep the algorithms the exact same for all input data.

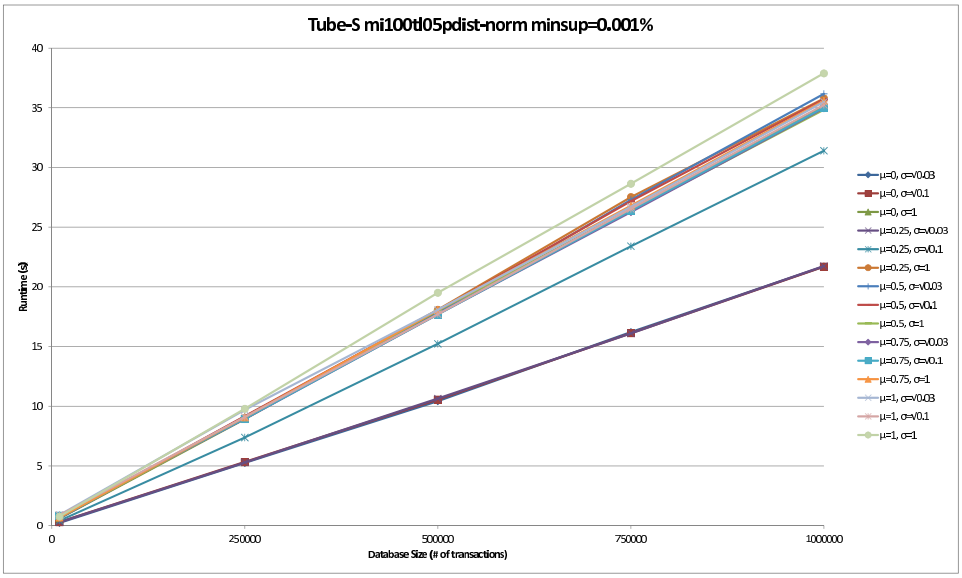
In terms of probability distribution, notice that a high variance (1) leads to higher run-times no matter where the distribution is centred. A high variance causes existential probability values to be almost uniformly distributed, leading to a high number of frequent patterns. As the variance is reduced, the mean becomes a more important determining factor, with higher means leading to higher runtimes. This is because a combination of a high mean and a low variance means that most existential probabilities will be high. On the other hand, a low mean and variance mean that most

existential probabilities will be low, leading to many infrequent patterns being pruned early in the algorithm and reducing the runtime. This effect will become even more clear with larger transaction sizes and discussed in more detail below.

Figures 5.3(b), 5.4(a) and 5.4(b) show the runtime when increasing the number of domain items to 1,000 but keeping the other parameters the same (effectively lowering the density by a factor of 10). In almost all cases, Tube-S is slightly faster than Tube-P, which is in turn faster than CUF-growth*. In this set of graphs, one distribution stands out. ($\mu = 1$, $\sigma = \sqrt{0.03}$) seems to spike to a higher runtime for sizes of 250k to 750k, then drop off for the largest size of a million transactions. In fact, this distribution on average is given to generating a lot of high valued existential probabilities. In this case, there is not much opportunity for pruning as the upper bound and the actual expected support are very close for most itemsets. The runtime drops off for the largest size because the extra transactions lower the gap relative to the increase in *minsup*. I can be sure this is the case due to *minsup* being reported in percent. Using a percentage of the database size allows me to combine different database sizes in the same graph since the relative frequency for an itemset to be considered frequent is the same. Instead of measuring an increase of runtime because of an obvious increase in item frequency leading to more frequent patterns, this allows me to notice effects due to the actual file I/O and memory use of my algorithms caused by the underlying distributions that would otherwise be hidden.



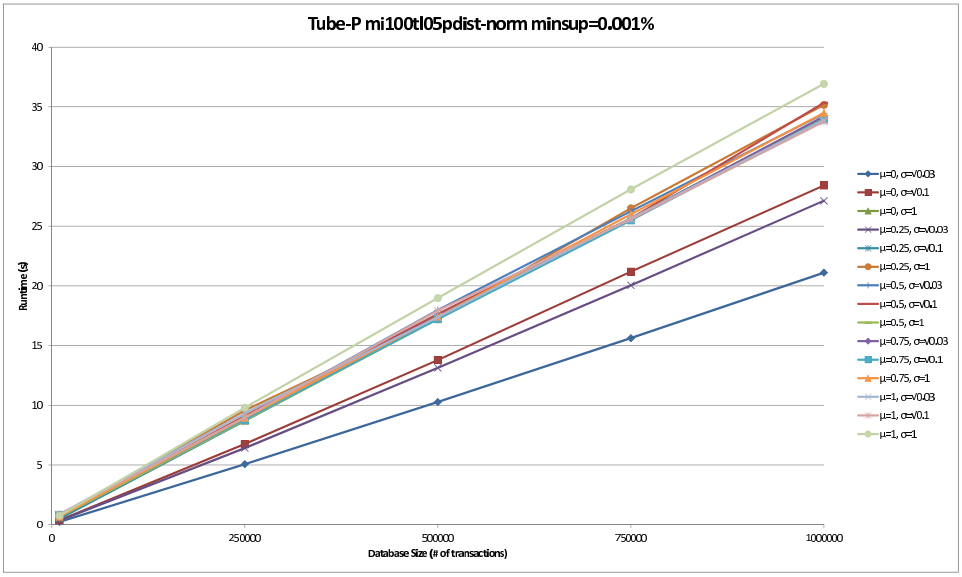
(a) CUF-growth*, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$



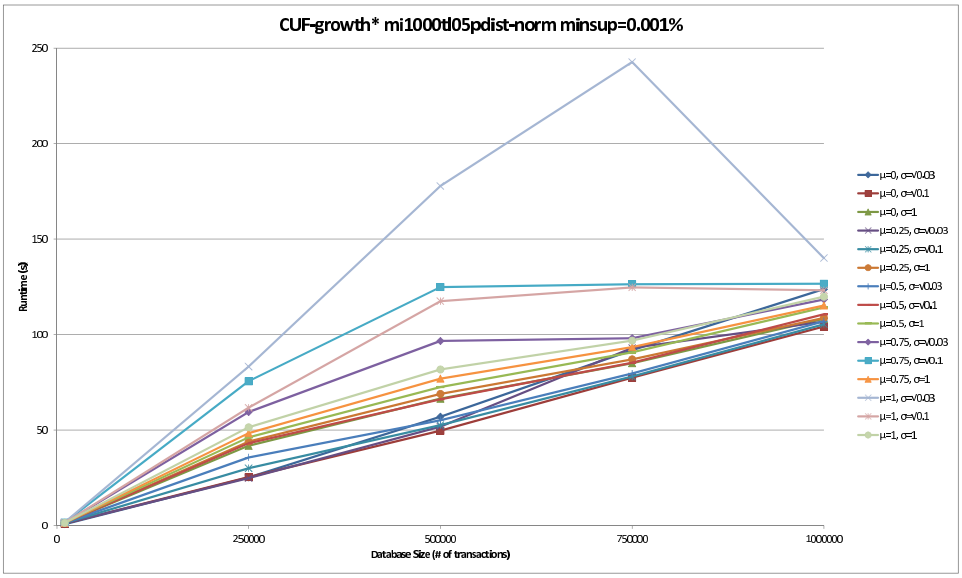
(b) Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$

Figure 5.2: Runtime: CUF-growth* & Tube-S

$|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$



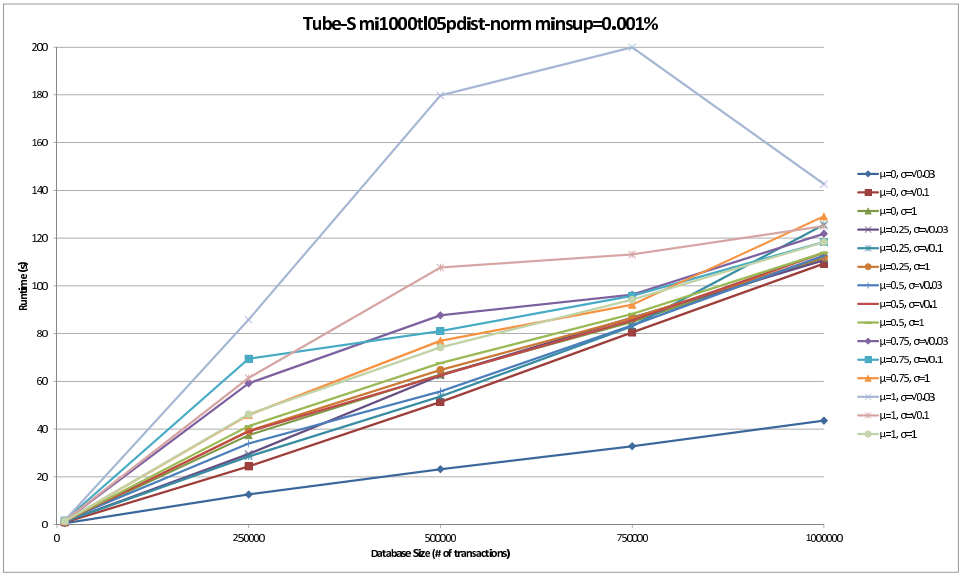
(a) Tube-P, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$



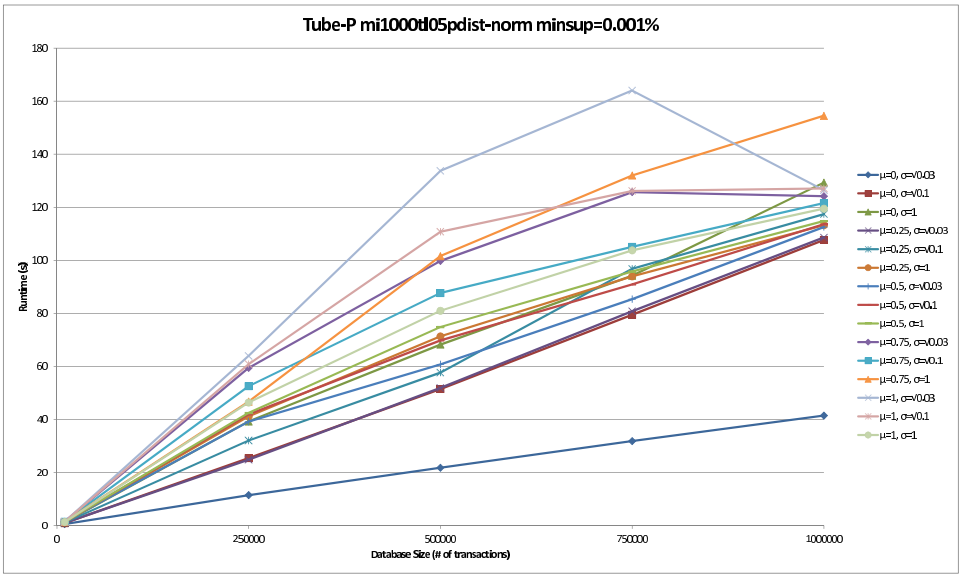
(b) CUF-growth*, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$

Figure 5.3: Runtime: Tube-P ($|\mathcal{I}| = 100$) & CUF-growth* ($|\mathcal{I}| = 1000$)

$\overline{|t_j|} = 5$, $minsup = 0.001\%$



(a) Tube-S, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$



(b) Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$

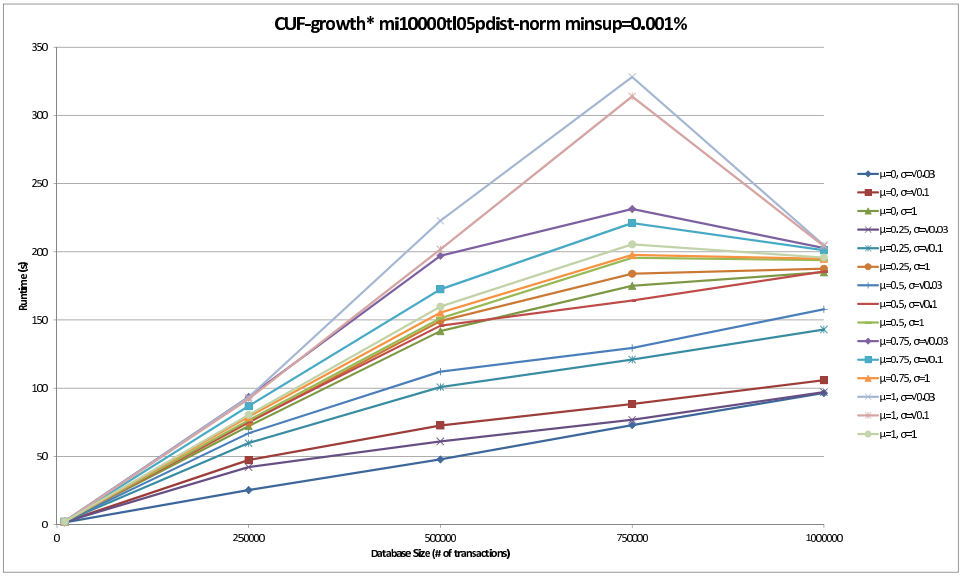
Figure 5.4: Runtime: Tube-S & Tube-P

$|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$

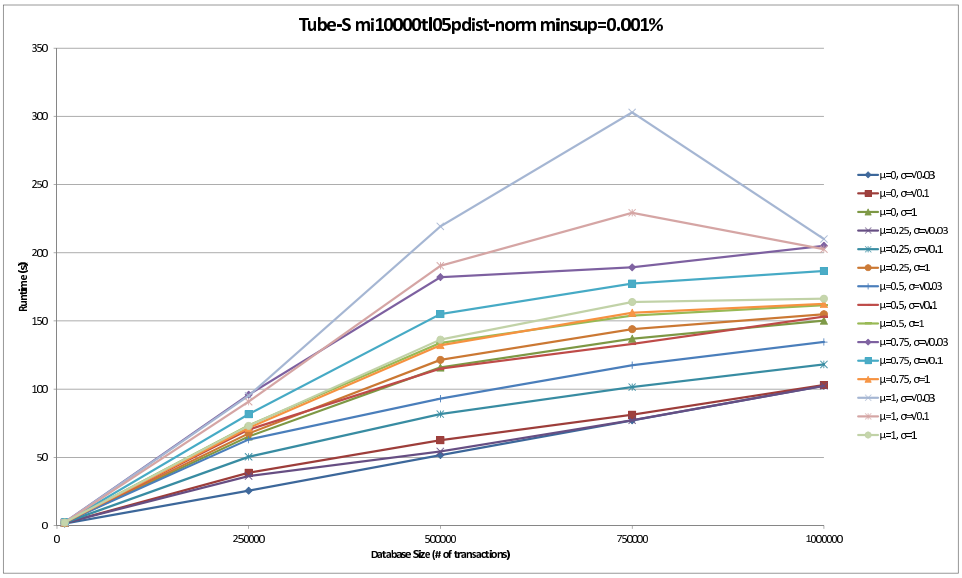
Figures 5.5(a), 5.5(b) and 5.6 show the runtime when increasing the number of domain items to 10,000 but keeping the other parameters the same (effectively lowering the density by another factor of 10). There is a more pronounced version of the effects noticed with one thousand domain items, both in terms of the spikes for distributions with high means, and in terms of greater distances between the runtimes for each distribution. At this stage, Tube-S and Tube-P are almost indistinguishable, but Tube-S still performs slightly better than Tube-P which in turn is noticeably faster than CUF-growth*. This is an affect of the low transaction length, which leads to low cardinality frequent patterns. In fact, for a transaction length of three, Tube-P's bound is guaranteed never to be tighter than Tube-S's. This is because if the second highest existential probability in the prefix of the terminating item is not always the second highest over all the transactions that share that path in the tree, it must be the highest in at least one transaction, therefore leading to a looser bound when Tube-P uses the higher value. The probability of this happening decreases significantly with longer transaction lengths and higher cardinality frequent patterns, however, as the max second highest existential probability may be drawn from an item not being considered in the current candidate pattern, while items actually in the pattern have lower single item maxes. Also noticeable at this stage is that there is enough separation between the runtimes that it now becomes possible to rank the different distributions in terms of lowest to highest runtime. Table 5.2 below shows the ordering.

Table 5.2: Distribution Rankings from Highest to Lowest Runtime

$\boldsymbol{\mu}$	$\boldsymbol{\sigma}$	**Ranking**
1	$\sqrt{0.03}$	15 (Highest)
1	$\sqrt{0.1}$	14
0.75	$\sqrt{0.03}$	13
0.75	$\sqrt{0.1}$	12
1	1	11
0.75	1	10
0.5	1	9
0.25	1	8
0.5	$\sqrt{0.1}$	7
0	1	6
0.5	$\sqrt{0.03}$	5
0.25	$\sqrt{0.1}$	4
0	$\sqrt{0.1}$	3
0.25	$\sqrt{0.03}$	2
0	$\sqrt{0.03}$	1 (Lowest)



(a) CUF-growth*, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$



(b) Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$

Figure 5.5: Runtime: CUF-growth* & Tube-S

$|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$

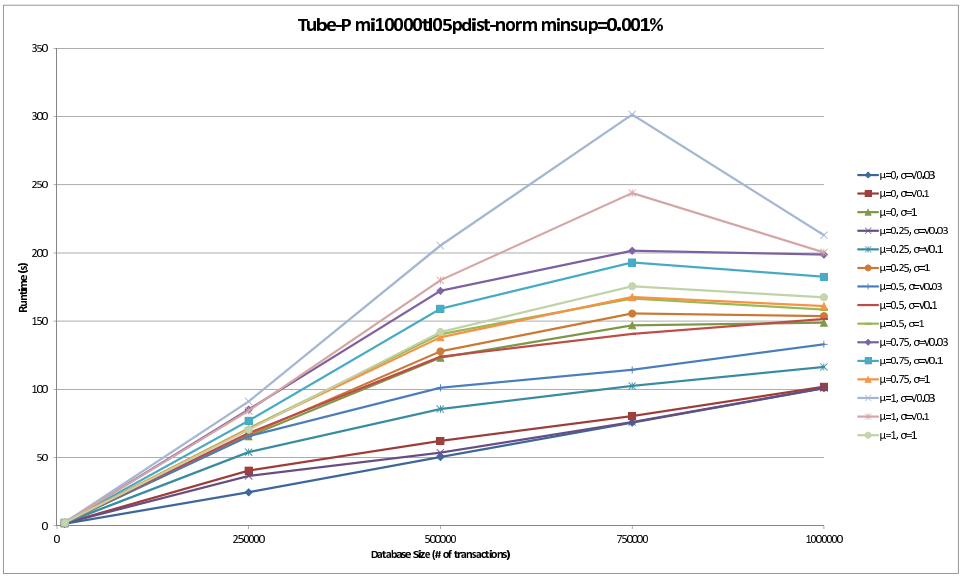


Figure 5.6: Runtime: Tube-P, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$

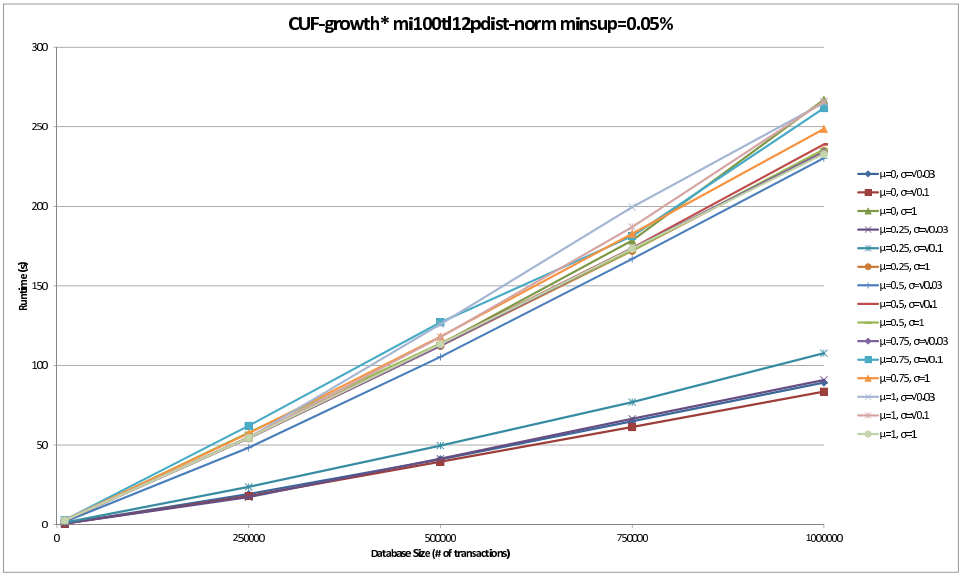
Transaction Length 12

Consider the graphs in Figures 5.7(a), 5.7(b) and 5.8(a). With a higher transaction length and 100 domain items, the density is now about 0.118, slightly more than double that with the same domain items for a transaction length of five. There are some other differences due to the longer transaction length. With a transaction length of five, CUF-growth* performed better than Tube-S and Tube-P with almost every distribution. That is no longer the case, with Tube-S and Tube-P beating or matching CUF-growth* in almost every distribution. Previously, most distributions were separated into two distinct clusters, with the majority taking a higher runtime. CUF-growth* continues this behaviour, but Tube-S and Tube-P take a bite out of that top cluster and spread it out more evenly with lower runtimes, keeping the bottom cluster the same. With a longer transaction length, the frequent patterns also get longer, allowing Tube-S and Tube-P to make use of their tighter upper bounds.

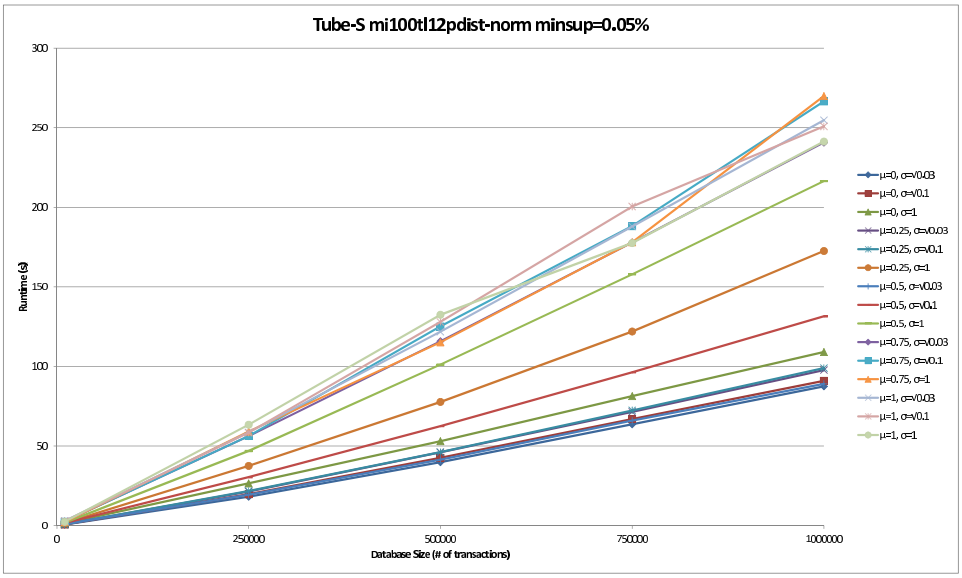
This is most effective for Tube-S on distributions: ($\mu = 0$, $\sigma = 1$), ($\mu = 0.25$, $\sigma = 1$), ($\mu = 0.5$, $\sigma = \sqrt{0.03}$), ($\mu = 0.5$, $\sigma = \sqrt{0.1}$) and ($\mu = 0.5$, $\sigma = 1$); and for Tube-P on distributions: ($\mu = 0$, $\sigma = 1$), ($\mu = 0.25$, $\sigma = 1$), ($\mu = 0.5$, $\sigma = \sqrt{0.03}$), ($\mu = 0.5$, $\sigma = \sqrt{0.1}$), ($\mu = 0.5$, $\sigma = 1$), ($\mu = 0.75$, $\sigma = 1$) and ($\mu = 1$, $\sigma = 1$). Interestingly, Tube-P lowers the runtimes of more distributions over CUF-growth* than Tube-S does, but Tube-S lowers the runtimes of ($\mu = 0.5$, $\sigma = \sqrt{0.03}$), ($\mu = 0$, $\sigma = 1$) and ($\mu = 0.5$, $\sigma = \sqrt{0.1}$) more than Tube-P does and Tube-P lowers the runtimes of ($\mu = 0.5$, $\sigma = 1$) more than Tube-S. ($\mu = 0.25$, $\sigma = 1$) is about the same on both Tube-S and Tube-P. Looking back at Table 5.2, Tube-S is more effective on distributions with lower ranks (higher variances and lower means) as transaction

length increases while Tube-P is more effective on distributions with higher ranks (higher variances and higher means). With Tube-P, a high variance and high mean indicates that most existential probabilities are uniformly distributed with a bias to high existential probabilities. In this case, the max of any single item is more likely to be lower than the max of all the second highest items in the prefix of transactions passing through a node. With low means and high variances, the bias is to low existential probabilities and the opposite is true.

Figures 5.8(b), 5.9(a) and 5.9(b) show the runtime when increasing the number of domain items to 1,000 but keeping the other parameters the same (effectively lowering the density by a factor of 10). Here the same effects are observed as when there were 100 domain items, but they are more pronounced. Going from CUF-growth* to Tube-S and Tube-P, instead of lower runtimes by spreading out the top cluster, Tube-S and Tube-P both cause most of the distributions in the top cluster to instead appear in the bottom cluster entirely. One interesting thing to note is that for the smallest database size (10k transactions), CUF-growth* shows a large spike for the two highest ranked distributions. Tube-S lowers the runtimes for this spike a little, but with Tube-P the spike does not even appear, reinforcing the effectiveness of Tube-P on high ranked distributions.



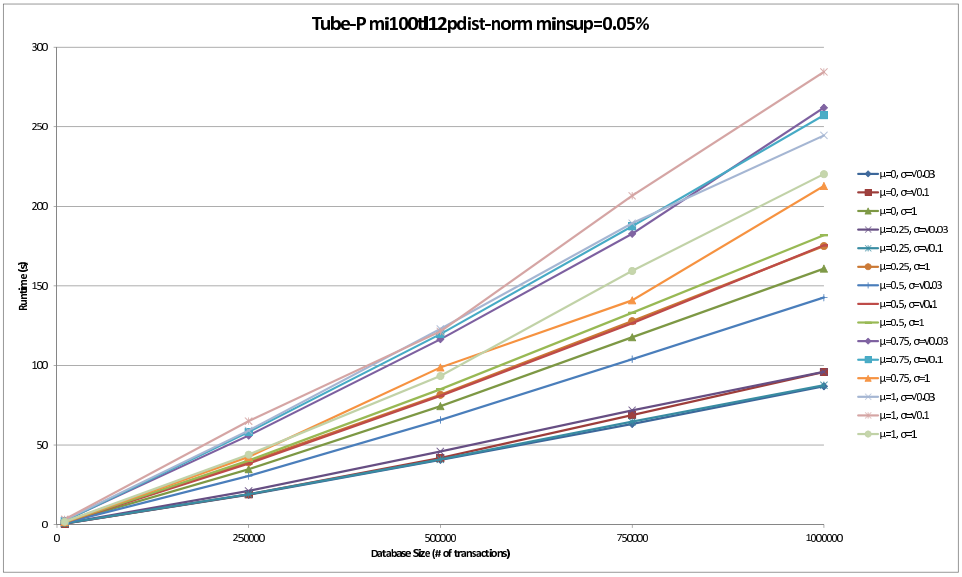
(a) CUF-growth*, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 12$, $minsup = 0.05\%$



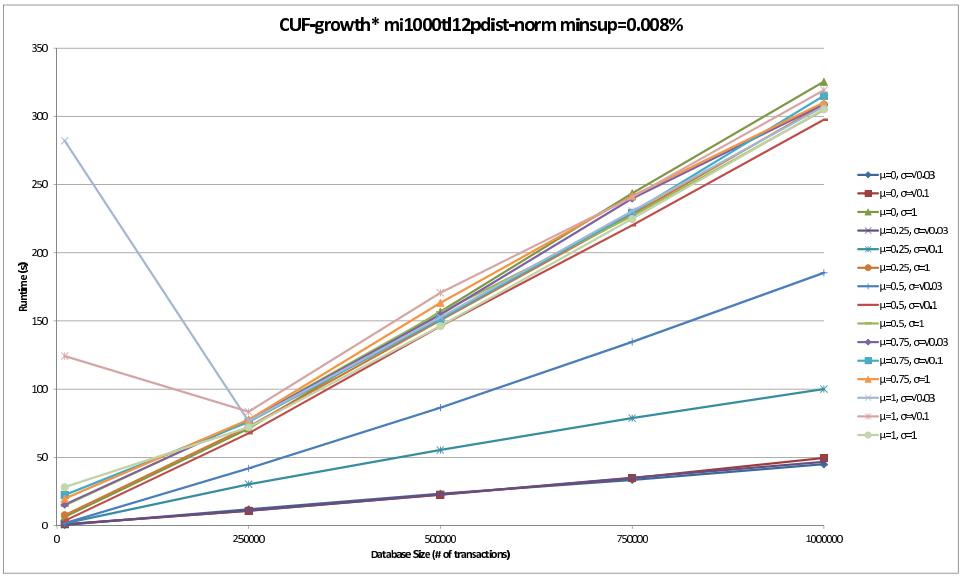
(b) Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 12$, $minsup = 0.05\%$

Figure 5.7: Runtime: CUF-growth* & Tube-S

$|\mathcal{I}| = 100$, $\overline{|t_j|} = 12$, $minsup = 0.05\%$

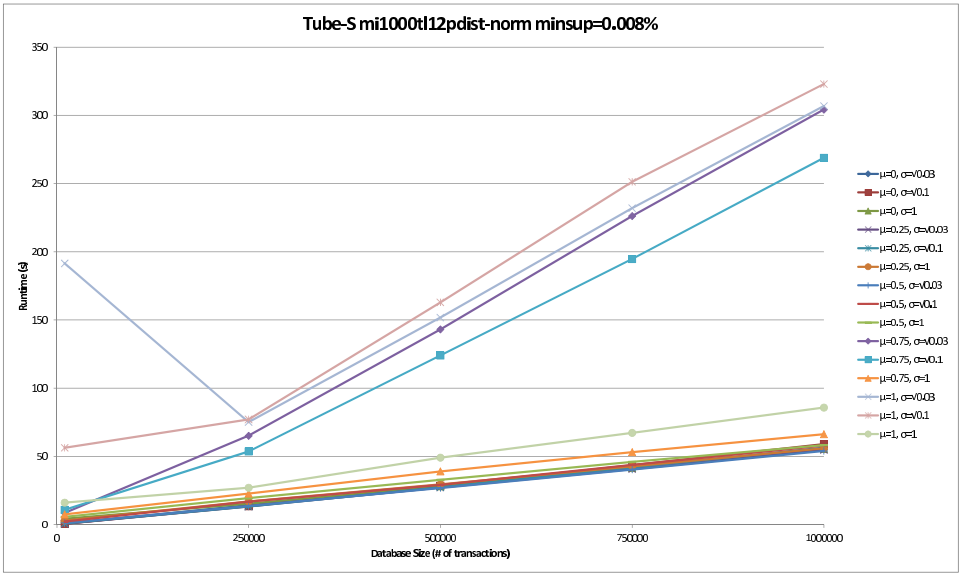


(a) Tube-P, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 12$, $minsup = 0.05\%$

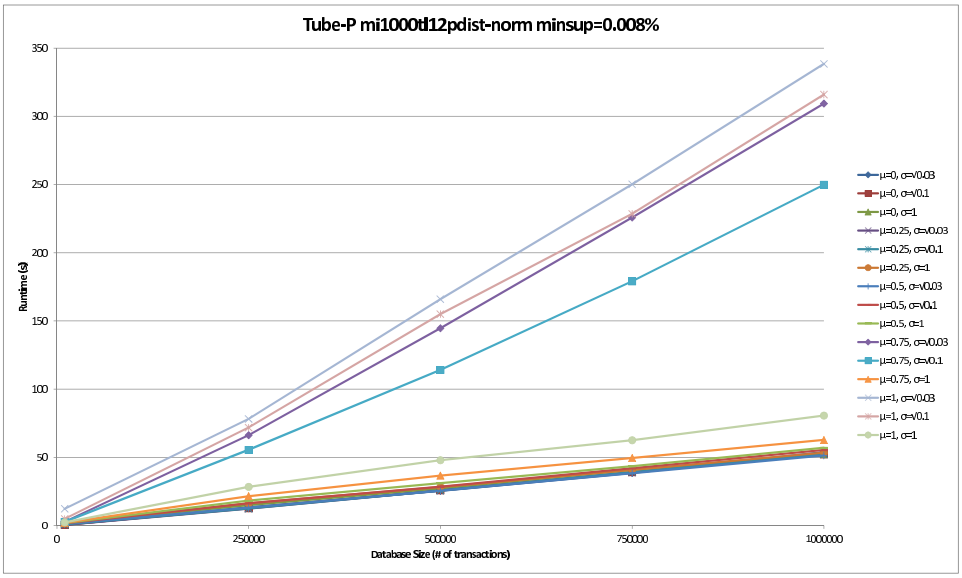


(b) CUF-growth*, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 12$, $minsup = 0.008\%$

Figure 5.8: Runtime: Tube-P ($|\mathcal{I}| = 100$, $minsup = 0.05\%$) & CUF-growth* ($|\mathcal{I}| = 1000$, $minsup = 0.008\%$), $\overline{|t_j|} = 12$



(a) Tube-S, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 12$, $minsup = 0.008\%$

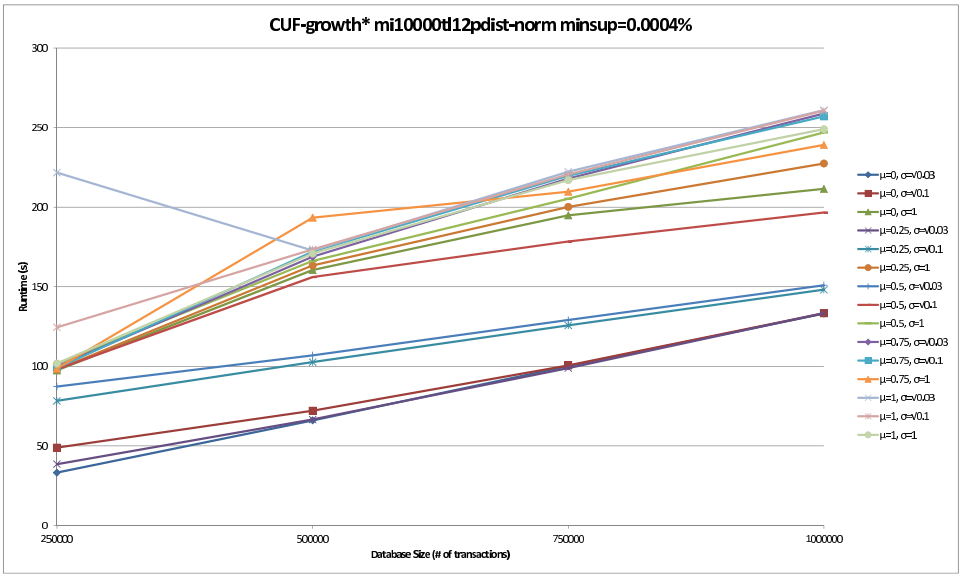


(b) Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 12$, $minsup = 0.008\%$

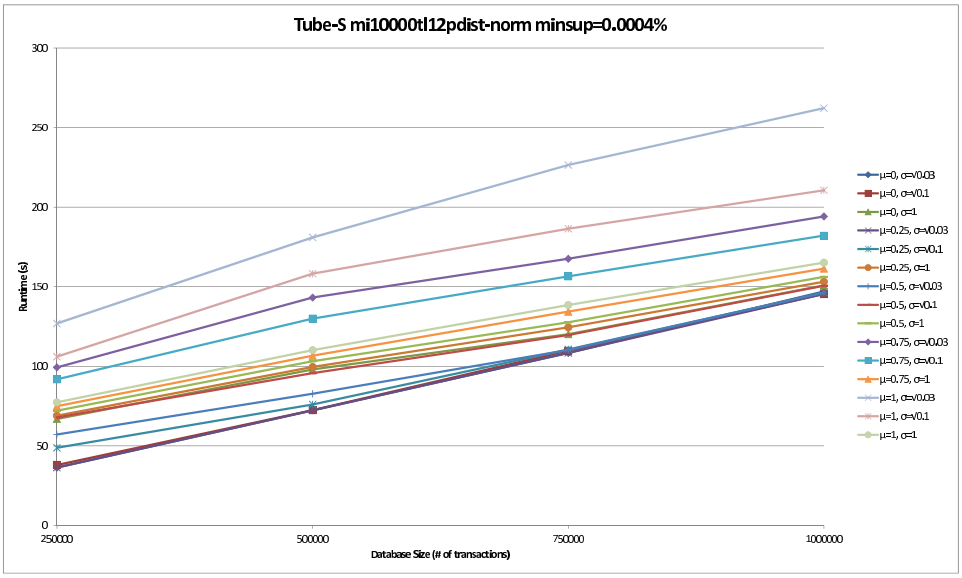
Figure 5.9: Runtime: Tube-S & Tube-P

$|\mathcal{I}| = 1000$, $\overline{|t_j|} = 12$, $minsup = 0.008\%$

Figures 5.10(a), 5.10(b) and 5.11 show the runtime when increasing the number of domain items to 10,000 but keeping the other parameters the same (effectively lowering the density by another factor of 10). Here the same effects are observed as when there were 1000 domain items.



(a) CUF-growth*, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 12$, $minsup = 0.0004\%$



(b) Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 12$, $minsup = 0.0004\%$

Figure 5.10: Runtime: CUF-growth* & Tube-S

$|\mathcal{I}| = 10000$, $\overline{|t_j|} = 12$, $minsup = 0.0004\%$

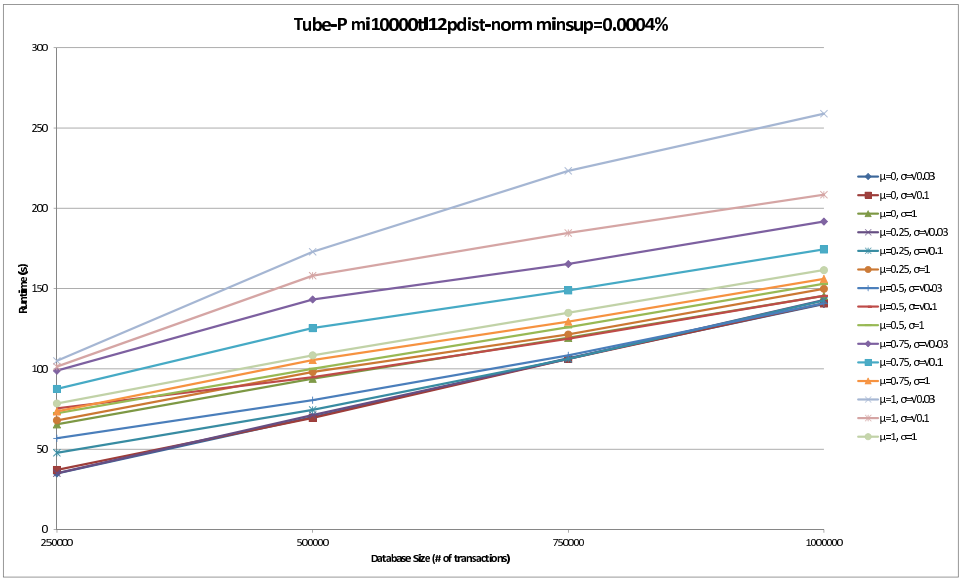


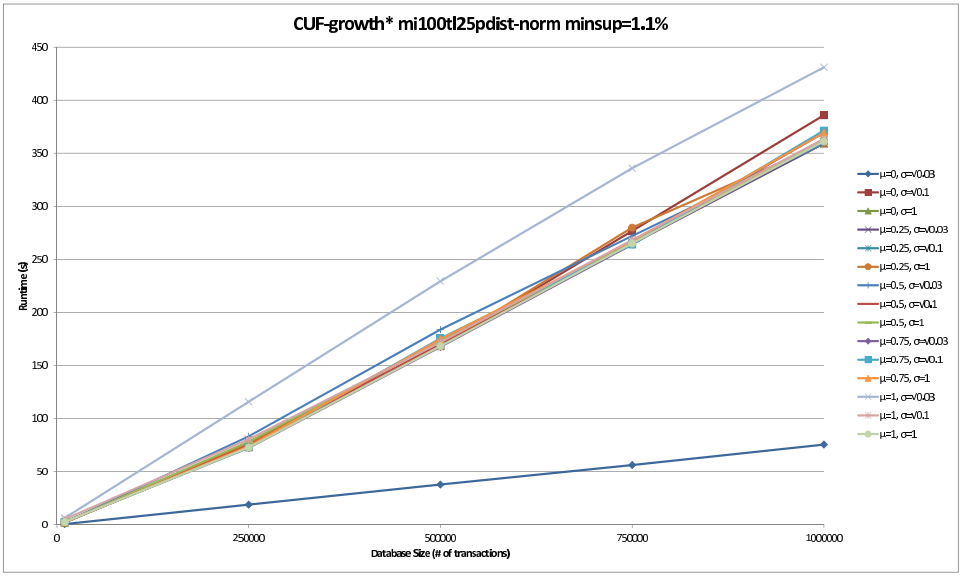
Figure 5.11: Runtime: Tube-P, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 12$, $minsup = 0.0004\%$

Transaction Length 25

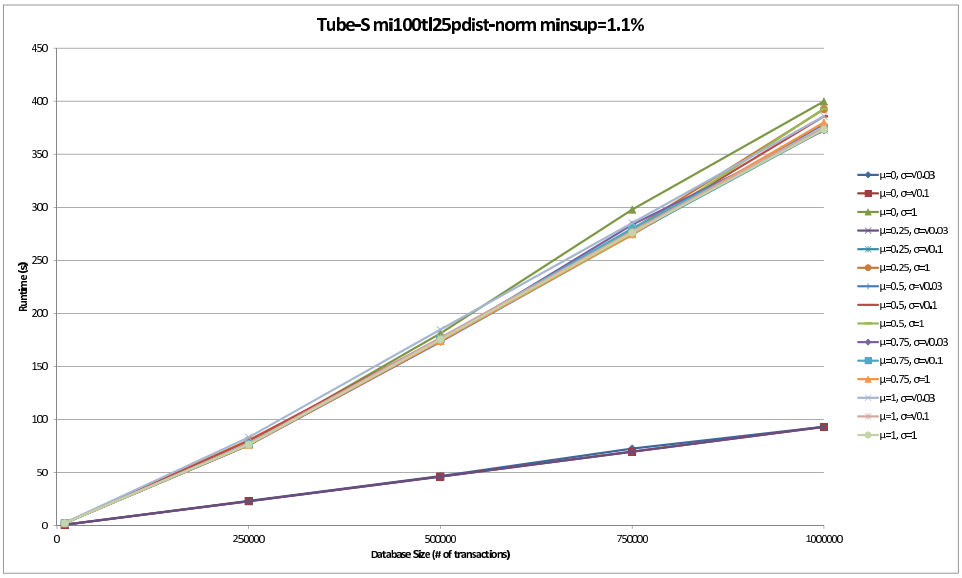
Consider the graphs in Figures 5.12(a), 5.12(b) and 5.13(a). With an even higher transaction length and 100 domain items, the density is now about 0.225, slightly more than double that with the same domain items for a transaction length of twelve. Recall that with 100 domain items and a transaction length of five and twelve, CUF-growth* performed better than or matched Tube-S and Tube-P for runtimes. In these graphs the same two clusters appear, but ($\mu = 1$, $\sigma = \sqrt{0.03}$) appears slightly higher than the main top cluster in CUF-growth*, and ($\mu = 0$, $\sigma = \sqrt{0.1}$) and ($\mu = 0.25$, $\sigma = \sqrt{0.03}$) both appear in the top cluster when previously they usually appeared in the bottom cluster. In this case, Tube-S and Tube-P manage to lower the runtimes of ($\mu = 0$, $\sigma = \sqrt{0.1}$) and ($\mu = 0.25$, $\sigma = \sqrt{0.03}$) back down to the bottom cluster and they lower the runtime of ($\mu = 1$, $\sigma = \sqrt{0.03}$) so it joins the other distributions in the top cluster again. The other distributions have their runtimes slightly increased over CUF-growth*, with CUF-growth* being the fastest, followed by Tube-P and then Tube-S. With only 100 domain items, no matter what the distribution is, items reappear with enough frequency that the advantage of restricting the existential probabilities used in the upper bounds to only a portion of the transaction is easily lost, as even those portions of the transactions will have similar maximum existential probabilities in a path sharing situation. Hence, in these situations, CUF-growth* may be just as effective as Tube-S or Tube-P.

Figures 5.13(b), 5.14(a) and 5.14(b) show the runtime when increasing the number of domain items to 1,000 but keeping the other parameters the same (effectively lowering the density by a factor of 10). As before, increasing the number of domain items

and going from CUF-growth* to Tube-S and Tube-P shows a lowering of runtimes of some distributions in the top cluster. Tube-S and Tube-P both outperform CUF-growth*, with Tube-P just slightly edging out Tube-S in the distributions ($\mu = 0.25$, $\sigma = 1$), ($\mu = 0.5$, $\sigma = 1$) and ($\mu = 0.75$, $\sigma = 1$), all of which are right around the middle in the distribution ranking from Table 5.2 with higher variances.



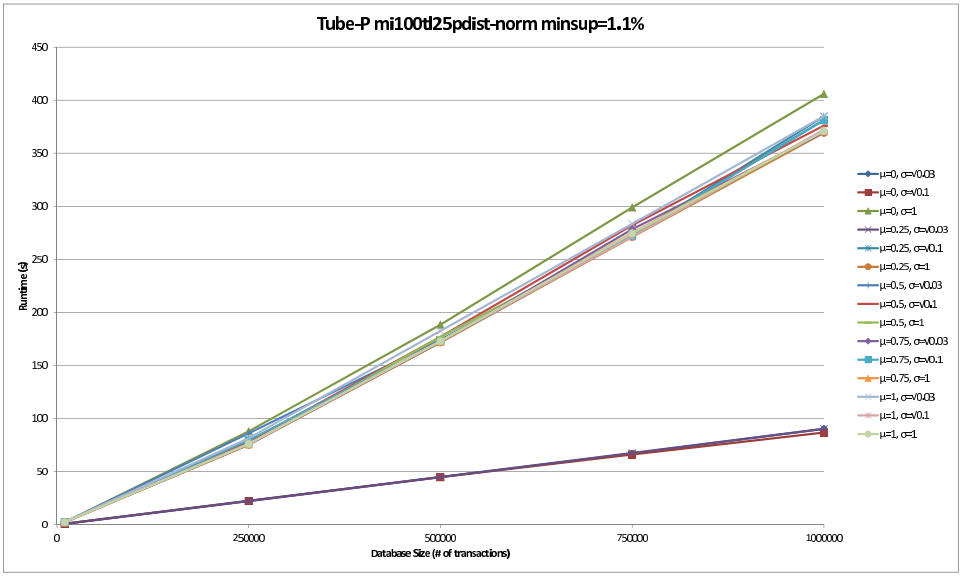
(a) CUF-growth*, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 25$, $minsup = 1.1\%$



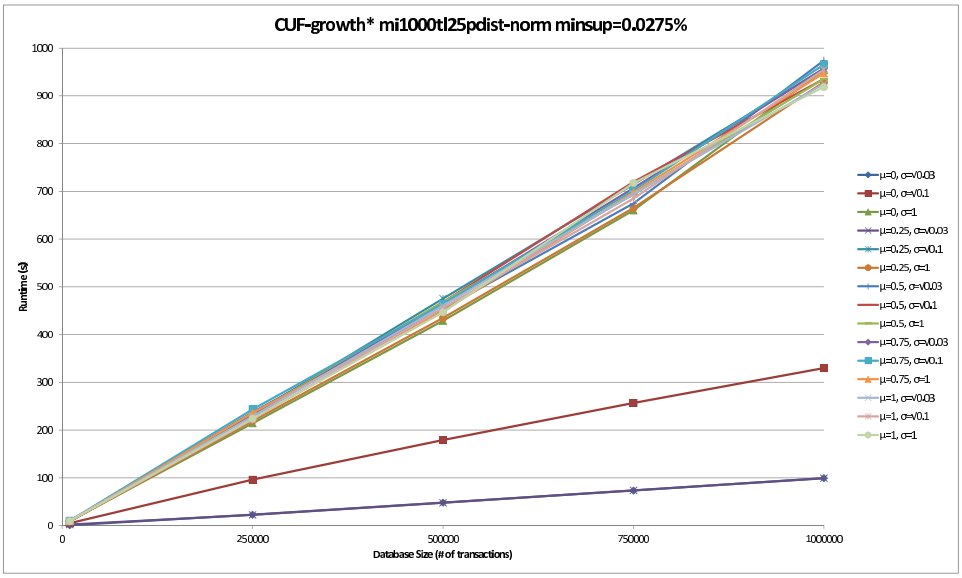
(b) Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 25$, $minsup = 1.1\%$

Figure 5.12: Runtime: CUF-growth* & Tube-S

$|\mathcal{I}| = 100$, $\overline{|t_j|} = 25$, $minsup = 1.1\%$

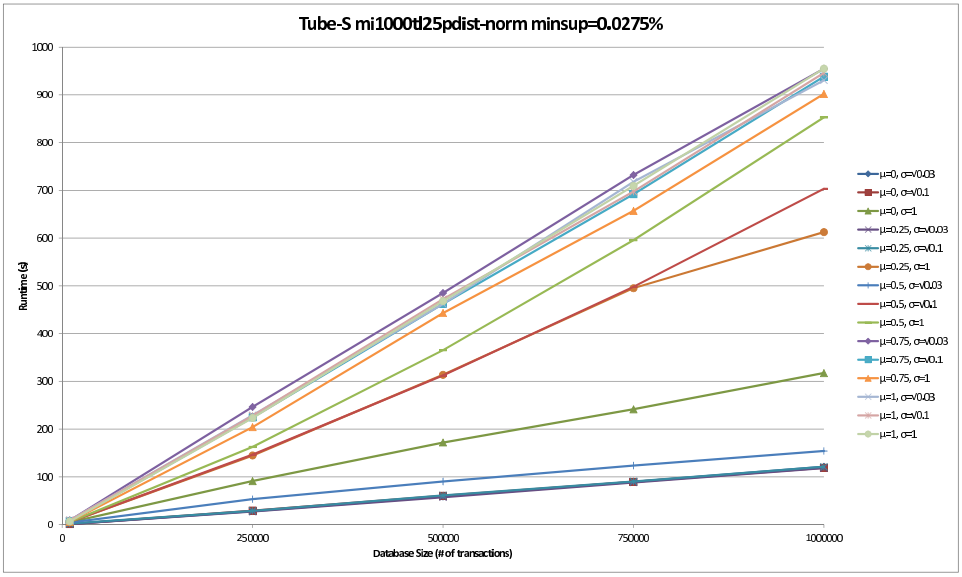


(a) Tube-P, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 25$, $minsup = 1.1\%$

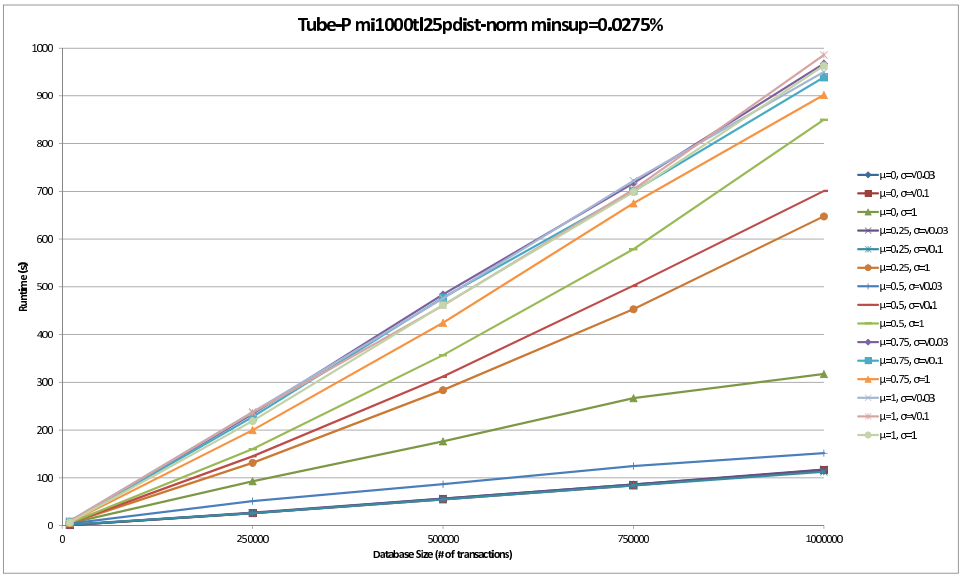


(b) CUF-growth*, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 25$, $minsup = 0.0275\%$

Figure 5.13: Runtime: Tube-P ($|\mathcal{I}| = 100$, $minsup = 1.1\%$) & CUF-growth* ($|\mathcal{I}| = 1000$, $minsup = 0.0275\%$), $\overline{|t_j|} = 25$



(a) Tube-S, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 25$, $minsup = 0.0275\%$

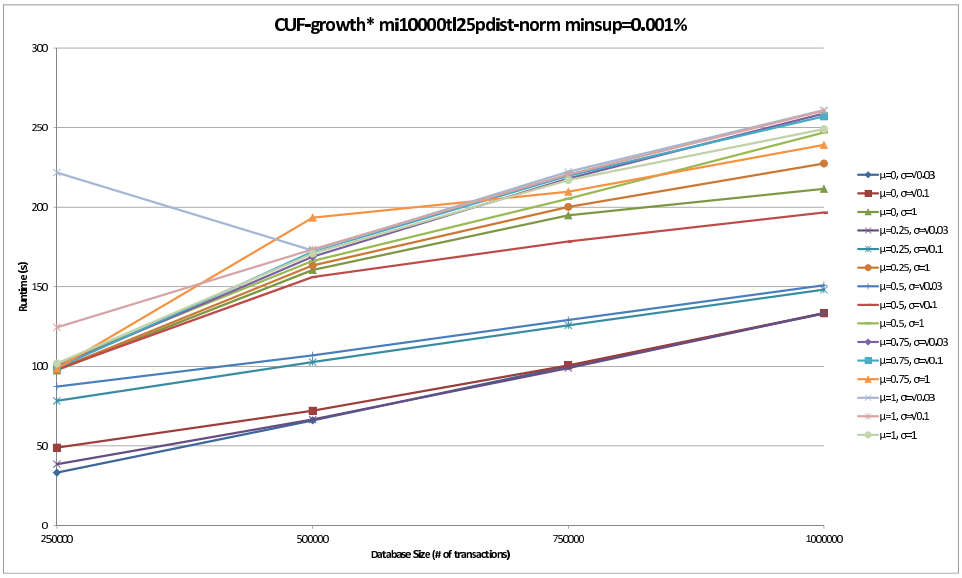


(b) Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 25$, $minsup = 0.0275\%$

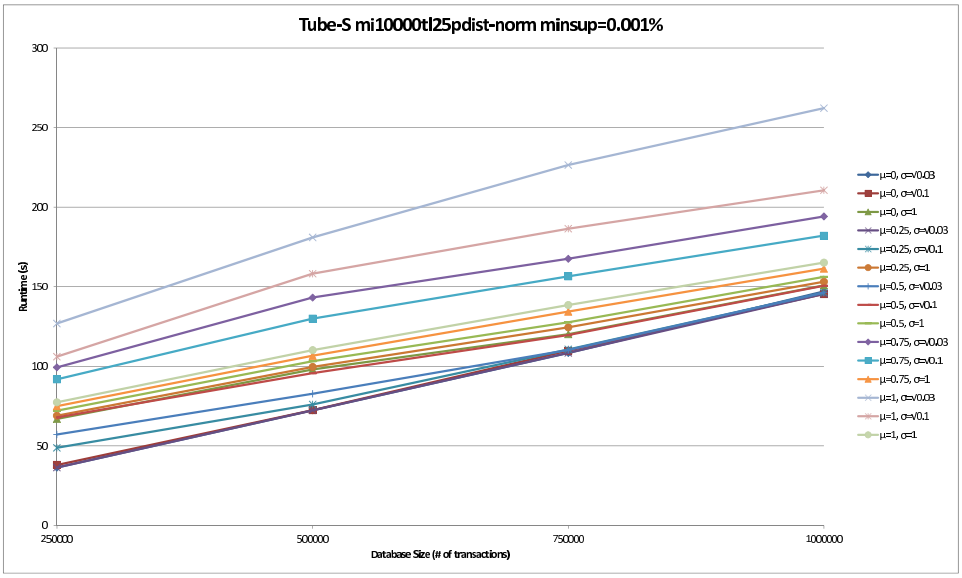
Figure 5.14: Runtime: Tube-S & Tube-P

$|\mathcal{I}| = 1000$, $\overline{|t_j|} = 25$, $minsup = 0.0275\%$

Figures 5.15(a), 5.15(b) and 5.16 show the runtime when increasing the number of domain items to 10,000 but keeping the other parameters the same (effectively lowering the density by another factor of 10). Here the same effects are observed as when there were 1000 domain items, however, even more distributions are lowered to the bottom cluster. Tube-S and Tube-P both outperform CUF-growth*, with Tube-P just slightly edging out Tube-S for all distributions.



(a) CUF-growth*, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 25$, $minsup = 0.0001\%$



(b) Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 25$, $minsup = 0.0001\%$

Figure 5.15: Runtime: CUF-growth* & Tube-S

$|\mathcal{I}| = 10000$, $\overline{|t_j|} = 25$, $minsup = 0.0001\%$

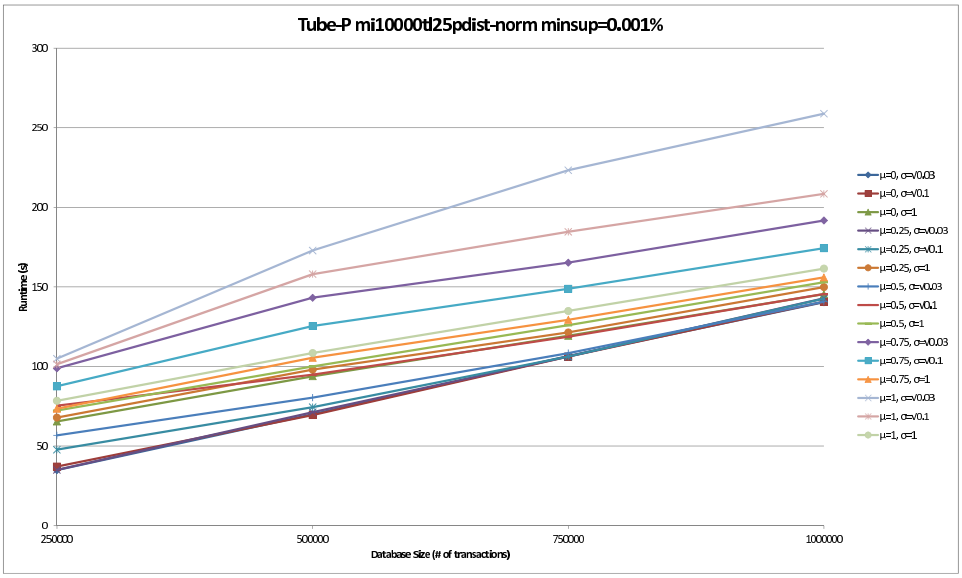


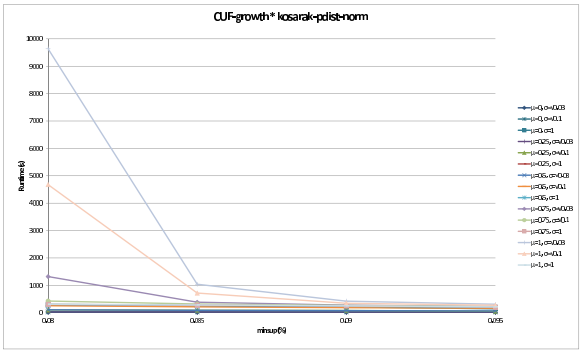
Figure 5.16: Runtime: Tube-P, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 25$, $minsup = 0.0001\%$

Real Data

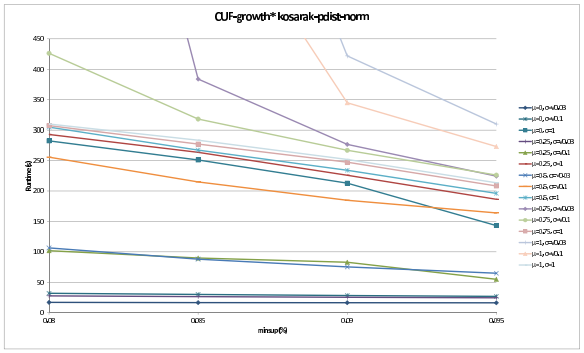
Consider the graphs in Figure 5.17. For the real life data, I did not chunk up the database to test different sizes since that could potentially destroy the characteristics of the probability distributions. Instead, I used the full size and different *minsup* values are included on the graphs. With a lower *minsup*, all algorithms had longer runtimes. Among the three algorithms, Tube-S and Tube-P performed between two to four times better than CUF-growth* depending on the distribution of existential probability values, with Tube-P outperforming Tube-S by a small amount on all distributions. Kosarak is a very sparse database, with a density of about 0.0002, comparable to my synthetic databases with 10,000 domain items and a transaction length between five and twelve. While an obvious separation between a top and bottom cluster in the CUF-growth* graph is less clear, the distributions between ($\mu = 0$, $\sigma = 1$) and ($\mu = 1$, $\sigma = 1$) form a loose cluster which Tube-S and Tube-P both distribute over the gap below it while lowering the runtime.

Consider the graphs in Figure 5.18. Mushroom is a very dense database, with a density of about 0.18, comparable to my synthetic databases with 100 domain items and a transaction length of 25. While in my synthetic databases there was not much difference between the three algorithms, with mushroom, Tube-S and Tube-P both result in runtimes of 1.2 to 2 times lower than CUF-growth* depending on the distribution. From the zoomed in graphs it is clear that the bottom cluster was not affected by the choice of algorithm but the effect was observed in the top cluster, which consists of the higher ranked distributions from Table 5.2.

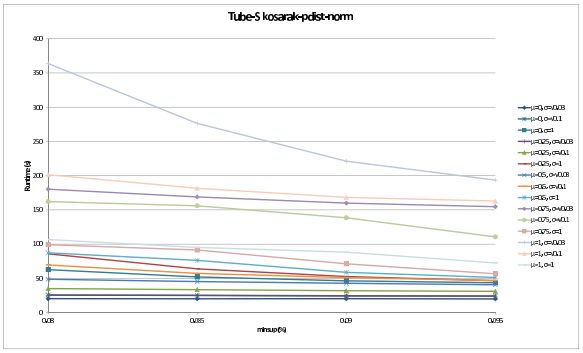
The last real database I tested was the retail database. The runtimes are shown



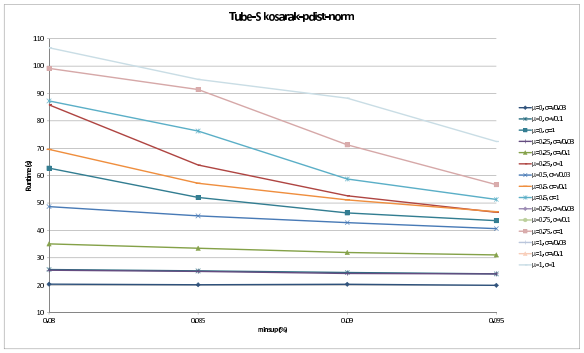
(a) CUF-growth*, Kosarak



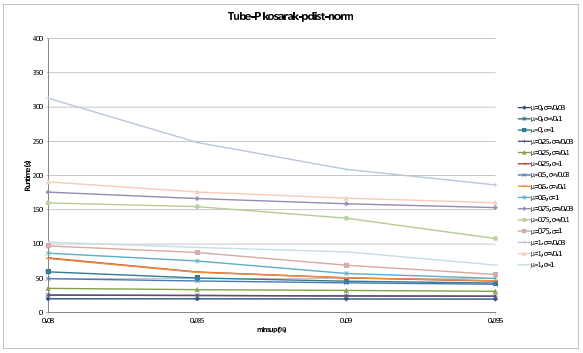
(b) CUF-growth*, Kosarak (zoomed in)



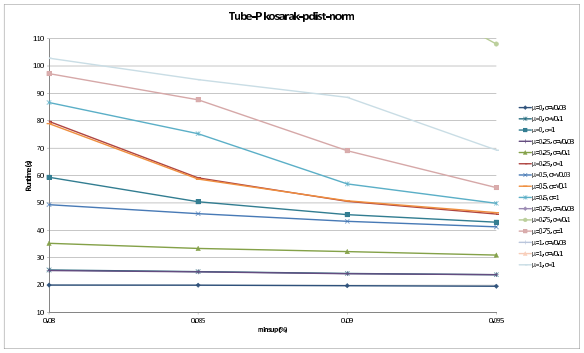
(c) Tube-S, Kosarak



(d) Tube-S, Kosarak (zoomed in)



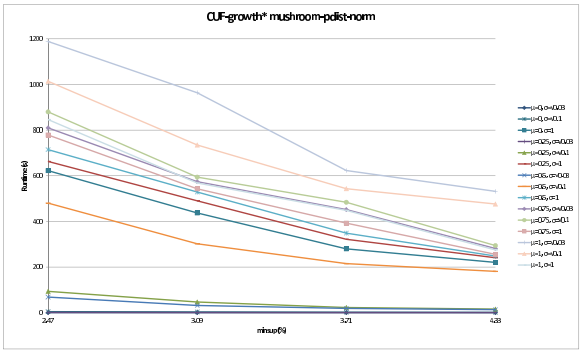
(e) Tube-P, Kosarak



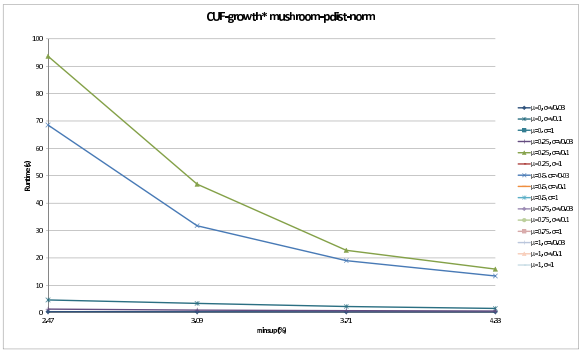
(f) Tube-P, Kosarak (zoomed in)

Figure 5.17: Kosarak Runtime: All Algorithms

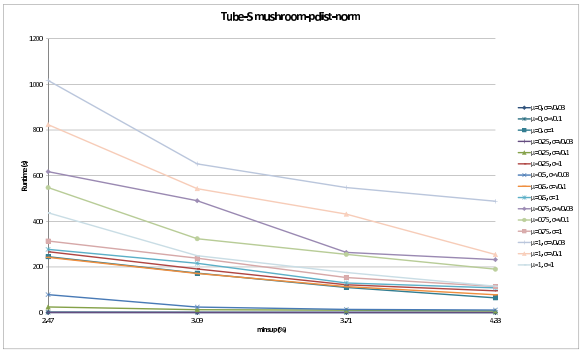
in Figures 5.19 and 5.20. Retail is a sparse database, with a density of about 0.0006, comparable to my synthetic databases with 10,000 domain items and a transaction length of five. There is a very clear progression on this database on CUF-growth*



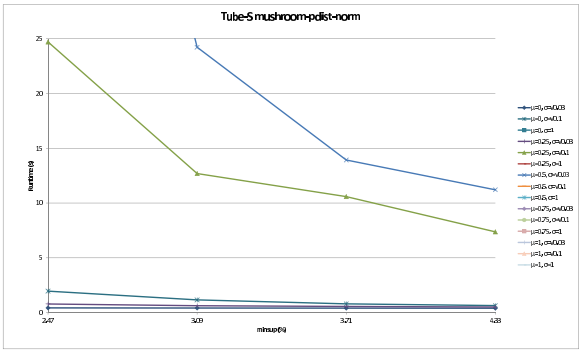
(a) CUF-growth*, Mushroom



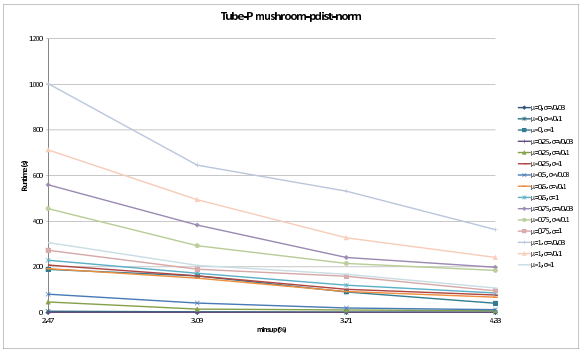
(b) CUF-growth*, Mushroom (zoomed in)



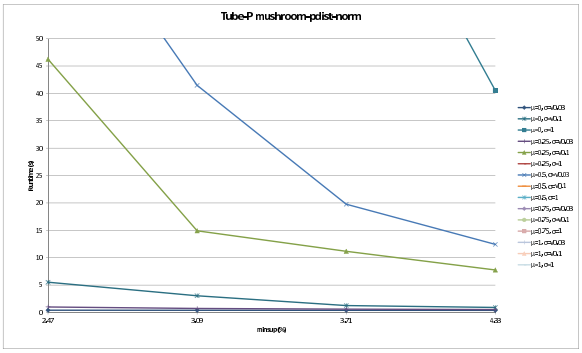
(c) Tube-S, Mushroom



(d) Tube-S, Mushroom (zoomed in)



(e) Tube-P, Mushroom

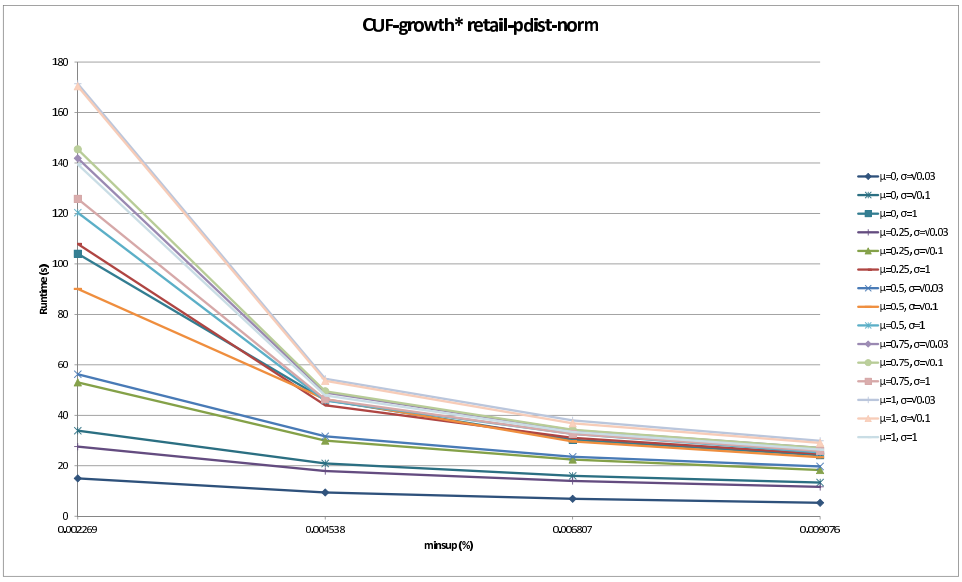


(f) Tube-P, Mushroom (zoomed in)

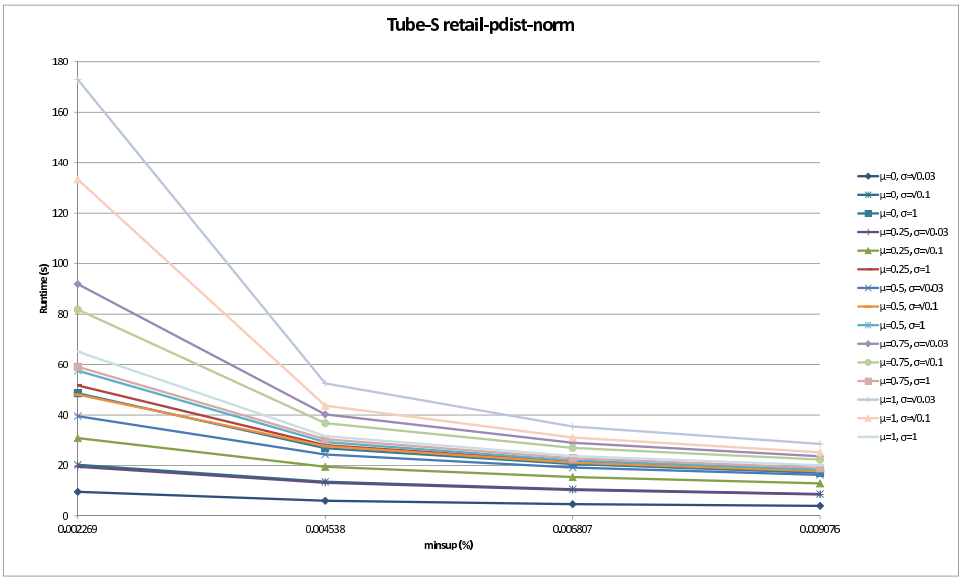
Figure 5.18: Mushroom Runtime: All Algorithms

having the slowest runtimes, followed by Tube-S, which lowers some distributions in the top cluster to the bottom cluster, followed by Tube-P, which lowers all the remaining distributions in the top cluster. On this dataset, notice the same interesting

quality that for the lower ranking distributions, Tube-S actually slightly outperforms Tube-P.

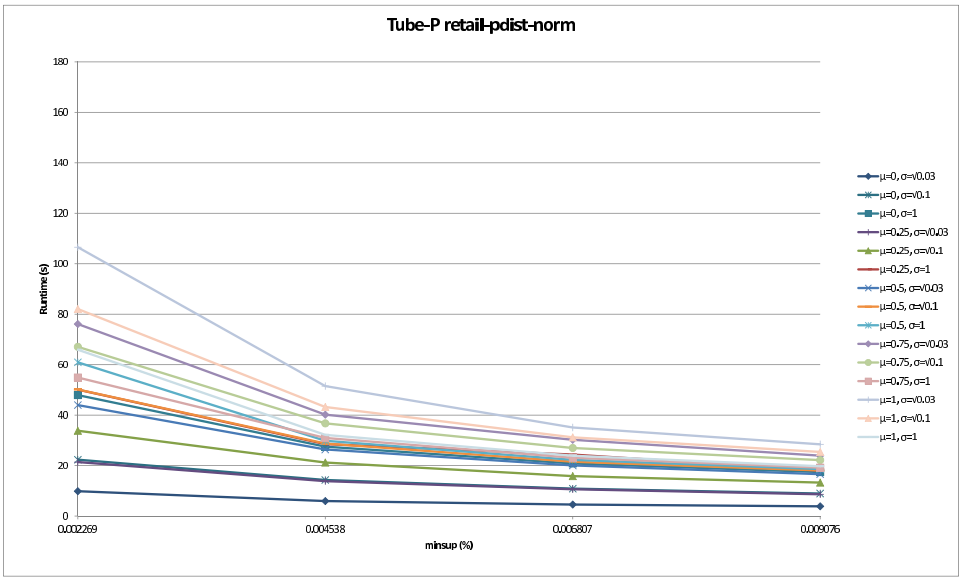


(a) CUF-growth*, Retail



(b) Tube-S, Retail

Figure 5.19: Retail Runtime: CUF-growth* and Tube-S



(a) Tube-P, Retail

Figure 5.20: Retail Runtime: Tube-P

Summary

Well, that was a lot of graphs. Here I summarize my findings into distinct observations.

1. **As database size increases and *minsup* stays the same, runtime increases.**

2. **As database size stays the same and *minsup* decreases, runtime increases.**

3. **Low variances and high means lead to the highest runtimes.** This is because as the variance is reduced, the mean becomes a more important determining factor, with higher means giving high existential probability values, leading to high numbers of candidate frequent patterns with few false positives and few opportunities for early pruning.

4. **High variances and middle means lead to the next highest runtimes.** This is because as the variance is increased, existential probability values become almost uniformly distributed, leading to a high number of candidate frequent patterns and a high number of false positives, but also more opportunities for early pruning that reduces the runtimes.

5. **A low mean and variance lead to the lowest runtimes.** In this case most existential probabilities will be low, leading to many infrequent patterns being pruned early in the algorithm and reducing the runtime. These rankings are summarized in Table 5.2.

6. **Tube-S and Tube-P both have lower runtimes when compared to CUF-growth*, except for the case where the number of domain items is small and the transaction length is small (i.e., 100 items and a transaction length of 5).** This is because the advantages gained by a tighter upper bound in Tube-S and Tube-P when considering only portions of a transaction are easily lost when there are few domain items, as the maximum existential probability values will be high regardless of whether the item is in the prefix or not due to each item appearing with higher frequencies.

7. **Tube-S matches or outperforms Tube-P when there are extremely low transaction lengths.** In this case, the max of all the second highest items' existential probabilities in the prefix of transactions passing through a node is more likely to be lower than the max of any single item's. In fact, for a pattern length of three, Tube-P's bound is guaranteed never to be tighter than Tube-S's. This is because if the second highest existential probability in the prefix of the terminating item is not always the second highest over all the transactions that share that path in the tree, it must be the highest in at least one transaction, therefore leading to a looser bound when Tube-P uses the higher value. The probability of this happening decreases significantly with longer transaction lengths and higher cardinality frequent patterns however, as the max of all the second highest items' existential probabilities may be drawn from an item not being considered in the current candidate pattern, while items actually in the pattern have lower single item maxes.

8. **Tube-P matches or outperforms the runtime of Tube-S for the higher**

ranked distributions in Table 5.2. With Tube-P, a high variance and high mean indicate that there is a wide range of higher existential probabilities among items. In this case, the max of any single item's existential probability is more likely to be lower than the max of all the second highest items' in the prefix of transactions passing through a node as the high variance will give low single item maximums with enough frequency.

9. **Tube-S matches or outperforms the runtime of Tube-P for the lower ranked distributions in Table 5.2.** With Tube-S, a low mean and high variance indicate that there is a wide range of lower existential probabilities among items. In this case, the max of all the second highest items' existential probabilities in the prefix of transactions passing through a node is more likely to be lower than the max of any single item's as the high variance will give high single item maximums with enough frequency.

10. **Tube-S and Tube-P match each other for performance on the very highest and very lowest ranked distributions in Table 5.2.** The low variance means that most existential probabilities are clustered around a single value, so there is not much difference between the max of all the second highest items' existential probabilities and the max of any single item.

11. **CUF-growth*, Tube-S and Tube-P are all scalable with respect to the database sizes (at least up to the tested 1,000,000 transactions).** The graphs for runtimes were almost linear in all cases for all three algorithms.

5.2.5 Memory Usage Analysis

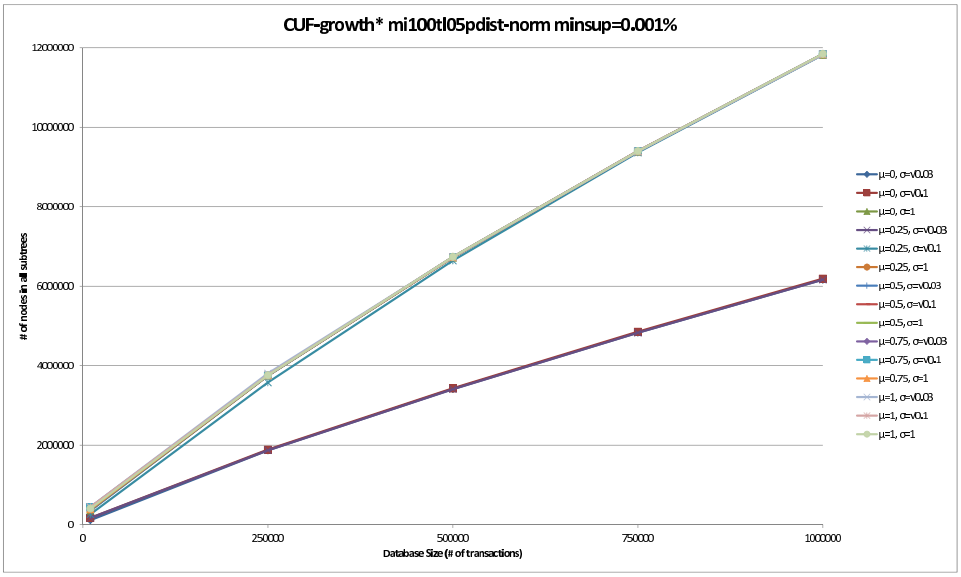
To measure the memory usage of the different algorithms, I tested the exact same databases in the exact same configurations as in Section 5.2.4, except that the total number of nodes created was the dependent variable. The graphs below show the memory usage of synthetic datasets with different probability distributions and densities for each of the three algorithms.

Transaction Length 5

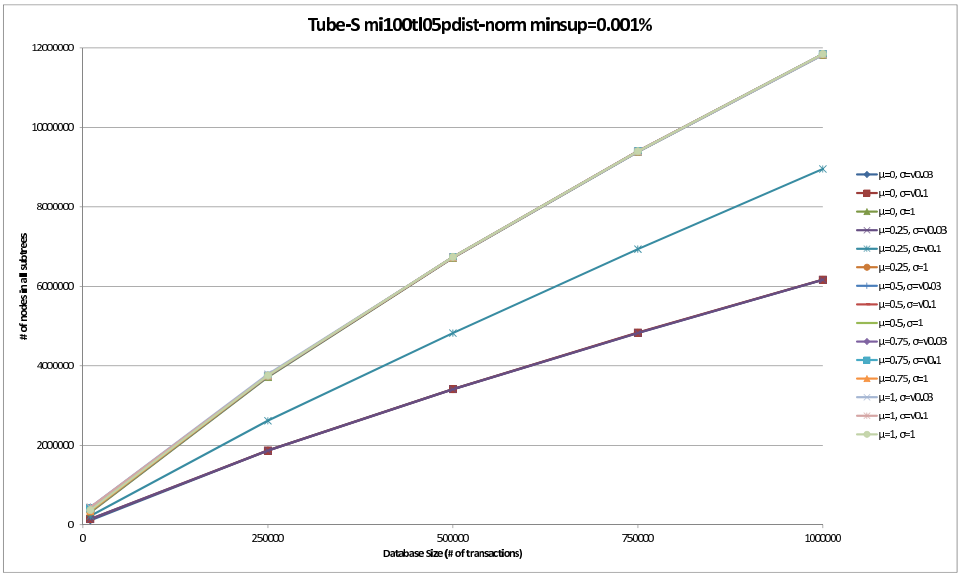
Consider the graphs in Figs. 5.21(a), 5.21(b) and 5.22(a). As the database size increases, the number of nodes also increases, however the rate of increase drops off as the size becomes larger. This is because new nodes are only created for prefixes that have not previously occurred in the database. If a prefix matches an already existing path, the nodes in that path are updated with new information without creating any new nodes. As the database size becomes larger, it becomes less likely that a prefix will not already have occurred. As in the runtime graphs, two clusters are apparent: a larger one with more total nodes consisting of most of the high ranking distributions in Table 5.2 and a smaller one with fewer total nodes consisting of low ranking distributions from the same table. As we move from CUF-growth* to Tube-S, the only noticeable change on this scale is that ($\mu = 0.25$, $\sigma = \sqrt{0.1}$) moves from the upper cluster to halfway in-between the upper and lower one. Looking at Tube-P, ($\mu = 0.25$, $\sigma = \sqrt{0.1}$) stays in the upper cluster and the previous three distributions in the lower cluster split apart with two of them showing an increased number of nodes over Tube-S and CUF-growth*. These observations mirror the runtime observations

for the same databases. The effect of increased pruning with a low mean and variance is the most prominent here. It should also be noted that CUF-growth* does not have more nodes than Tube-S or Tube-P for any distribution, which, given the same basic algorithmic framework was used to program all three algorithms, backs up my earlier assertion that the reason for the increase in the very first set of runtime graphs was due to the overhead of applying my optimization.

Figs. 5.22(b), 5.23(a) and 5.23(b) show the memory usage when increasing the number of domain items to 1,000 but keeping the other parameters the same (effectively lowering the density by a factor of 10). The drop off in the number of nodes is far less noticeable here, as the higher number of domain items allows for more combinations of items in the prefix of the trees. Here, notice that Tube-S and Tube-P both produce fewer nodes than CUF-growth*, but Tube-S beats Tube-P in terms of a lower number of nodes. Again, the relationship that a lower number of nodes leads to a lower runtime is borne out with the graphs from Section 5.2.4. The same observations are found in Figs. 5.24(a), 5.24(b) and 5.25: that distributions with lower runtimes are the same distributions with fewer nodes and are those lower ranked distributions in Table 5.2.



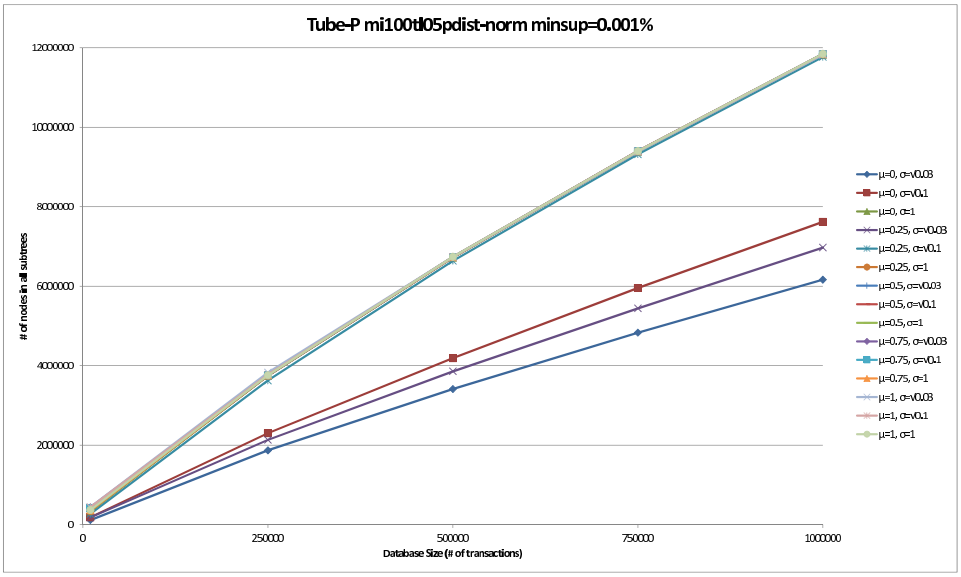
(a) CUF-growth*, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$



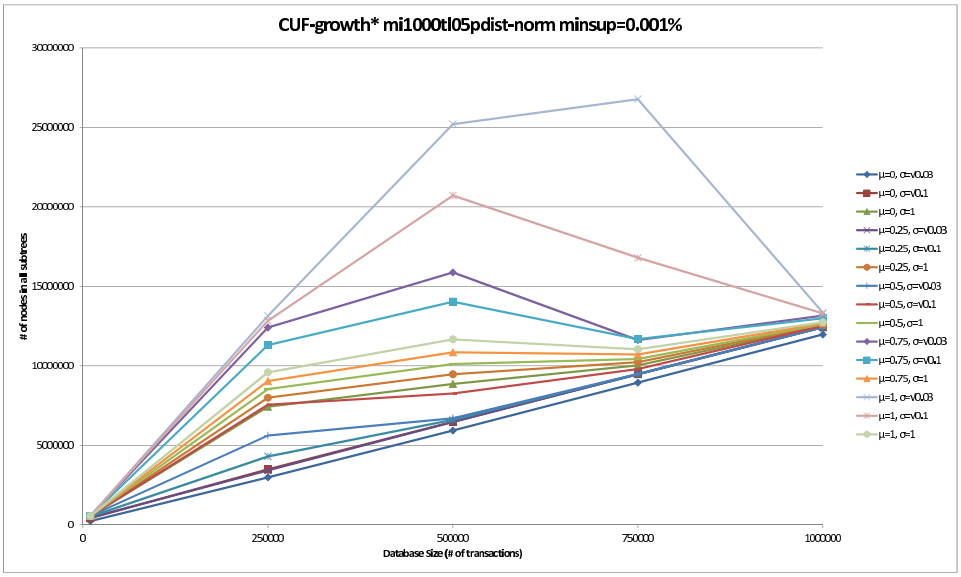
(b) Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$

Figure 5.21: Memory: CUF-growth* & Tube-S

$|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$

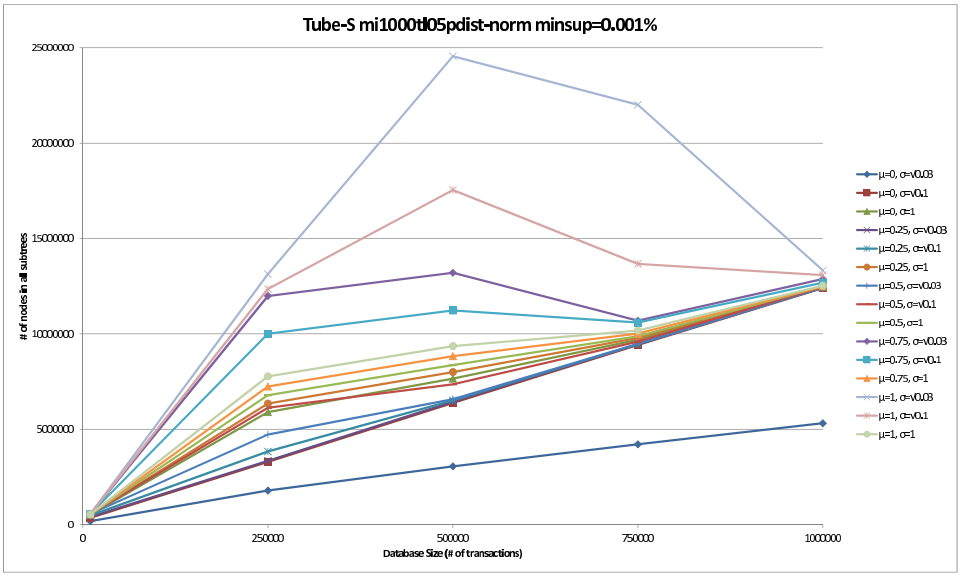


(a) Tube-P, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$

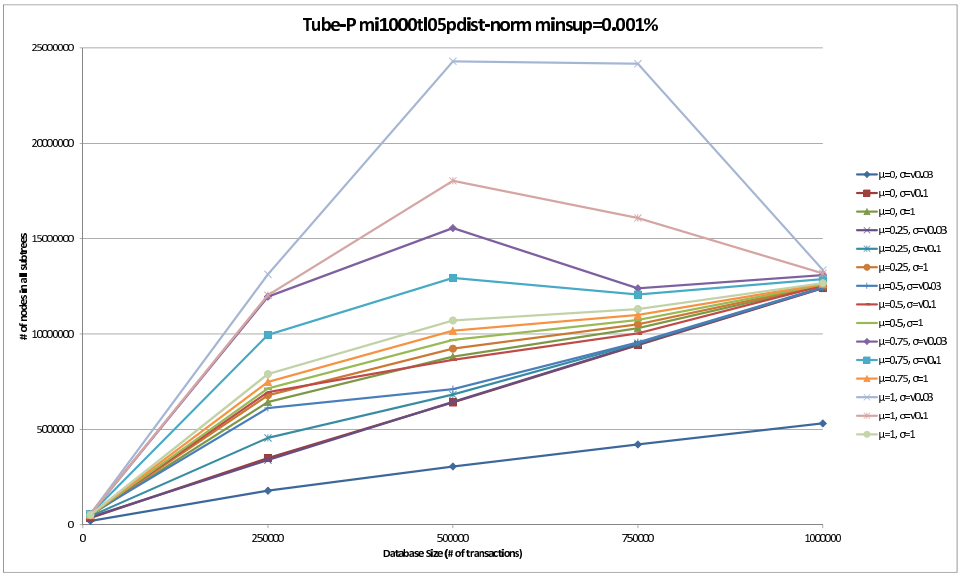


(b) CUF-growth*, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$

Figure 5.22: Memory: Tube-P ($|\mathcal{I}| = 100$) & CUF-growth* ($|\mathcal{I}| = 1000$)
$\overline{|t_j|} = 5$, $minsup = 0.001\%$



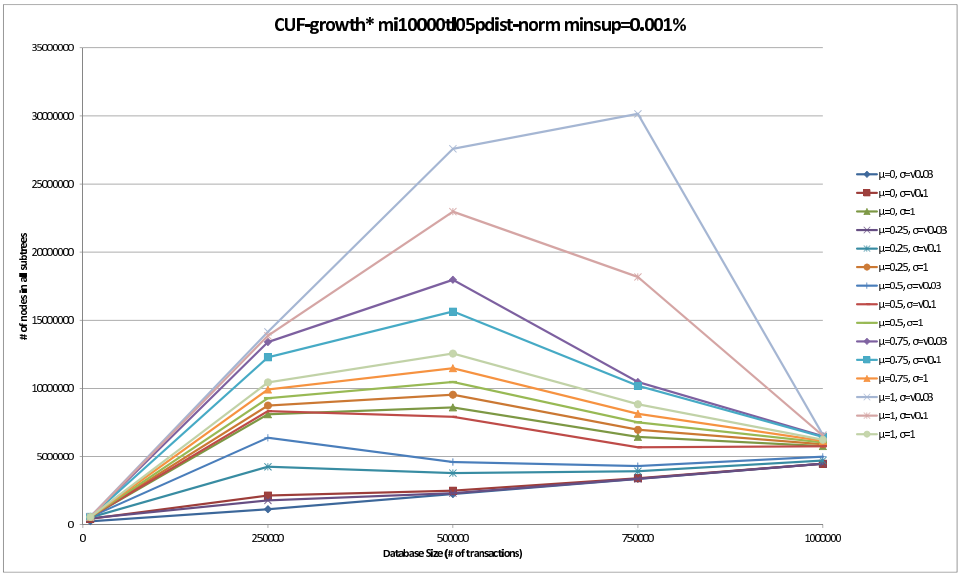
(a) Tube-S, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$



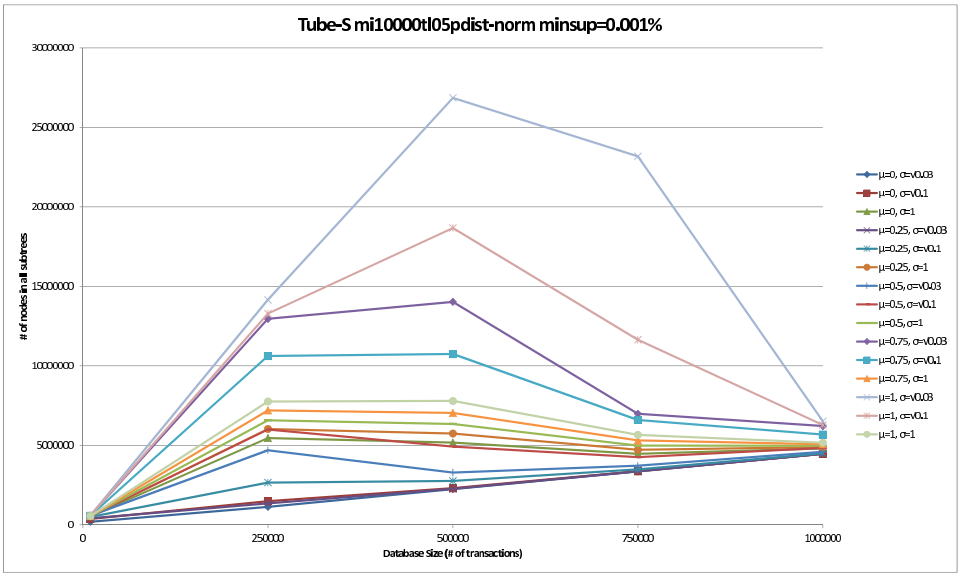
(b) Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$

Figure 5.23: Memory: Tube-S & Tube-P

$|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$



(a) CUF-growth*, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$



(b) Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$

Figure 5.24: Memory: CUF-growth* & Tube-S

$|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$

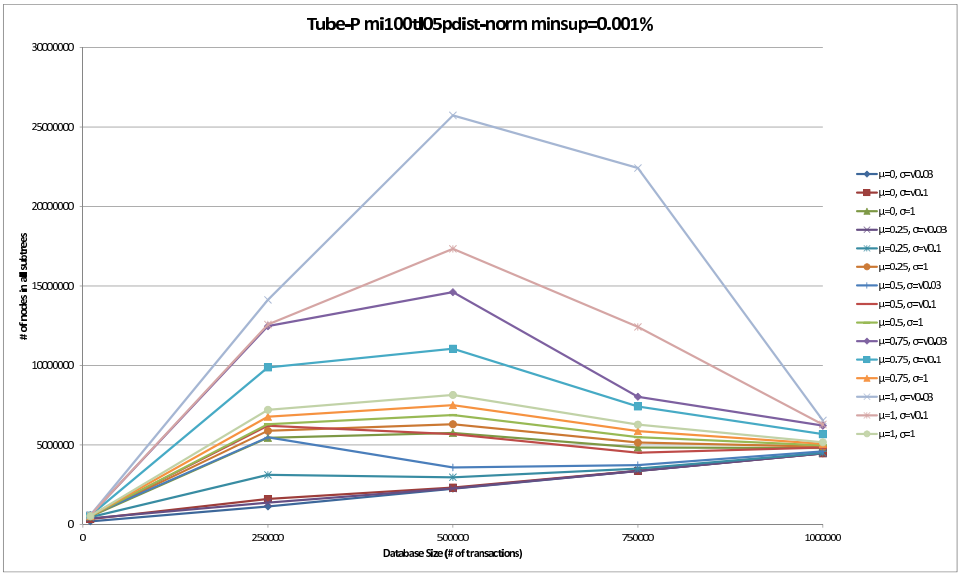
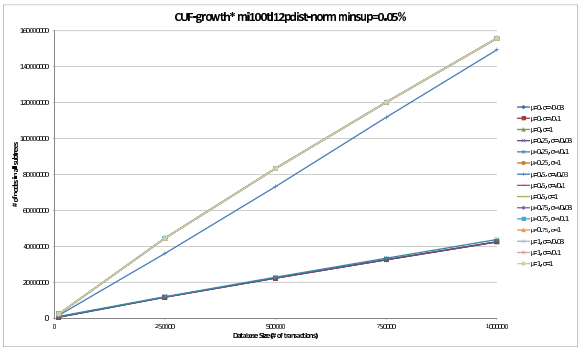


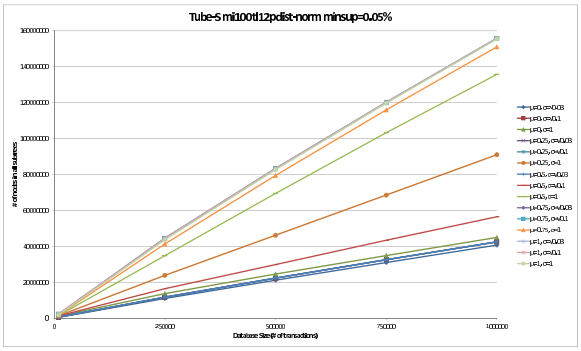
Figure 5.25: Memory: Tube-P, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$, $minsup = 0.001\%$

Transaction Length 12

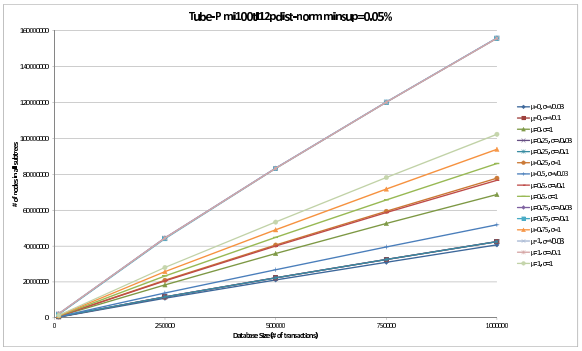
Consider the graphs in Figs. 5.26(a), 5.26(b) and 5.26(c). This is the clearest picture yet of the relationship between the three algorithms: Tube-S and Tube-P both showing fewer nodes than CUF-growth*, Tube-S beating Tube-P on lower ranked distributions and Tube-P beating Tube-S on higher ranked distributions. Figs. 5.26(d), 5.26(e) and 5.26(f) show the memory usage when increasing the number of domain items to 1,000 but keeping the other parameters the same (effectively lowering the density by a factor of 10). Finally, Figs. 5.27(a), 5.27(b) and 5.27(c) show the runtime when increasing the number of domain items to 10,000 but keeping the other parameters the same (effectively lowering the density by another factor of 10). Nothing significant appears in the graphs not already discussed in Section 5.2.4.



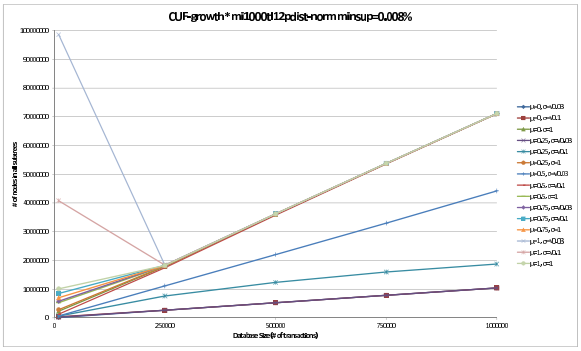
(a) CUF-growth*, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 12$, $minsup = 0.05\%$



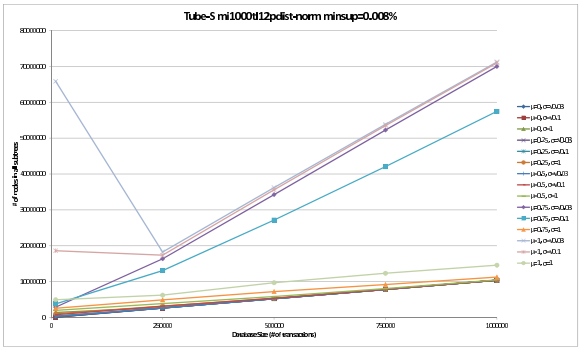
(b) Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 12$, $minsup = 0.05\%$



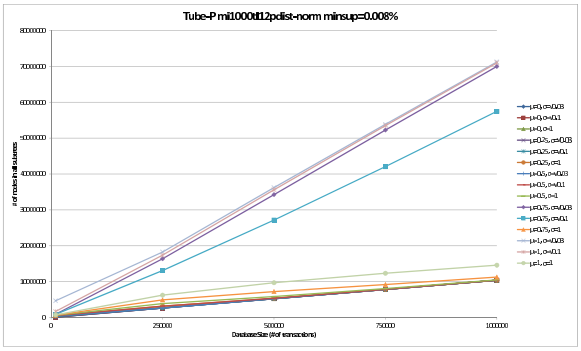
(c) Tube-P, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 12$, $minsup = 0.05\%$



(d) CUF-growth*, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 12$, $minsup = 0.008\%$



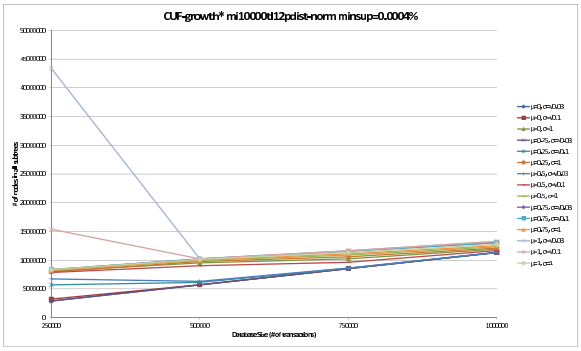
(e) Tube-S, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 12$, $minsup = 0.008\%$



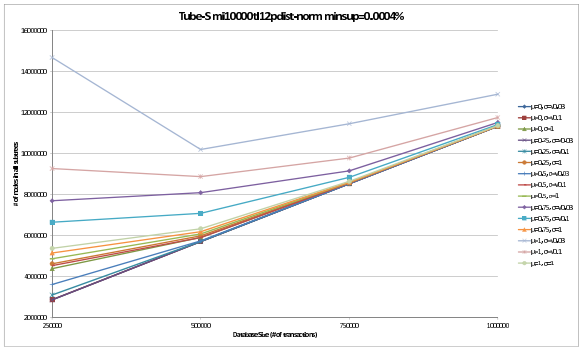
(f) Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 12$, $minsup = 0.008\%$

Figure 5.26: Memory: All Algorithms

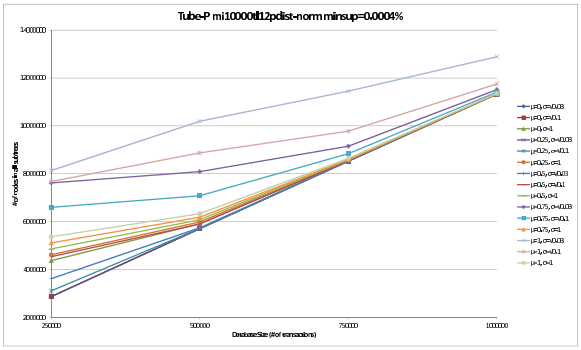
($|\mathcal{I}| = 100$, $minsup = 0.05\%$) & ($|\mathcal{I}| = 1000$, $minsup = 0.008\%$), $\overline{|t_j|} = 12$



(a) CUF-growth*, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 12$, $minsup = 0.0004\%$



(b) Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 12$, $minsup = 0.0004\%$



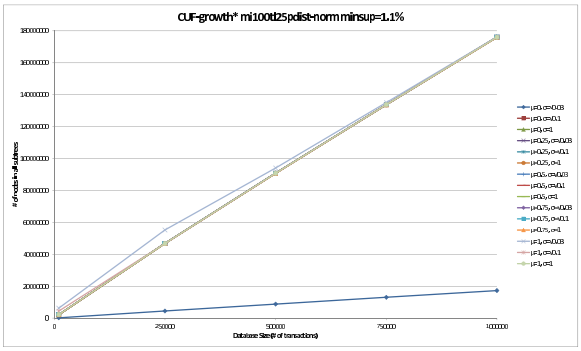
(c) Tube-P, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 12$, $minsup = 0.0004\%$

Figure 5.27: Memory: All Algorithms

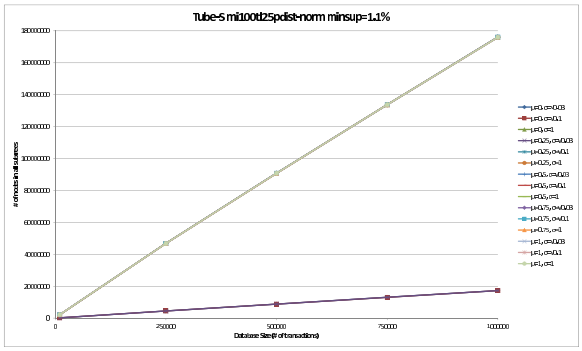
$|\mathcal{I}| = 10000$, $minsup = 0.0004\%$, $\overline{|t_j|} = 12$

Transaction Length 25

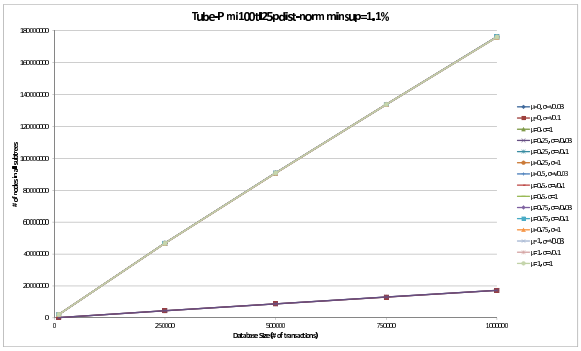
Consider the graphs in Figs. 5.28 and 5.29. The number of nodes in each case show trends corresponding to the runtimes in Section 5.2.4.



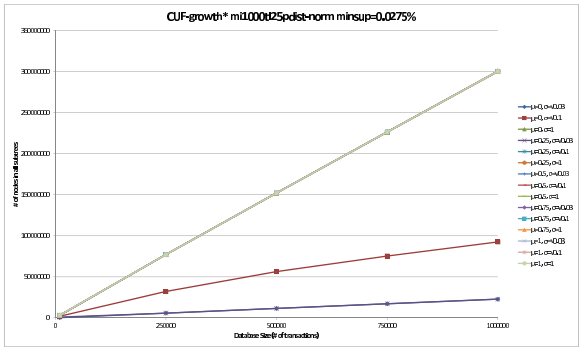
(a) CUF-growth*, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 25$, $minsup = 1.1\%$



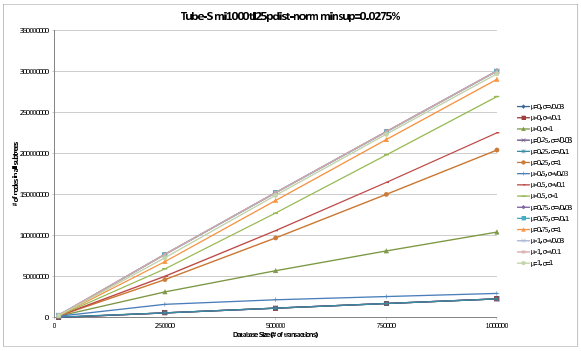
(b) Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 25$, $minsup = 1.1\%$



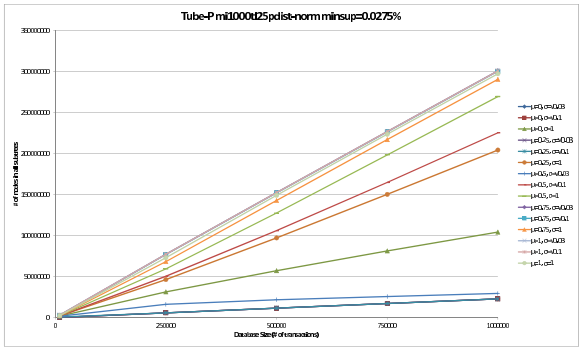
(c) Tube-P, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 25$, $minsup = 1.1\%$



(d) CUF-growth*, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 25$, $minsup = 0.0275\%$



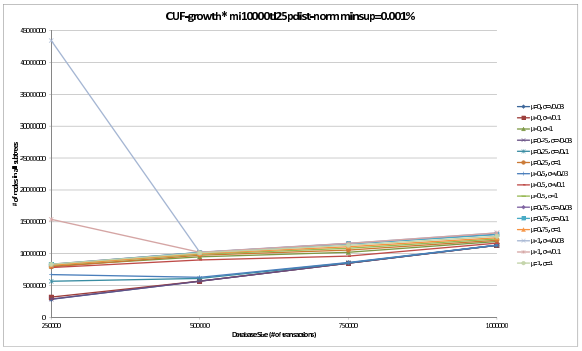
(e) Tube-S, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 25$, $minsup = 0.0275\%$



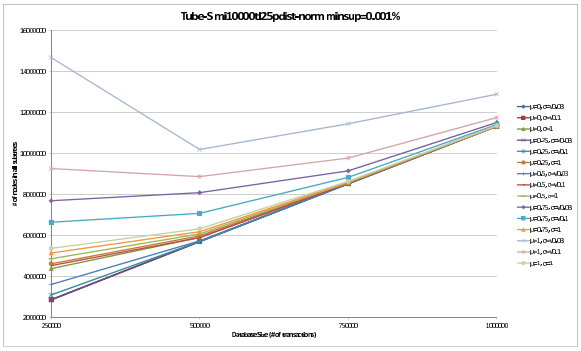
(f) Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 25$, $minsup = 0.0275\%$

Figure 5.28: Memory: All Algorithms

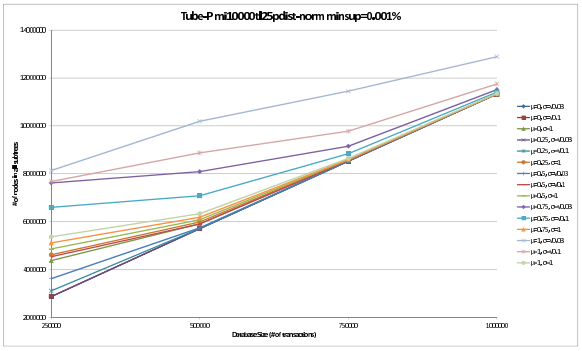
($|\mathcal{I}| = 100$, $minsup = 1.1\%$) & ($|\mathcal{I}| = 1000$, $minsup = 0.0275\%$), $\overline{|t_j|} = 25$



(a) CUF-growth*, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 25$, $minsup = 0.0001\%$



(b) Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 25$, $minsup = 0.0001\%$



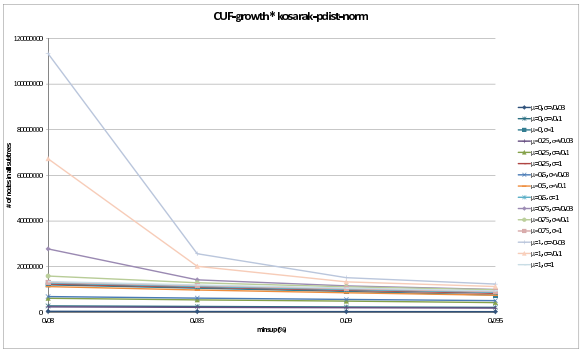
(c) Tube-P, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 25$, $minsup = 0.0001\%$

Figure 5.29: Memory: All Algorithms

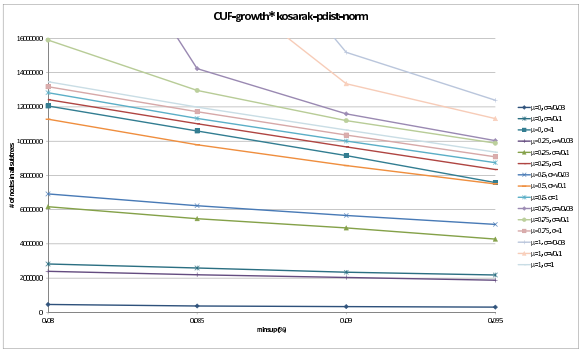
$|\mathcal{I}| = 10000$, $minsup = 0.0001\%$, $\overline{|t_j|} = 25$

Real Data

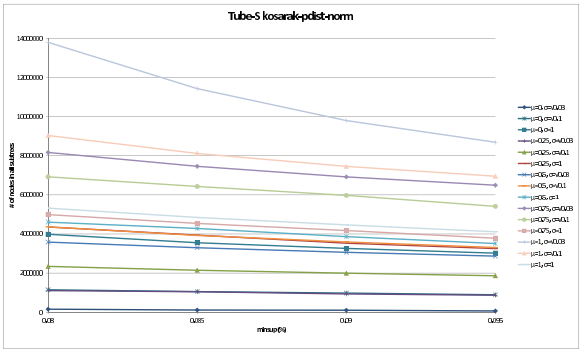
Consider the graphs in Figs. 5.30, 5.31 and 5.32. There is no noticeable different between these graphs and the corresponding runtime graphs in Section 5.2.4. A summary of the memory usage specific observations follows.



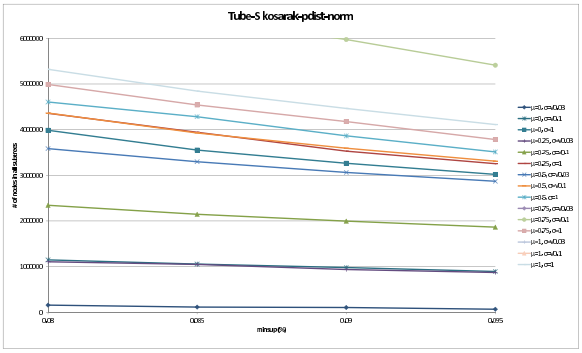
(a) CUF-growth*, Kosarak



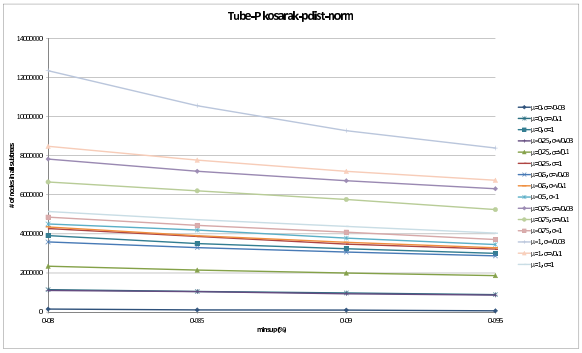
(b) CUF-growth*, Kosarak (zoomed in)



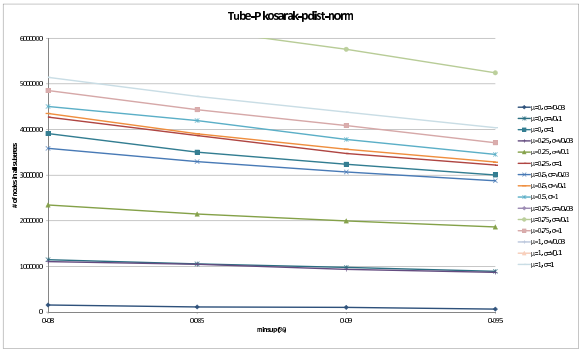
(c) Tube-S, Kosarak



(d) Tube-S, Kosarak (zoomed in)

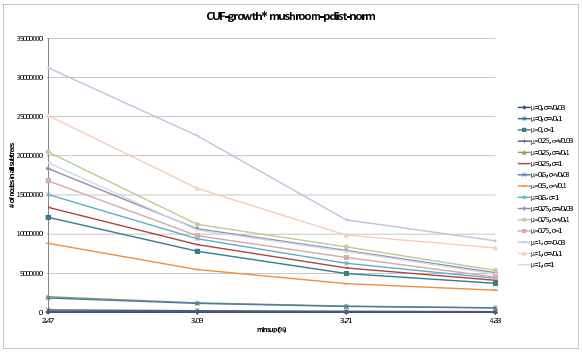


(e) Tube-P, Kosarak

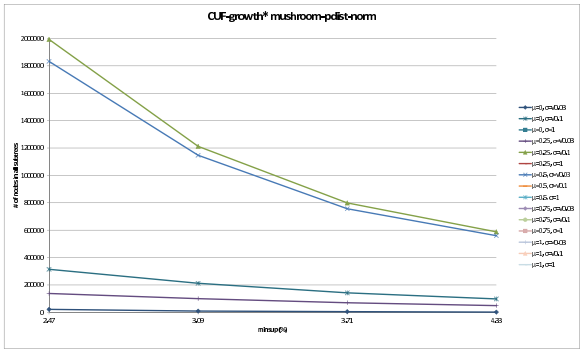


(f) Tube-P, Kosarak (zoomed in)

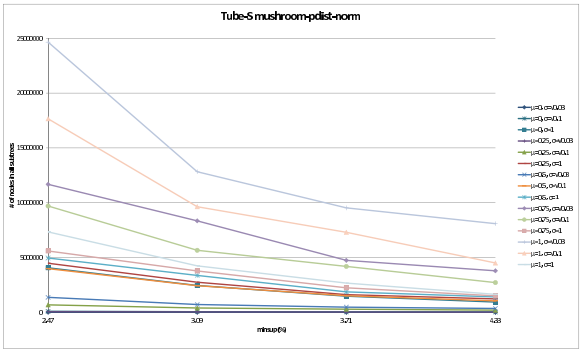
Figure 5.30: Kosarak Memory: All Algorithms



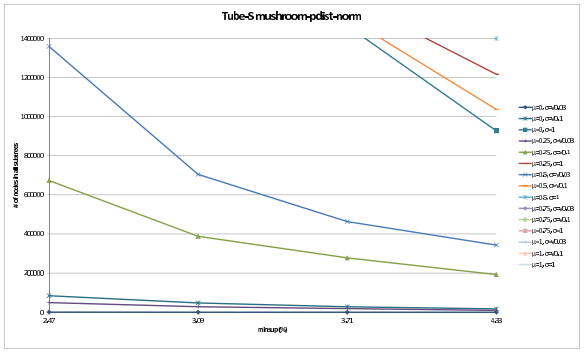
(a) CUF-growth*, Mushroom



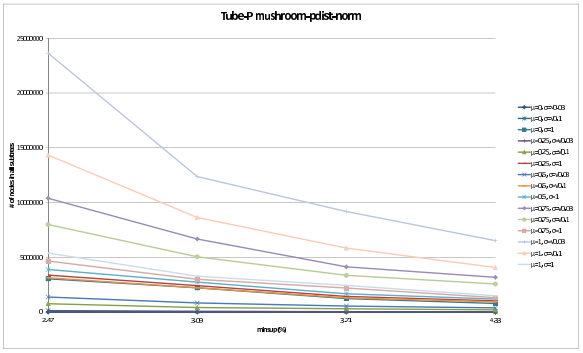
(b) CUF-growth*, Mushroom (zoomed in)



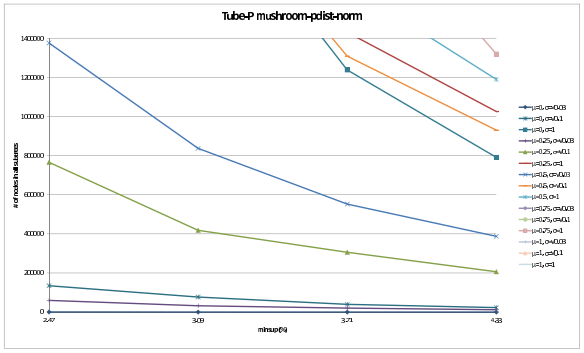
(c) Tube-S, Mushroom



(d) Tube-S, Mushroom (zoomed in)

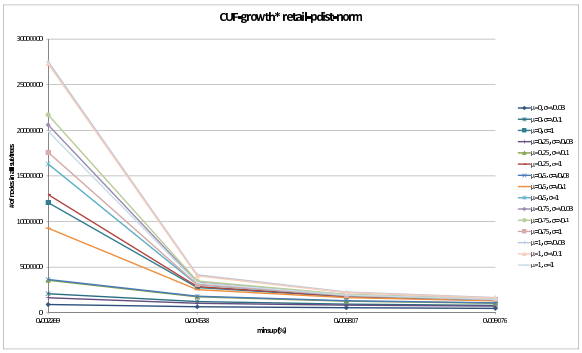


(e) Tube-P, Mushroom

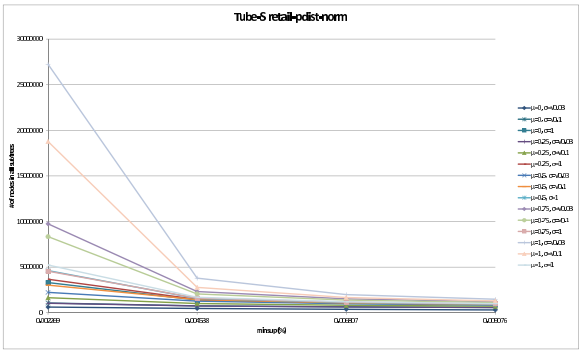


(f) Tube-P, Mushroom (zoomed in)

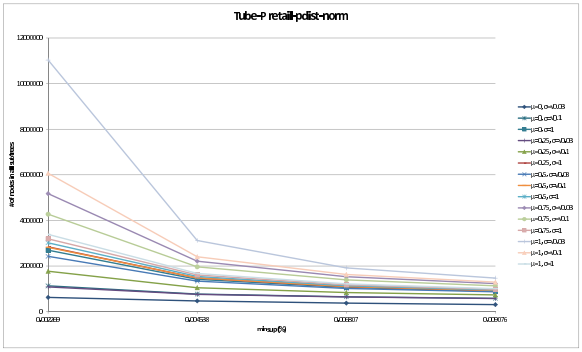
Figure 5.31: Mushroom Memory: All Algorithms



(a) CUF-growth*, Retail



(b) Tube-S, Retail



(c) Tube-P, Retail

Figure 5.32: Retail Memory: All Algorithms

Summary

1. **As the database size increases, the number of nodes also increases, however the rate of increase drops off as the size becomes larger and/or the number of domain items increases.** This is because new nodes are only created for prefixes that have not previously occurred in the database. If a prefix matches an already existing path, the nodes in that path are updated with new information without creating any new nodes. As the database size becomes larger, it becomes less likely that a prefix will not already have occurred. Additionally, as the number of domain items increases, the number of possible combinations of prefix items also increases, raising the size of database after which it becomes possible to easily observe a drop off in memory usage.

2. **Tube-S and Tube-P both use less or equal memory when compared to CUF-growth*.** This is because the upper bounds on expected support are tighter in the Tube algorithms than they are in CUF-growth*.

3. **Tube-S and Tube-P trade off outperforming each other depending on the transaction length and number of domain items of the tested database.** The reasons have already been discussed in the aforementioned summary in Section 5.2.4.

5.2.6 False Positive Analysis

The final experiment I ran measured the number of false positives in each cardinality level for different synthetic and real data distributions. This allowed me

to understand the behaviour of each algorithm on a deeper level — instead of talking about total memory usage, analyzing false positives allowed me to see at what cardinality level when mining candidate patterns memory is either being saved or increased.

Transaction Length 5

Consider Figures 5.33(a), 5.33(b) and 5.34(a). There are no false positives for a cardinality of one in all three algorithms. This is exactly what I would expect since the expected support of all the single frequent patterns is calculated during the construction of the global tree in each algorithm before the mining even begins, thus upper bound calculations are not even performed. Another thing to note is that as the cardinality increases, fewer and fewer false positives are found, and those false positives are mostly found in the high ranked distributions in Table 5.2 (as these would be the ones most likely to not yet be pruned after multiple multiplications through higher level projected trees). This is because the upper bound in all three algorithms becomes tighter as the cardinality increases due to continued multiplications of CUF-growth*'s bronze value, Tube-S's M_2 value, and Tube-P's max existential probability values.

Comparing CUF-growth* to Tube-S, there are fewer false positives in Tube-S in all distributions for cardinalities two to four. After that, most distributions still experience a drop in false positives, but for some distributions there is actually a small increase in false positives, namely: ($\mu = 0$, $\sigma = 1$), ($\mu = 0.5$, $\sigma = \sqrt{0.1}$), ($\mu = 0.75$, $\sigma = \sqrt{0.1}$), ($\mu = 1$, $\sigma = 1$) and ($\mu = 1$, $\sigma = \sqrt{0.03}$), all of which are mostly mid to

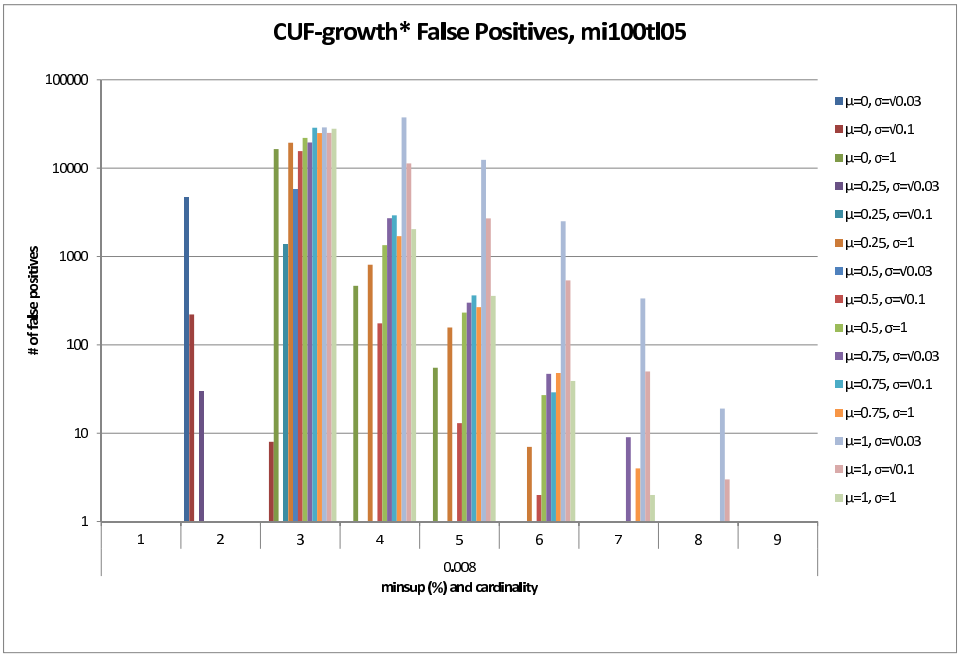
higher ranked distributions. When considering Tube-P, there are again far fewer false positives than CUF-growth*, with the biggest difference (from both CUF-growth* and Tube-S) being that for the high cardinality itemsets there are only one or two distributions that show any false positives for 7- or 8-itemsets. I have already shown in Section 5.2.4 that Tube-P is most effective on higher ranked distributions and higher cardinality itemsets, so this makes sense. On the other hand, there are again a couple distributions that show increased false positives over CUF-growth* and Tube-S for low cardinalities, namely: ($\mu = 0$, $\sigma = \sqrt{0.1}$), ($\mu = 0.25$, $\sigma = \sqrt{0.03}$), ($\mu = 0.25$, $\sigma = \sqrt{0.1}$), ($\mu = 0$, $\sigma = \sqrt{0.03}$), ($\mu = 0.5$, $\sigma = \sqrt{0.1}$) and ($\mu = 0.75$, $\sigma = \sqrt{0.1}$), all of which are mostly lower ranked distributions. To show how Tube-S and Tube-P can produce more false positives for some cardinalities than CUF-growth*, consider the following quick example.

Example 5.1. Consider a database containing a single transaction $t_1 = \{a : 0.8, b : 0.8, c : 0.5, d : 0.6, e : 0.5\}$. When inserted into the global tree in CUF-growth* with an alphabetical ordering, the transaction cap stored in the node for d will be the product of the two highest values in the transaction, $0.8 \times 0.8 = 0.64$, and the bronze value will be 0.6. When inserted into either a Tube-S or Tube-P tree the prefixed item cap for the node d will be the product of the existential probability value of d and the highest existential probability in d's prefix, $0.6 \times 0.8 = 0.48$, and the M_2 value for Tube-S will be the second highest existential probability value in d's prefix, 0.8. When calculating $expSup^{Cap}(\{a, b, c, d\})$, CUF-growth* will give $0.64 \times 0.6^2 = 0.2304$, while both Tube-S and Tube-P will give $0.48 \times 0.8^2 = 0.3072$, a looser bound. So while Tube-S and Tube-P are both guaranteed to have a tighter bound for all 2-itemsets,

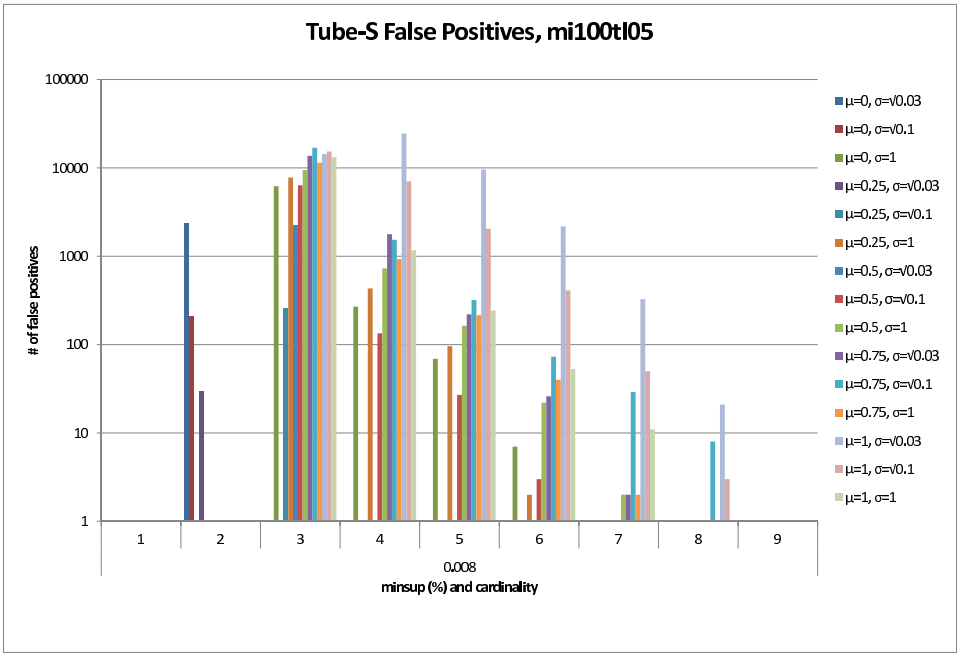
this is not necessarily so for 3-itemsets or higher cardinalities and depends on factors discussed in Section 5.2.4. ■

Finally, also note that both Tube-S and Tube-P have the exact same number of false positives for all candidate 2-itemsets. This is expected as the calculation of the upper bound on expected support for 2-itemsets in the same in both algorithms.

Now consider Figures 5.34(b), 5.35(a) and 5.35(b). The same trends are visible but are perhaps even easier to see in these charts when there are more domain items. The same is true for Figures 5.36(a), 5.36(b) and 5.37.

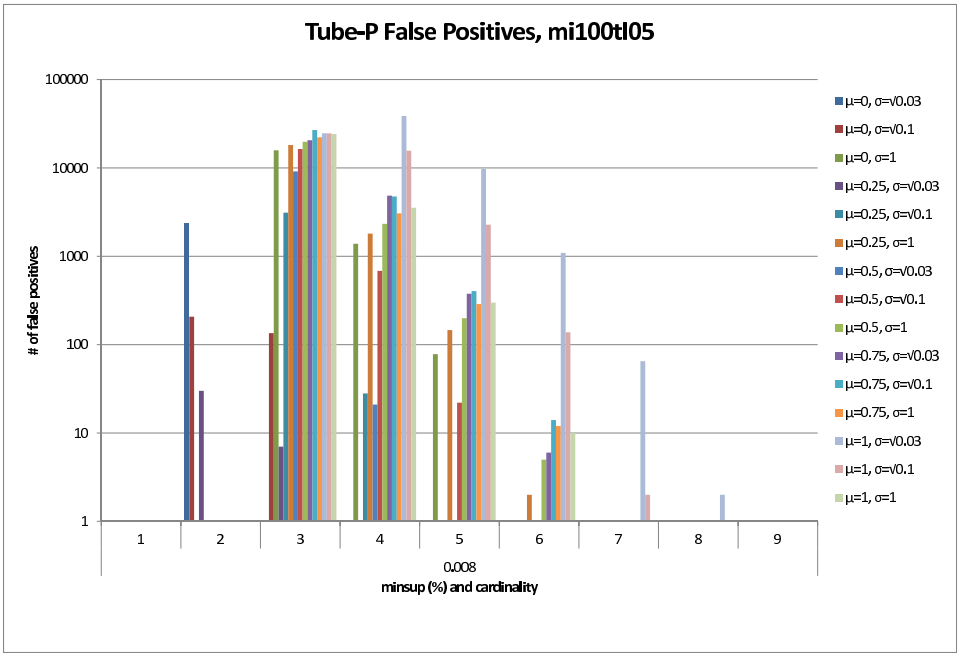


(a) CUF-growth*, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$

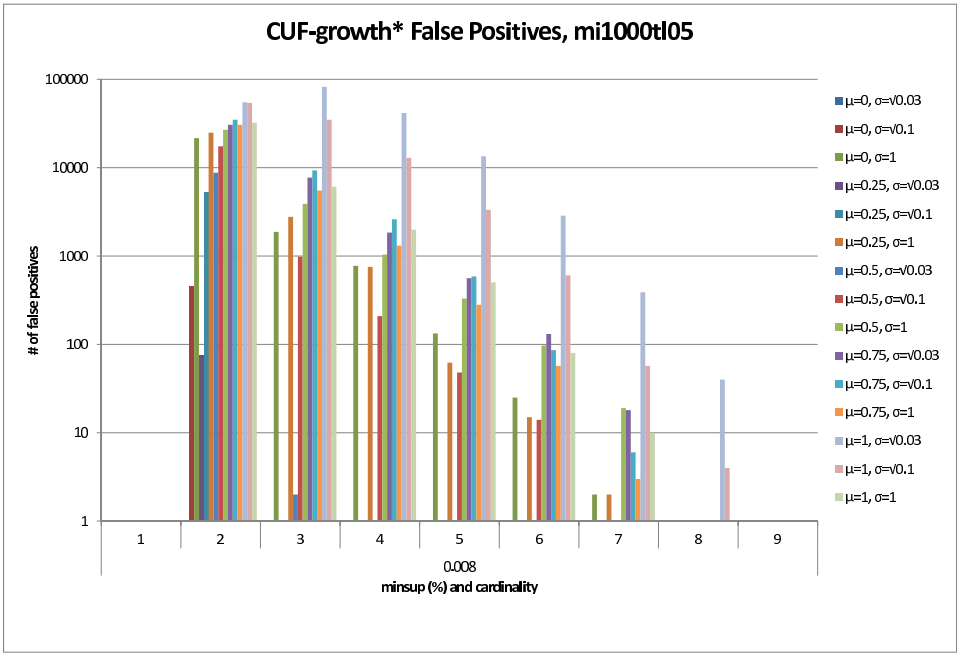


(b) Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$

Figure 5.33: False Positives: CUF-growth* & Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$

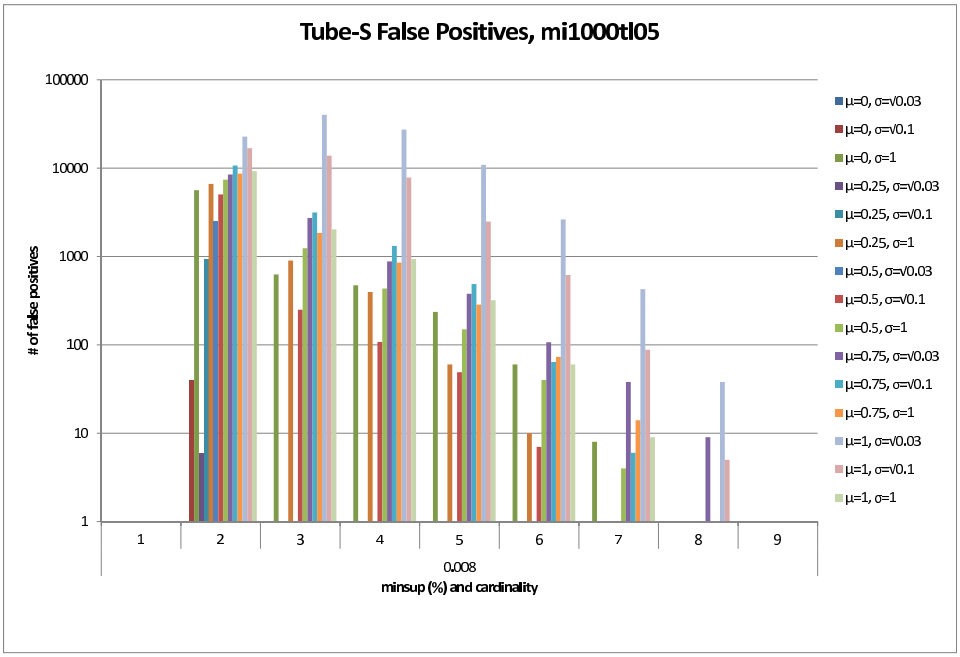


(a) Tube-P, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$

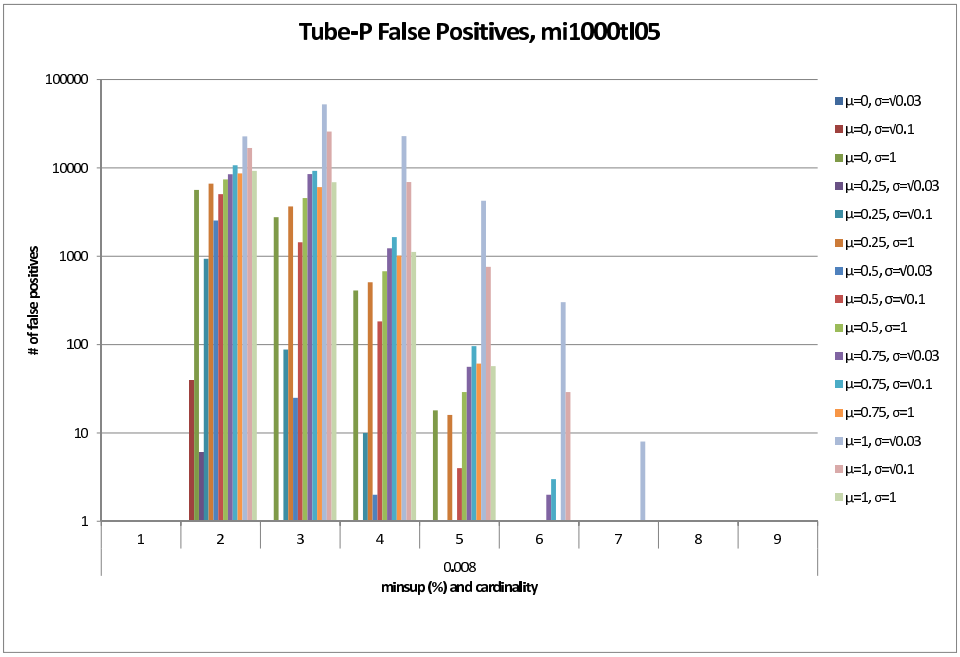


(b) CUF-growth*, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$

Figure 5.34: False Positives: Tube-P ($|\mathcal{I}| = 100$) & CUF-growth* ($|\mathcal{I}| = 1000$), $\overline{|t_j|} = 5$

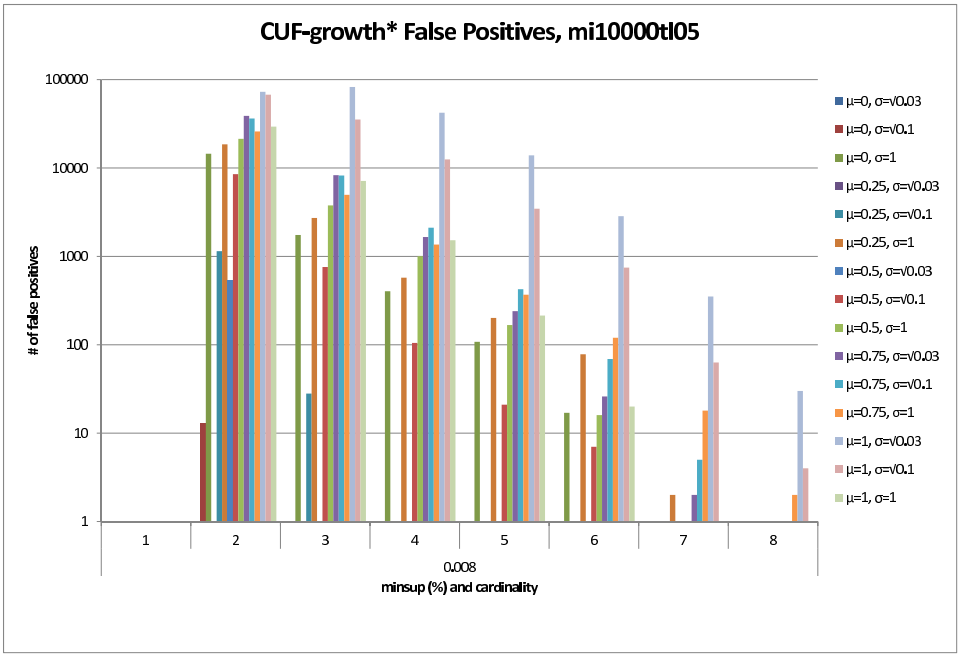


(a) Tube-S, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$

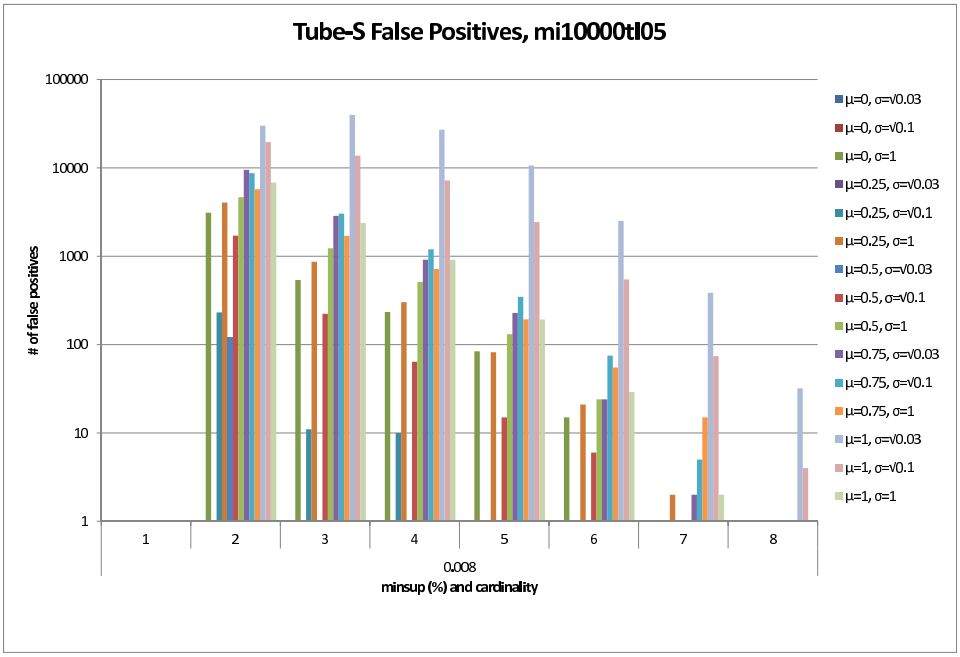


(b) Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$

Figure 5.35: False Positives: Tube-S & Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$



(a) CUF-growth*, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$



(b) Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$

Figure 5.36: False Positives: CUF-growth* & Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$

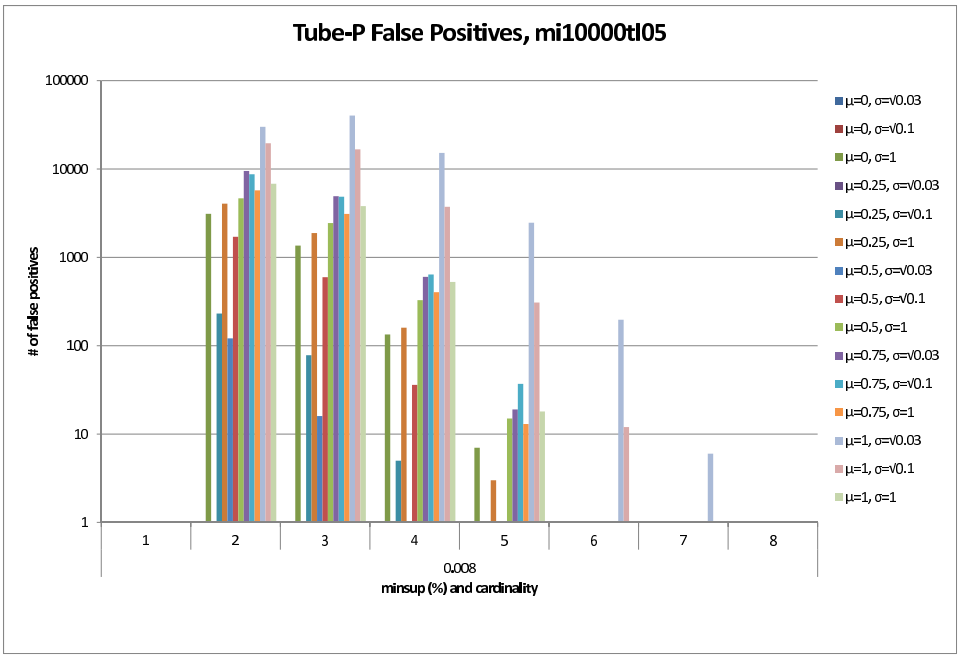


Figure 5.37: False Positives: Tube-P, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$

Transaction Length 12

Consider Figures 5.38(a), 5.38(b) and 5.39(a). Like with Section 5.2.4, there is now a much clearer picture of Tube-S and Tube-P outperforming CUF-growth*. Tube-P especially makes this easy to see with its reduction to zero false positives on so many distributions for high cardinalities. It may seem at a glance that Tube-P has so few false positives compared to Tube-S that there should be a bigger difference in their runtimes, however, this is not the case for two reasons:

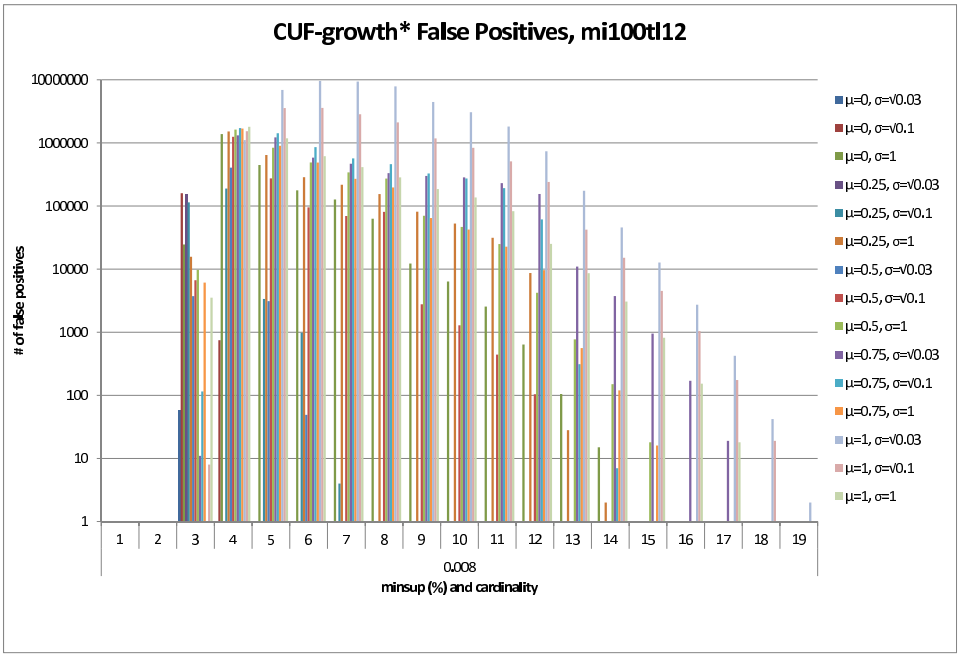
1. When false positives are mostly reduced in the high cardinalities, it still requires many levels of projected trees to be created before any pruning can happen. Hence, the reduction in runtimes associated with fewer high cardinality false positives is not as great as the reduction in runtimes associated with the same amount of fewer low cardinality false positives.

2. Because of the log scale on the y-axis, a decrease in the height of a bar by the same amount from Tube-S to Tube-P as from CUF-growth* to Tube-S actually represents a much smaller decrease in the number of false positives.

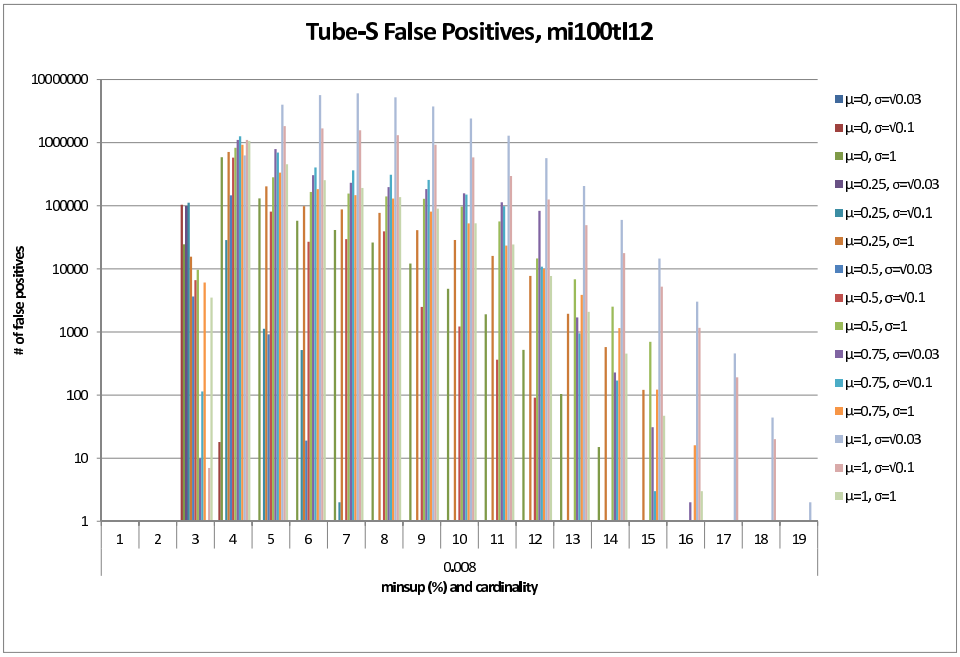
Continuing the trends seen with a transaction length of five, Tube-S overall shows much fewer false positives than CUF-growth*, but for higher cardinalities for some higher ranked distributions, there are a few more false positives. Similarly, Tube-P overall shows much fewer false positives than CUF-growth* and Tube-S, but for lower cardinalities for some lower ranked distributions, there are a few more false positives.

As the number of domain items increases; first to 1000 (cf. Figures 5.39(b), 5.40(a) and 5.40(b)), then to 10000 (cf. Figures 5.41(a), 5.41(b) and 5.42); I make the same

observations.

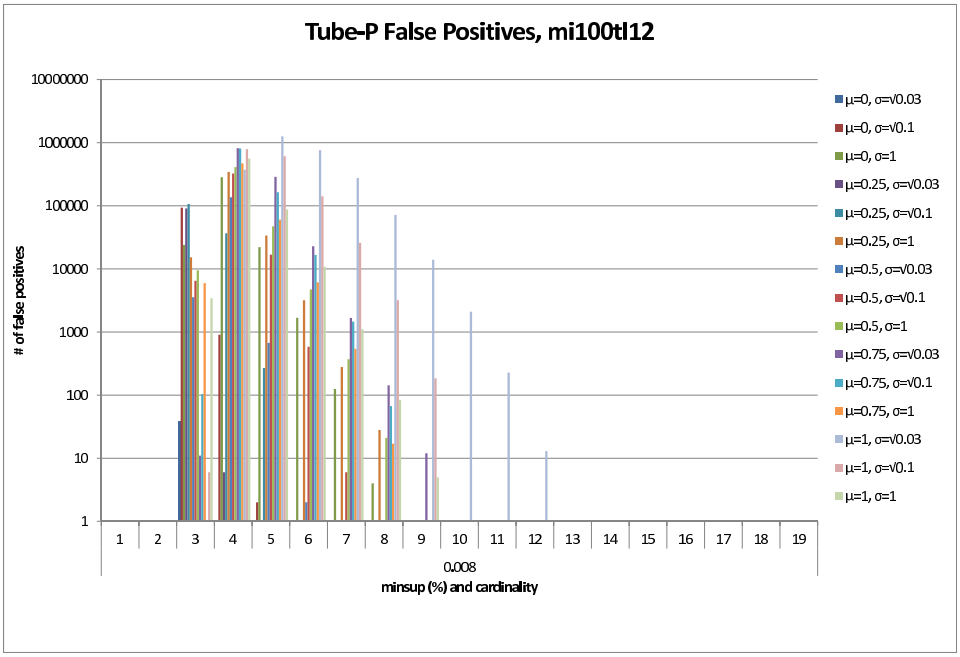


(a) CUF-growth*, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 12$

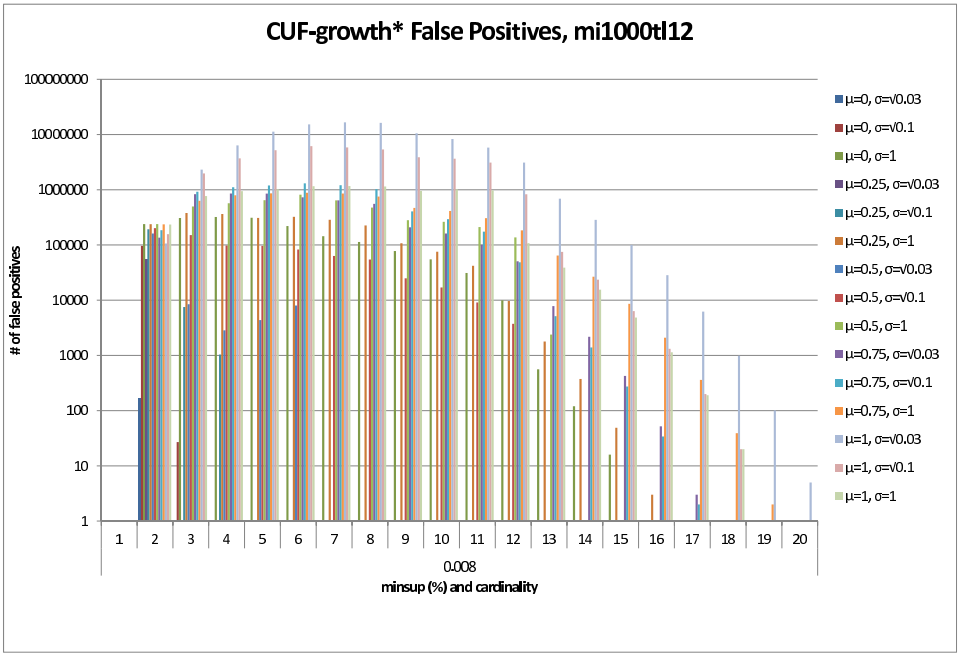


(b) Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 12$

Figure 5.38: False Positives: CUF-growth* & Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 12$

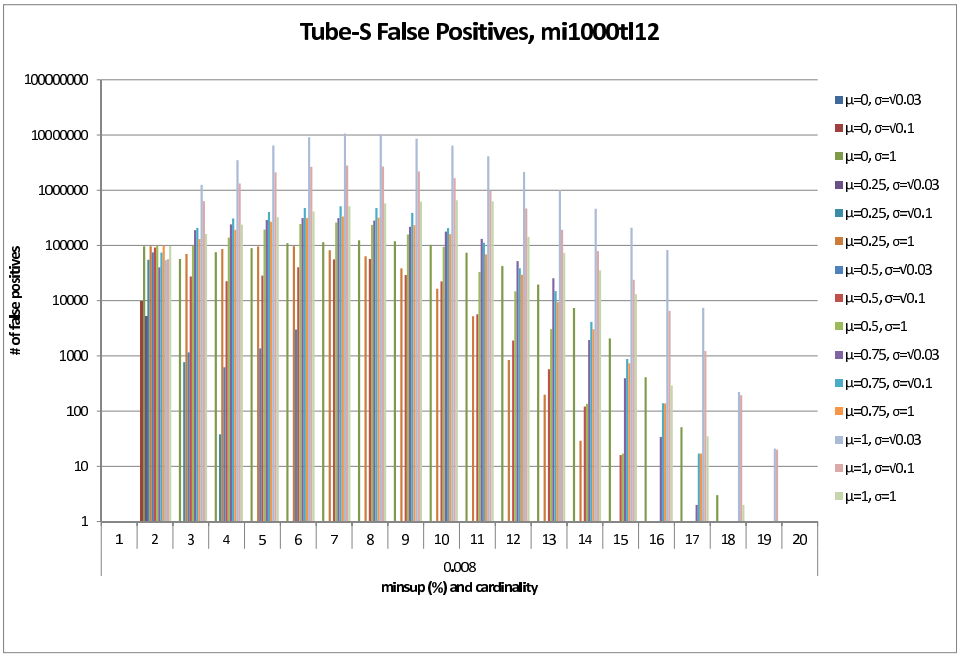


(a) Tube-P, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 12$

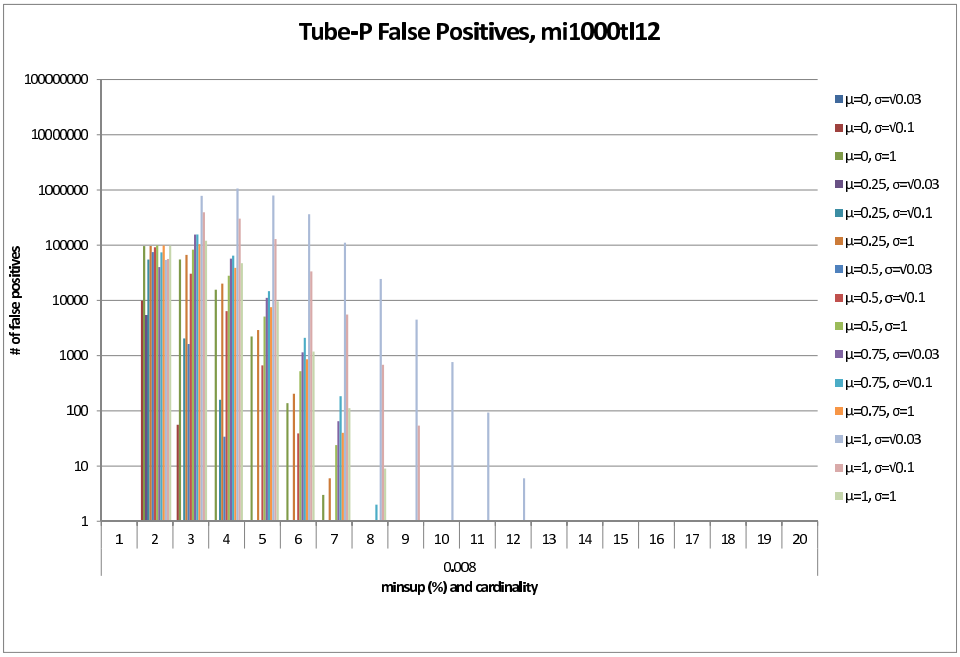


(b) CUF-growth*, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 12$

Figure 5.39: False Positives: Tube-P ($|\mathcal{I}| = 100$) & CUF-growth* ($|\mathcal{I}| = 1000$), $\overline{|t_j|} = 12$

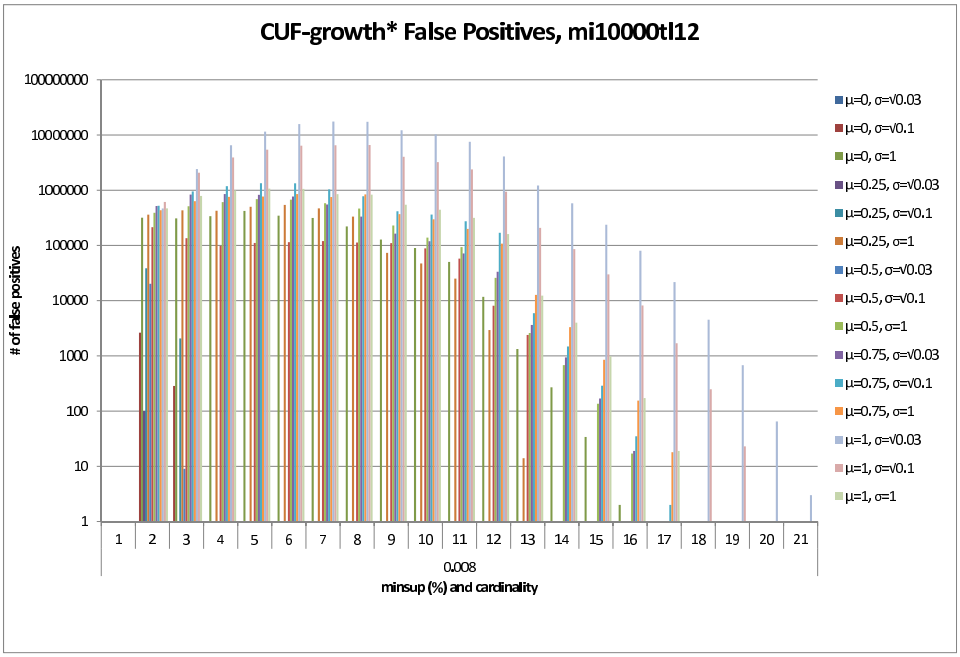


(a) Tube-S, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 12$

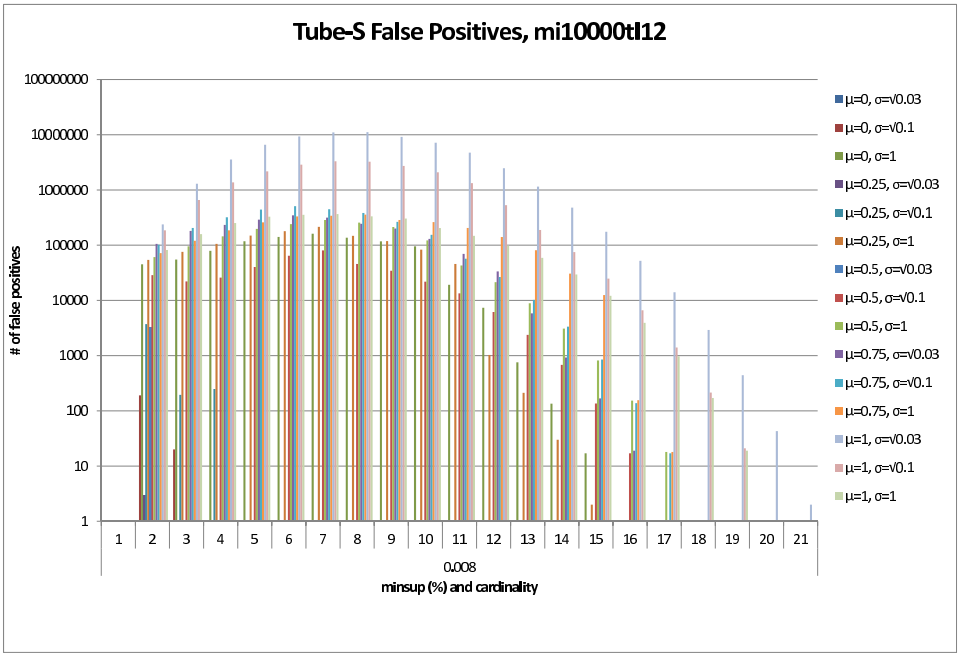


(b) Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 12$

Figure 5.40: False Positives: Tube-S & Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 12$



(a) CUF-growth*, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 12$



(b) Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 12$

Figure 5.41: False Positives: CUF-growth* & Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 12$

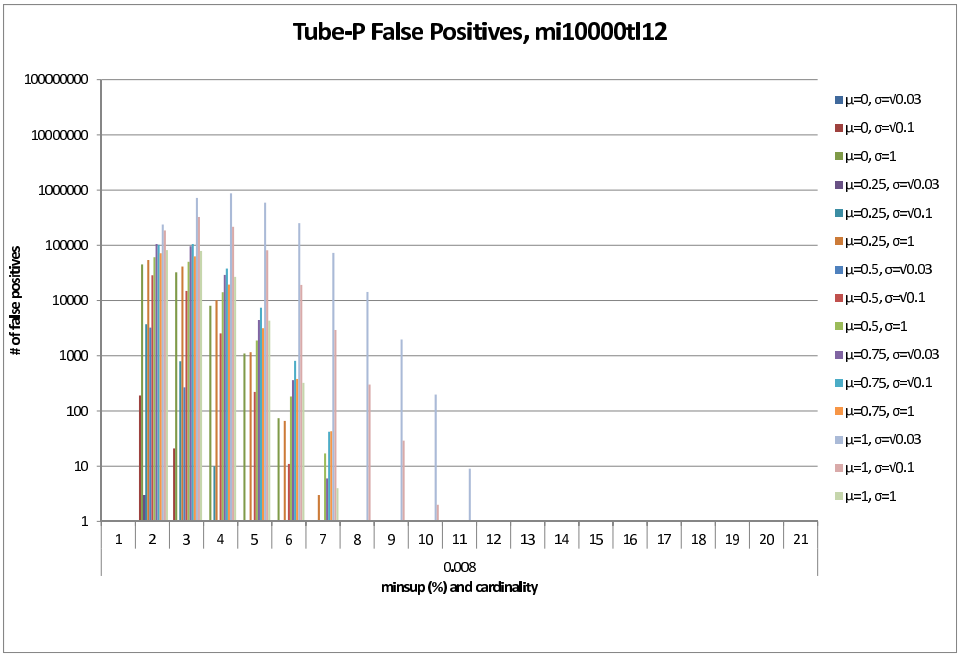
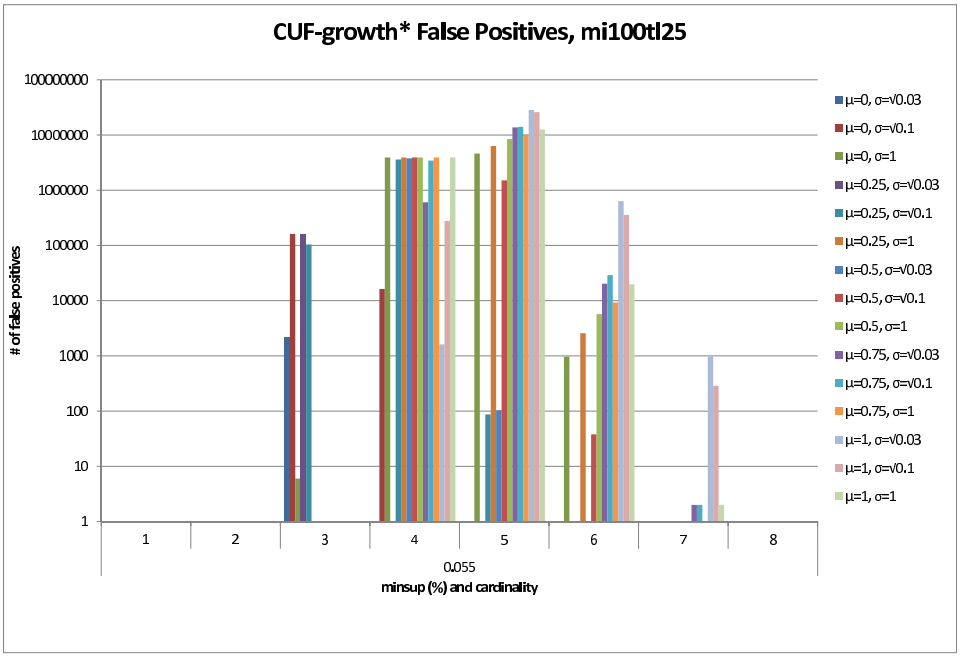


Figure 5.42: False Positives: Tube-P, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 12$

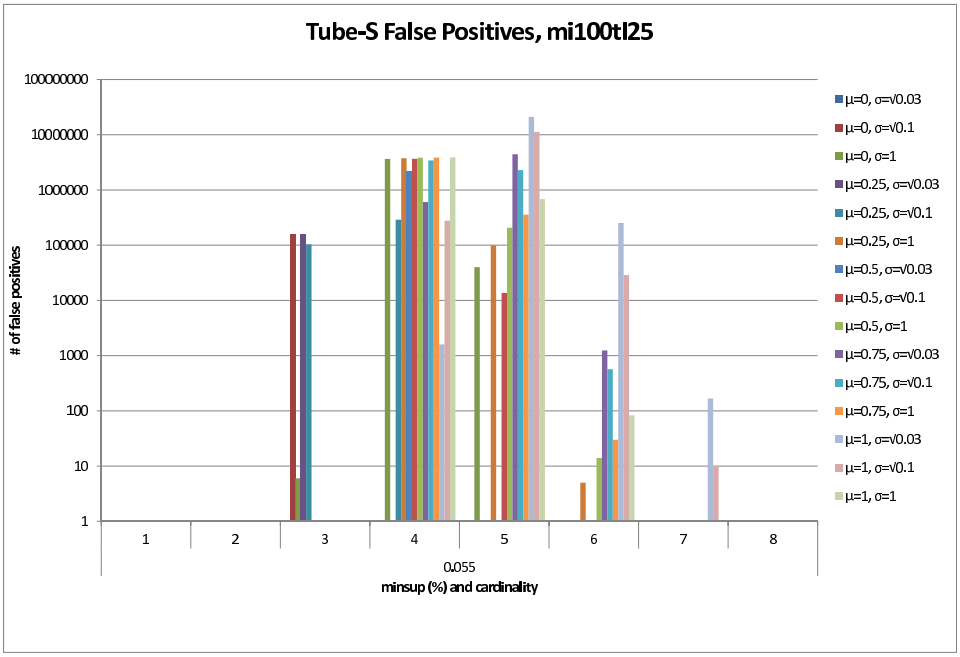
Transaction Length 25

Consider Figures 5.43(a), 5.43(b) and 5.44(a). Tube-S and Tube-P both show reduced or equal numbers of false positives compared to CUF-growth* for all distributions, with Tube-S showing a little bit more reduction on some of the lower ranked distributions, and Tube-P showing a little bit more reduction on some of the higher ranked distributions.

As the number of domain items increases to 1000 (cf. Figures 5.44(b), 5.45(a) and 5.45(b)) and later to 10000 (cf. Figures 5.46(a), 5.46(b) and 5.47) notice that an increase in the number of domain items while keeping *minsup* the same decreases the maximum cardinality reported for false positives, but if *minsup* is similarly decreased, the maximum cardinality is not affected as much. This is because these higher cardinality patterns do not occur with enough frequency when there are so many more domain items to choose from and *minsup* is unchanged. Lowering *minsup* allows the lower frequencies to exceed the upper bound and thus perhaps become false positives.

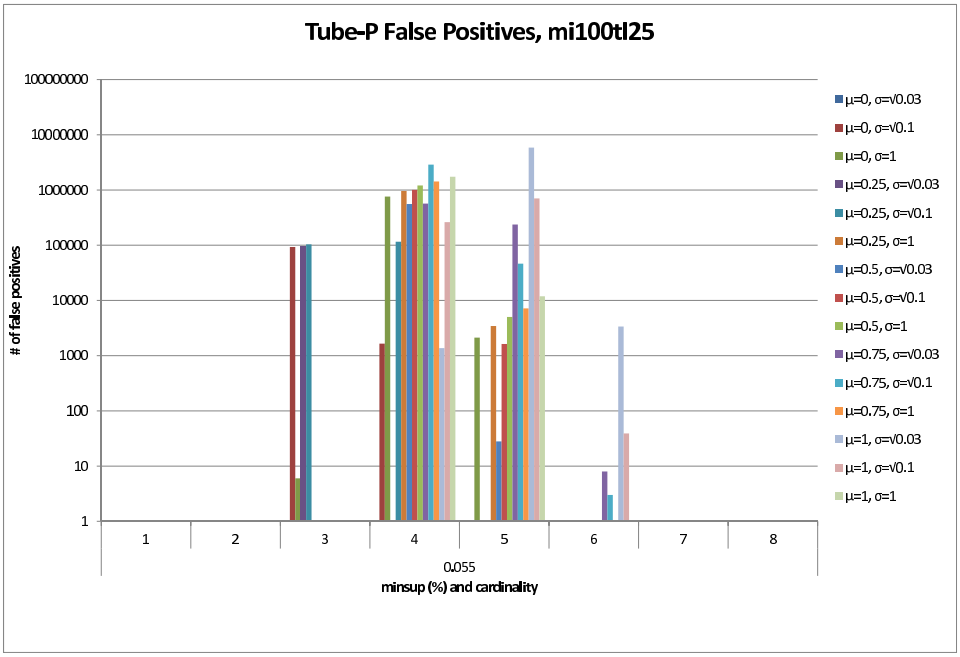


(a) CUF-growth*, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 25$

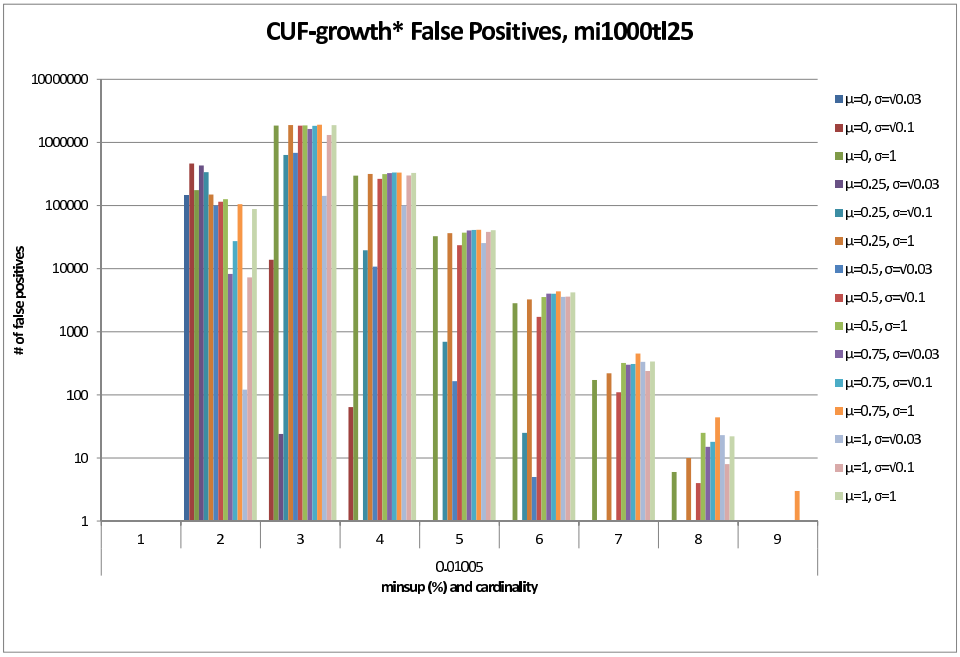


(b) Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 25$

Figure 5.43: False Positives: CUF-growth* & Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 25$

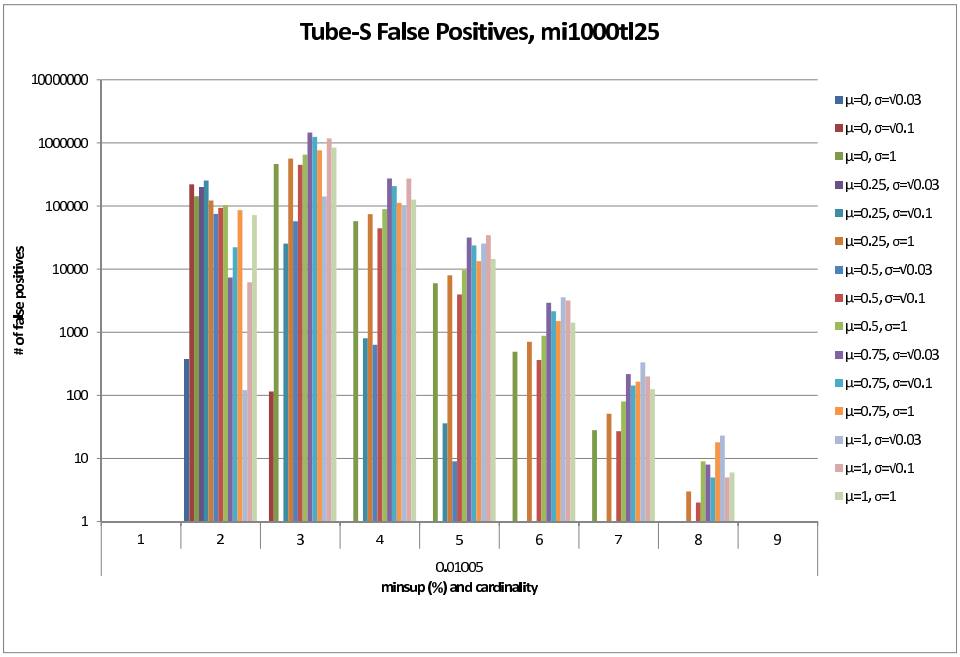


(a) Tube-P, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 25$

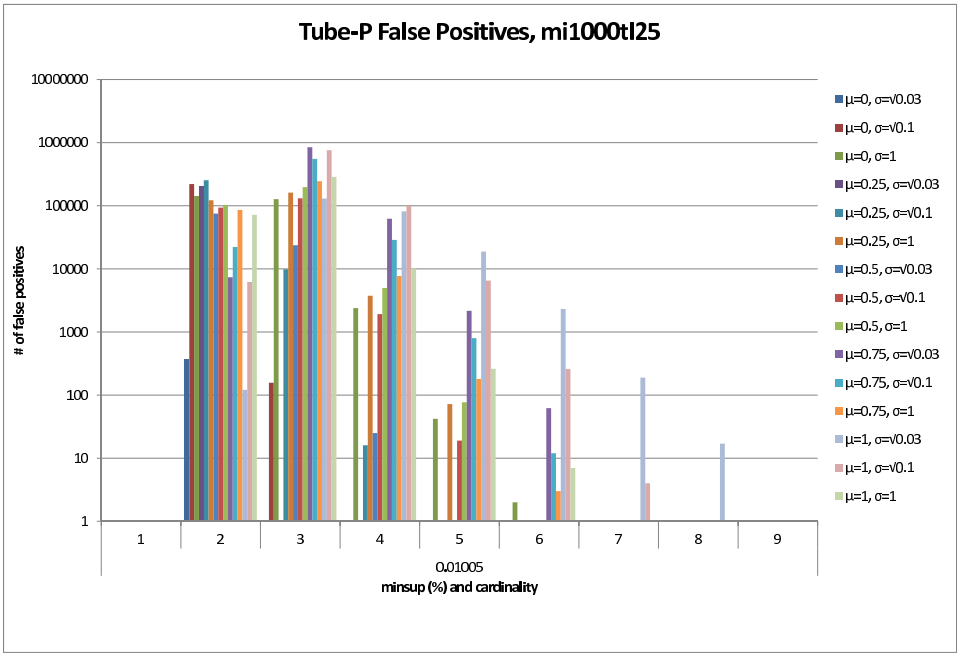


(b) CUF-growth*, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 25$

Figure 5.44: False Positives: Tube-P ($|\mathcal{I}| = 100$) & CUF-growth* ($|\mathcal{I}| = 1000$), $\overline{|t_j|} = 25$

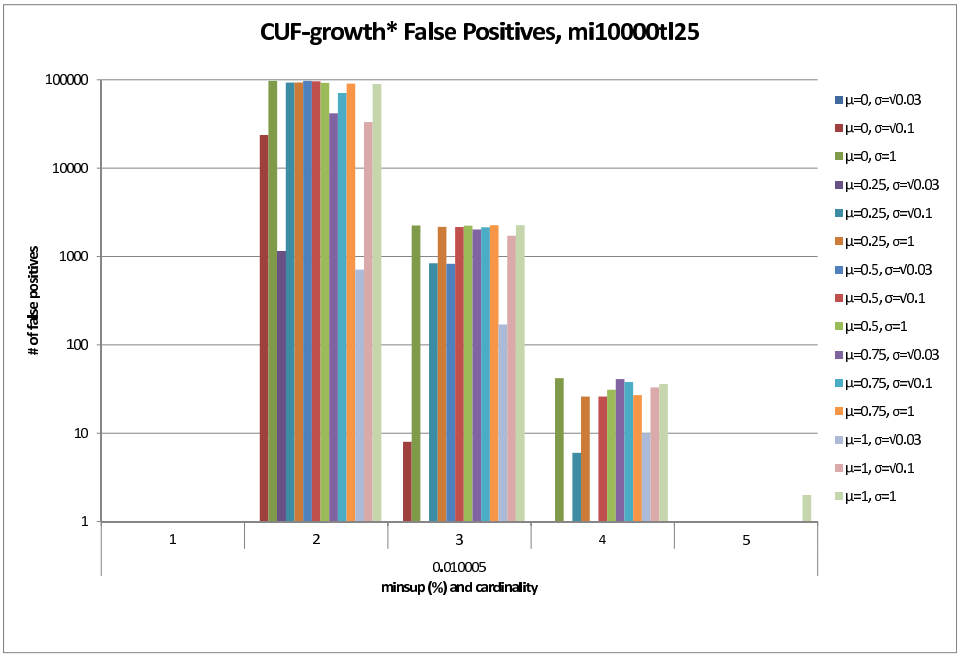


(a) Tube-S, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 25$

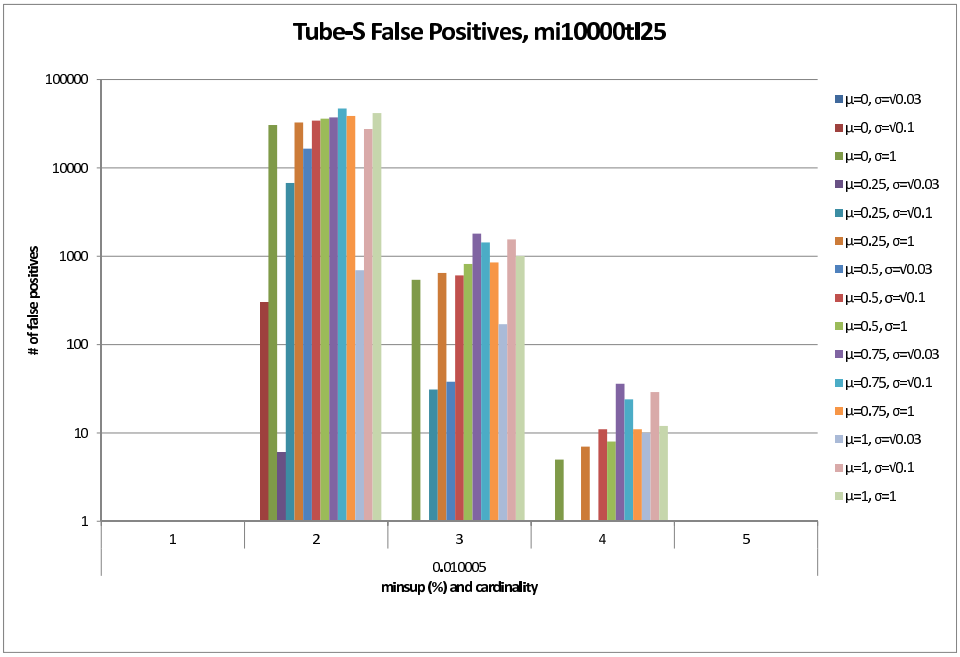


(b) Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 25$

Figure 5.45: False Positives: Tube-S & Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 25$



(a) CUF-growth*, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 25$



(b) Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 25$

Figure 5.46: False Positives: CUF-growth* & Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 25$

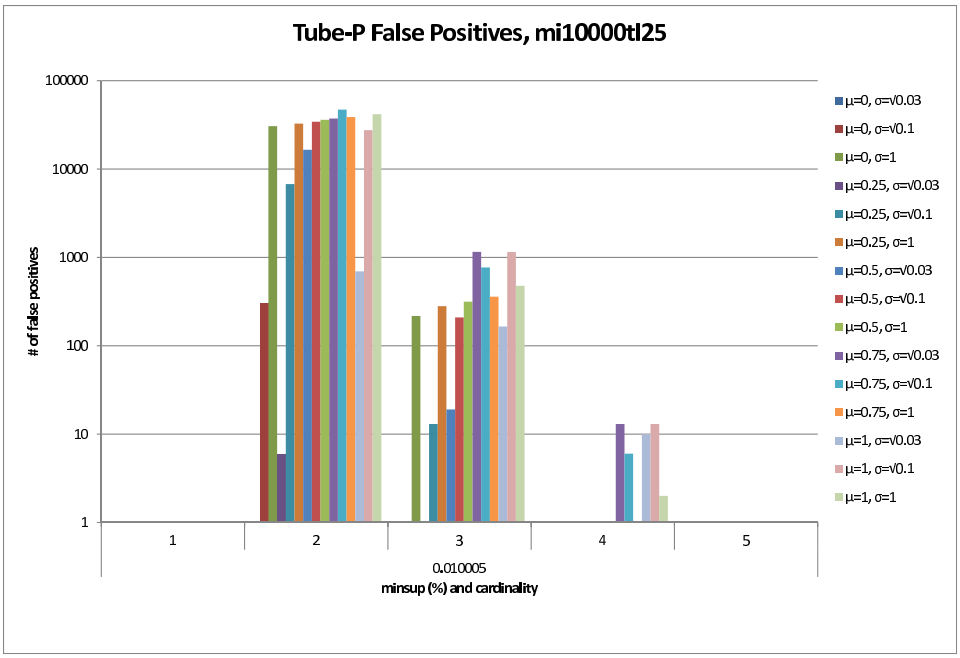


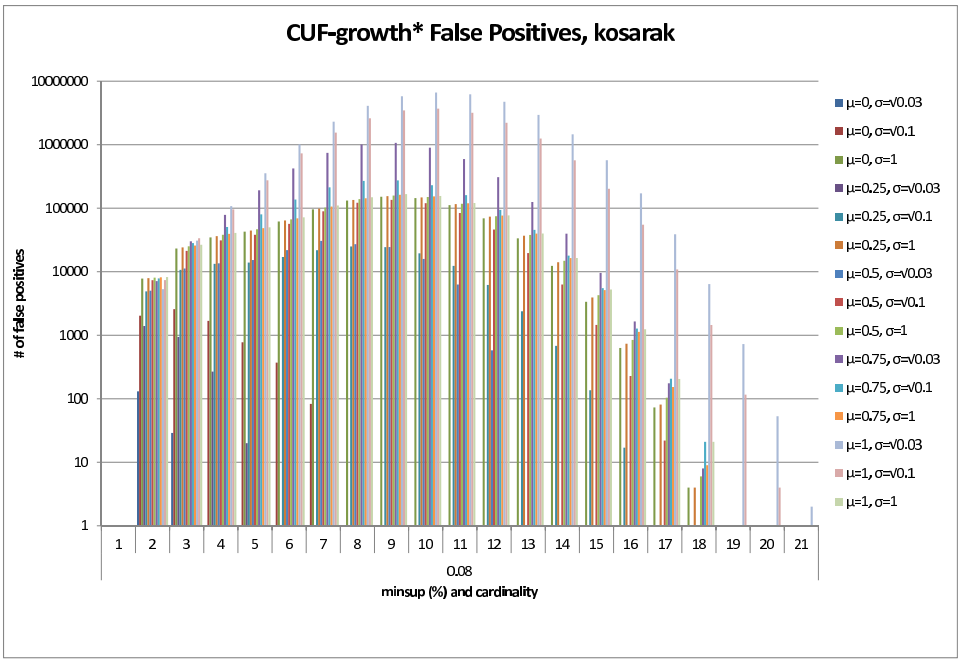
Figure 5.47: False Positives: Tube-P, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 25$

Real Data

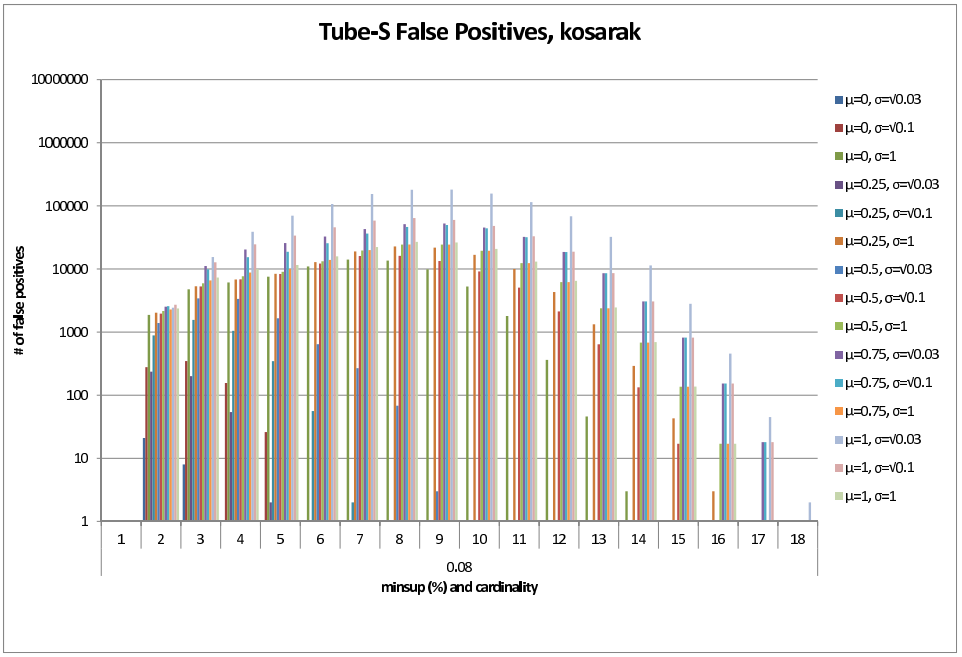
Consider the kosarak database in Figures 5.48(a), 5.48(b) and 5.49(a). Tube-S and Tube-P both reduced the numbers of false positives over CUF-growth* by a huge amount for all distributions, dropping to zero for the three highest cardinalities. Again, Tube-S shows more reduction on some of the lower ranked distributions, and Tube-P shows more reduction on some of the higher ranked distributions.

While not removing quite so many false positives as in kosarak, Tube-S and Tube-P still both clearly outperform CUF-growth* in the mushroom database, shown in Figures 5.49(b), 5.50(a) and 5.50(b). In these charts, the tendency of Tube-P to perform better on higher ranked distributions and Tube-S to perform better on lower ranked distributions in much more pronounced.

When considering the retail database in Figures 5.51(a), 5.51(b) and 5.52, there is a not insignificant reduction in false positives from CUF-growth* to Tube-S and a significant reduction when moving to Tube-P, mirroring the runtime behaviour of the two algorithms in Section 5.2.4.

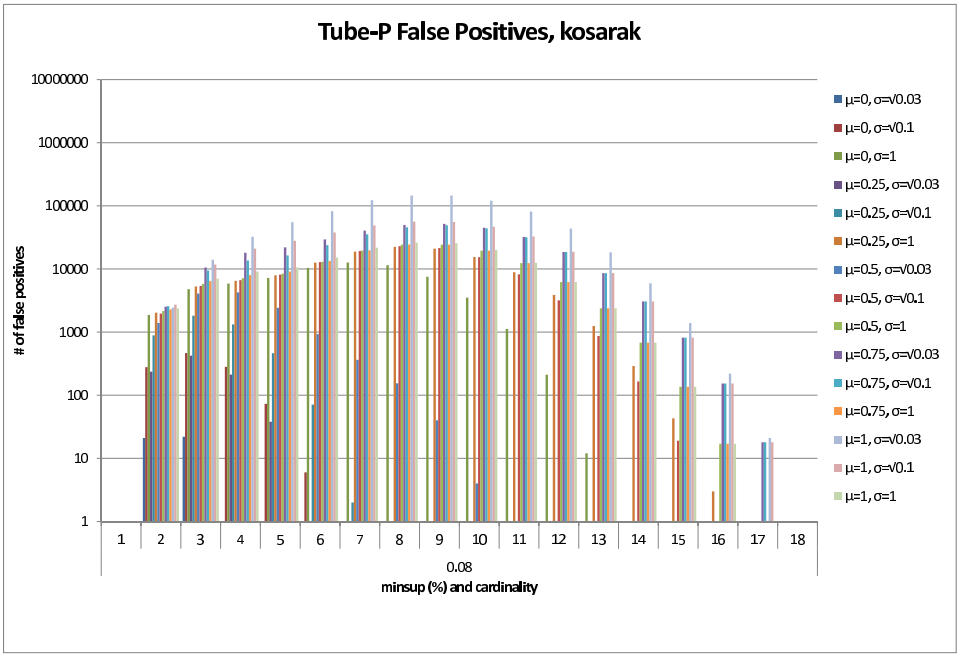


(a) CUF-growth* False Positives

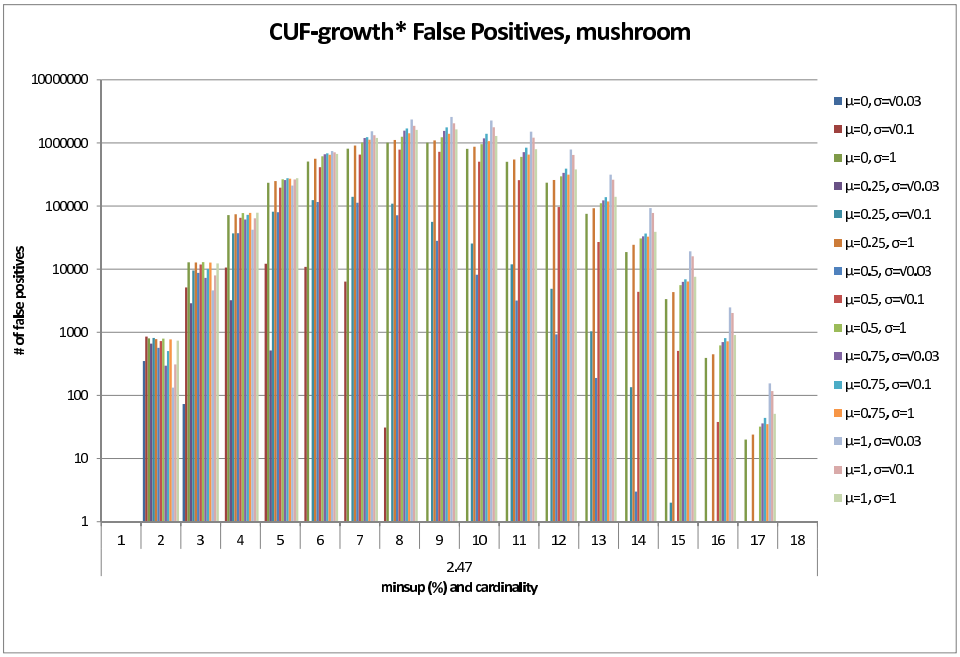


(b) Tube-S False Positives

Figure 5.48: False Positives: Kosarak, CUF-growth* & Tube-S

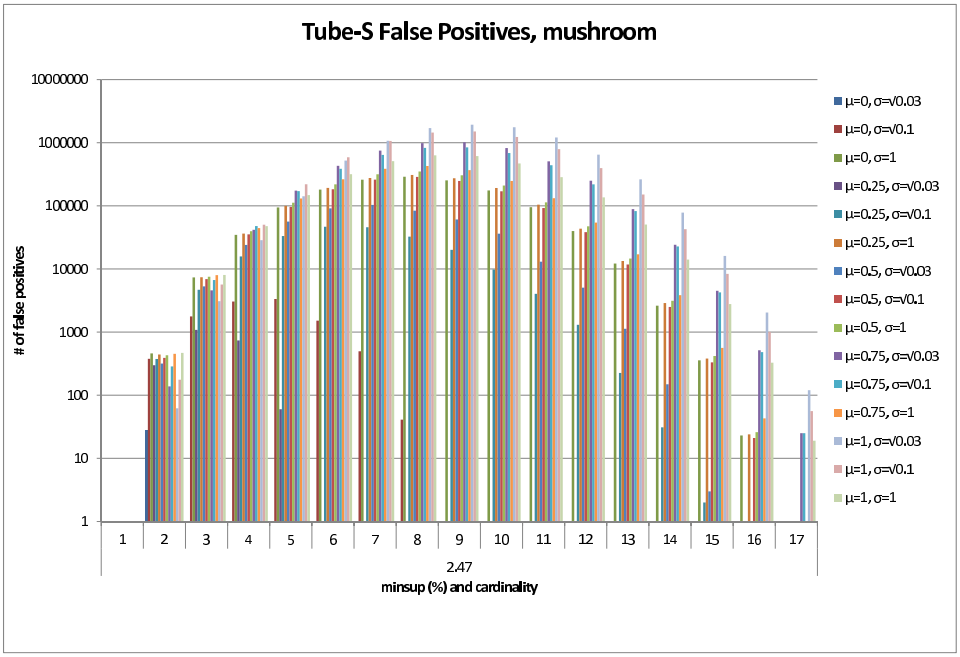


(a) Kosarak Tube-P False Positives

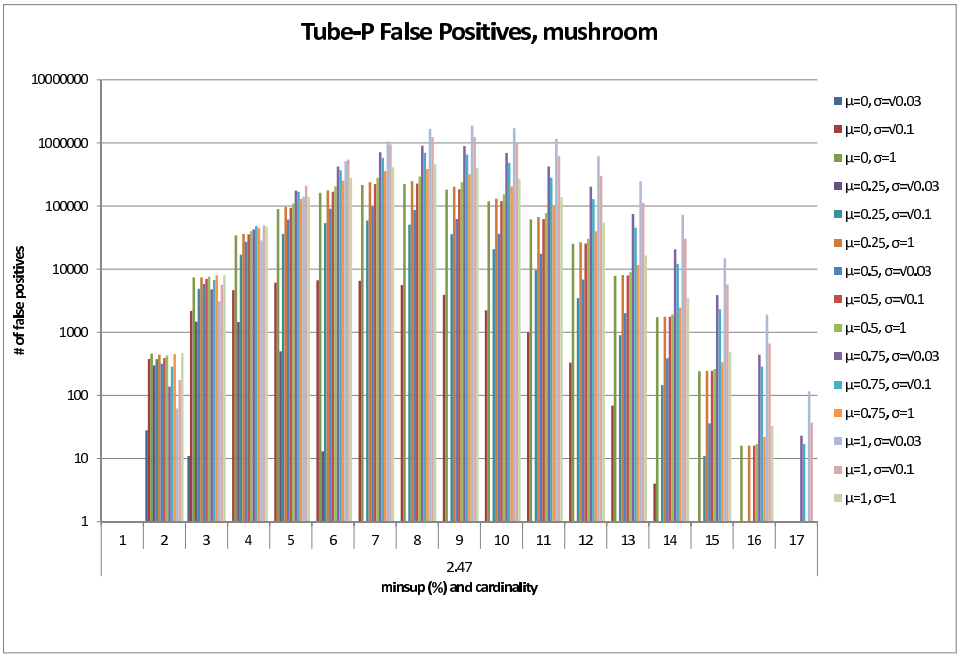


(b) Mushroom CUF-growth* False Positives

Figure 5.49: False Positives: (Kosarak, Tube-P) & (Mushroom, CUF-growth*)

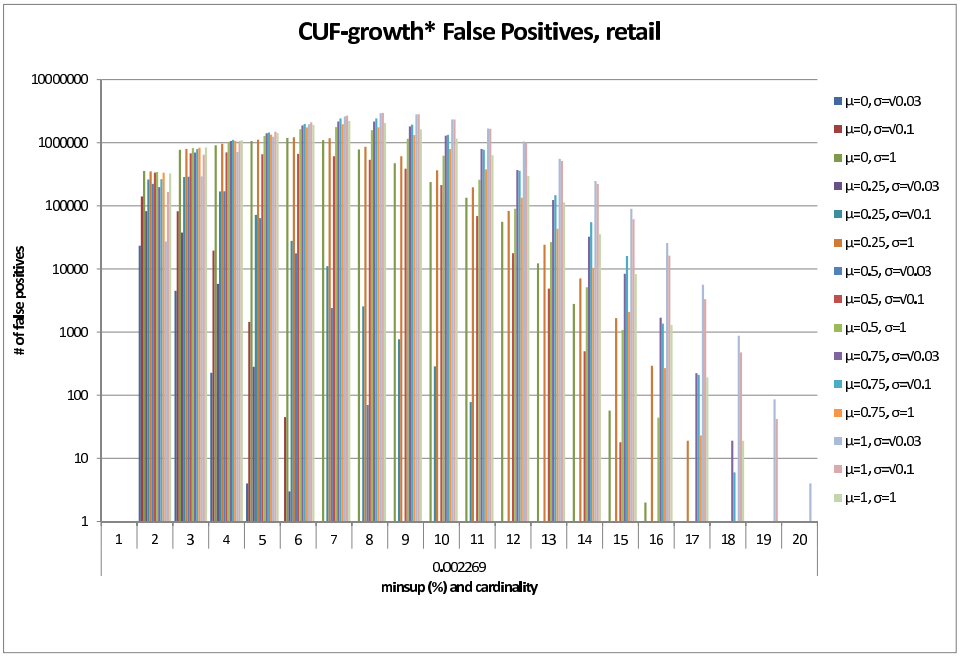


(a) Tube-S False Positives

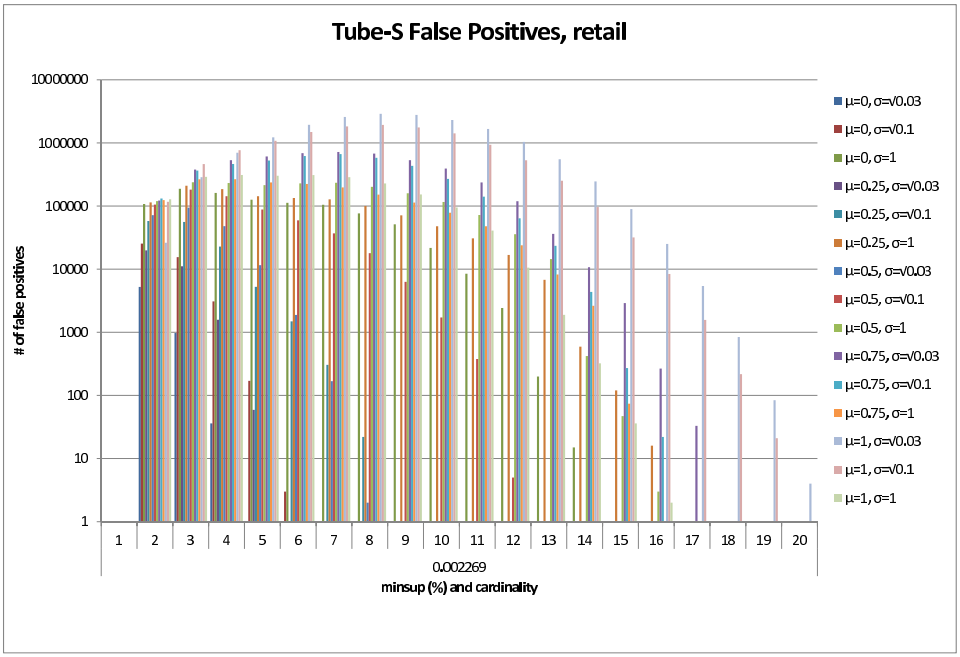


(b) Tube-P False Positives

Figure 5.50: False Positives: Mushroom, Tube-S & Tube-P



(a) CUF-growth* False Positives



(b) Tube-S False Positives

Figure 5.51: False Positives: Retail, CUF-growth* & Tube-S

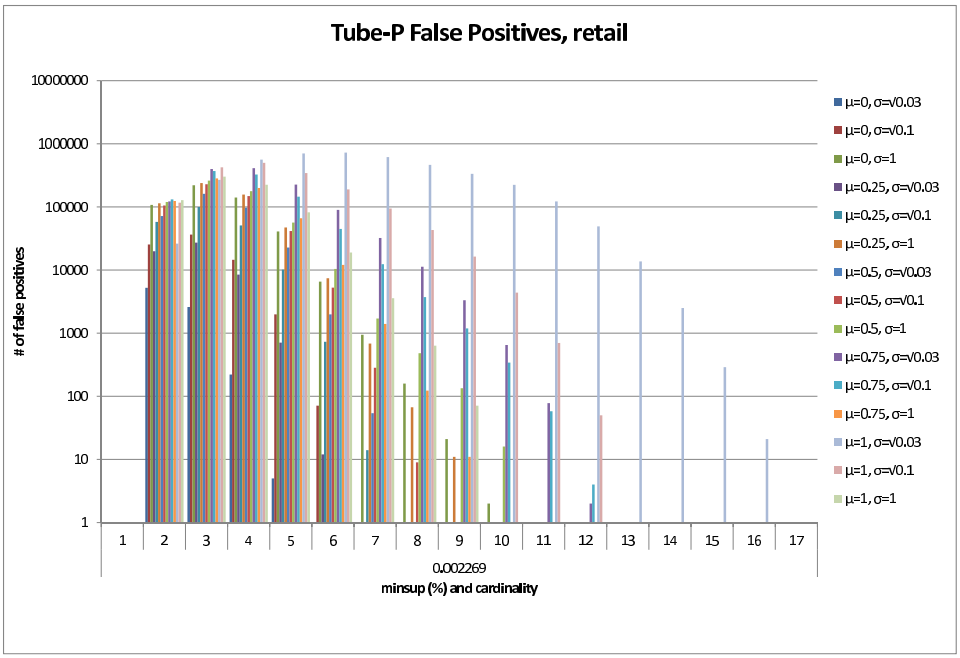


Figure 5.52: False Positives: Retail, Tube-P

Summary

My false positive observations are summarized below:

1. **No algorithm shows any false positives for a cardinality of one.** This is because the expected supports of all the single frequent patterns are calculated during the construction of the global tree in each algorithm before the mining even begins, thus upper bound calculations are not performed.

2. **As the cardinality increases towards its maximum, fewer and fewer false positives are found, and those false positives are mostly found in the high ranked distributions in Table 5.2.** This is because the upper bound in all three algorithms becomes tighter as the cardinality increases due to continued multiplications of CUF-growth*'s bronze value, Tube-S's M_2 value, and Tube-P's max existential probability values. The reason higher ranked distributions last longer (i.e., show false positives at higher cardinalities than lower ranked distributions) is because these are the ones most likely to not yet be pruned after multiple multiplications through higher level projected trees due to containing higher existential probability values.

3. **Both Tube-S and Tube-P have the exact same number of false positives for all candidate 2-itemsets.** This is because the calculation of the upper bound on expected support for 2-itemsets is the same in both algorithms.

4. **For short transaction lengths, Tube-S and Tube-P may actually produce more false positives than CUF-growth* for some distributions.** This is because while the upper bound provided by Tube-S and Tube-P is

tighter than CUF-growth* in the majority of cases, situations still exist where the bound provided by CUF-growth* is tighter, as in Example 5.1.

5. **Tube-S overall shows much fewer false positives than CUF-growth*, but for higher cardinalities for some higher ranked distributions, there are a few more false positives; similarly, Tube-P overall shows much fewer false positives than CUF-growth*, but for lower cardinalities for some lower ranked distributions, there are a few more false positives.** This is because the upper bound of Tube-S is more effective than Tube-P for lower cardinalities and lower ranked distributions, and vice-versa (cf. Section 5.2.4).

6. **For longer transaction lengths an increase in the number of domain items while keeping *minsup* the same decreases the maximum cardinality reported for false positives, but if *minsup* is similarly decreased, the maximum cardinality is not affected as much.** This is because these higher cardinality patterns do not occur with enough frequency when there are so many more domain items to choose from and *minsup* is unchanged. Lowering *minsup* allows the lower frequency candidate patterns to exceed the upper bound and thus perhaps become false positives. This effect is not as strong for shorter transaction lengths because not enough higher cardinality patterns occur in the first place.

5.2.7 Comparison of Algorithms

Given the above analyses, I can now offer a succinct recommendation of which situations favour the use of which algorithm. This is most easily expressed in a table format below. The columns delineate transaction length and number of domain items while the rows delineate distribution ranking and algorithm. For each of the inner columns and grouping of three rows, a cell will appear black to mark the best performing algorithm. If there are multiple algorithms that perform equally well in a given situation, multiple black cells will appear.

Table 5.3: Algorithm Comparison

			Transaction Length								
			Low			Medium			High		
		Alg./$\lvert\mathcal{I}\rvert$	*L.*	*M.*	*H.*	*L.*	*M.*	*H.*	*L.*	*M.*	*H.*
Distribution Ranking	Low	*CUF-growth**	■			■			■		
		Tube-S	■	■	■	■	■	■	■	■	■
		Tube-P	■			■			■		
	Med.	*CUF-growth**	■			■			■		
		Tube-S	■	■	■	■	■	■	■	■	■
		Tube-P	■			■			■		
	High	*CUF-growth**	■			■			■		
		Tube-S	■	■	■	■			■		
		Tube-P	■	■	■	■	■	■	■	■	■

5.3 Summary

From a theoretical analysis I conclude that Tube-S and Tube-P cannot be directly compared since cases can be constructed where each algorithm outperforms the other. In terms of items per node, Tube-S and Tube-P have fewer than algorithms with exact bounds but more than algorithms with looser upper bounds. Nevertheless, memory usage is still less due to fewer nodes being created in projected trees during the mining process. Before running my main experiments, I performed a correctness analysis to test the accuracy of my implementation. The analysis showed that Tube-S, Tube-P and CUF-growth* returned the same frequent patterns and expected supports as FP-growth and UF-growth, tested with the same input databases and *minsup*s. My main experiments consisted of a runtime analysis, a memory usage analysis focusing on the number of nodes and a second memory usage analysis focusing on the number of false positives at different cardinality levels. For each experiment multiple database sizes, densities, *minsup*s and probability distributions were tested. For detailed summaries from each experiment refer to Sections 5.2.4, 5.2.5 and 5.2.6 respectively. The experimental results showed that ***Tube-S and Tube-P outperformed CUF-growth**** in the vast majority of cases, the only exception (where CUF-growth*, Tube-S and Tube-P were equally effective) was in the case of a low transaction length and low number of domain items. Otherwise, Tube-S was faster than Tube-P for the low ranking distributions in Table 5.2 whereas Tube-P was faster for the high ranking distributions.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Data is useless if we do not use it. Before we use it, we need to understand it. And before that, we need to process and transform it in a way that makes its meaning apparent. Frequent pattern mining is an example of this transformation. Frequent pattern mining is the nontrivial extraction of implicit, previously unknown, and potentially useful information, *in the form of frequent patterns*, from data. This data can either be precise or uncertain, however, oftentimes what most people initially consider to be precise data are actually uncertain data when more details about its provenance are considered. For instance, data collection methodologies, measurement accuracies, instrument precision and changes in the environment all lend uncertainty to reported values. Moreover, forecasts, lossy compression, noise and any modelling of natural phenomena are inherently uncertain.

Previously proposed methods of frequent pattern mining from uncertain data in-

clude candidate-generate-and-test (Apriori-based) and pattern growth (largely tree-based) approaches. At the outset, I investigated four research questions and developed four algorithm qualities that could be evaluated theoretically and experimentally to guide this research. In response, I have explored improvements and new methods to mine frequent patterns from uncertain data using tree-based algorithms.

In Question 1 I asked: ***Can the upper bound on expected support be further reduced beyond the current state-of-the-art (cf. CUF-growth* [24])?*** In fact, Tube-S (cf. Section 4.1) and Tube-P (cf. Section 4.2) both offer tightened upper bounds on expected support over CUF-growth*. Tube-S achieves a tightened upper bound on expected support by considering the second highest existential probability in the proper prefix of an item in a transaction (the M_2 value). Alternatively, Tube-P achieves a tightened upper bound on expected support by using a new data layout and tree construction process.

In Question 2 I asked: ***Can the resulting tree still be as compact as CUF-growth*?***. In fact, Tube-S and Tube-P both have the exact same number of elements per node as CUF-growth*. In terms of the total number of nodes, Tube-S and Tube-P both have fewer total nodes than CUF-growth* for the vast majority of mining problems due to their tightened upper bounds.

In Question 3 I asked: ***How would frequent itemsets be mined from such a tree?***. In fact, the two mining algorithms I have proposed have a recursive structure, akin to previous tree-based frequent pattern mining algorithms but with some differences. The details are given in Sections 4.1.2 and 4.2.2 for Tube-S and Tube-P respectively, with worked out examples.

In Question 4 I asked: ***How would such an algorithm compare with previous uncertain data frequent pattern mining algorithms?***. In fact, I have carried out theoretical analyses (cf. Section 5.1) and experimental evaluations (cf. Sections 5.2.4, 5.2.5 and 5.2.6) on Tube-S and Tube-P, comparing them to previous work. To evaluate my algorithms, I performed a complexity analysis that focused on memory usage. In particular, I showed that because Tube-S and Tube-P both maintain the same amount of information in each tree node, the total amount of memory used depends on the extra nodes created in projected databases during the mining. I showed that it was impossible to determine analytically whether one algorithm used more or less memory than the other, however, when compared to previous algorithms, Tube-S and Tube-P both had tighter bounds than the other non-exact algorithms. To help fill in the gaps, I conducted multiple experiments with various synthetic and real databases testing different densities (varying transaction length and number of domain items) and probability distributions. The results showed that Tube-P outperforms Tube-S on datasets where the existential probability has a high variance and a high mean, while Tube-S outperforms Tube-P on datasets with a high variance and a low mean. For lower variance datasets the two algorithms' performances are comparable. Both algorithms were faster than the previous state-of-the-art, CUF-growth* [24].

Overall, Tube-S and Tube-P fulfill the four qualities I was looking for in algorithms that might improve upon the previous work. They each *discover all frequent patterns from uncertain data*. Specifically, due to the initial global tree being a lossy compressed representation of the input database in each case, Tube-S and Tube-P first

find a superset of the frequent patterns from the uncertain data before performing a final database scan to eliminate any false positives, thus discovering all and only those frequent pattern from the uncertain data. As mentioned in the answer to Question 1, Tube-S and Tube-P both *demonstrate new approaches to calculating upper bounds on expected support.* Per the answer to Question 3, they *implement data structures that facilitate mining using the upper bounds.* Finally, in part drawn from the answers of every Question, they *offer competitive run-times while keeping memory requirements at a minimum.*

6.2 Future Work

There are several areas I am considering for which this thesis work can be expanded in the future. In particular, Big data is an area of much research and industry interest at the moment. In order to scale my algorithms to operate on the terabyte sized datasets that are increasingly becoming the norm in large Internet-based companies, research and modification to the fundamental data structures and algorithms will be necessary. Secondly, while the algorithms I have discussed (mine and others) have so far all used an expected support probability model, other probability models exist. Supporting these other probability models directly or by approximation and comparing the quality of the results between them would be an important contribution to the field. Finally, my algorithms find the entire set of frequent itemsets within a dataset, but there are other kinds of itemsets besides frequent itemsets that provide utility to users. Examples include: constrained, maximal, and/or closed frequent itemsets. Adapting and optimizing my work to mine these other itemsets as well is

another future direction.

6.2.1 Big Data and Distributed Environments

One of the most promising areas for extending this work is in adapting it to handle Big data. My current implementations of the Tube algorithms run on a single machine on a single thread, which do not take advantage of the full computing power even on that single machine environment. Distributed and/or MapReduce implementations will more easily be able to handle larger quantities of data by distributing the memory and computational burden among multiple machines. A few algorithms have already been proposed [20], [38], [41] for tree-based frequent pattern mining using the current Big data frameworks. The common approach in all of them is to mine sub-trees on different worker nodes. This makes an assumption that the individual sub-trees will be small enough and use little enough memory so as not to overwhelm the resources available on the worker nodes, however, even these sub-trees may require more resources than are available to them when operating on large datasets. I believe that a better, albeit less obvious, partitioning of the problem exists besides dividing the work by subtree that will evenly distribute the memory load. Identifying heavily- or under-used portions of the tree will be an important first step to understanding this partitioning, enabling the problem to be split over multiple workers and swapped to disk when needed to manage memory. Developing new techniques to allow tree-based mining algorithms to perform efficiently in a distributed environment is a future direction with implications far beyond a direct extension of this work.

6.2.2 Another Probability Model

While the expected support or frequency is a useful statistic in limiting the amount of frequent itemsets returned to the user, this statistic loses information about the number of possible worlds an itemset is frequent in. An alternative probability model that finds so called probabilistic frequent itemsets maintains this information by requiring the user to specify a *minsup* and a *minprob*, the minimum probability with which an itemset appears more than *minsup* times in the database [5]. FP-tree based algorithms using these semantics (cf. ProFP-Growth [6]) already exist, however, it is been shown that any expected support based algorithm can be extended to handle this case with no additional computational complexity if the variance of the support of each itemset is additionally tracked [33]. An interesting future direction would be to incorporate this functionality into our algorithms and to compare the results with algorithms developed solely for the probabilistic model. Concurrent to the development of this thesis, Xu *et al.* [38] have extended the PUF-growth algorithm [25] to do just this in a MapReduce environment. Considering that the Tube algorithms are further improvements on PUF-growth, a similar adaptation and comparison among all the algorithms would likely yield positive results as well offer a meaningful comparison with the previous work due to the similar data structures used.

6.2.3 Other Types of Itemsets

There may be some cases where finding all the frequent itemsets is not necessary, or it may be the case that mining other types of itemsets is less resource intensive due to the natures of these other itemsets. It is for these situations that constrained [10],

maximal [18], closed and gradual itemsets [11] among others have been proposed. Constrained mining is actually a more general view of frequent itemset mining, as frequent itemset mining can be considered to be itemset mining with a minimum frequency constraint, but other types of constraints exist such as: subset, max, min, sum and average, to name a few. Combinations of these constraints reduce the search space and resource requirements of a frequent itemset mining algorithm so that only itemsets satisfying the user-specified constraints are found. The sets of maximal and closed itemsets are compact representations of the set of frequent itemsets for a given problem. Maximal frequent itemsets require that no supersets of a maximal frequent itemset be frequent, while closed frequent itemsets require the slightly looser condition that no supersets of a closed frequent itemset have the same expected support as it. Maximal and closed itemsets are useful in space critical applications because of their compact portrayal of all frequent itemsets, however only closed itemsets contain enough information to reconstruct the entire set of frequent itemsets if necessary. Gradual frequent itemsets pair a direction with each item to identify trends in data such as: when the air pressure decreases, precipitation increases. Extending my algorithms with additional pruning capabilities so that they can be used to mine other types of itemsets is another possible direction of future work.

Appendix A

Database Characteristics

Table A.1: Probability Distribution Mappings for Existential Support Values

μ	σ	**Distribution Code**
0	$\sqrt{0.03}$	01
0	$\sqrt{0.1}$	02
0	1	03
0.25	$\sqrt{0.03}$	04
0.25	$\sqrt{0.1}$	05
0.25	1	06
0.5	$\sqrt{0.03}$	07
0.5	$\sqrt{0.1}$	08
0.5	1	09
0.75	$\sqrt{0.03}$	10
0.75	$\sqrt{0.1}$	11
0.75	1	12
1	$\sqrt{0.03}$	13
1	$\sqrt{0.1}$	14
1	1	15

Table A.2: Synthetic Database Characteristics

Database	Min Item	Max Item	Avg. Length	Density
s10kmi10ktl05pdist01	2	9	5.506700	0.00055061
s250kmi10ktl05pdist01	1	10	5.496212	0.00054962
s500kmi10ktl05pdist01	1	10	5.498548	0.00054985
s750kmi10ktl05pdist01	1	10	5.497453	0.00054974
s1Mmi10ktl05pdist01	1	10	5.498138	0.00054981
s10kmi10ktl05pdist02	2	9	5.497000	0.00054965
s250kmi10ktl05pdist02	1	10	5.495624	0.00054956
s500kmi10ktl05pdist02	1	10	5.497084	0.00054971
s750kmi10ktl05pdist02	1	10	5.497341	0.00054973
s1Mmi10ktl05pdist02	1	10	5.497607	0.00054976
s10kmi10ktl05pdist03	2	9	5.501400	0.00055008
s250kmi10ktl05pdist03	1	10	5.497740	0.00054977
s500kmi10ktl05pdist03	1	10	5.497538	0.00054975
s750kmi10ktl05pdist03	1	10	5.496689	0.00054967
s1Mmi10ktl05pdist03	1	10	5.496828	0.00054968
s10kmi10ktl05pdist04	2	9	5.505700	0.00055050
s250kmi10ktl05pdist04	1	10	5.500996	0.00055010
s500kmi10ktl05pdist04	1	10	5.500630	0.00055006
s750kmi10ktl05pdist04	1	10	5.500905	0.00055009

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi10ktl05pdist04	1	10	5.499980	0.00055000
s10kmi10ktl05pdist05	2	10	5.508800	0.00055084
s250kmi10ktl05pdist05	1	10	5.498984	0.00054990
s500kmi10ktl05pdist05	1	10	5.499832	0.00054998
s750kmi10ktl05pdist05	1	10	5.497993	0.00054980
s1Mmi10ktl05pdist05	1	10	5.497611	0.00054976
s10kmi10ktl05pdist06	2	9	5.502300	0.00055018
s250kmi10ktl05pdist06	1	10	5.502692	0.00055027
s500kmi10ktl05pdist06	1	10	5.502016	0.00055020
s750kmi10ktl05pdist06	1	10	5.501112	0.00055011
s1Mmi10ktl05pdist06	1	10	5.500508	0.00055005
s10kmi10ktl05pdist07	2	9	5.503900	0.00055034
s250kmi10ktl05pdist07	1	10	5.497856	0.00054978
s500kmi10ktl05pdist07	1	10	5.496906	0.00054969
s750kmi10ktl05pdist07	1	10	5.497177	0.00054972
s1Mmi10ktl05pdist07	1	10	5.497382	0.00054974
s10kmi10ktl05pdist08	2	10	5.492000	0.00054915
s250kmi10ktl05pdist08	1	10	5.497684	0.00054977
s500kmi10ktl05pdist08	1	10	5.498976	0.00054990
s750kmi10ktl05pdist08	1	10	5.498127	0.00054981

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi10ktl05pdist08	1	10	5.497718	0.00054977
s10kmi10ktl05pdist09	1	9	5.490100	0.00054896
s250kmi10ktl05pdist09	1	10	5.495528	0.00054955
s500kmi10ktl05pdist09	1	10	5.498020	0.00054980
s750kmi10ktl05pdist09	1	10	5.498296	0.00054983
s1Mmi10ktl05pdist09	1	10	5.497978	0.00054980
s10kmi10ktl05pdist10	1	9	5.519700	0.00055191
s250kmi10ktl05pdist10	1	10	5.495796	0.00054958
s500kmi10ktl05pdist10	1	10	5.496286	0.00054963
s750kmi10ktl05pdist10	1	10	5.498749	0.00054987
s1Mmi10ktl05pdist10	1	10	5.498814	0.00054988
s10kmi10ktl05pdist11	1	9	5.491600	0.00054910
s250kmi10ktl05pdist11	1	10	5.497776	0.00054978
s500kmi10ktl05pdist11	1	10	5.499300	0.00054993
s750kmi10ktl05pdist11	1	10	5.498677	0.00054987
s1Mmi10ktl05pdist11	1	10	5.499253	0.00054993
s10kmi10ktl05pdist12	1	9	5.491900	0.00054913
s250kmi10ktl05pdist12	1	10	5.502804	0.00055028
s500kmi10ktl05pdist12	1	10	5.502694	0.00055027
s750kmi10ktl05pdist12	1	10	5.501837	0.00055018

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi10ktl05pdist12	1	10	5.501708	0.00055017
s10kmi10ktl05pdist13	2	9	5.501800	0.00055012
s250kmi10ktl05pdist13	1	10	5.499028	0.00054990
s500kmi10ktl05pdist13	1	10	5.498848	0.00054988
s750kmi10ktl05pdist13	1	10	5.498688	0.00054987
s1Mmi10ktl05pdist13	1	10	5.498725	0.00054987
s10kmi10ktl05pdist14	2	9	5.492000	0.00054914
s250kmi10ktl05pdist14	1	10	5.501304	0.00055013
s500kmi10ktl05pdist14	1	10	5.500792	0.00055008
s750kmi10ktl05pdist14	1	10	5.499616	0.00054996
s1Mmi10ktl05pdist14	1	10	5.499681	0.00054997
s10kmi10ktl05pdist15	2	10	5.481700	0.00054811
s250kmi10ktl05pdist15	1	10	5.496932	0.00054969
s500kmi10ktl05pdist15	1	10	5.500352	0.00055003
s750kmi10ktl05pdist15	1	10	5.498616	0.00054986
s1Mmi10ktl05pdist15	1	10	5.498956	0.00054990
s10kmi10ktl12pdist01	2	21	12.513200	0.00125123
s250kmi10ktl12pdist01	1	24	12.494992	0.00124949
s500kmi10ktl12pdist01	1	24	12.496814	0.00124968
s750kmi10ktl12pdist01	1	24	12.495280	0.00124953

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi10ktl12pdist01	1	24	12.494585	0.00124946
s10kmi10ktl12pdist02	3	21	12.458500	0.00124570
s250kmi10ktl12pdist02	2	23	12.487048	0.00124870
s500kmi10ktl12pdist02	2	23	12.493328	0.00124933
s750kmi10ktl12pdist02	1	24	12.492257	0.00124922
s1Mmi10ktl12pdist02	1	24	12.490747	0.00124907
s10kmi10ktl12pdist03	4	21	12.494400	0.00124927
s250kmi10ktl12pdist03	2	24	12.491068	0.00124910
s500kmi10ktl12pdist03	1	24	12.492218	0.00124922
s750kmi10ktl12pdist03	1	24	12.489727	0.00124897
s1Mmi10ktl12pdist03	1	24	12.489797	0.00124898
s10kmi10ktl12pdist04	4	23	12.466600	0.00124660
s250kmi10ktl12pdist04	2	23	12.490672	0.00124906
s500kmi10ktl12pdist04	2	24	12.488116	0.00124881
s750kmi10ktl12pdist04	1	24	12.487431	0.00124874
s1Mmi10ktl12pdist04	1	24	12.487070	0.00124871
s10kmi10ktl12pdist05	4	22	12.536400	0.00125352
s250kmi10ktl12pdist05	2	23	12.489368	0.00124893
s500kmi10ktl12pdist05	2	24	12.492126	0.00124921
s750kmi10ktl12pdist05	1	24	12.493760	0.00124937

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi10ktl12pdist05	1	24	12.494914	0.00124949
s10kmi10ktl12pdist06	4	22	12.521100	0.00125200
s250kmi10ktl12pdist06	3	23	12.495700	0.00124956
s500kmi10ktl12pdist06	2	23	12.494844	0.00124948
s750kmi10ktl12pdist06	2	23	12.493780	0.00124938
s1Mmi10ktl12pdist06	2	23	12.492044	0.00124920
s10kmi10ktl12pdist07	4	21	12.481400	0.00124802
s250kmi10ktl12pdist07	2	24	12.491380	0.00124913
s500kmi10ktl12pdist07	2	24	12.489482	0.00124895
s750kmi10ktl12pdist07	1	24	12.490555	0.00124905
s1Mmi10ktl12pdist07	1	24	12.489614	0.00124896
s10kmi10ktl12pdist08	3	22	12.465500	0.00124641
s250kmi10ktl12pdist08	2	23	12.496820	0.00124968
s500kmi10ktl12pdist08	1	24	12.492826	0.00124928
s750kmi10ktl12pdist08	1	24	12.494088	0.00124941
s1Mmi10ktl12pdist08	1	24	12.495663	0.00124957
s10kmi10ktl12pdist09	4	23	12.470200	0.00124685
s250kmi10ktl12pdist09	1	23	12.490668	0.00124906
s500kmi10ktl12pdist09	1	24	12.491538	0.00124915
s750kmi10ktl12pdist09	1	24	12.490112	0.00124901

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi10ktl12pdist09	1	24	12.492428	0.00124924
s10kmi10ktl12pdist10	4	22	12.474700	0.00124736
s250kmi10ktl12pdist10	2	24	12.490976	0.00124909
s500kmi10ktl12pdist10	2	24	12.491222	0.00124912
s750kmi10ktl12pdist10	2	24	12.492176	0.00124922
s1Mmi10ktl12pdist10	2	24	12.488748	0.00124887
s10kmi10ktl12pdist11	2	22	12.485400	0.00124840
s250kmi10ktl12pdist11	2	24	12.496240	0.00124962
s500kmi10ktl12pdist11	1	24	12.491432	0.00124914
s750kmi10ktl12pdist11	1	24	12.490568	0.00124906
s1Mmi10ktl12pdist11	1	24	12.490926	0.00124909
s10kmi10ktl12pdist12	4	22	12.517400	0.00125167
s250kmi10ktl12pdist12	2	23	12.484848	0.00124848
s500kmi10ktl12pdist12	2	24	12.484180	0.00124841
s750kmi10ktl12pdist12	1	24	12.483615	0.00124836
s1Mmi10ktl12pdist12	1	24	12.485302	0.00124853
s10kmi10ktl12pdist13	2	21	12.505000	0.00125039
s250kmi10ktl12pdist13	2	23	12.504152	0.00125041
s500kmi10ktl12pdist13	2	23	12.497210	0.00124972
s750kmi10ktl12pdist13	2	23	12.494308	0.00124943

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi10ktl12pdist13	1	23	12.494260	0.00124943
s10kmi10ktl12pdist14	4	21	12.517900	0.00125165
s250kmi10ktl12pdist14	2	23	12.500020	0.00125000
s500kmi10ktl12pdist14	2	23	12.496406	0.00124964
s750kmi10ktl12pdist14	1	23	12.495517	0.00124955
s1Mmi10ktl12pdist14	1	23	12.495576	0.00124956
s10kmi10ktl12pdist15	3	21	12.488300	0.00124870
s250kmi10ktl12pdist15	2	23	12.489628	0.00124896
s500kmi10ktl12pdist15	2	23	12.492530	0.00124925
s750kmi10ktl12pdist15	1	24	12.495960	0.00124959
s1Mmi10ktl12pdist15	1	24	12.495189	0.00124952
s10kmi10ktl25pdist01	6	44	25.485700	0.00254831
s250kmi10ktl25pdist01	3	47	25.458284	0.00254582
s500kmi10ktl25pdist01	2	48	25.461568	0.00254615
s750kmi10ktl25pdist01	2	48	25.466619	0.00254666
s1Mmi10ktl25pdist01	1	48	25.468359	0.00254684
s10kmi10ktl25pdist02	7	44	25.451100	0.00254485
s250kmi10ktl25pdist02	3	48	25.462804	0.00254628
s500kmi10ktl25pdist02	2	48	25.461816	0.00254617
s750kmi10ktl25pdist02	1	48	25.461604	0.00254616

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi10ktl25pdist02	1	48	25.461882	0.00254619
s10kmi10ktl25pdist03	7	44	25.535600	0.00255325
s250kmi10ktl25pdist03	4	48	25.476268	0.00254762
s500kmi10ktl25pdist03	4	49	25.482006	0.00254820
s750kmi10ktl25pdist03	3	50	25.476831	0.00254768
s1Mmi10ktl25pdist03	2	50	25.473730	0.00254737
s10kmi10ktl25pdist04	9	46	25.428100	0.00254258
s250kmi10ktl25pdist04	2	47	25.460660	0.00254606
s500kmi10ktl25pdist04	2	49	25.452946	0.00254529
s750kmi10ktl25pdist04	2	49	25.458580	0.00254585
s1Mmi10ktl25pdist04	2	49	25.457914	0.00254579
s10kmi10ktl25pdist05	3	43	25.506300	0.00255042
s250kmi10ktl25pdist05	3	47	25.479264	0.00254792
s500kmi10ktl25pdist05	2	48	25.471674	0.00254716
s750kmi10ktl25pdist05	2	48	25.473632	0.00254736
s1Mmi10ktl25pdist05	2	48	25.477618	0.00254776
s10kmi10ktl25pdist06	6	43	25.360300	0.00253573
s250kmi10ktl25pdist06	5	47	25.459452	0.00254594
s500kmi10ktl25pdist06	4	47	25.462590	0.00254625
s750kmi10ktl25pdist06	1	48	25.460432	0.00254604

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi10ktl25pdist06	1	48	25.462812	0.00254628
s10kmi10ktl25pdist07	4	44	25.441900	0.00254391
s250kmi10ktl25pdist07	3	48	25.467116	0.00254670
s500kmi10ktl25pdist07	2	48	25.467526	0.00254675
s750kmi10ktl25pdist07	2	48	25.464053	0.00254640
s1Mmi10ktl25pdist07	2	48	25.463789	0.00254638
s10kmi10ktl25pdist08	8	43	25.483200	0.00254812
s250kmi10ktl25pdist08	2	49	25.459580	0.00254595
s500kmi10ktl25pdist08	2	50	25.473176	0.00254731
s750kmi10ktl25pdist08	2	50	25.469867	0.00254698
s1Mmi10ktl25pdist08	1	50	25.468875	0.00254689
s10kmi10ktl25pdist09	8	43	25.451500	0.00254488
s250kmi10ktl25pdist09	3	49	25.469520	0.00254694
s500kmi10ktl25pdist09	2	49	25.470712	0.00254707
s750kmi10ktl25pdist09	2	49	25.470044	0.00254700
s1Mmi10ktl25pdist09	2	49	25.474310	0.00254743
s10kmi10ktl25pdist10	6	44	25.465700	0.00254637
s250kmi10ktl25pdist10	4	48	25.465448	0.00254653
s500kmi10ktl25pdist10	2	48	25.458192	0.00254582
s750kmi10ktl25pdist10	2	48	25.460761	0.00254607

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi10ktl25pdist10	2	48	25.461243	0.00254612
s10kmi10ktl25pdist11	7	46	25.388900	0.00253863
s250kmi10ktl25pdist11	3	48	25.469628	0.00254695
s500kmi10ktl25pdist11	3	48	25.467992	0.00254679
s750kmi10ktl25pdist11	3	48	25.463396	0.00254634
s1Mmi10ktl25pdist11	1	48	25.465176	0.00254652
s10kmi10ktl25pdist12	7	47	25.414300	0.00254115
s250kmi10ktl25pdist12	2	50	25.459788	0.00254597
s500kmi10ktl25pdist12	2	50	25.460614	0.00254606
s750kmi10ktl25pdist12	1	50	25.463675	0.00254636
s1Mmi10ktl25pdist12	1	50	25.463321	0.00254633
s10kmi10ktl25pdist13	5	46	25.434500	0.00254320
s250kmi10ktl25pdist13	2	46	25.462244	0.00254621
s500kmi10ktl25pdist13	2	49	25.467354	0.00254673
s750kmi10ktl25pdist13	2	49	25.470001	0.00254700
s1Mmi10ktl25pdist13	1	49	25.470941	0.00254709
s10kmi10ktl25pdist14	8	42	25.582500	0.00255805
s250kmi10ktl25pdist14	2	49	25.463260	0.00254631
s500kmi10ktl25pdist14	2	49	25.463280	0.00254632
s750kmi10ktl25pdist14	1	49	25.469927	0.00254699

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi10ktl25pdist14	1	49	25.469165	0.00254692
s10kmi10ktl25pdist15	6	46	25.537300	0.00255353
s250kmi10ktl25pdist15	4	48	25.484344	0.00254842
s500kmi10ktl25pdist15	4	48	25.478570	0.00254785
s750kmi10ktl25pdist15	4	48	25.468112	0.00254681
s1Mmi10ktl25pdist15	4	48	25.468183	0.00254682
s10kmi1ktl05pdist01	2	9	5.469900	0.00546940
s250kmi1ktl05pdist01	1	10	5.489472	0.00548946
s500kmi1ktl05pdist01	1	10	5.488328	0.00548832
s750kmi1ktl05pdist01	1	10	5.487327	0.00548732
s1Mmi1ktl05pdist01	1	10	5.487532	0.00548753
s10kmi1ktl05pdist02	2	9	5.492600	0.00549180
s250kmi1ktl05pdist02	1	10	5.486908	0.00548688
s500kmi1ktl05pdist02	1	10	5.487600	0.00548759
s750kmi1ktl05pdist02	1	10	5.486613	0.00548661
s1Mmi1ktl05pdist02	1	10	5.486509	0.00548651
s10kmi1ktl05pdist03	2	9	5.489800	0.00548920
s250kmi1ktl05pdist03	1	10	5.488204	0.00548818
s500kmi1ktl05pdist03	1	10	5.488692	0.00548868
s750kmi1ktl05pdist03	1	10	5.488600	0.00548859

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi1ktl05pdist03	1	10	5.487595	0.00548760
s10kmi1ktl05pdist04	2	10	5.499100	0.00549850
s250kmi1ktl05pdist04	1	10	5.485568	0.00548554
s500kmi1ktl05pdist04	1	10	5.484790	0.00548478
s750kmi1ktl05pdist04	1	10	5.486408	0.00548640
s1Mmi1ktl05pdist04	1	10	5.487146	0.00548715
s10kmi1ktl05pdist05	2	10	5.468500	0.00546810
s250kmi1ktl05pdist05	1	10	5.483164	0.00548314
s500kmi1ktl05pdist05	1	10	5.484348	0.00548434
s750kmi1ktl05pdist05	1	10	5.485261	0.00548525
s1Mmi1ktl05pdist05	1	10	5.486463	0.00548646
s10kmi1ktl05pdist06	2	9	5.491100	0.00549060
s250kmi1ktl05pdist06	1	10	5.487728	0.00548770
s500kmi1ktl05pdist06	1	10	5.487286	0.00548728
s750kmi1ktl05pdist06	1	10	5.487885	0.00548788
s1Mmi1ktl05pdist06	1	10	5.487524	0.00548752
s10kmi1ktl05pdist07	2	9	5.493100	0.00549260
s250kmi1ktl05pdist07	1	10	5.487516	0.00548749
s500kmi1ktl05pdist07	1	10	5.486074	0.00548606
s750kmi1ktl05pdist07	1	10	5.486184	0.00548618

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi1ktl05pdist07	1	10	5.486609	0.00548661
s10kmi1ktl05pdist08	2	9	5.496000	0.00549530
s250kmi1ktl05pdist08	1	10	5.487088	0.00548706
s500kmi1ktl05pdist08	1	10	5.487560	0.00548755
s750kmi1ktl05pdist08	1	10	5.486408	0.00548640
s1Mmi1ktl05pdist08	1	10	5.487302	0.00548730
s10kmi1ktl05pdist09	2	9	5.484100	0.00548360
s250kmi1ktl05pdist09	1	10	5.487396	0.00548737
s500kmi1ktl05pdist09	1	10	5.486610	0.00548660
s750kmi1ktl05pdist09	1	10	5.486360	0.00548635
s1Mmi1ktl05pdist09	1	10	5.486879	0.00548688
s10kmi1ktl05pdist10	2	10	5.491400	0.00549090
s250kmi1ktl05pdist10	1	10	5.488100	0.00548808
s500kmi1ktl05pdist10	1	10	5.486826	0.00548681
s750kmi1ktl05pdist10	1	10	5.486773	0.00548677
s1Mmi1ktl05pdist10	1	10	5.487858	0.00548786
s10kmi1ktl05pdist11	2	9	5.487200	0.00548670
s250kmi1ktl05pdist11	1	10	5.488080	0.00548806
s500kmi1ktl05pdist11	1	10	5.490316	0.00549031
s750kmi1ktl05pdist11	1	10	5.489911	0.00548990

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi1ktl05pdist11	1	10	5.489489	0.00548949
s10kmi1ktl05pdist12	1	9	5.468300	0.00546760
s250kmi1ktl05pdist12	1	10	5.483924	0.00548390
s500kmi1ktl05pdist12	1	10	5.485770	0.00548576
s750kmi1ktl05pdist12	1	10	5.485880	0.00548587
s1Mmi1ktl05pdist12	1	10	5.486351	0.00548635
s10kmi1ktl05pdist13	2	9	5.479800	0.00547930
s250kmi1ktl05pdist13	1	10	5.486780	0.00548675
s500kmi1ktl05pdist13	1	10	5.486044	0.00548603
s750kmi1ktl05pdist13	1	10	5.485872	0.00548586
s1Mmi1ktl05pdist13	1	10	5.486547	0.00548655
s10kmi1ktl05pdist14	2	10	5.480300	0.00547980
s250kmi1ktl05pdist14	1	10	5.488308	0.00548829
s500kmi1ktl05pdist14	1	10	5.487560	0.00548755
s750kmi1ktl05pdist14	1	10	5.487856	0.00548785
s1Mmi1ktl05pdist14	1	10	5.487175	0.00548718
s10kmi1ktl05pdist15	2	10	5.472400	0.00547170
s250kmi1ktl05pdist15	1	10	5.487700	0.00548768
s500kmi1ktl05pdist15	1	10	5.486996	0.00548699
s750kmi1ktl05pdist15	1	10	5.486476	0.00548647

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi1ktl05pdist15	1	10	5.486428	0.00548643
s10kmi1ktl12pdist01	2	21	12.421100	0.01241990
s250kmi1ktl12pdist01	2	24	12.419120	0.01241908
s500kmi1ktl12pdist01	1	24	12.421034	0.01242101
s750kmi1ktl12pdist01	1	24	12.422733	0.01242272
s1Mmi1ktl12pdist01	1	24	12.424088	0.01242409
s10kmi1ktl12pdist02	4	22	12.382300	0.01238110
s250kmi1ktl12pdist02	1	23	12.426244	0.01242620
s500kmi1ktl12pdist02	1	24	12.429502	0.01242948
s750kmi1ktl12pdist02	1	24	12.427668	0.01242765
s1Mmi1ktl12pdist02	1	24	12.426951	0.01242695
s10kmi1ktl12pdist03	4	23	12.453900	0.01245310
s250kmi1ktl12pdist03	2	23	12.419680	0.01241962
s500kmi1ktl12pdist03	1	23	12.422868	0.01242285
s750kmi1ktl12pdist03	1	24	12.422200	0.01242218
s1Mmi1ktl12pdist03	1	24	12.422992	0.01242299
s10kmi1ktl12pdist04	3	23	12.437500	0.01243580
s250kmi1ktl12pdist04	1	23	12.414940	0.01241488
s500kmi1ktl12pdist04	1	23	12.416678	0.01241665
s750kmi1ktl12pdist04	1	24	12.422225	0.01242221

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi1ktl12pdist04	1	24	12.421586	0.01242159
s10kmi1ktl12pdist05	2	21	12.438800	0.01243740
s250kmi1ktl12pdist05	1	23	12.427788	0.01242774
s500kmi1ktl12pdist05	1	23	12.425854	0.01242583
s750kmi1ktl12pdist05	1	23	12.428084	0.01242807
s1Mmi1ktl12pdist05	1	23	12.426951	0.01242695
s10kmi1ktl12pdist06	4	21	12.368200	0.01236700
s250kmi1ktl12pdist06	1	23	12.423248	0.01242322
s500kmi1ktl12pdist06	1	23	12.420890	0.01242087
s750kmi1ktl12pdist06	1	23	12.423379	0.01242336
s1Mmi1ktl12pdist06	1	24	12.425109	0.01242511
s10kmi1ktl12pdist07	4	23	12.424000	0.01242230
s250kmi1ktl12pdist07	2	23	12.422840	0.01242278
s500kmi1ktl12pdist07	2	24	12.426470	0.01242644
s750kmi1ktl12pdist07	2	24	12.426109	0.01242609
s1Mmi1ktl12pdist07	1	24	12.426839	0.01242684
s10kmi1ktl12pdist08	3	23	12.432500	0.01243130
s250kmi1ktl12pdist08	2	23	12.424468	0.01242442
s500kmi1ktl12pdist08	2	24	12.426534	0.01242651
s750kmi1ktl12pdist08	1	24	12.427668	0.01242766

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi1ktl12pdist08	1	24	12.427982	0.01242798
s10kmi1ktl12pdist09	1	21	12.442400	0.01244150
s250kmi1ktl12pdist09	1	24	12.428860	0.01242881
s500kmi1ktl12pdist09	1	24	12.425100	0.01242507
s750kmi1ktl12pdist09	1	24	12.426093	0.01242608
s1Mmi1ktl12pdist09	1	24	12.425575	0.01242558
s10kmi1ktl12pdist10	4	22	12.405100	0.01240370
s250kmi1ktl12pdist10	1	23	12.419668	0.01241962
s500kmi1ktl12pdist10	1	23	12.423008	0.01242298
s750kmi1ktl12pdist10	1	23	12.423019	0.01242301
s1Mmi1ktl12pdist10	1	23	12.422973	0.01242297
s10kmi1ktl12pdist11	2	23	12.430200	0.01242910
s250kmi1ktl12pdist11	1	23	12.430104	0.01243005
s500kmi1ktl12pdist11	1	24	12.427356	0.01242734
s750kmi1ktl12pdist11	1	24	12.427076	0.01242706
s1Mmi1ktl12pdist11	1	24	12.428379	0.01242838
s10kmi1ktl12pdist12	3	21	12.423000	0.01242120
s250kmi1ktl12pdist12	1	24	12.427636	0.01242758
s500kmi1ktl12pdist12	1	24	12.426612	0.01242659
s750kmi1ktl12pdist12	1	24	12.424904	0.01242488

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi1ktl12pdist12	1	24	12.425749	0.01242575
s10kmi1ktl12pdist13	4	23	12.420300	0.01241920
s250kmi1ktl12pdist13	2	23	12.428996	0.01242894
s500kmi1ktl12pdist13	2	23	12.427694	0.01242767
s750kmi1ktl12pdist13	2	23	12.429932	0.01242992
s1Mmi1ktl12pdist13	1	24	12.430397	0.01243040
s10kmi1ktl12pdist14	3	23	12.437300	0.01243610
s250kmi1ktl12pdist14	2	23	12.430128	0.01243008
s500kmi1ktl12pdist14	2	23	12.426114	0.01242609
s750kmi1ktl12pdist14	2	23	12.425421	0.01242541
s1Mmi1ktl12pdist14	1	23	12.423674	0.01242367
s10kmi1ktl12pdist15	3	22	12.424800	0.01242350
s250kmi1ktl12pdist15	2	23	12.430996	0.01243094
s500kmi1ktl12pdist15	2	24	12.427302	0.01242727
s750kmi1ktl12pdist15	2	24	12.427219	0.01242720
s1Mmi1ktl12pdist15	2	24	12.426077	0.01242608
s10kmi1ktl25pdist01	7	44	25.159700	0.02515740
s250kmi1ktl25pdist01	2	47	25.174884	0.02517480
s500kmi1ktl25pdist01	2	48	25.178838	0.02517880
s750kmi1ktl25pdist01	2	48	25.174613	0.02517458

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi1ktl25pdist01	2	50	25.174508	0.02517451
s10kmi1ktl25pdist02	8	49	25.138600	0.02513650
s250kmi1ktl25pdist02	2	49	25.182060	0.02518197
s500kmi1ktl25pdist02	2	49	25.180474	0.02518044
s750kmi1ktl25pdist02	2	49	25.179909	0.02517988
s1Mmi1ktl25pdist02	2	49	25.179586	0.02517959
s10kmi1ktl25pdist03	7	43	25.125300	0.02512320
s250kmi1ktl25pdist03	3	47	25.169376	0.02516930
s500kmi1ktl25pdist03	3	47	25.177650	0.02517762
s750kmi1ktl25pdist03	3	49	25.179121	0.02517909
s1Mmi1ktl25pdist03	2	49	25.182115	0.02518212
s10kmi1ktl25pdist04	8	44	25.176000	0.02517390
s250kmi1ktl25pdist04	5	47	25.168432	0.02516832
s500kmi1ktl25pdist04	3	48	25.175722	0.02517566
s750kmi1ktl25pdist04	3	48	25.179900	0.02517986
s1Mmi1ktl25pdist04	3	48	25.181496	0.02518150
s10kmi1ktl25pdist05	5	43	25.230000	0.02522650
s250kmi1ktl25pdist05	1	47	25.180652	0.02518053
s500kmi1ktl25pdist05	1	50	25.181230	0.02518119
s750kmi1ktl25pdist05	1	50	25.183057	0.02518303

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi1ktl25pdist05	1	50	25.180802	0.02518080
s10kmi1ktl25pdist06	7	46	25.193700	0.02519070
s250kmi1ktl25pdist06	3	49	25.164136	0.02516402
s500kmi1ktl25pdist06	3	49	25.172774	0.02517272
s750kmi1ktl25pdist06	1	49	25.170893	0.02517087
s1Mmi1ktl25pdist06	1	49	25.164765	0.02516477
s10kmi1ktl25pdist07	6	49	25.145300	0.02514220
s250kmi1ktl25pdist07	3	49	25.197888	0.02519779
s500kmi1ktl25pdist07	3	49	25.188756	0.02518870
s750kmi1ktl25pdist07	3	49	25.189005	0.02518896
s1Mmi1ktl25pdist07	3	49	25.192001	0.02519200
s10kmi1ktl25pdist08	5	43	25.190900	0.02518910
s250kmi1ktl25pdist08	4	47	25.180700	0.02518060
s500kmi1ktl25pdist08	3	48	25.182138	0.02518207
s750kmi1ktl25pdist08	2	48	25.182133	0.02518210
s1Mmi1ktl25pdist08	1	49	25.181242	0.02518124
s10kmi1ktl25pdist09	5	43	25.160000	0.02515730
s250kmi1ktl25pdist09	3	46	25.169752	0.02516961
s500kmi1ktl25pdist09	2	48	25.175772	0.02517572
s750kmi1ktl25pdist09	2	48	25.183588	0.02518355

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi1ktl25pdist09	2	49	25.184635	0.02518464
s10kmi1ktl25pdist10	5	43	25.232700	0.02523020
s250kmi1ktl25pdist10	3	49	25.184364	0.02518426
s500kmi1ktl25pdist10	3	49	25.189570	0.02518952
s750kmi1ktl25pdist10	1	49	25.189489	0.02518945
s1Mmi1ktl25pdist10	1	49	25.192256	0.02519226
s10kmi1ktl25pdist11	5	43	25.237000	0.02523530
s250kmi1ktl25pdist11	1	46	25.169500	0.02516939
s500kmi1ktl25pdist11	1	47	25.168604	0.02516856
s750kmi1ktl25pdist11	1	49	25.170076	0.02517004
s1Mmi1ktl25pdist11	1	49	25.174958	0.02517496
s10kmi1ktl25pdist12	6	45	25.242100	0.02523920
s250kmi1ktl25pdist12	4	47	25.183080	0.02518300
s500kmi1ktl25pdist12	3	48	25.171178	0.02517113
s750kmi1ktl25pdist12	3	48	25.174627	0.02517460
s1Mmi1ktl25pdist12	3	48	25.175504	0.02517550
s10kmi1ktl25pdist13	4	42	25.200500	0.02519830
s250kmi1ktl25pdist13	2	47	25.166948	0.02516685
s500kmi1ktl25pdist13	2	47	25.174118	0.02517405
s750kmi1ktl25pdist13	2	49	25.174477	0.02517444

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi1ktl25pdist13	2	49	25.175136	0.02517514
s10kmi1ktl25pdist14	6	44	25.109800	0.02510790
s250kmi1ktl25pdist14	1	47	25.182268	0.02518215
s500kmi1ktl25pdist14	1	47	25.186594	0.02518654
s750kmi1ktl25pdist14	1	48	25.187875	0.02518785
s1Mmi1ktl25pdist14	1	48	25.187421	0.02518742
s10kmi1ktl25pdist15	8	44	25.217300	0.02521420
s250kmi1ktl25pdist15	2	47	25.178748	0.02517865
s500kmi1ktl25pdist15	1	48	25.180230	0.02518017
s750kmi1ktl25pdist15	1	48	25.180797	0.02518077
s1Mmi1ktl25pdist15	1	48	25.179457	0.02517946
s10kmi100tl05pdist01	1	9	5.375400	0.05374900
s250kmi100tl05pdist01	1	10	5.373080	0.05373064
s500kmi100tl05pdist01	1	10	5.373030	0.05373018
s750kmi100tl05pdist01	1	10	5.371620	0.05371611
s1Mmi100tl05pdist01	1	10	5.371590	0.05371590
s10kmi100tl05pdist02	2	10	5.360500	0.05359900
s250kmi100tl05pdist02	1	10	5.374520	0.05374496
s500kmi100tl05pdist02	1	10	5.373150	0.05373140
s750kmi100tl05pdist02	1	10	5.374019	0.05374012

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi100tl05pdist02	1	10	5.373485	0.05373485
s10kmi100tl05pdist03	2	10	5.364400	0.05363700
s250kmi100tl05pdist03	1	10	5.371116	0.05371096
s500kmi100tl05pdist03	1	10	5.370422	0.05370408
s750kmi100tl05pdist03	1	10	5.371471	0.05371461
s1Mmi100tl05pdist03	1	10	5.370800	0.05370800
s10kmi100tl05pdist04	2	9	5.373400	0.05373000
s250kmi100tl05pdist04	1	10	5.370928	0.05370916
s500kmi100tl05pdist04	1	10	5.370712	0.05370700
s750kmi100tl05pdist04	1	10	5.369651	0.05369645
s1Mmi100tl05pdist04	1	10	5.370780	0.05370780
s10kmi100tl05pdist05	2	9	5.357200	0.05356800
s250kmi100tl05pdist05	1	10	5.372772	0.05372756
s500kmi100tl05pdist05	1	10	5.372194	0.05372184
s750kmi100tl05pdist05	1	10	5.373187	0.05373180
s1Mmi100tl05pdist05	1	10	5.372475	0.05372475
s10kmi100tl05pdist06	2	9	5.377400	0.05376900
s250kmi100tl05pdist06	1	10	5.373204	0.05373188
s500kmi100tl05pdist06	1	10	5.373278	0.05373268
s750kmi100tl05pdist06	1	10	5.372768	0.05372763

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi100tl05pdist06	1	10	5.371922	0.05371922
s10kmi100tl05pdist07	2	9	5.387100	0.05386500
s250kmi100tl05pdist07	1	10	5.372384	0.05372360
s500kmi100tl05pdist07	1	10	5.371430	0.05371418
s750kmi100tl05pdist07	1	10	5.371775	0.05371768
s1Mmi100tl05pdist07	1	10	5.372101	0.05372101
s10kmi100tl05pdist08	2	9	5.372300	0.05371700
s250kmi100tl05pdist08	1	10	5.371384	0.05371356
s500kmi100tl05pdist08	1	10	5.371432	0.05371424
s750kmi100tl05pdist08	1	10	5.372827	0.05372820
s1Mmi100tl05pdist08	1	10	5.372200	0.05372200
s10kmi100tl05pdist09	1	9	5.368000	0.05367500
s250kmi100tl05pdist09	1	10	5.372840	0.05372816
s500kmi100tl05pdist09	1	10	5.373284	0.05373276
s750kmi100tl05pdist09	1	10	5.372345	0.05372335
s1Mmi100tl05pdist09	1	10	5.372396	0.05372396
s10kmi100tl05pdist10	1	10	5.361700	0.05361300
s250kmi100tl05pdist10	1	10	5.372444	0.05372428
s500kmi100tl05pdist10	1	10	5.374032	0.05374022
s750kmi100tl05pdist10	1	10	5.373493	0.05373485

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi100tl05pdist10	1	10	5.373602	0.05373602
s10kmi100tl05pdist11	2	9	5.365400	0.05364800
s250kmi100tl05pdist11	1	10	5.371104	0.05371080
s500kmi100tl05pdist11	1	10	5.372502	0.05372494
s750kmi100tl05pdist11	1	10	5.372388	0.05372381
s1Mmi100tl05pdist11	1	10	5.373022	0.05373022
s10kmi100tl05pdist12	2	9	5.379100	0.05378600
s250kmi100tl05pdist12	1	10	5.374124	0.05374100
s500kmi100tl05pdist12	1	10	5.371530	0.05371520
s750kmi100tl05pdist12	1	10	5.371947	0.05371939
s1Mmi100tl05pdist12	1	10	5.372396	0.05372396
s10kmi100tl05pdist13	2	9	5.367900	0.05367200
s250kmi100tl05pdist13	1	10	5.372448	0.05372420
s500kmi100tl05pdist13	1	10	5.370886	0.05370878
s750kmi100tl05pdist13	1	10	5.369931	0.05369924
s1Mmi100tl05pdist13	1	10	5.369926	0.05369926
s10kmi100tl05pdist14	2	9	5.371100	0.05370700
s250kmi100tl05pdist14	1	10	5.370928	0.05370908
s500kmi100tl05pdist14	1	10	5.372082	0.05372072
s750kmi100tl05pdist14	1	10	5.373143	0.05373133

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi100tl05pdist14	1	10	5.372970	0.05372970
s10kmi100tl05pdist15	1	9	5.375600	0.05375000
s250kmi100tl05pdist15	1	10	5.376384	0.05376368
s500kmi100tl05pdist15	1	10	5.375288	0.05375276
s750kmi100tl05pdist15	1	10	5.373652	0.05373643
s1Mmi100tl05pdist15	1	10	5.374243	0.05374243
s10kmi100tl12pdist01	4	20	11.743400	0.11741800
s250kmi100tl12pdist01	1	21	11.776756	0.11776712
s500kmi100tl12pdist01	1	22	11.776706	0.11776684
s750kmi100tl12pdist01	1	22	11.775783	0.11775767
s1Mmi100tl12pdist01	1	23	11.777158	0.11777158
s10kmi100tl12pdist02	3	21	11.773600	0.11772300
s250kmi100tl12pdist02	1	22	11.776488	0.11776432
s500kmi100tl12pdist02	1	23	11.782796	0.11782766
s750kmi100tl12pdist02	1	23	11.783847	0.11783835
s1Mmi100tl12pdist02	1	23	11.784423	0.11784423
s10kmi100tl12pdist03	4	20	11.787600	0.11786700
s250kmi100tl12pdist03	2	22	11.781388	0.11781352
s500kmi100tl12pdist03	2	22	11.784308	0.11784282
s750kmi100tl12pdist03	2	22	11.783548	0.11783528

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi100tl12pdist03	2	22	11.782871	0.11782871
s10kmi100tl12pdist04	2	21	11.787700	0.11786700
s250kmi100tl12pdist04	1	22	11.782692	0.11782640
s500kmi100tl12pdist04	1	23	11.781094	0.11781070
s750kmi100tl12pdist04	1	23	11.781193	0.11781175
s1Mmi100tl12pdist04	1	23	11.779817	0.11779817
s10kmi100tl12pdist05	4	21	11.797000	0.11796000
s250kmi100tl12pdist05	2	22	11.783124	0.11783072
s500kmi100tl12pdist05	2	23	11.784600	0.11784572
s750kmi100tl12pdist05	1	23	11.782860	0.11782844
s1Mmi100tl12pdist05	1	23	11.781912	0.11781912
s10kmi100tl12pdist06	3	21	11.747800	0.11746900
s250kmi100tl12pdist06	1	22	11.783544	0.11783492
s500kmi100tl12pdist06	1	22	11.779888	0.11779864
s750kmi100tl12pdist06	1	23	11.778900	0.11778884
s1Mmi100tl12pdist06	1	23	11.779090	0.11779090
s10kmi100tl12pdist07	3	20	11.767900	0.11767300
s250kmi100tl12pdist07	2	22	11.782056	0.11782008
s500kmi100tl12pdist07	1	22	11.776230	0.11776214
s750kmi100tl12pdist07	1	22	11.777988	0.11777975

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi100tl12pdist07	1	22	11.779688	0.11779688
s10kmi100tl12pdist08	3	21	11.792200	0.11791000
s250kmi100tl12pdist08	3	23	11.776748	0.11776700
s500kmi100tl12pdist08	2	23	11.779060	0.11779036
s750kmi100tl12pdist08	2	23	11.780695	0.11780679
s1Mmi100tl12pdist08	2	23	11.779988	0.11779988
s10kmi100tl12pdist09	3	20	11.756700	0.11755400
s250kmi100tl12pdist09	2	22	11.775268	0.11775216
s500kmi100tl12pdist09	2	22	11.779052	0.11779028
s750kmi100tl12pdist09	2	22	11.782065	0.11782043
s1Mmi100tl12pdist09	2	22	11.781063	0.11781063
s10kmi100tl12pdist10	3	21	11.775900	0.11775100
s250kmi100tl12pdist10	1	23	11.781132	0.11781092
s500kmi100tl12pdist10	1	23	11.779910	0.11779892
s750kmi100tl12pdist10	1	23	11.777379	0.11777367
s1Mmi100tl12pdist10	1	23	11.779599	0.11779599
s10kmi100tl12pdist11	3	20	11.762900	0.11761900
s250kmi100tl12pdist11	2	23	11.772588	0.11772552
s500kmi100tl12pdist11	1	23	11.775772	0.11775740
s750kmi100tl12pdist11	1	23	11.777712	0.11777691

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi100tl12pdist11	1	23	11.778943	0.11778943
s10kmi100tl12pdist12	4	20	11.756700	0.11755700
s250kmi100tl12pdist12	1	23	11.779532	0.11779484
s500kmi100tl12pdist12	1	23	11.778030	0.11778004
s750kmi100tl12pdist12	1	23	11.778820	0.11778807
s1Mmi100tl12pdist12	1	23	11.777968	0.11777968
s10kmi100tl12pdist13	4	20	11.823300	0.11822200
s250kmi100tl12pdist13	1	22	11.775432	0.11775368
s500kmi100tl12pdist13	1	23	11.780002	0.11779974
s750kmi100tl12pdist13	1	23	11.779529	0.11779507
s1Mmi100tl12pdist13	1	23	11.780011	0.11780011
s10kmi100tl12pdist14	2	20	11.770400	0.11769300
s250kmi100tl12pdist14	2	23	11.777008	0.11776960
s500kmi100tl12pdist14	2	23	11.775288	0.11775262
s750kmi100tl12pdist14	1	23	11.776357	0.11776349
s1Mmi100tl12pdist14	1	23	11.776079	0.11776079
s10kmi100tl12pdist15	3	21	11.805400	0.11804400
s250kmi100tl12pdist15	2	23	11.778612	0.11778560
s500kmi100tl12pdist15	2	23	11.778146	0.11778126
s750kmi100tl12pdist15	2	23	11.779796	0.11779779

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi100tl12pdist15	2	23	11.779902	0.11779902
s10kmi100tl25pdist01	8	39	22.486500	0.22484200
s250kmi100tl25pdist01	3	41	22.498824	0.22498712
s500kmi100tl25pdist01	2	41	22.506918	0.22506874
s750kmi100tl25pdist01	2	41	22.510468	0.22510443
s1Mmi100tl25pdist01	1	41	22.511428	0.22511428
s10kmi100tl25pdist02	4	39	22.515500	0.22513000
s250kmi100tl25pdist02	1	41	22.513384	0.22513300
s500kmi100tl25pdist02	1	41	22.508532	0.22508474
s750kmi100tl25pdist02	1	41	22.507252	0.22507223
s1Mmi100tl25pdist02	1	43	22.505012	0.22505012
s10kmi100tl25pdist03	6	37	22.511400	0.22508900
s250kmi100tl25pdist03	2	41	22.518772	0.22518688
s500kmi100tl25pdist03	2	41	22.513862	0.22513818
s750kmi100tl25pdist03	1	42	22.516717	0.22516687
s1Mmi100tl25pdist03	1	42	22.516752	0.22516752
s10kmi100tl25pdist04	6	41	22.521200	0.22518700
s250kmi100tl25pdist04	2	41	22.516744	0.22516664
s500kmi100tl25pdist04	2	41	22.510368	0.22510300
s750kmi100tl25pdist04	2	41	22.511563	0.22511536

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi100tl25pdist04	2	47	22.512072	0.22512072
s10kmi100tl25pdist05	5	37	22.491300	0.22489200
s250kmi100tl25pdist05	4	40	22.510420	0.22510332
s500kmi100tl25pdist05	3	42	22.504366	0.22504326
s750kmi100tl25pdist05	3	42	22.504263	0.22504229
s1Mmi100tl25pdist05	2	42	22.506910	0.22506910
s10kmi100tl25pdist06	4	37	22.475900	0.22474100
s250kmi100tl25pdist06	1	40	22.511464	0.22511372
s500kmi100tl25pdist06	1	40	22.498564	0.22498524
s750kmi100tl25pdist06	1	41	22.501599	0.22501569
s1Mmi100tl25pdist06	1	43	22.500618	0.22500618
s10kmi100tl25pdist07	7	39	22.492300	0.22489700
s250kmi100tl25pdist07	4	42	22.518368	0.22518260
s500kmi100tl25pdist07	1	42	22.518948	0.22518904
s750kmi100tl25pdist07	1	42	22.515271	0.22515237
s1Mmi100tl25pdist07	1	42	22.515572	0.22515572
s10kmi100tl25pdist08	5	38	22.444600	0.22442300
s250kmi100tl25pdist08	2	41	22.513044	0.22512936
s500kmi100tl25pdist08	2	43	22.509580	0.22509536
s750kmi100tl25pdist08	2	44	22.507557	0.22507536

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi100tl25pdist08	2	44	22.510957	0.22510957
s10kmi100tl25pdist09	4	38	22.534800	0.22532400
s250kmi100tl25pdist09	4	42	22.515864	0.22515772
s500kmi100tl25pdist09	3	43	22.518112	0.22518070
s750kmi100tl25pdist09	3	43	22.509823	0.22509788
s1Mmi100tl25pdist09	2	43	22.509706	0.22509706
s10kmi100tl25pdist10	8	38	22.535400	0.22533600
s250kmi100tl25pdist10	2	40	22.523488	0.22523400
s500kmi100tl25pdist10	2	41	22.523052	0.22522988
s750kmi100tl25pdist10	2	41	22.517849	0.22517820
s1Mmi100tl25pdist10	2	42	22.517130	0.22517130
s10kmi100tl25pdist11	2	37	22.458500	0.22456800
s250kmi100tl25pdist11	2	40	22.504080	0.22503988
s500kmi100tl25pdist11	2	41	22.505964	0.22505922
s750kmi100tl25pdist11	2	41	22.505431	0.22505401
s1Mmi100tl25pdist11	2	41	22.511406	0.22511406
s10kmi100tl25pdist12	7	40	22.493900	0.22492000
s250kmi100tl25pdist12	2	43	22.494156	0.22494060
s500kmi100tl25pdist12	2	43	22.498250	0.22498200
s750kmi100tl25pdist12	2	43	22.504087	0.22504056

Continued on next page

Table A.2 – continued from previous page

Database	Min Item	Max Item	Avg. Length	Density
s1Mmi100tl25pdist12	2	43	22.501548	0.22501548
s10kmi100tl25pdist13	6	39	22.459500	0.22457000
s250kmi100tl25pdist13	2	43	22.496040	0.22495936
s500kmi100tl25pdist13	1	43	22.502594	0.22502530
s750kmi100tl25pdist13	1	43	22.503673	0.22503636
s1Mmi100tl25pdist13	1	43	22.500598	0.22500598
s10kmi100tl25pdist14	4	40	22.512700	0.22510100
s250kmi100tl25pdist14	2	41	22.511404	0.22511328
s500kmi100tl25pdist14	2	41	22.511722	0.22511672
s750kmi100tl25pdist14	1	42	22.510077	0.22510044
s1Mmi100tl25pdist14	1	42	22.511219	0.22511219
s10kmi100tl25pdist15	7	36	22.492500	0.22490700
s250kmi100tl25pdist15	2	42	22.496236	0.22496168
s500kmi100tl25pdist15	2	42	22.507064	0.22507020
s750kmi100tl25pdist15	2	42	22.505551	0.22505523
s1Mmi100tl25pdist15	2	42	22.503749	0.22503749

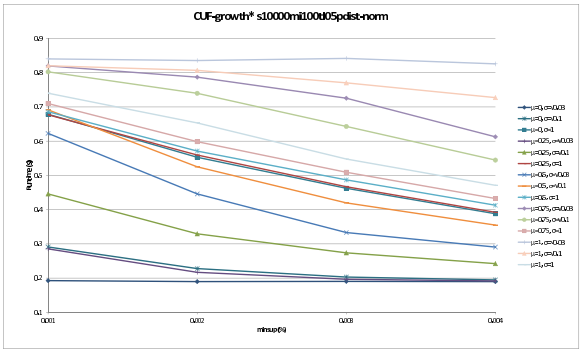
Table A.3: Real Database Characteristics

Database	**Min Item**	**Max Item**	**Avg. Length**	**Density**
kosarak	1	2498	8.099999	0.00019627
mushroom	23	23	23.000000	0.17968750
retail	1	76	10.305695	0.00062576

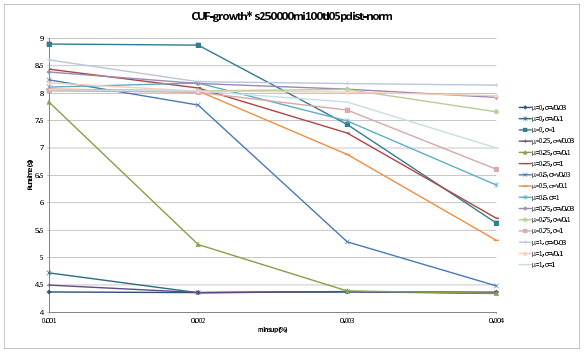
Appendix B

Extra Experimental Results

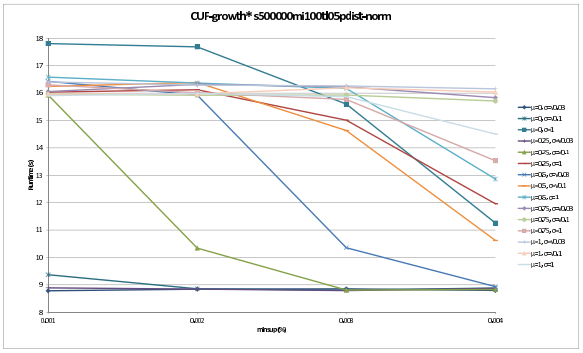
B.1 Transaction Length 5 Runtimes



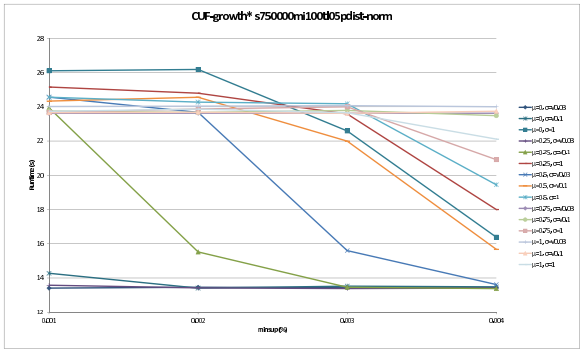
(a) 10,000 Transactions



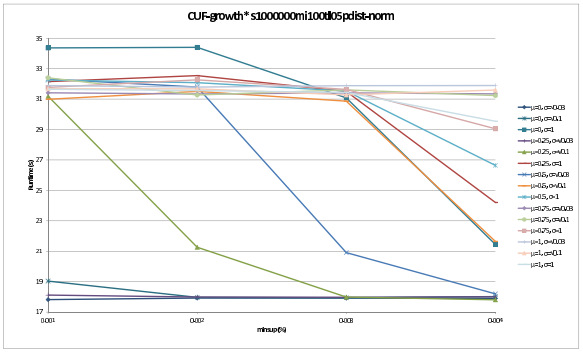
(b) 250,000 Transactions



(c) 500,000 Transactions

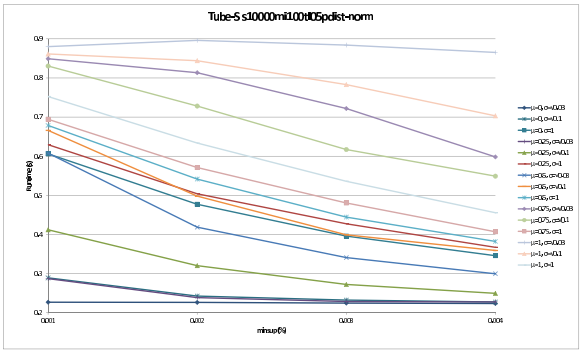


(d) 750,000 Transactions

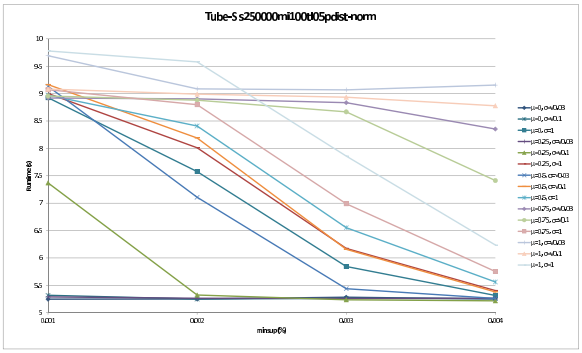


(e) 1,000,000 Transactions

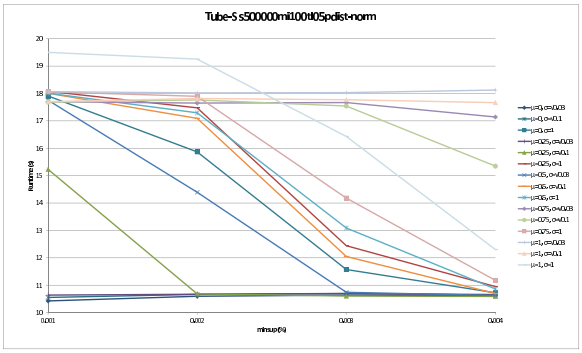
Figure B.1: Runtime: CUF-growth*, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$



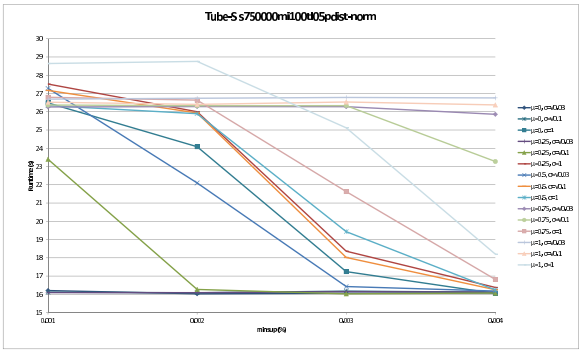
(a) 10,000 Transactions



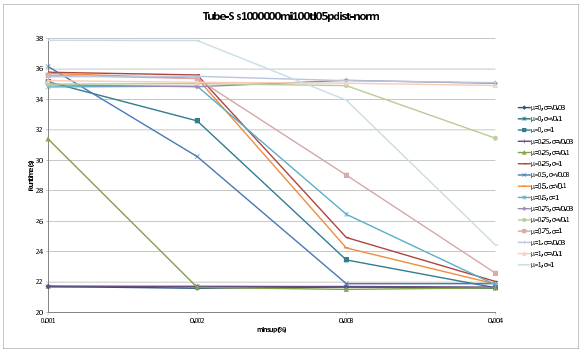
(b) 250,000 Transactions



(c) 500,000 Transactions

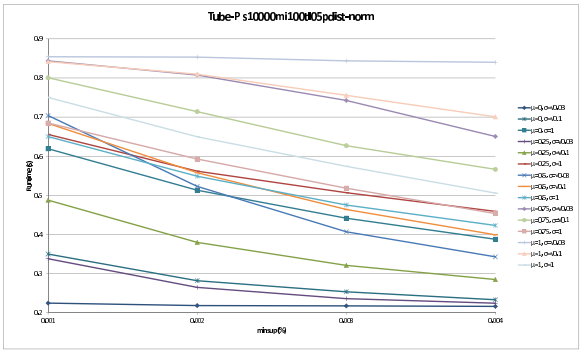


(d) 750,000 Transactions

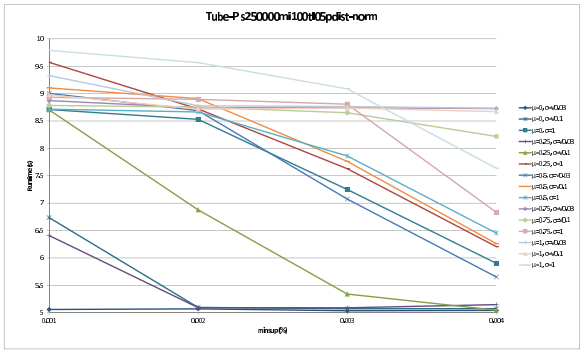


(e) 1,000,000 Transactions

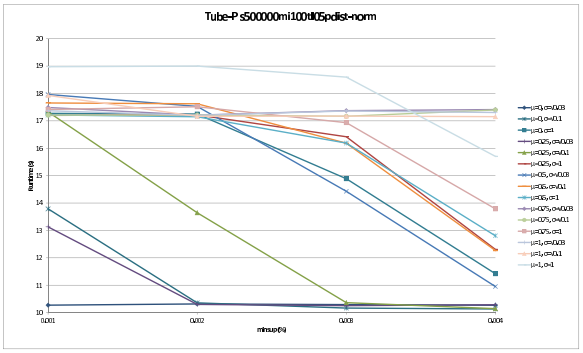
Figure B.2: Runtime: Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$



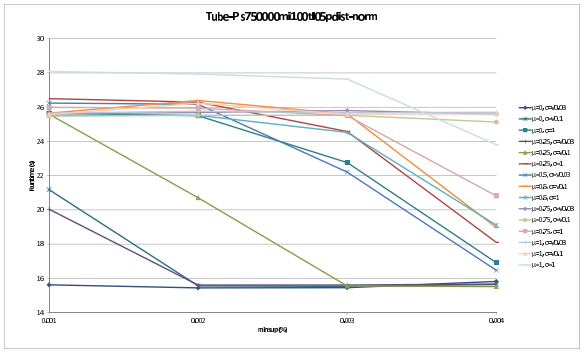
(a) 10,000 Transactions



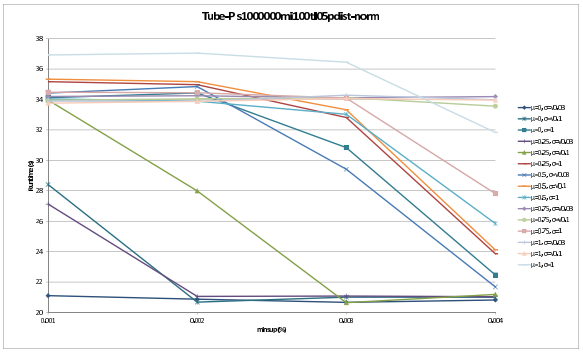
(b) 250,000 Transactions



(c) 500,000 Transactions

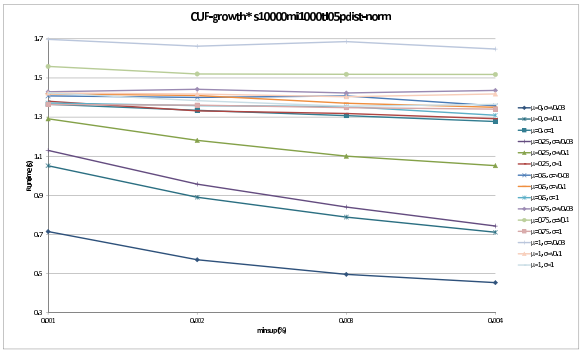


(d) 750,000 Transactions

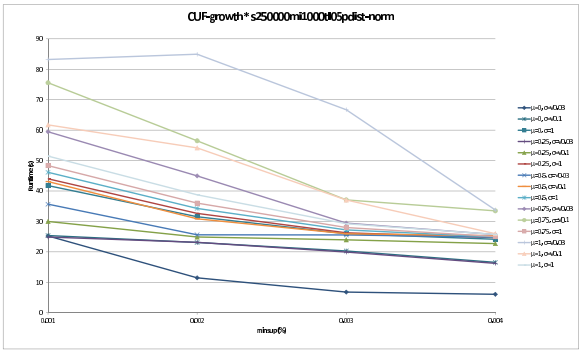


(e) 1,000,000 Transactions

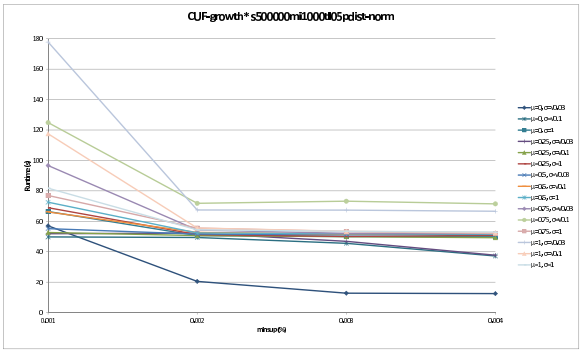
Figure B.3: Runtime: Tube-P, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$



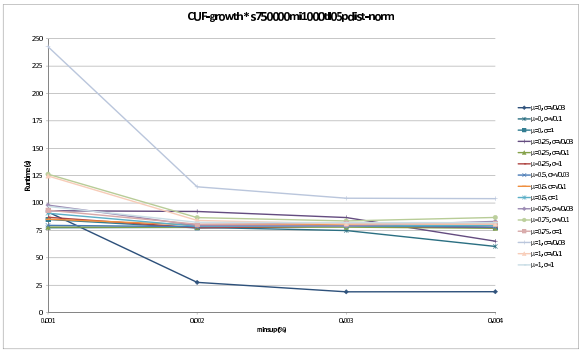
(a) 10,000 Transactions



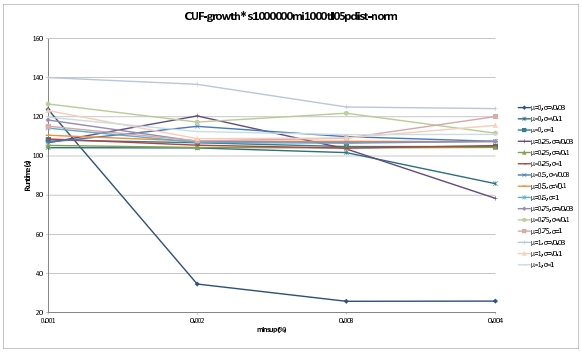
(b) 250,000 Transactions



(c) 500,000 Transactions

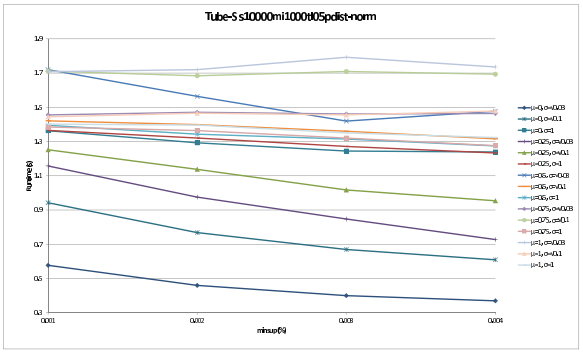


(d) 750,000 Transactions

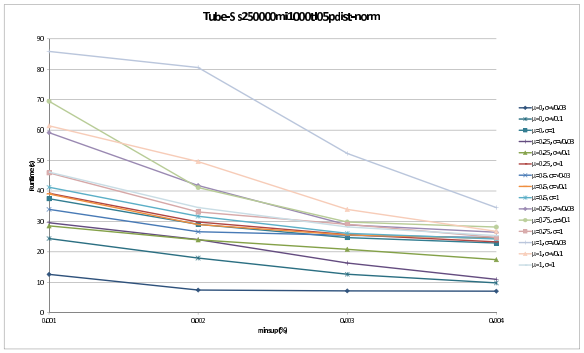


(e) 1,000,000 Transactions

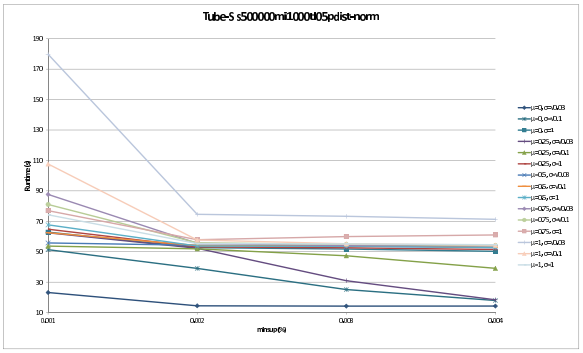
Figure B.4: Runtime: CUF-growth*, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$



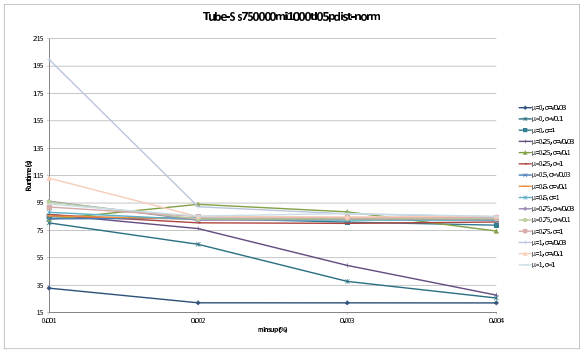
(a) 10,000 Transactions



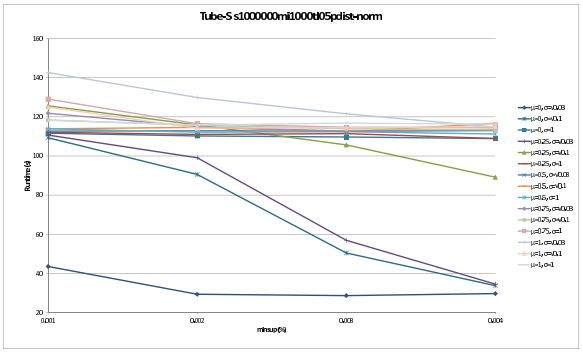
(b) 250,000 Transactions



(c) 500,000 Transactions

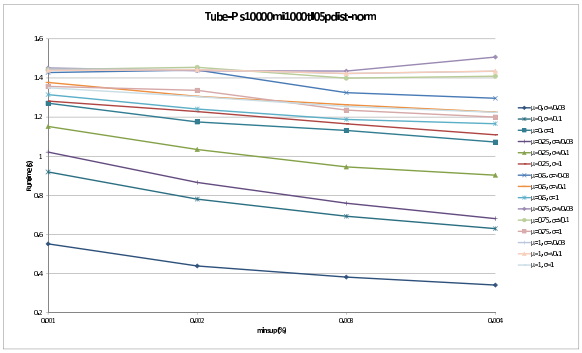


(d) 750,000 Transactions

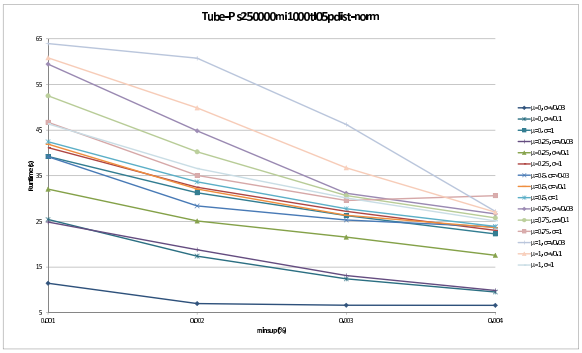


(e) 1,000,000 Transactions

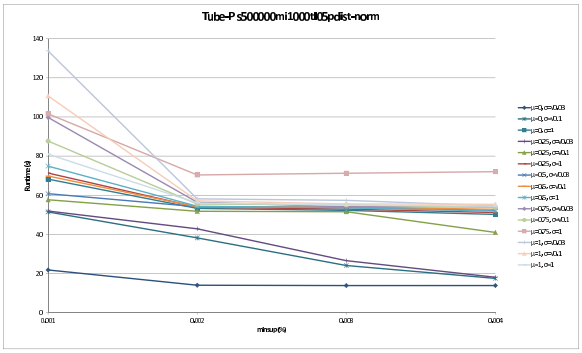
Figure B.5: Runtime: Tube-S, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$



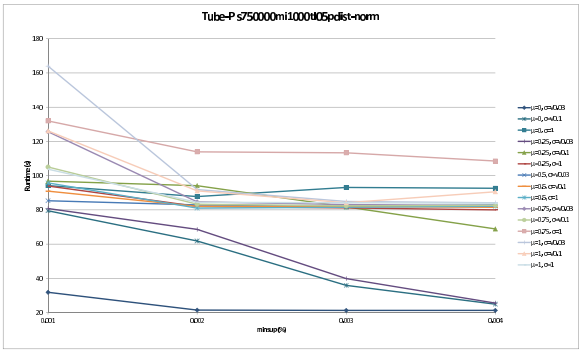
(a) 10,000 Transactions



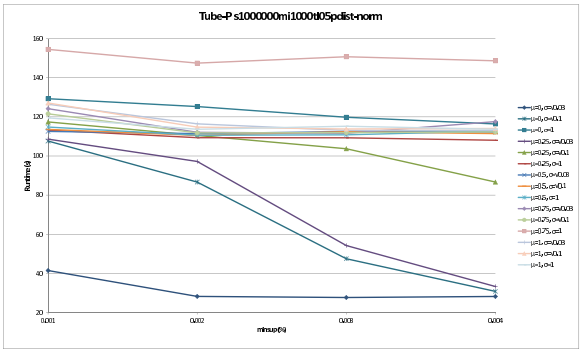
(b) 250,000 Transactions



(c) 500,000 Transactions

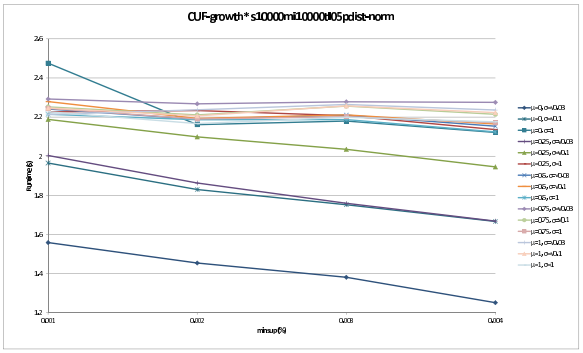


(d) 750,000 Transactions

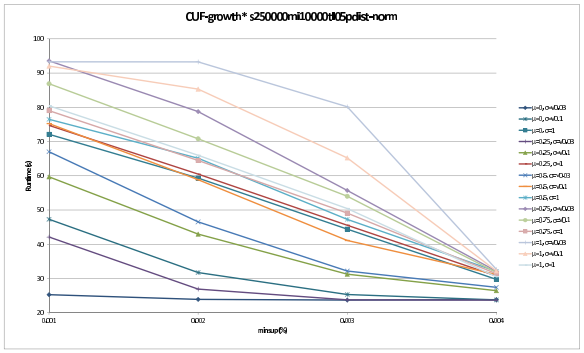


(e) 1,000,000 Transactions

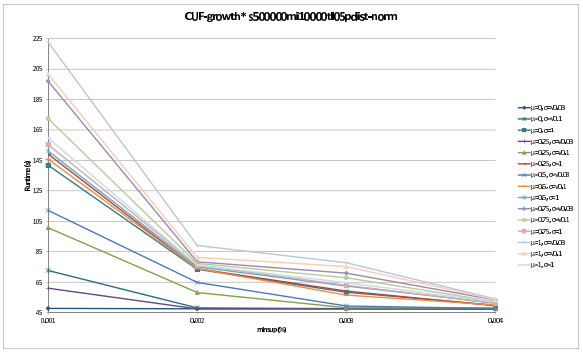
Figure B.6: Runtime: Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$



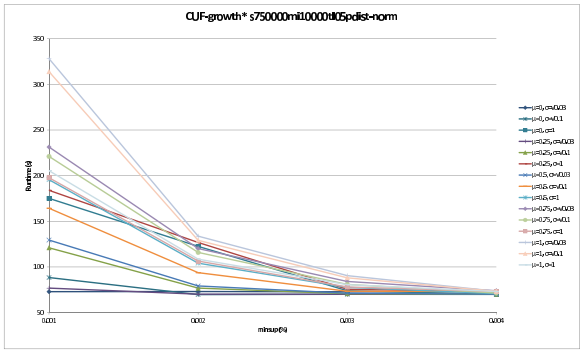
(a) 10,000 Transactions



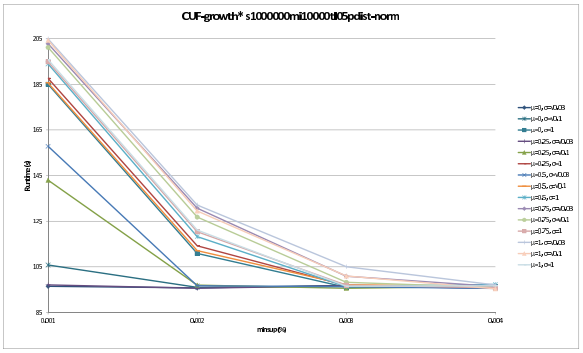
(b) 250,000 Transactions



(c) 500,000 Transactions

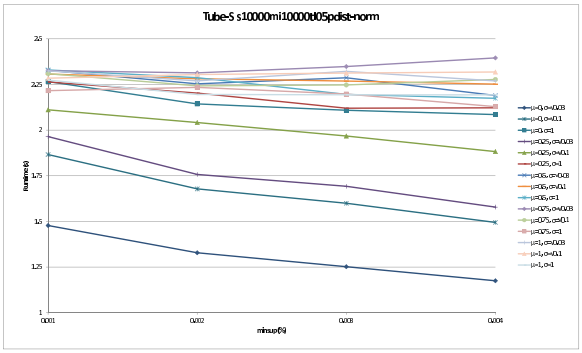


(d) 750,000 Transactions

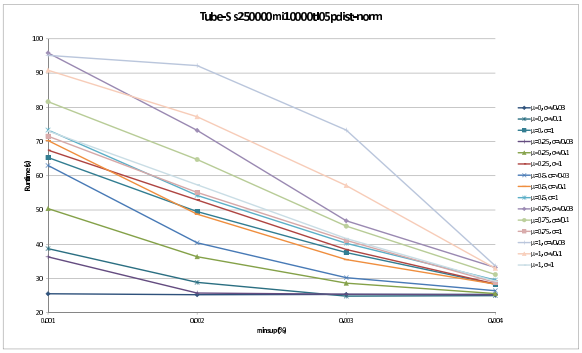


(e) 1,000,000 Transactions

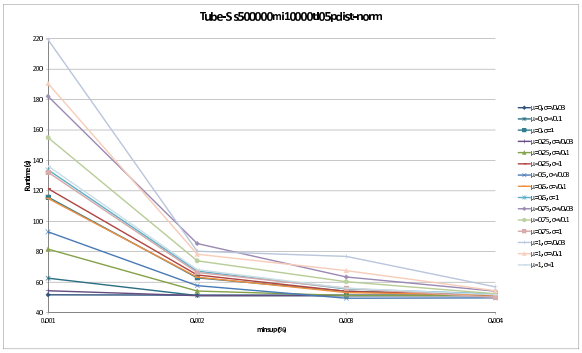
Figure B.7: Runtime: CUF-growth*, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$



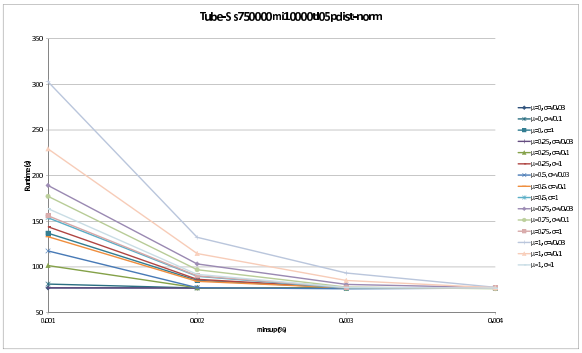
(a) 10,000 Transactions



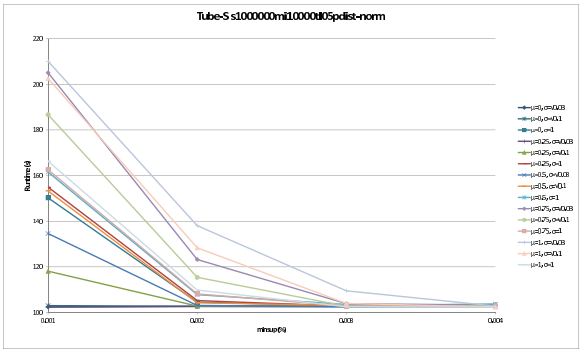
(b) 250,000 Transactions



(c) 500,000 Transactions

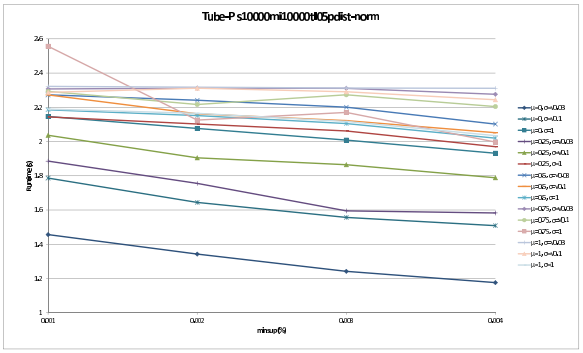


(d) 750,000 Transactions

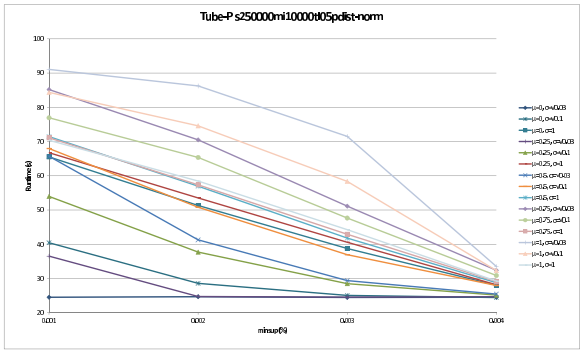


(e) 1,000,000 Transactions

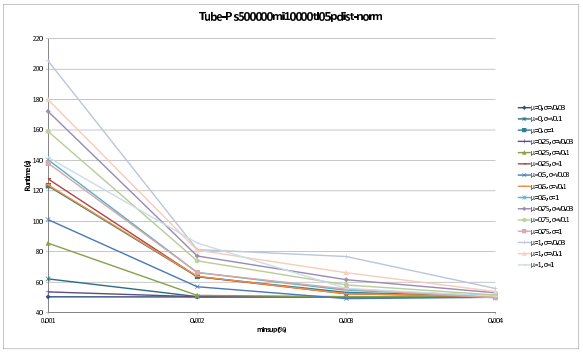
Figure B.8: Runtime: Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$



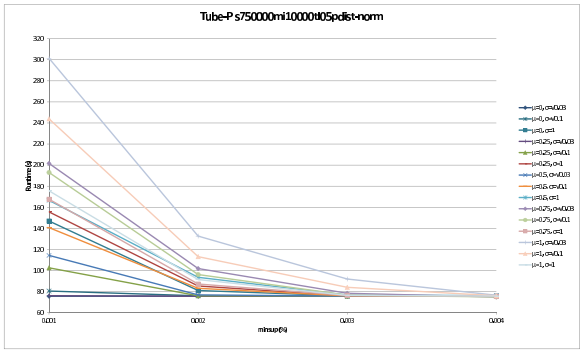
(a) 10,000 Transactions



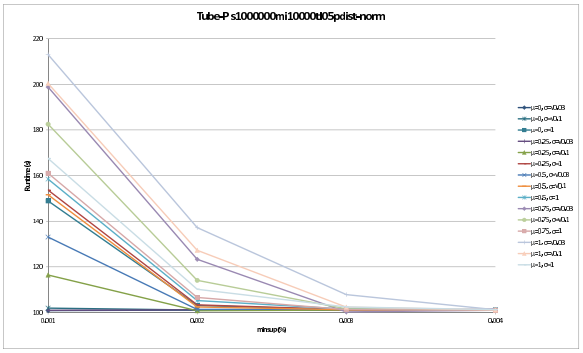
(b) 250,000 Transactions



(c) 500,000 Transactions



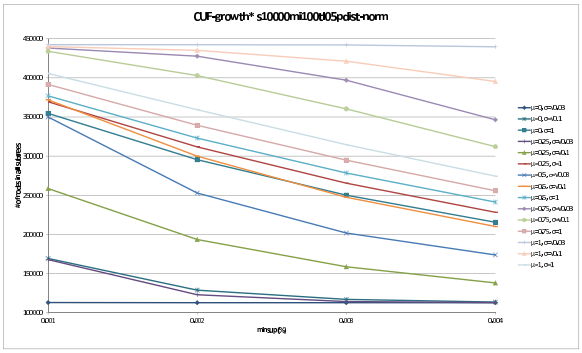
(d) 750,000 Transactions



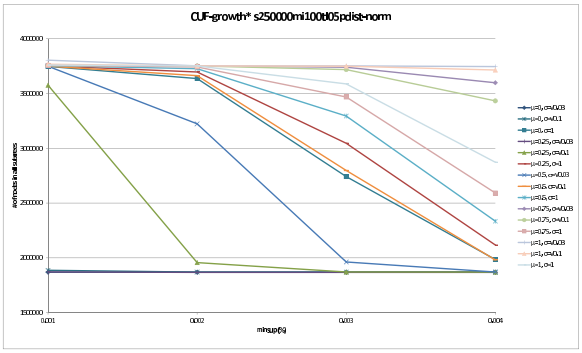
(e) 1,000,000 Transactions

Figure B.9: Runtime: Tube-P, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$

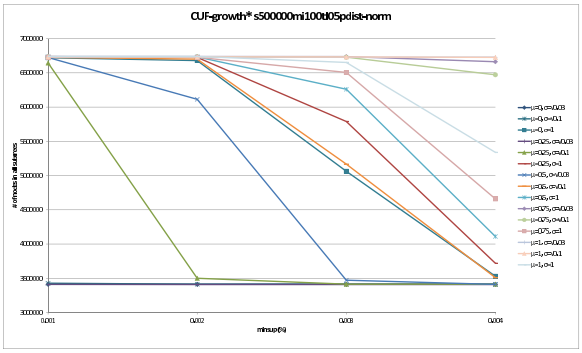
B.2 Transaction Length 5 Memory Usage



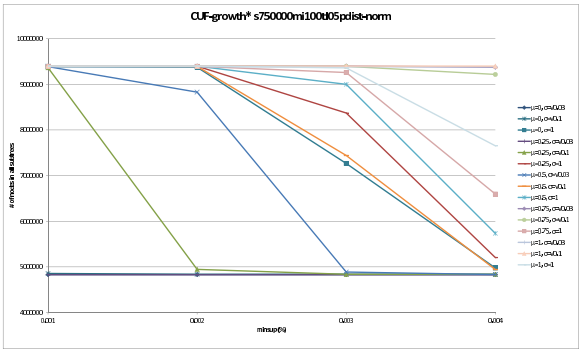
(a) 10,000 Transactions



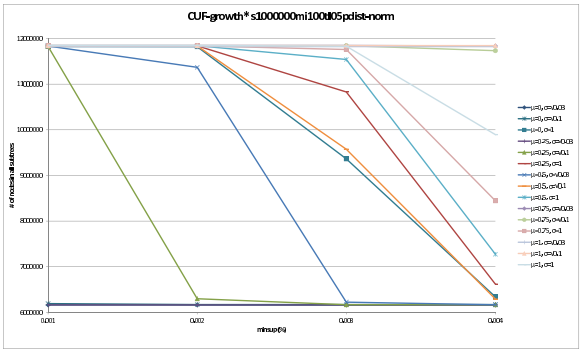
(b) 250,000 Transactions



(c) 500,000 Transactions

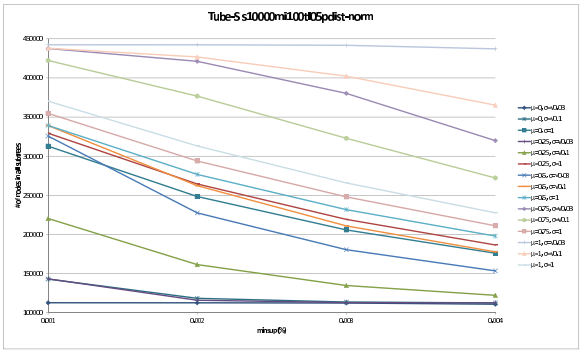


(d) 750,000 Transactions

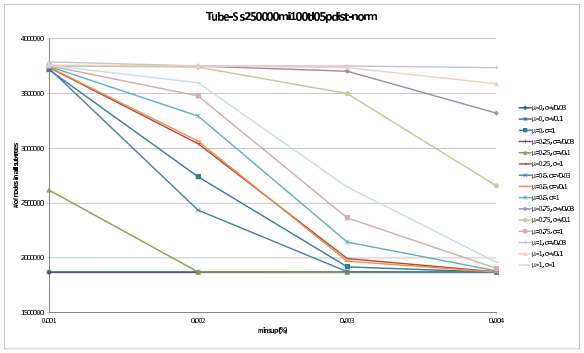


(e) 1,000,000 Transactions

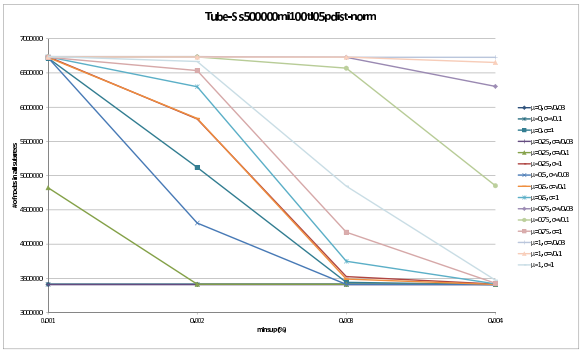
Figure B.10: Memory: CUF-growth*, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$



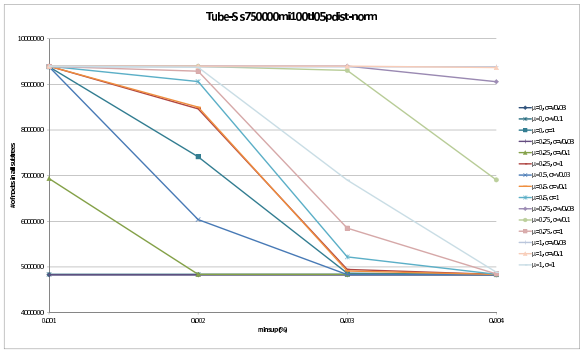
(a) 10,000 Transactions



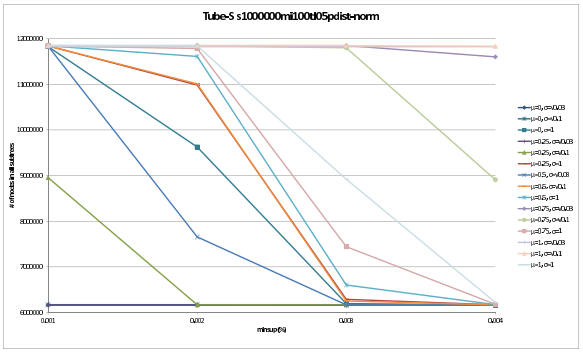
(b) 250,000 Transactions



(c) 500,000 Transactions

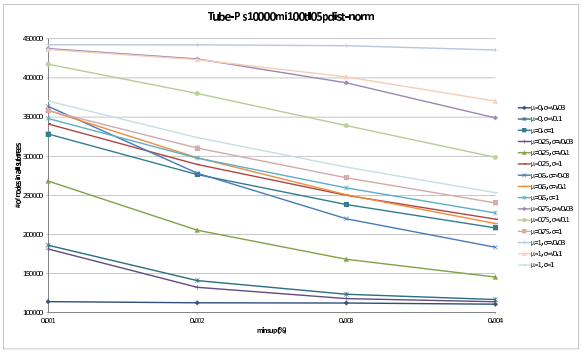


(d) 750,000 Transactions

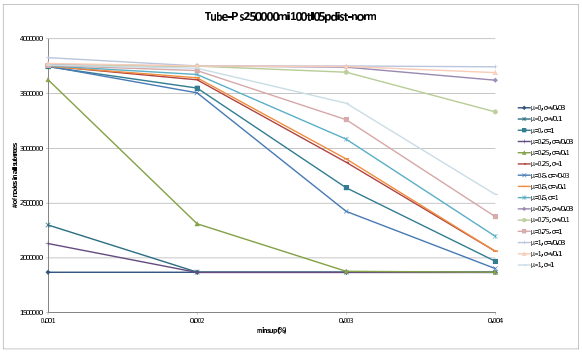


(e) 1,000,000 Transactions

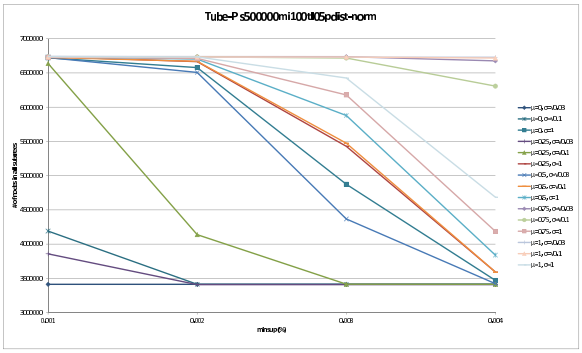
Figure B.11: Memory: Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$



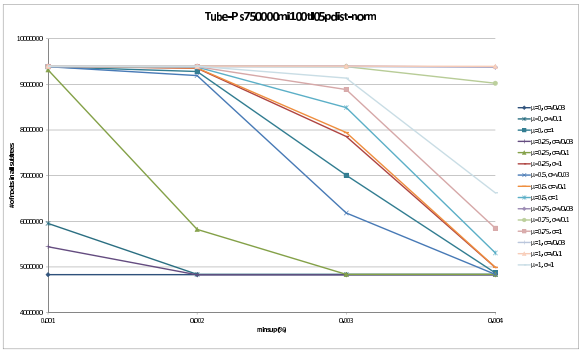
(a) 10,000 Transactions



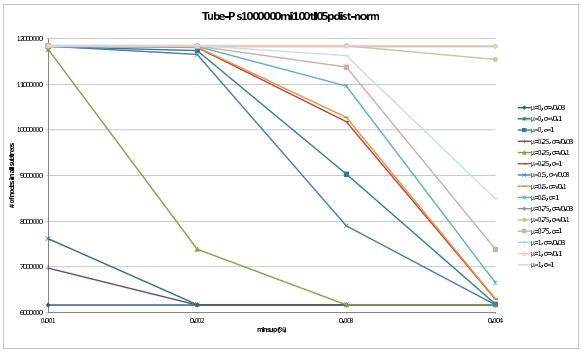
(b) 250,000 Transactions



(c) 500,000 Transactions

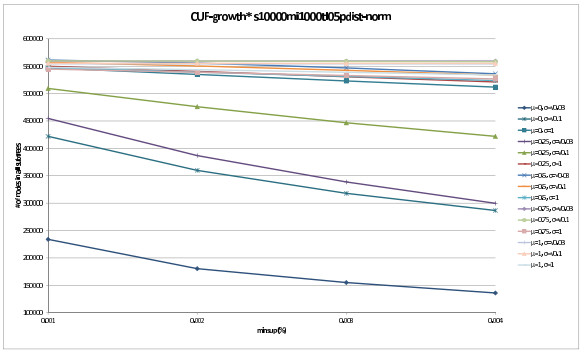


(d) 750,000 Transactions

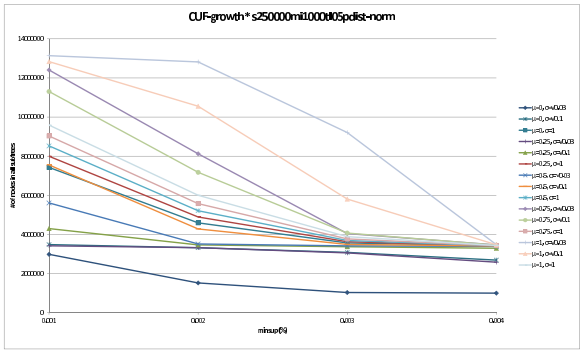


(e) 1,000,000 Transactions

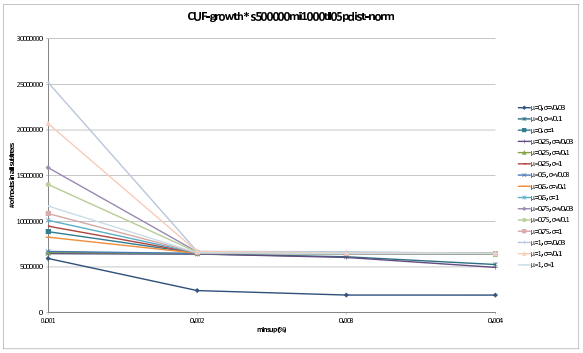
Figure B.12: Memory: Tube-P, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 5$



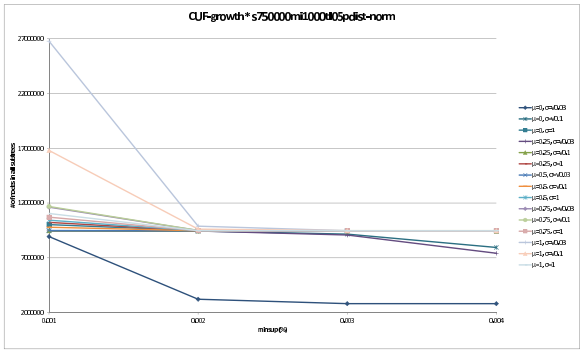
(a) 10,000 Transactions



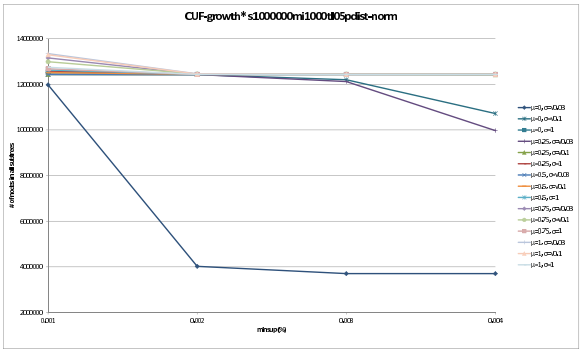
(b) 250,000 Transactions



(c) 500,000 Transactions

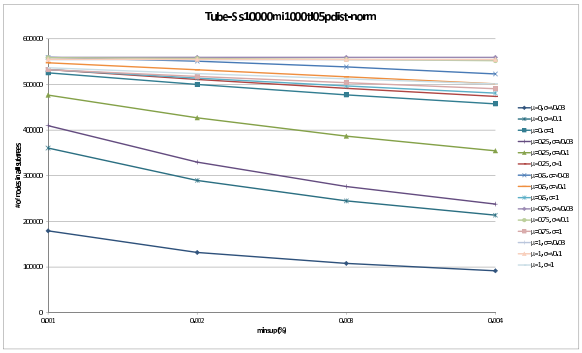


(d) 750,000 Transactions

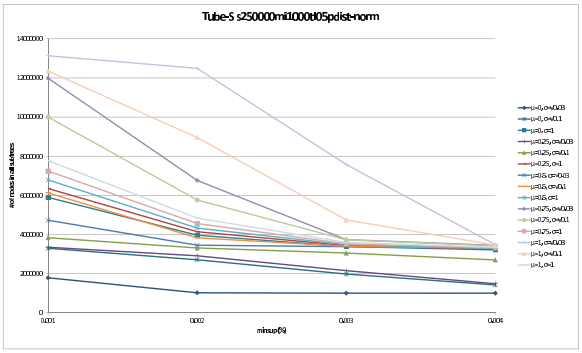


(e) 1,000,000 Transactions

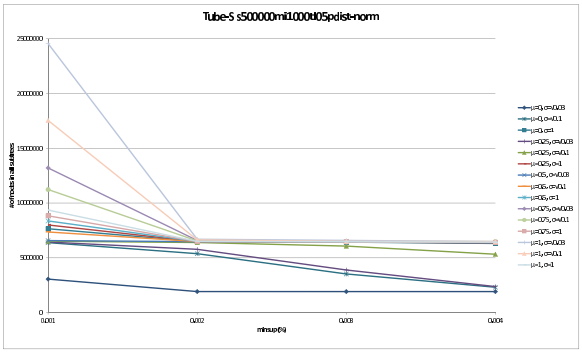
Figure B.13: Memory: CUF-growth*, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$



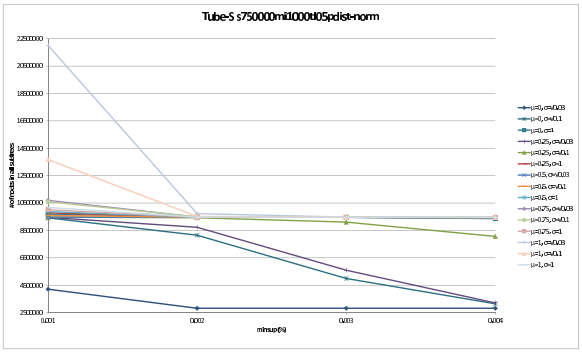
(a) 10,000 Transactions



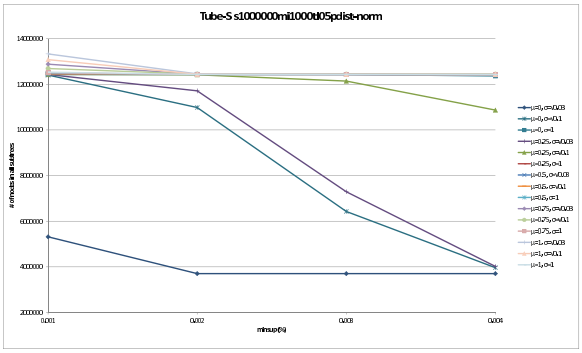
(b) 250,000 Transactions



(c) 500,000 Transactions

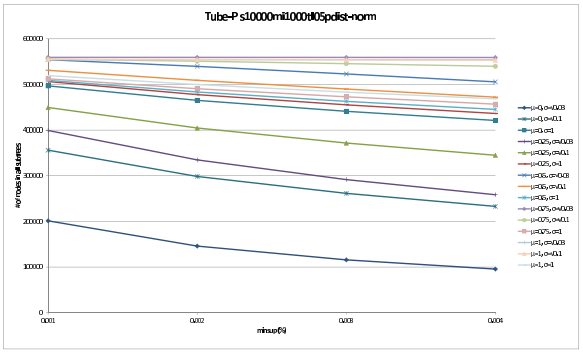


(d) 750,000 Transactions

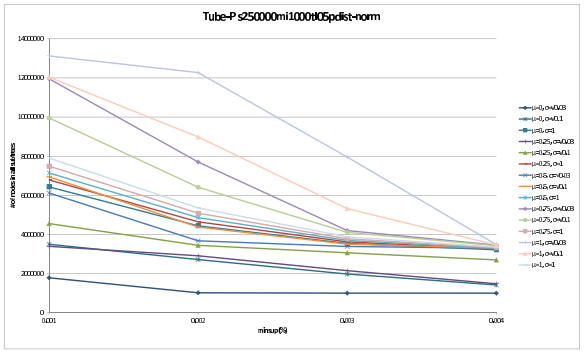


(e) 1,000,000 Transactions

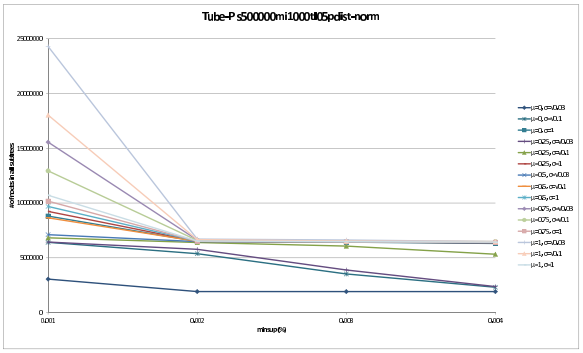
Figure B.14: Memory: Tube-S, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$



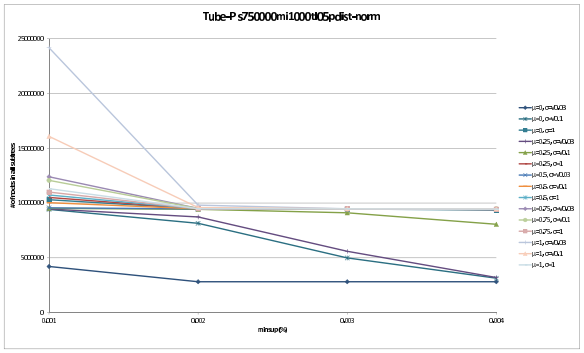
(a) 10,000 Transactions



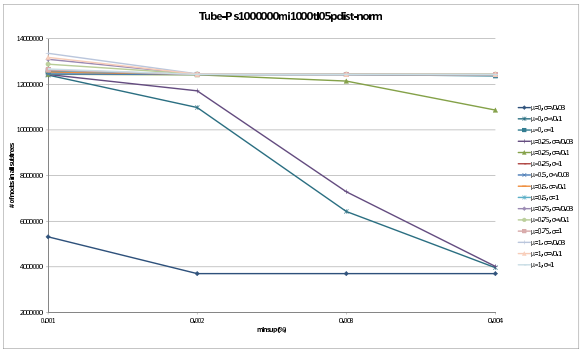
(b) 250,000 Transactions



(c) 500,000 Transactions

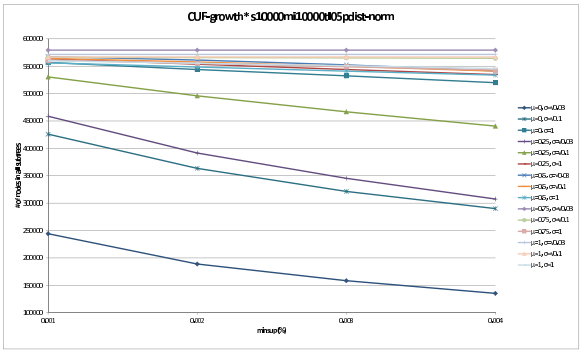


(d) 750,000 Transactions

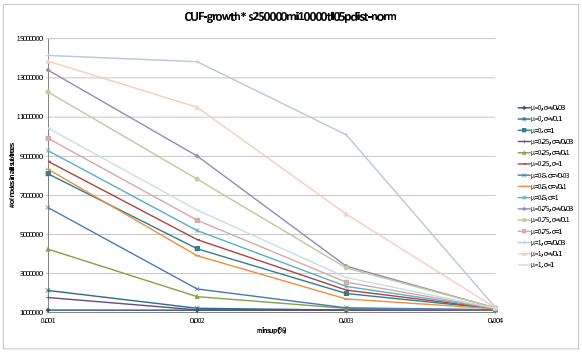


(e) 1,000,000 Transactions

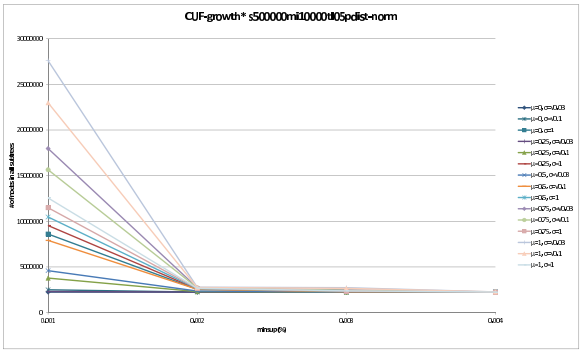
Figure B.15: Memory: Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 5$



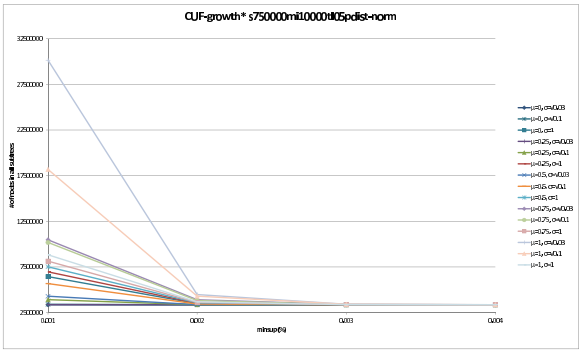
(a) 10,000 Transactions



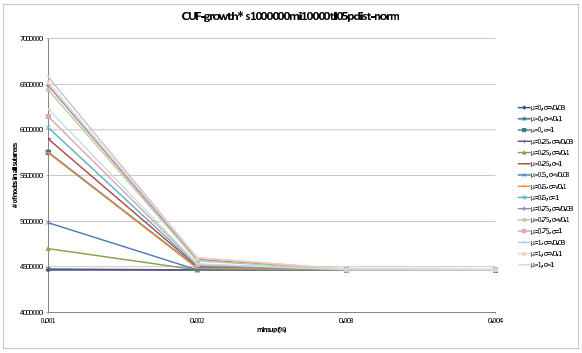
(b) 250,000 Transactions



(c) 500,000 Transactions

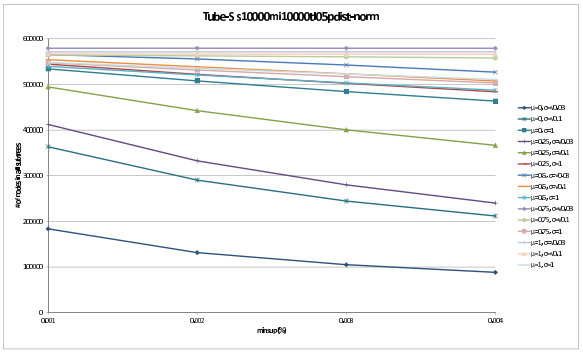


(d) 750,000 Transactions

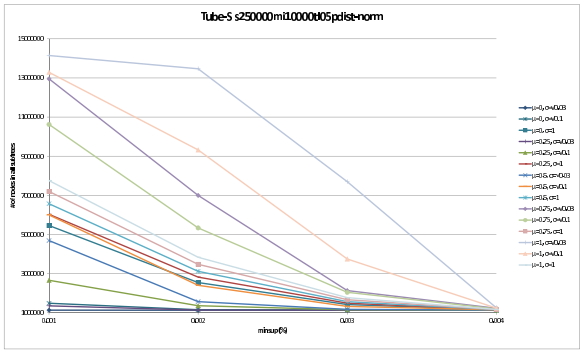


(e) 1,000,000 Transactions

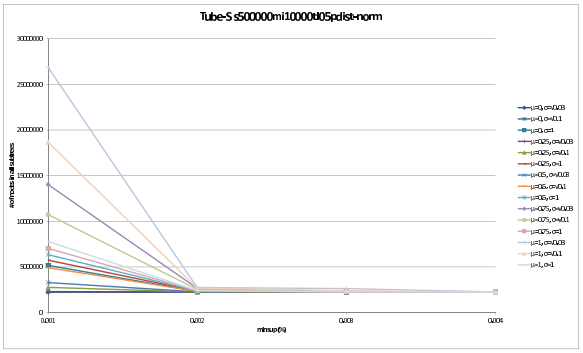
Figure B.16: Memory: CUF-growth*, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$



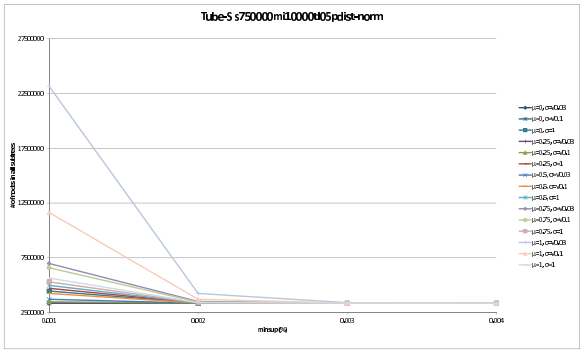
(a) 10,000 Transactions



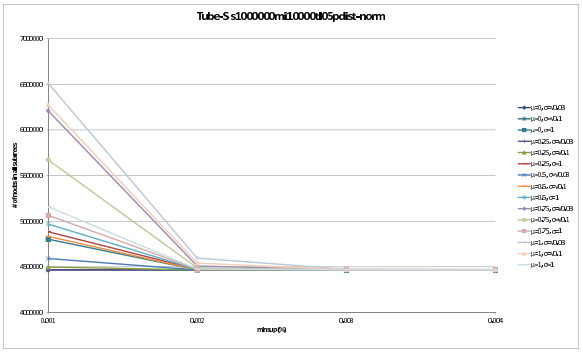
(b) 250,000 Transactions



(c) 500,000 Transactions

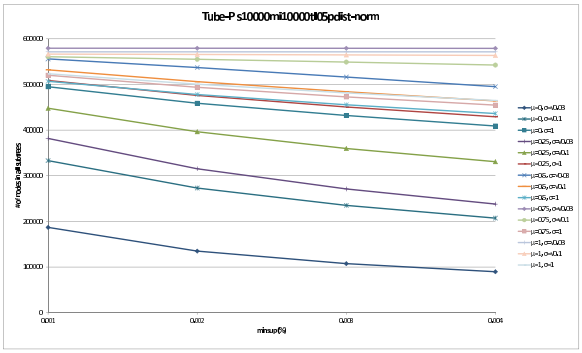


(d) 750,000 Transactions

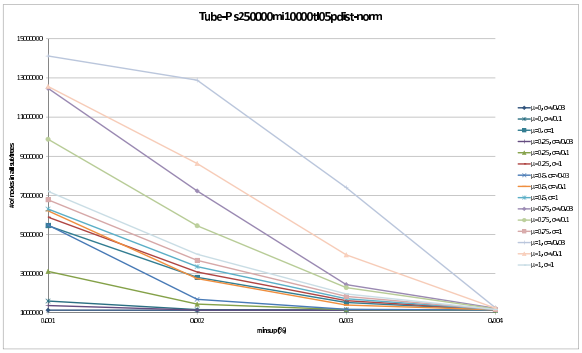


(e) 1,000,000 Transactions

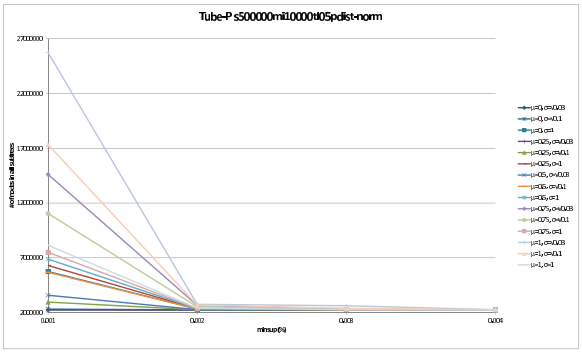
Figure B.17: Memory: Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$



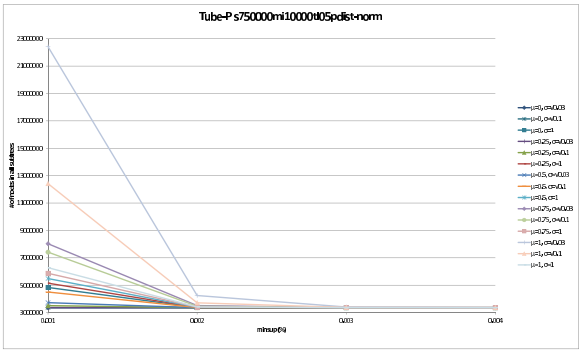
(a) 10,000 Transactions



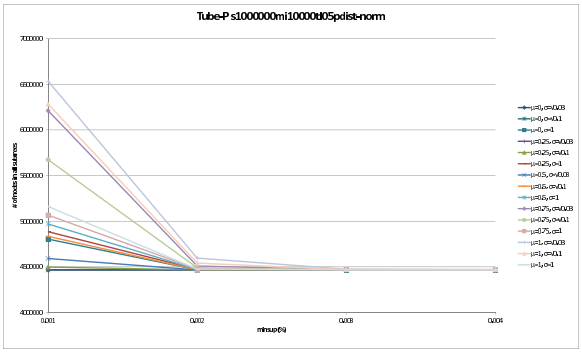
(b) 250,000 Transactions



(c) 500,000 Transactions



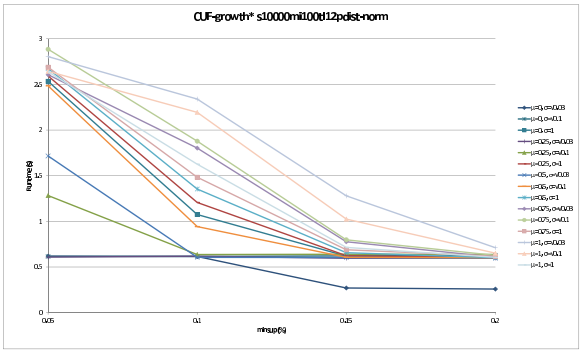
(d) 750,000 Transactions



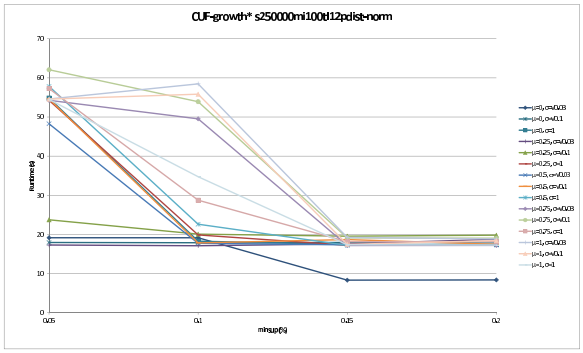
(e) 1,000,000 Transactions

Figure B.18: Memory: Tube-P, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 5$

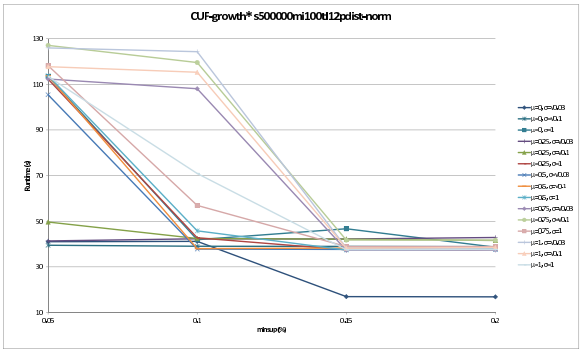
B.3 Transaction Length 12 Runtimes



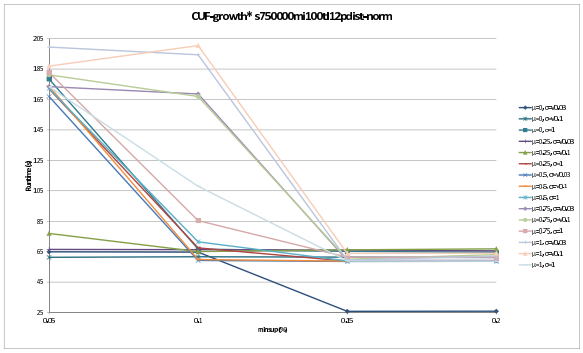
(a) 10,000 Transactions



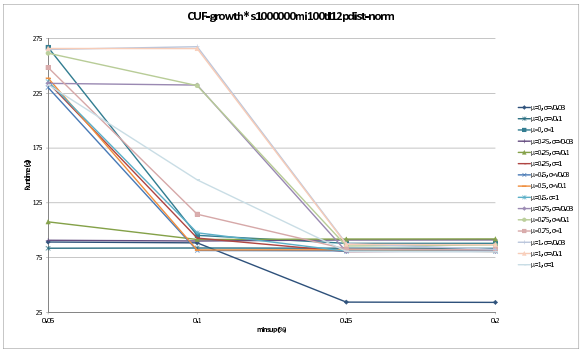
(b) 250,000 Transactions



(c) 500,000 Transactions

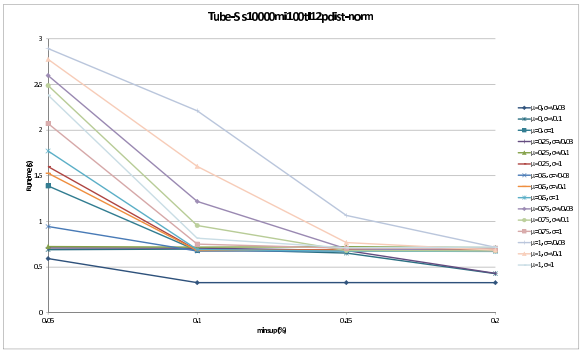


(d) 750,000 Transactions

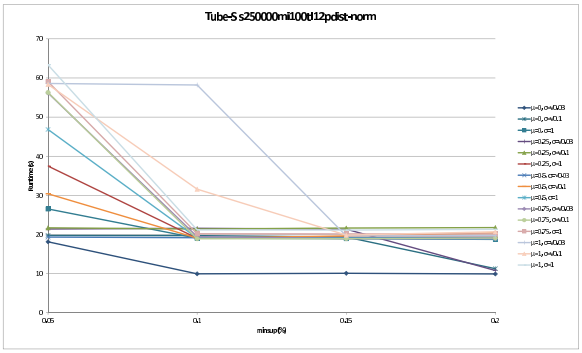


(e) 1,000,000 Transactions

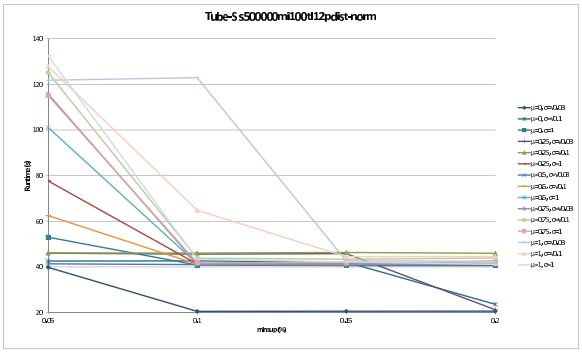
Figure B.19: Runtime: CUF-growth*, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 12$



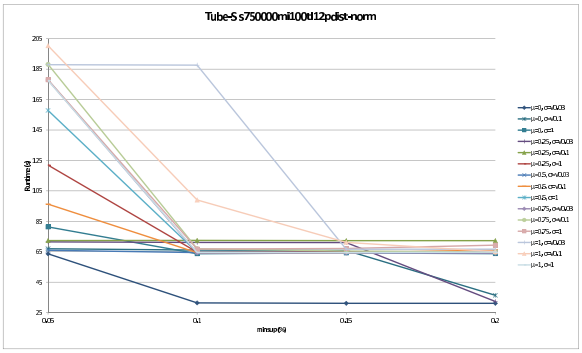
(a) 10,000 Transactions



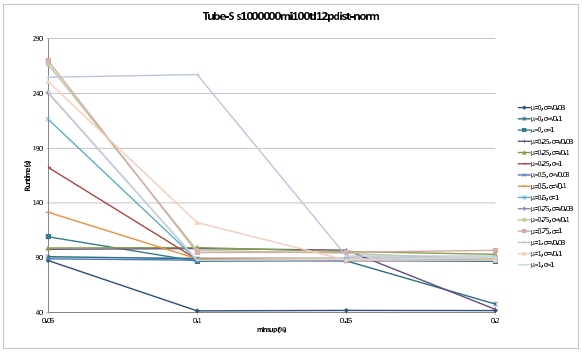
(b) 250,000 Transactions



(c) 500,000 Transactions

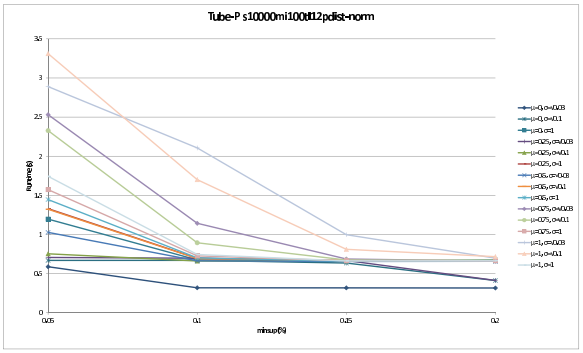


(d) 750,000 Transactions

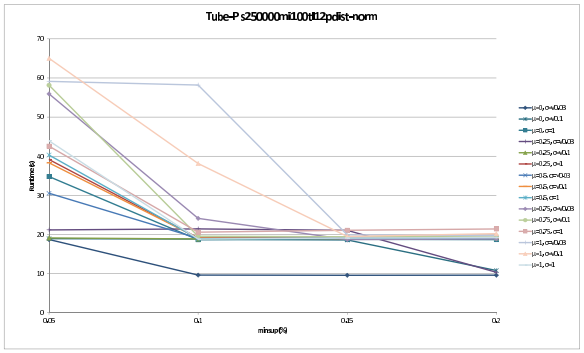


(e) 1,000,000 Transactions

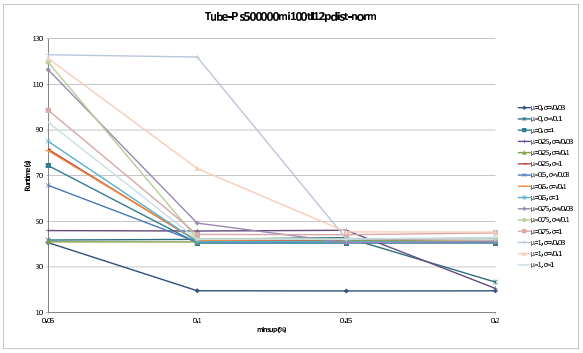
Figure B.20: Runtime: Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 12$



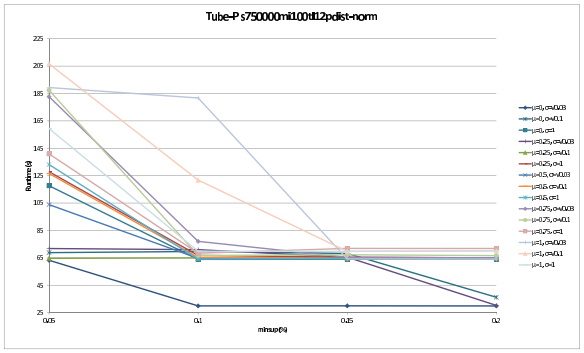
(a) 10,000 Transactions



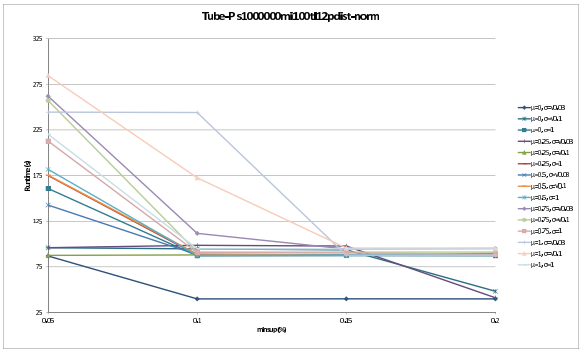
(b) 250,000 Transactions



(c) 500,000 Transactions

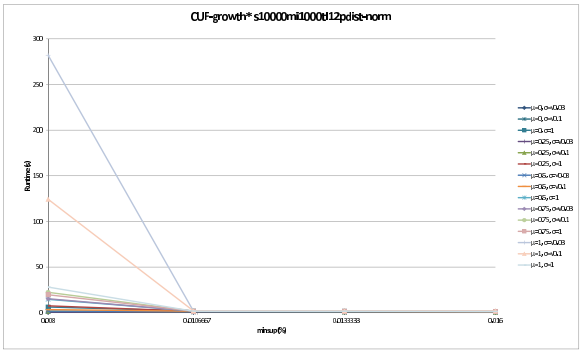


(d) 750,000 Transactions

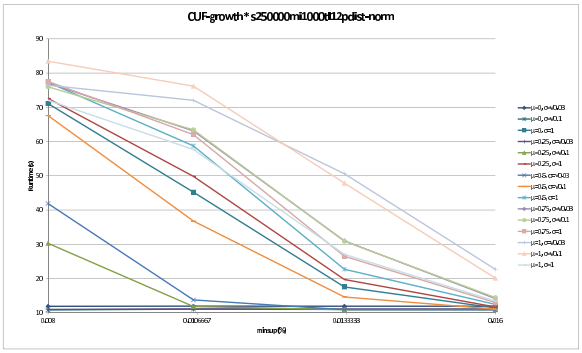


(e) 1,000,000 Transactions

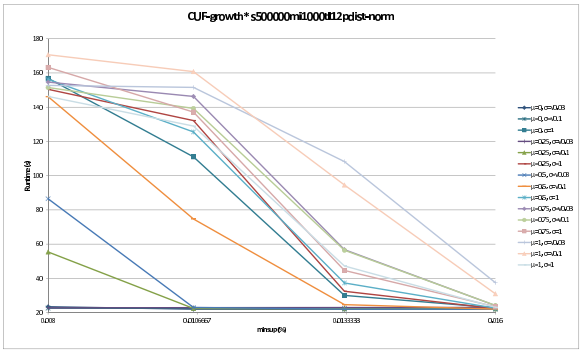
Figure B.21: Runtime: Tube-P, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 12$



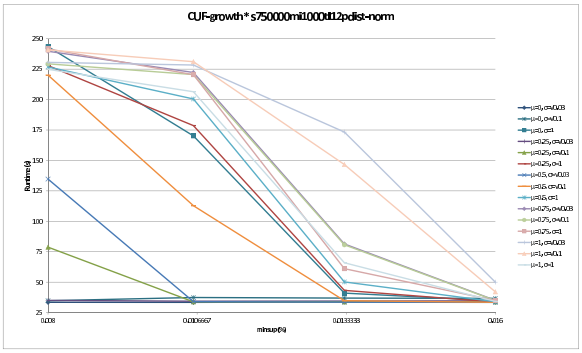
(a) 10,000 Transactions



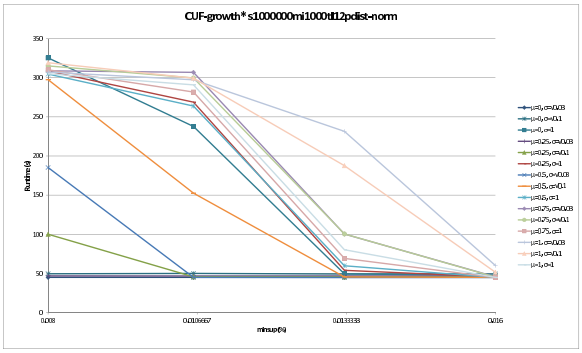
(b) 250,000 Transactions



(c) 500,000 Transactions

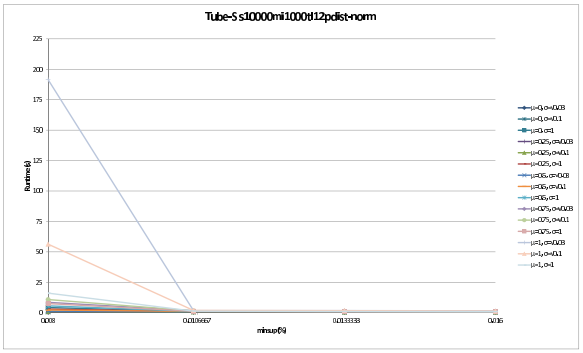


(d) 750,000 Transactions

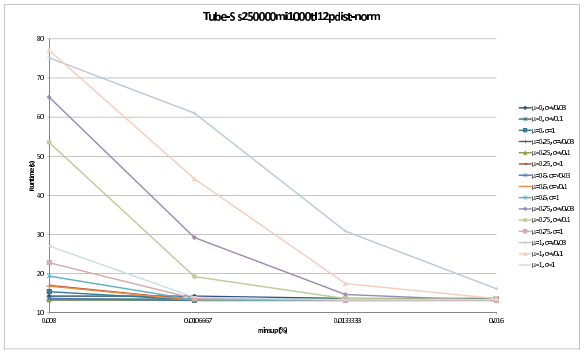


(e) 1,000,000 Transactions

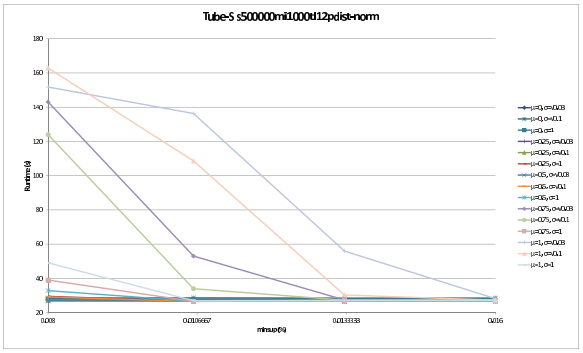
Figure B.22: Runtime: CUF-growth*, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 12$



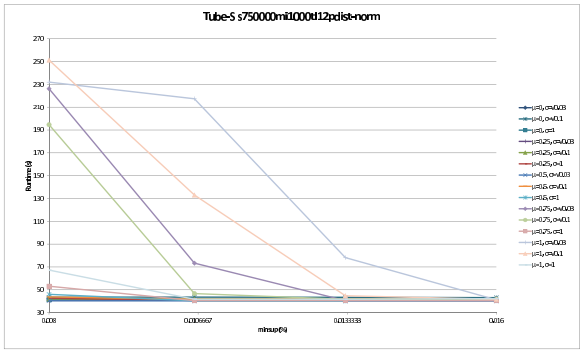
(a) 10,000 Transactions



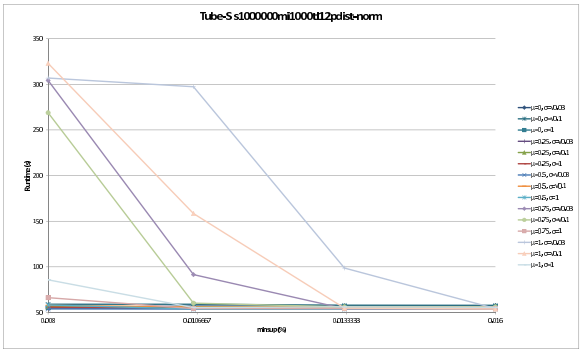
(b) 250,000 Transactions



(c) 500,000 Transactions

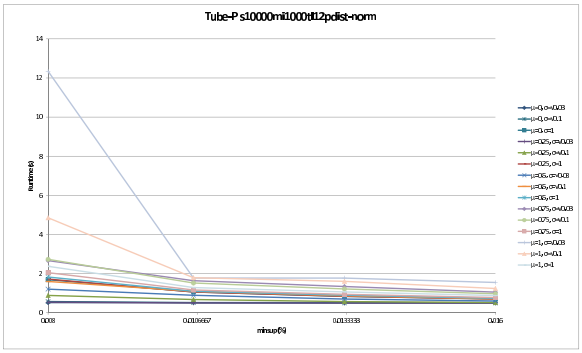


(d) 750,000 Transactions

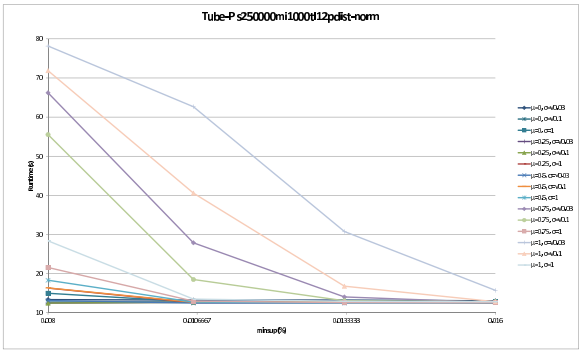


(e) 1,000,000 Transactions

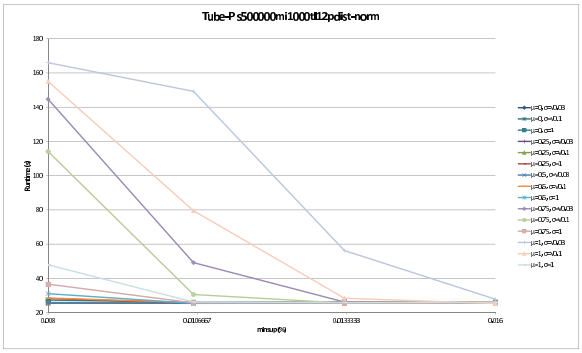
Figure B.23: Runtime: Tube-S, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 12$



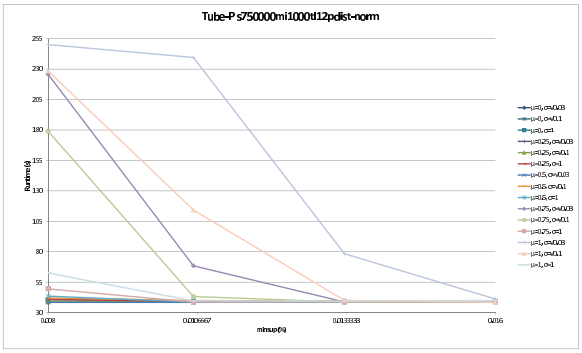
(a) 10,000 Transactions



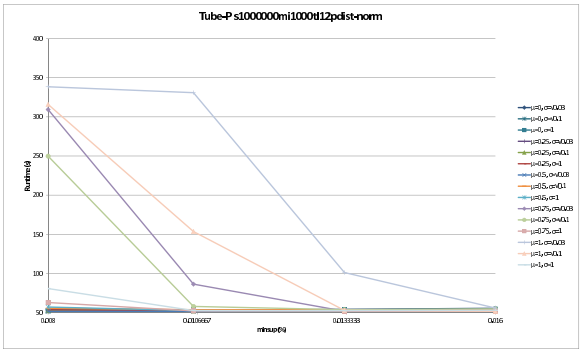
(b) 250,000 Transactions



(c) 500,000 Transactions

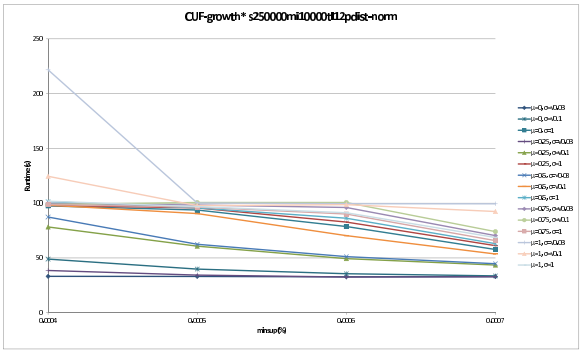


(d) 750,000 Transactions

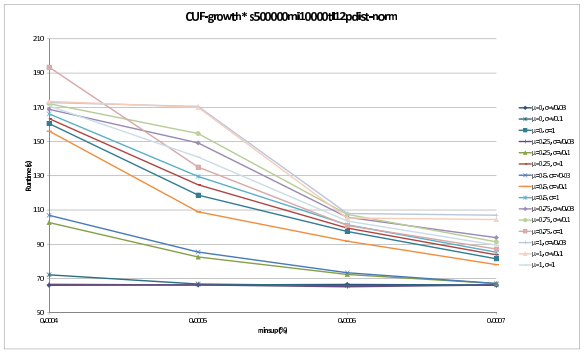


(e) 1,000,000 Transactions

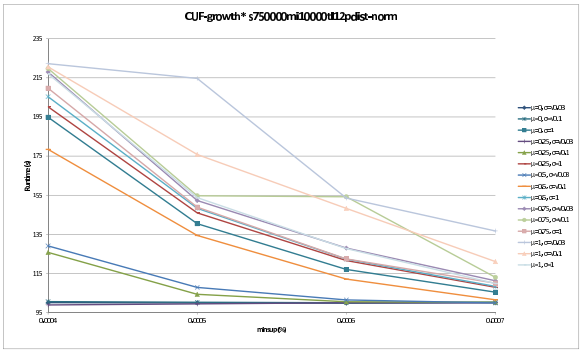
Figure B.24: Runtime: Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 12$



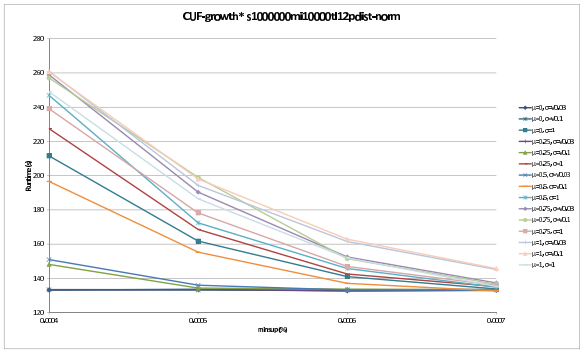
(a) 250,000 Transactions



(b) 500,000 Transactions

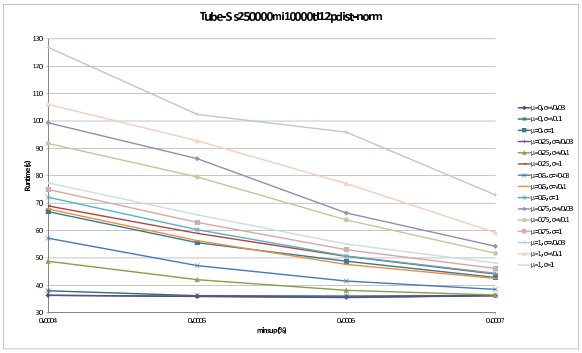


(c) 750,000 Transactions

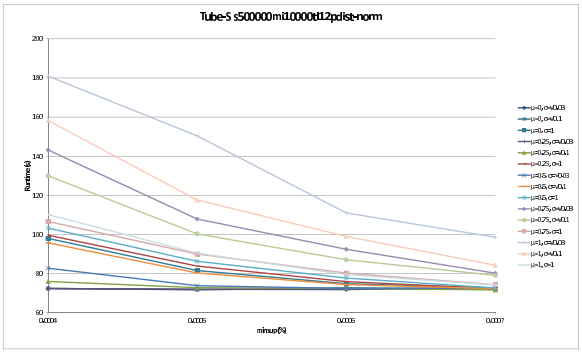


(d) 1,000,000 Transactions

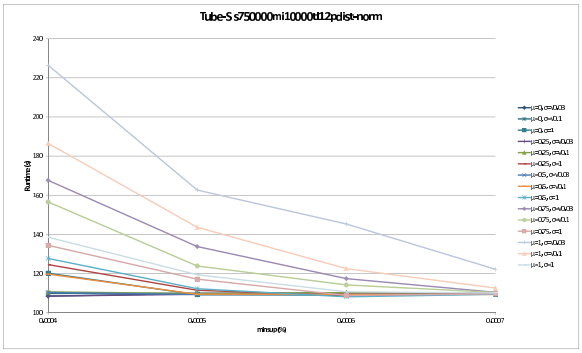
Figure B.25: Runtime: CUF-growth*, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 12$



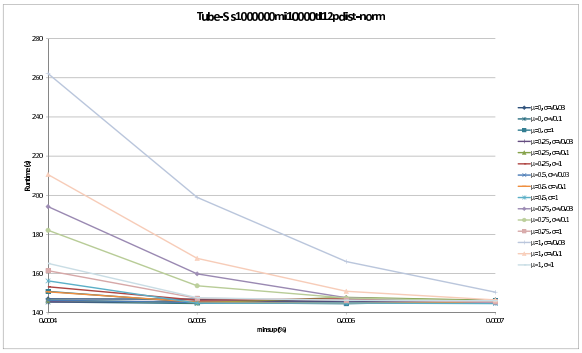
(a) 250,000 Transactions



(b) 500,000 Transactions

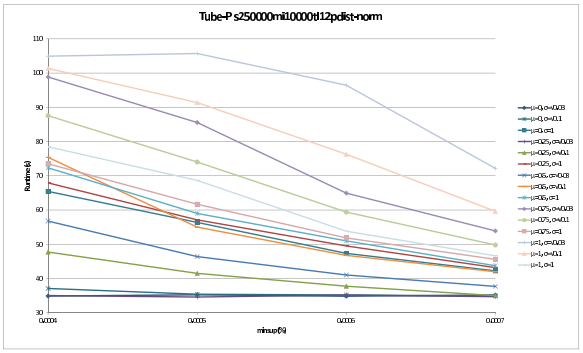


(c) 750,000 Transactions

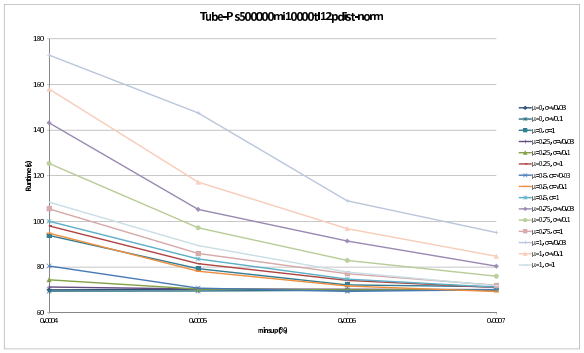


(d) 1,000,000 Transactions

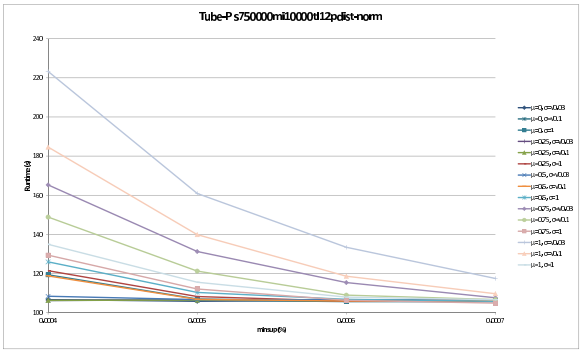
Figure B.26: Runtime: Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 12$



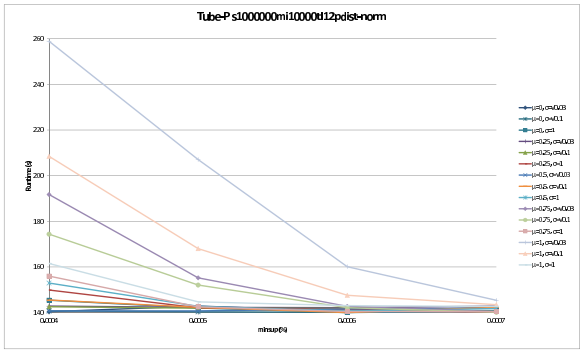
(a) 250,000 Transactions



(b) 500,000 Transactions



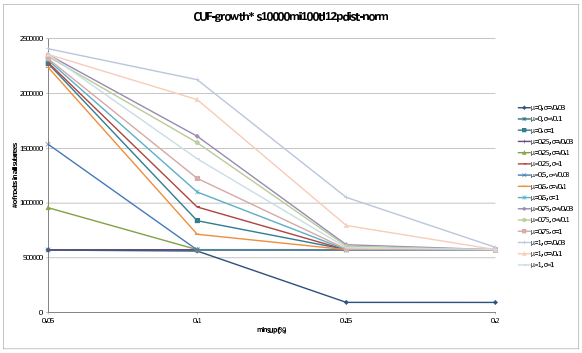
(c) 750,000 Transactions



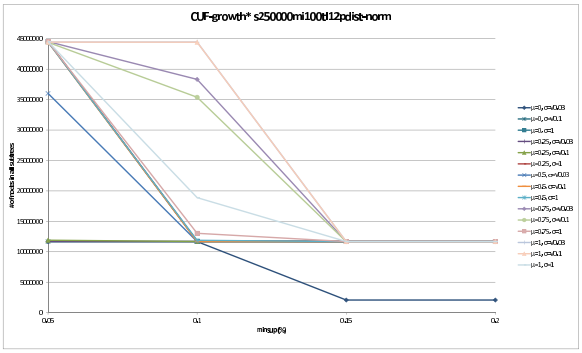
(d) 1,000,000 Transactions

Figure B.27: Runtime: Tube-P, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 12$

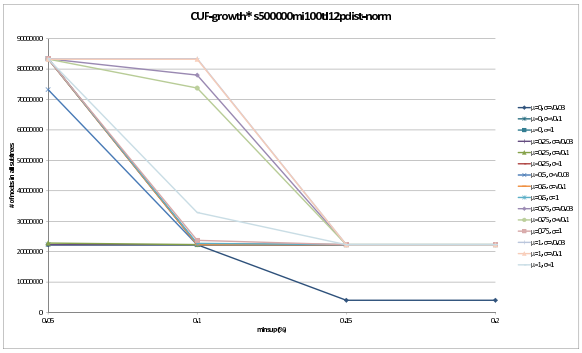
B.4 Transaction Length 12 Memory Usage



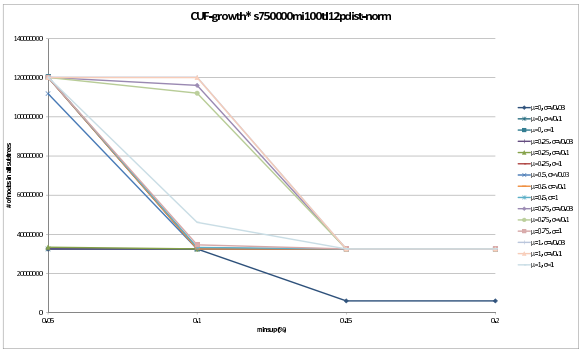
(a) 10,000 Transactions



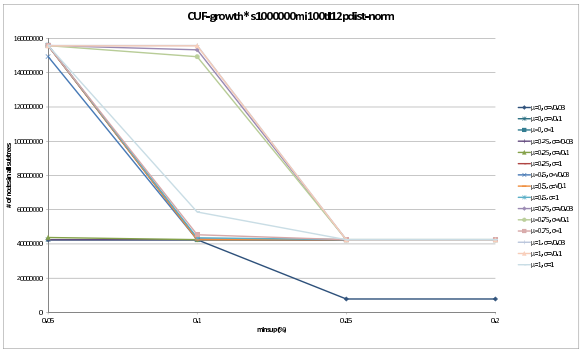
(b) 250,000 Transactions



(c) 500,000 Transactions

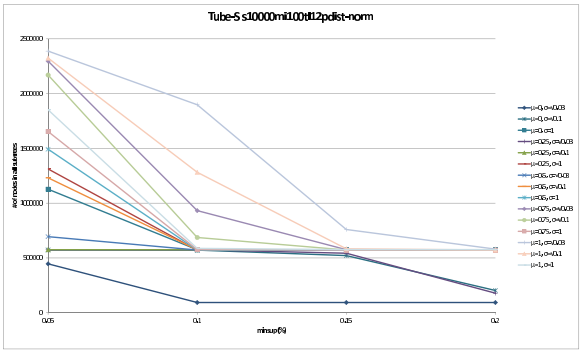


(d) 750,000 Transactions

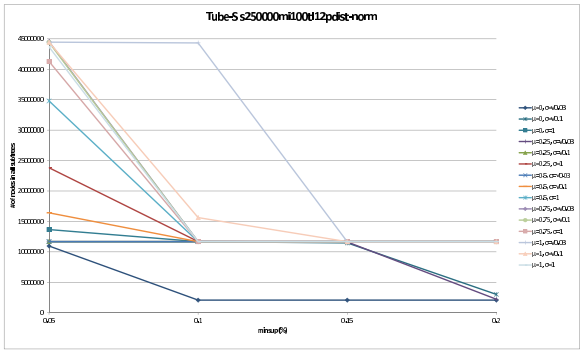


(e) 1,000,000 Transactions

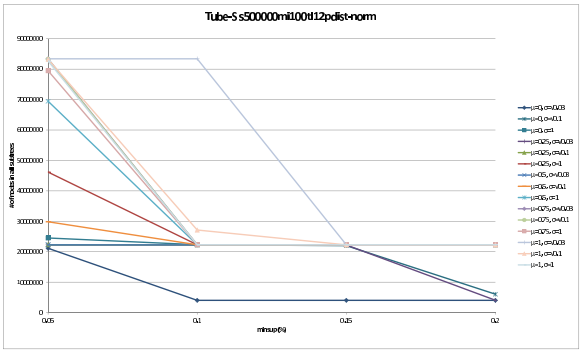
Figure B.28: Memory: CUF-growth*, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 12$



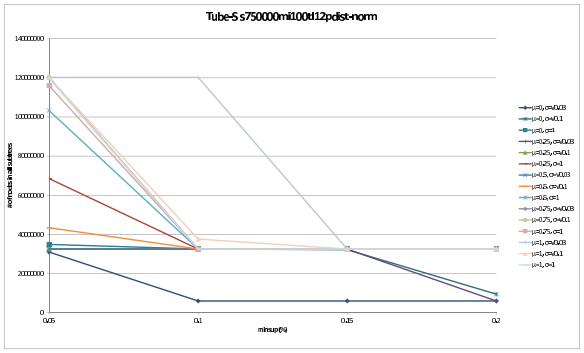
(a) 10,000 Transactions



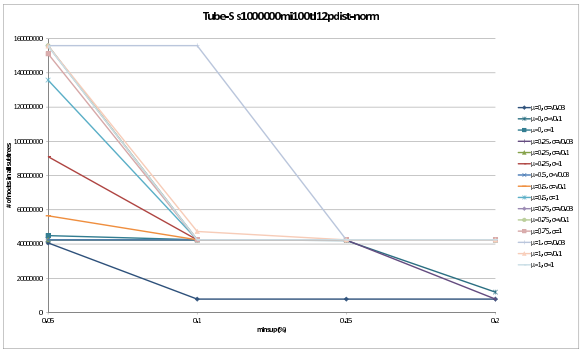
(b) 250,000 Transactions



(c) 500,000 Transactions

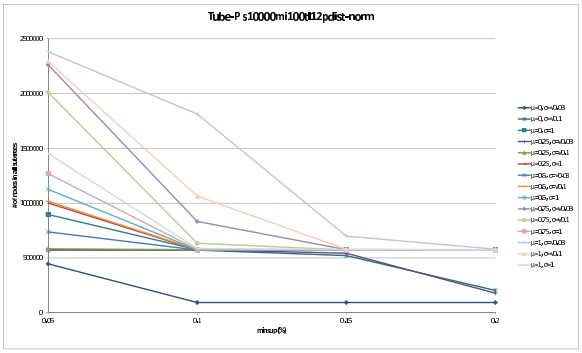


(d) 750,000 Transactions

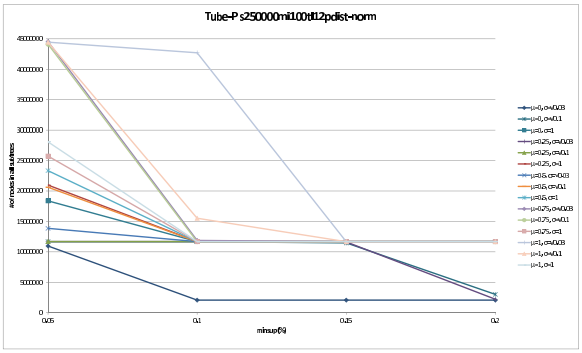


(e) 1,000,000 Transactions

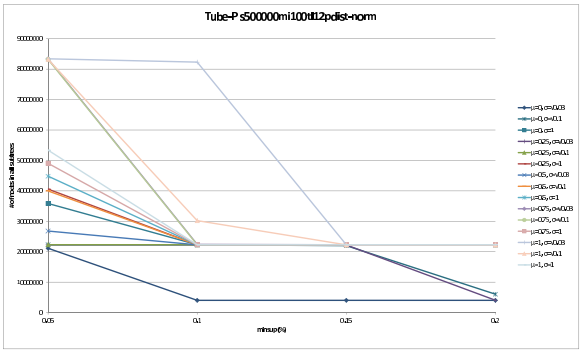
Figure B.29: Memory: Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 12$



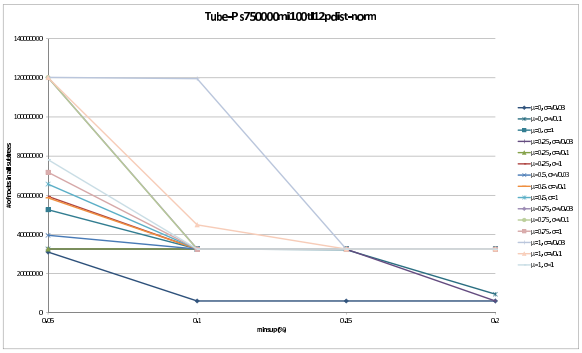
(a) 10,000 Transactions



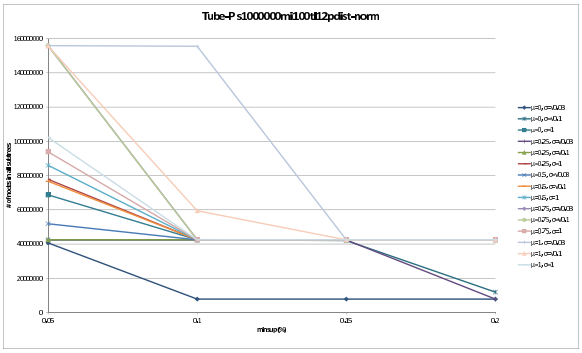
(b) 250,000 Transactions



(c) 500,000 Transactions

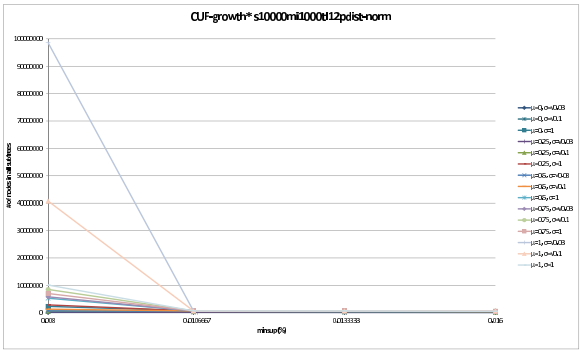


(d) 750,000 Transactions

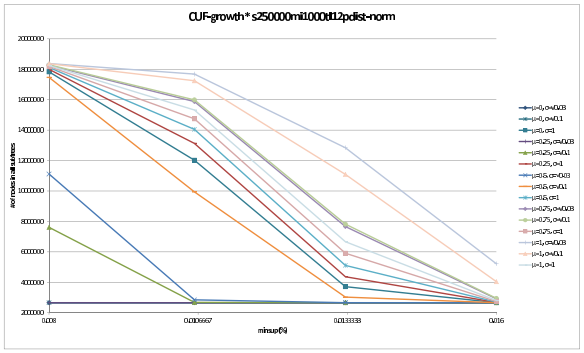


(e) 1,000,000 Transactions

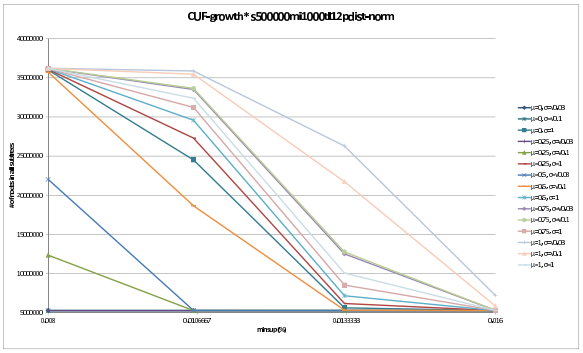
Figure B.30: Memory: Tube-P, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 12$



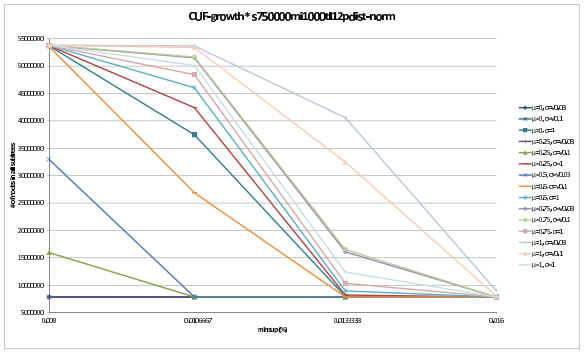
(a) 10,000 Transactions



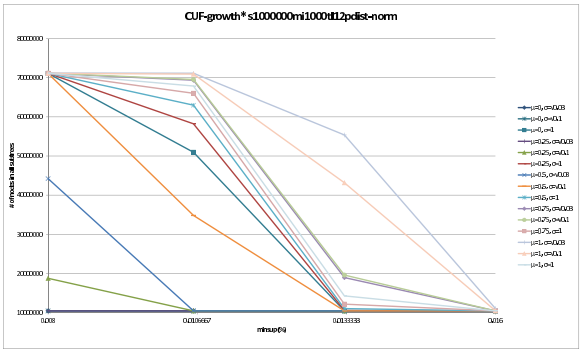
(b) 250,000 Transactions



(c) 500,000 Transactions

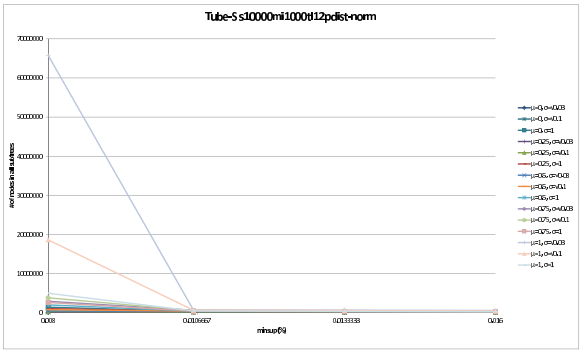


(d) 750,000 Transactions

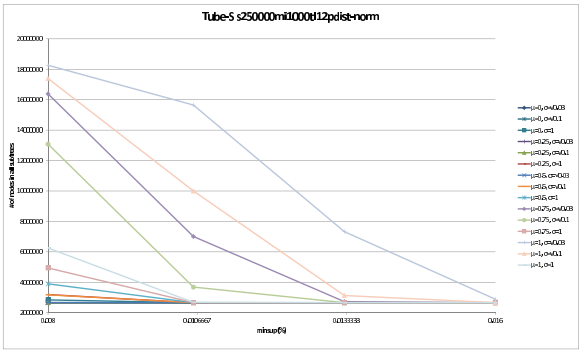


(e) 1,000,000 Transactions

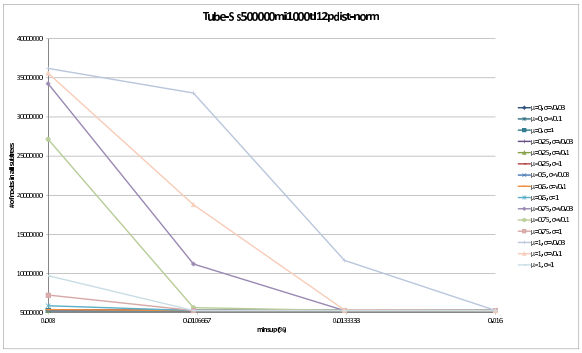
Figure B.31: Memory: CUF-growth*, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 12$



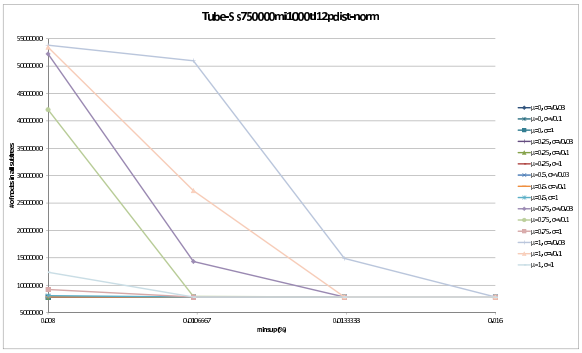
(a) 10,000 Transactions



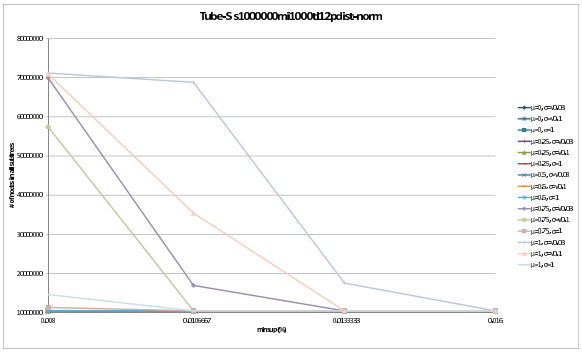
(b) 250,000 Transactions



(c) 500,000 Transactions

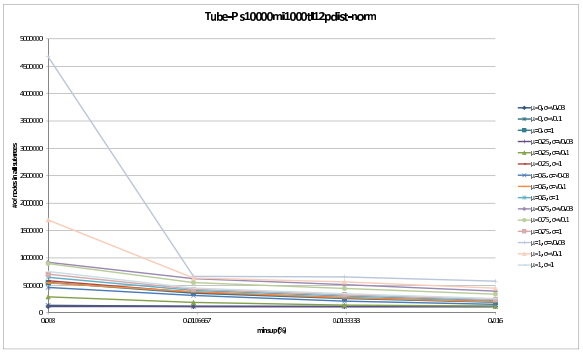


(d) 750,000 Transactions

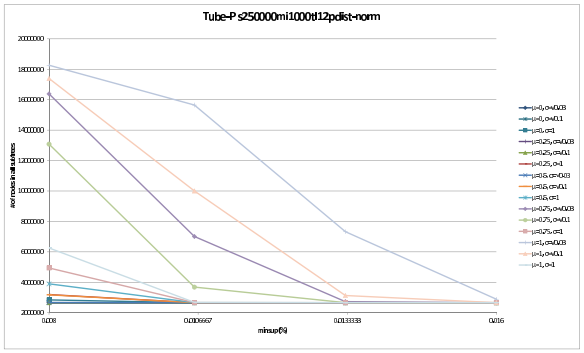


(e) 1,000,000 Transactions

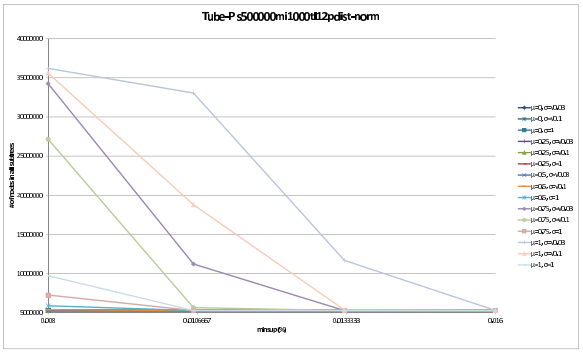
Figure B.32: Memory: Tube-S, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 12$



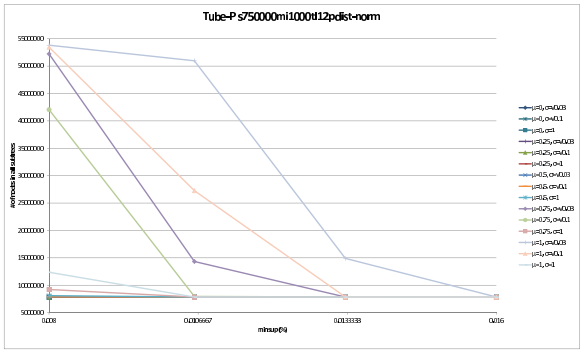
(a) 10,000 Transactions



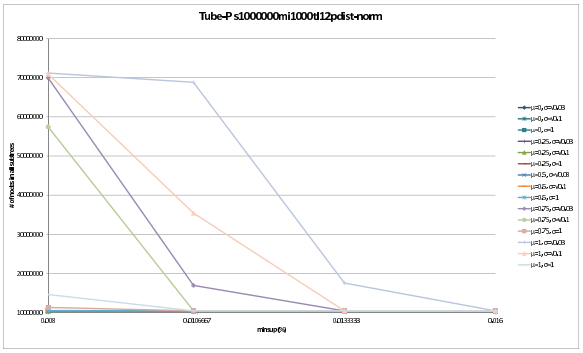
(b) 250,000 Transactions



(c) 500,000 Transactions

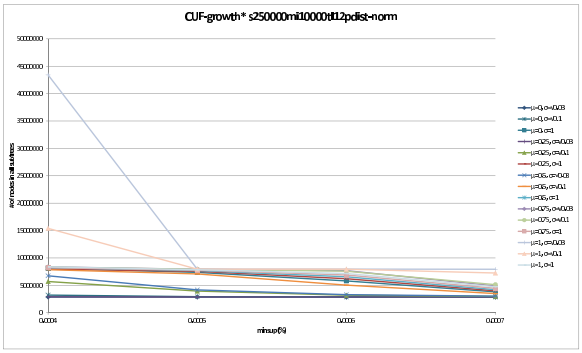


(d) 750,000 Transactions

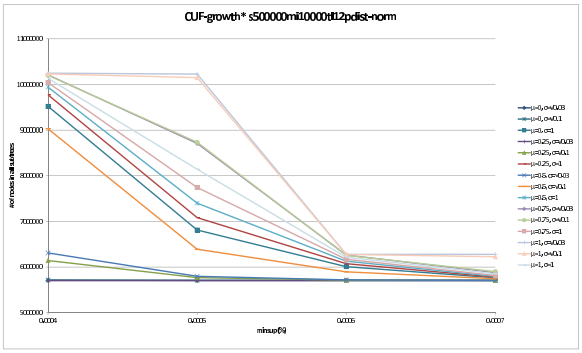


(e) 1,000,000 Transactions

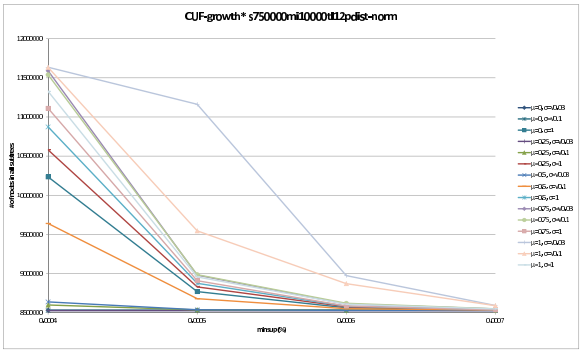
Figure B.33: Memory: Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 12$



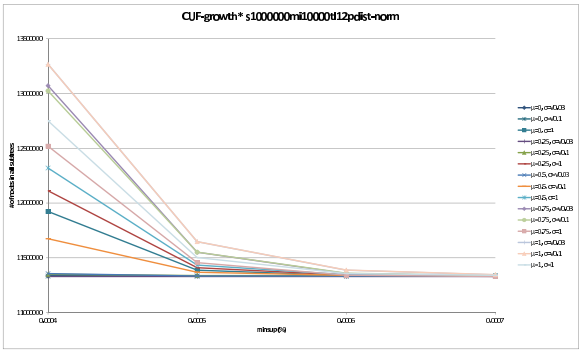
(a) 250,000 Transactions



(b) 500,000 Transactions

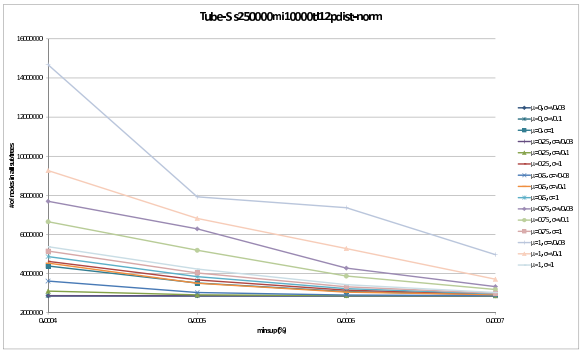


(c) 750,000 Transactions

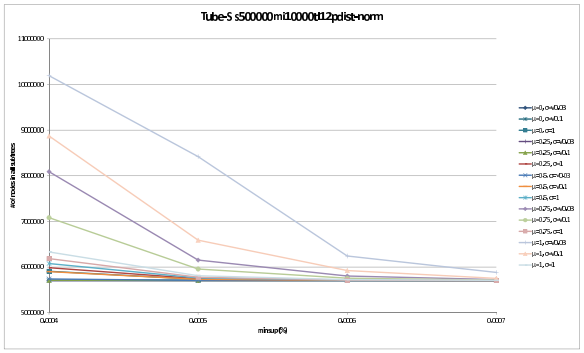


(d) 1,000,000 Transactions

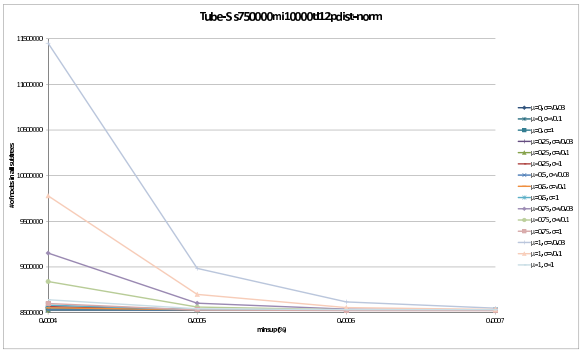
Figure B.34: Memory: CUF-growth*, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 12$



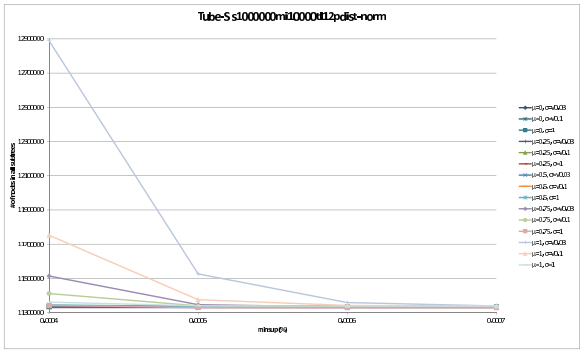
(a) 250,000 Transactions



(b) 500,000 Transactions

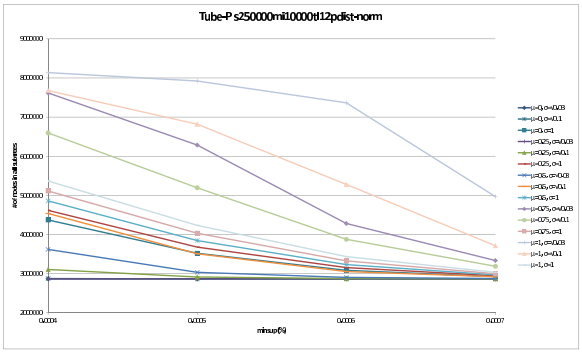


(c) 750,000 Transactions

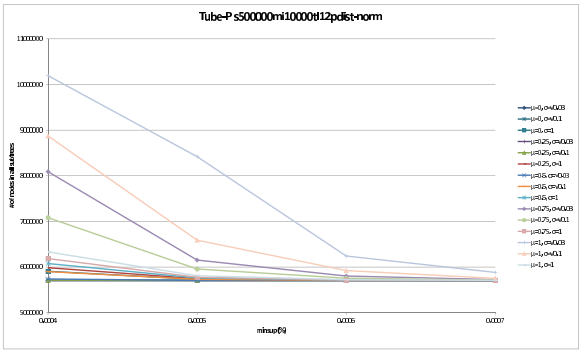


(d) 1,000,000 Transactions

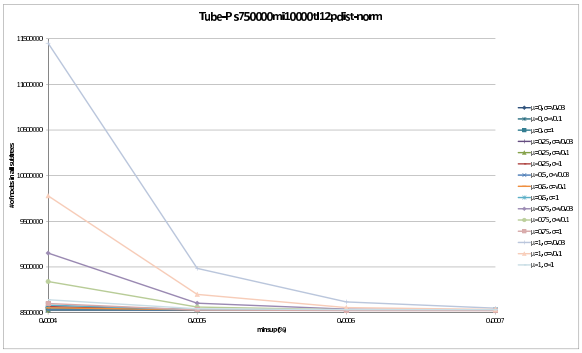
Figure B.35: Memory: Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 12$



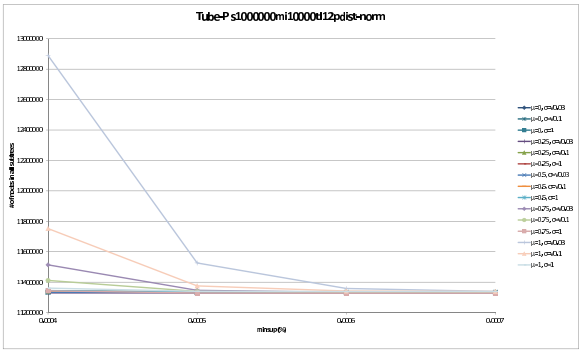
(a) 250,000 Transactions



(b) 500,000 Transactions



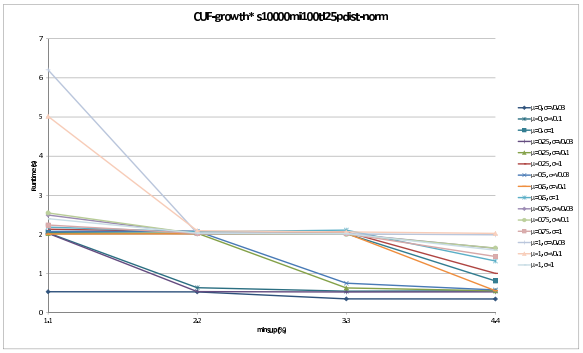
(c) 750,000 Transactions



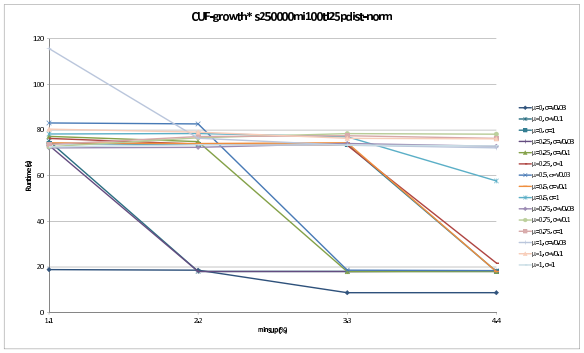
(d) 1,000,000 Transactions

Figure B.36: Memory: Tube-P, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 12$

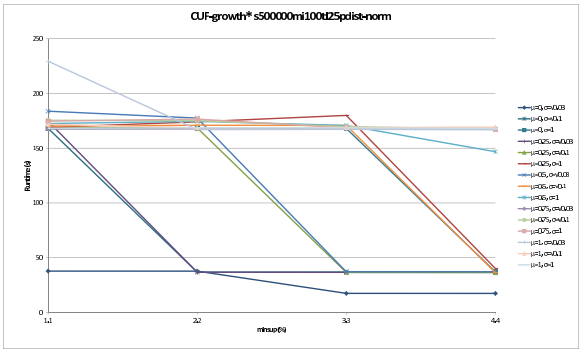
B.5 Transaction Length 25 Runtimes



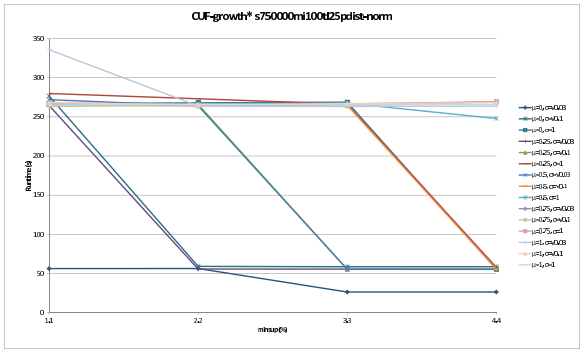
(a) 10,000 Transactions



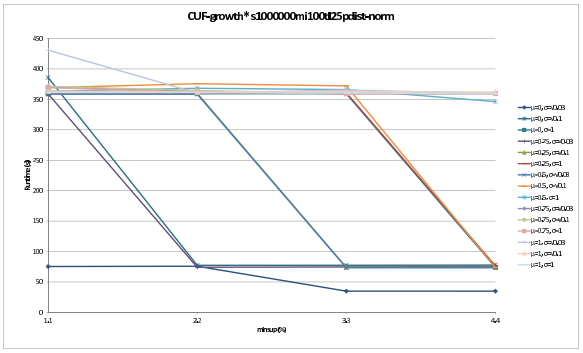
(b) 250,000 Transactions



(c) 500,000 Transactions

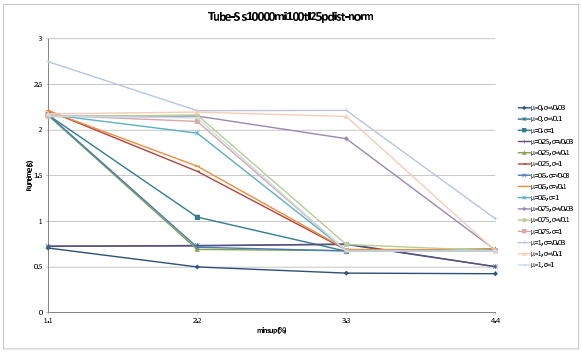


(d) 750,000 Transactions

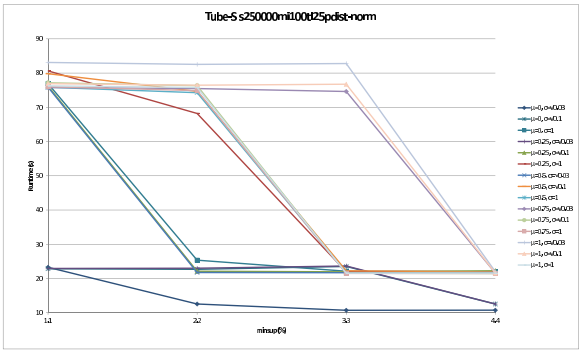


(e) 1,000,000 Transactions

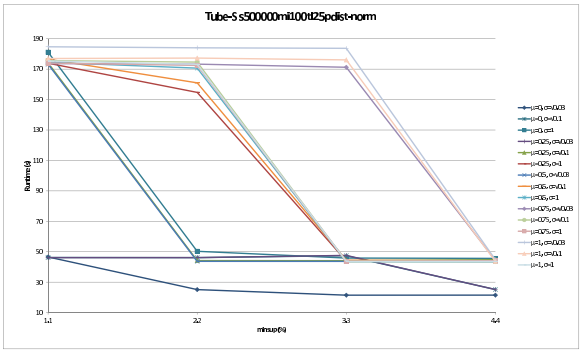
Figure B.37: Runtime: CUF-growth*, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 25$



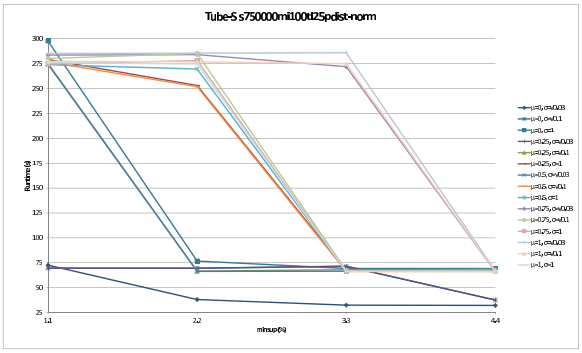
(a) 10,000 Transactions



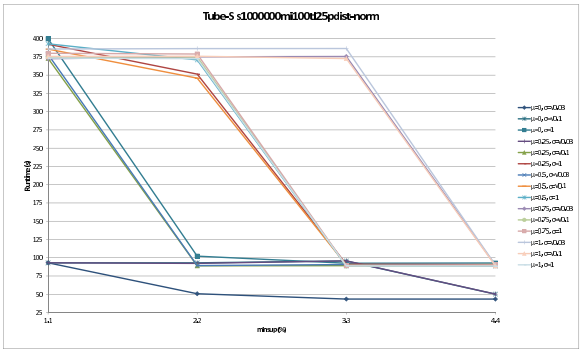
(b) 250,000 Transactions



(c) 500,000 Transactions

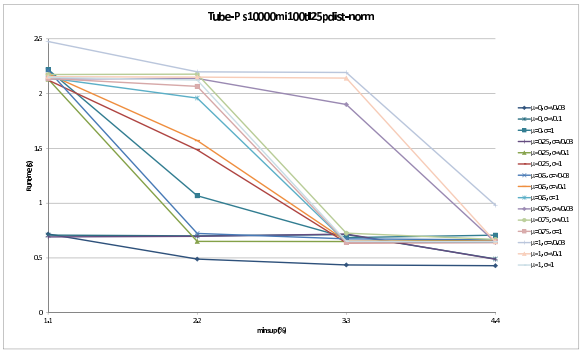


(d) 750,000 Transactions

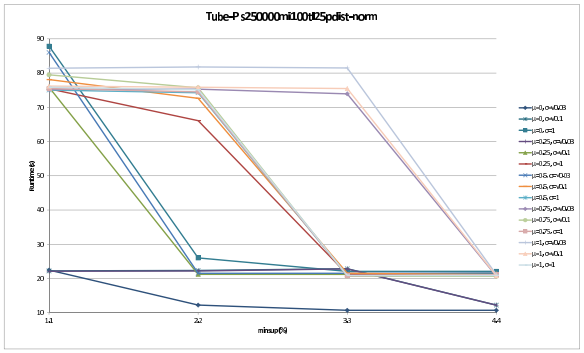


(e) 1,000,000 Transactions

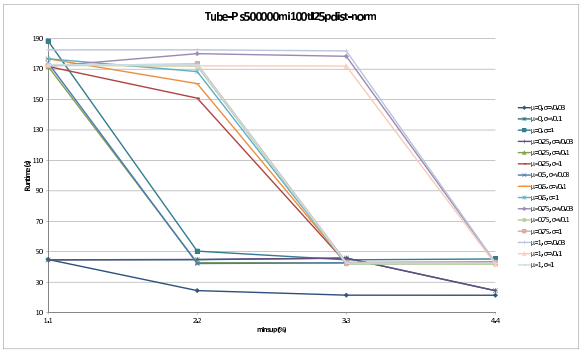
Figure B.38: Runtime: Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 25$



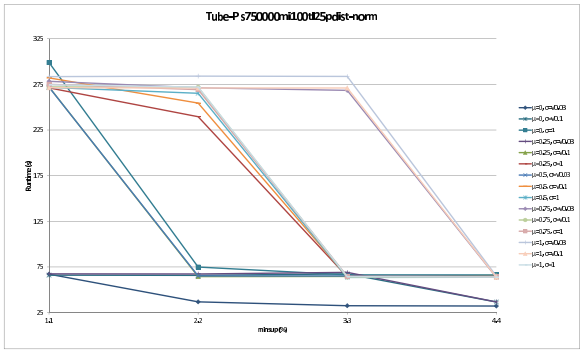
(a) 10,000 Transactions



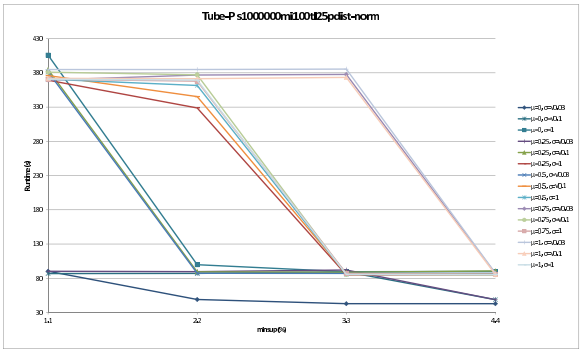
(b) 250,000 Transactions



(c) 500,000 Transactions

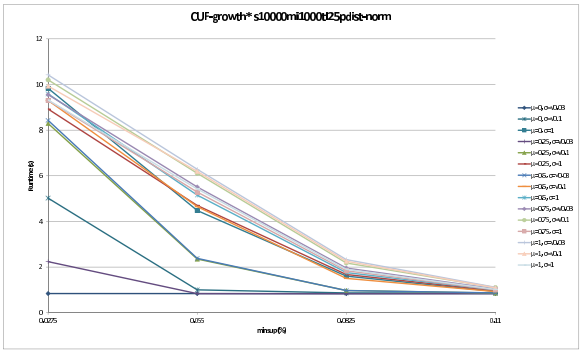


(d) 750,000 Transactions

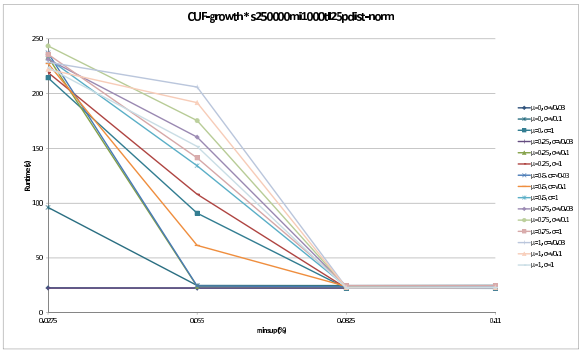


(e) 1,000,000 Transactions

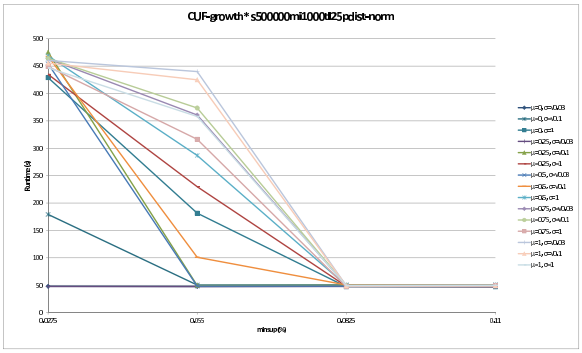
Figure B.39: Runtime: Tube-P, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 25$



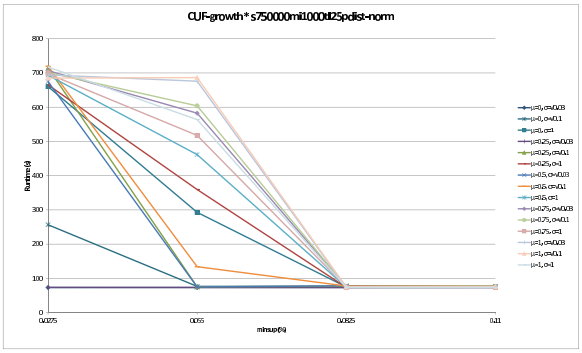
(a) 10,000 Transactions



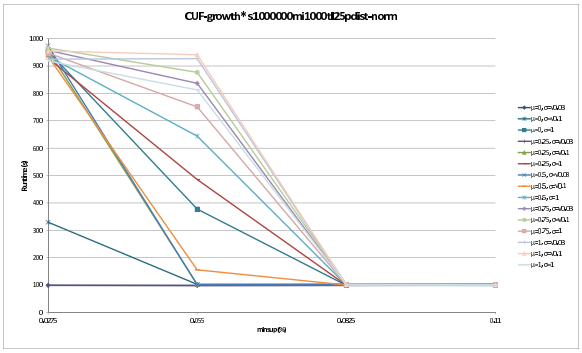
(b) 250,000 Transactions



(c) 500,000 Transactions

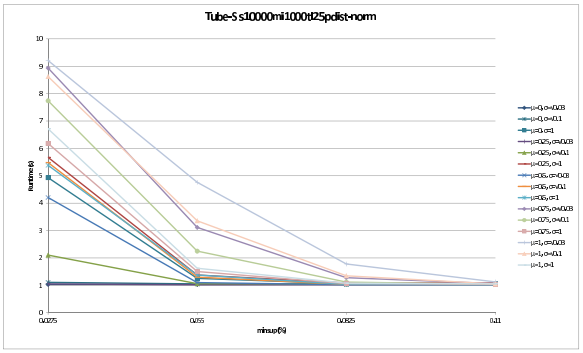


(d) 750,000 Transactions

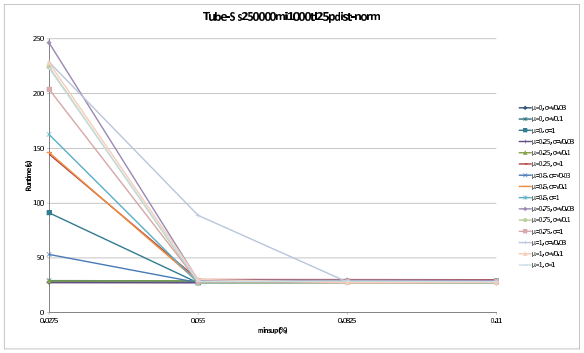


(e) 1,000,000 Transactions

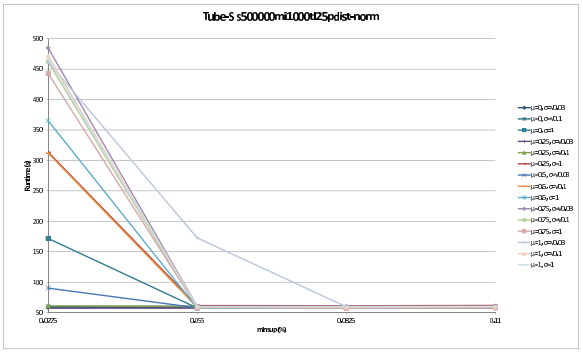
Figure B.40: Runtime: CUF-growth*, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 25$



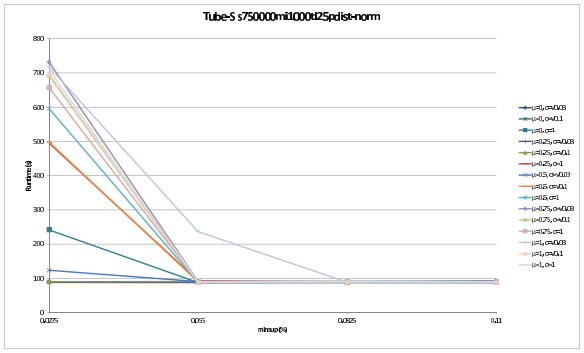
(a) 10,000 Transactions



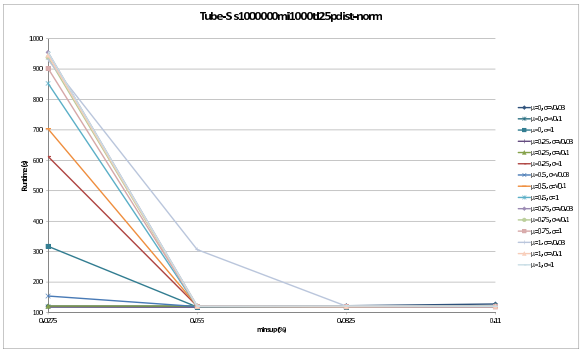
(b) 250,000 Transactions



(c) 500,000 Transactions

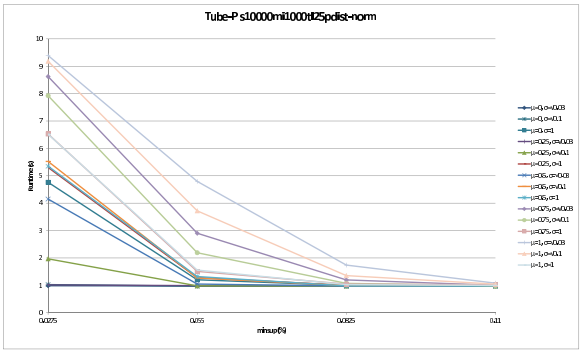


(d) 750,000 Transactions

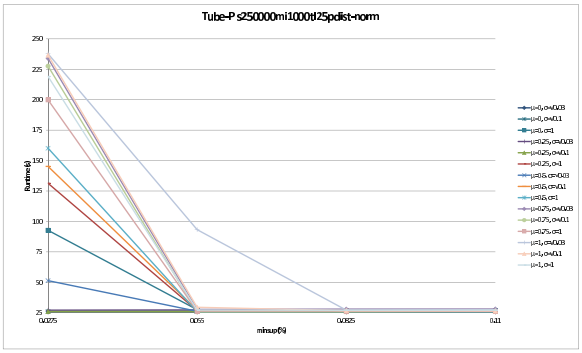


(e) 1,000,000 Transactions

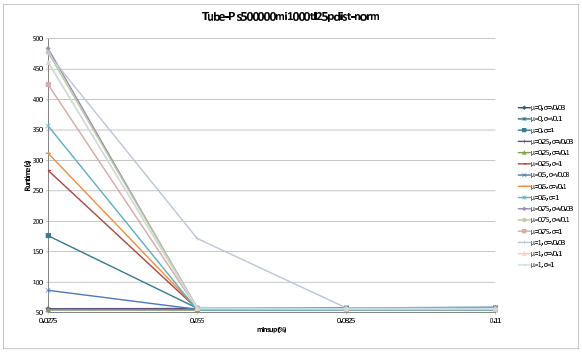
Figure B.41: Runtime: Tube-S, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 25$



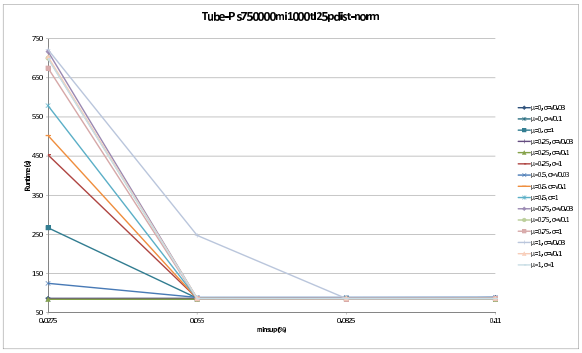
(a) 10,000 Transactions



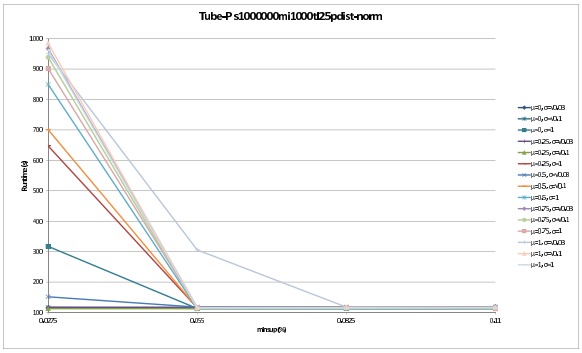
(b) 250,000 Transactions



(c) 500,000 Transactions

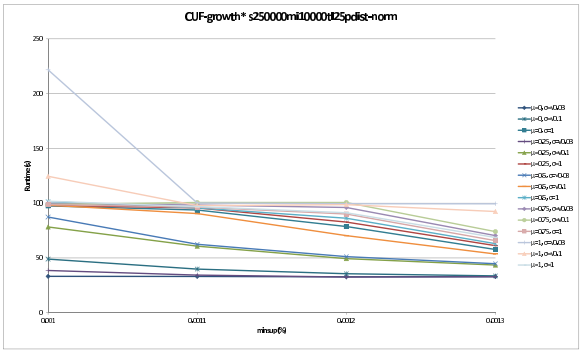


(d) 750,000 Transactions

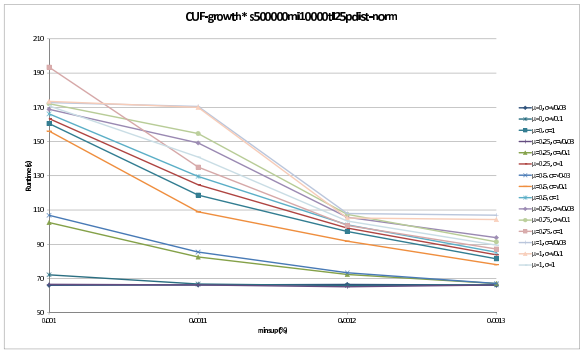


(e) 1,000,000 Transactions

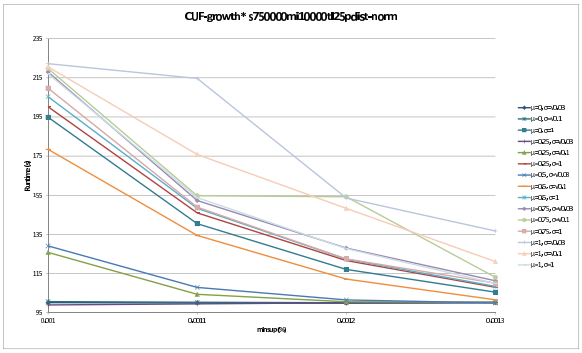
Figure B.42: Runtime: Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 25$



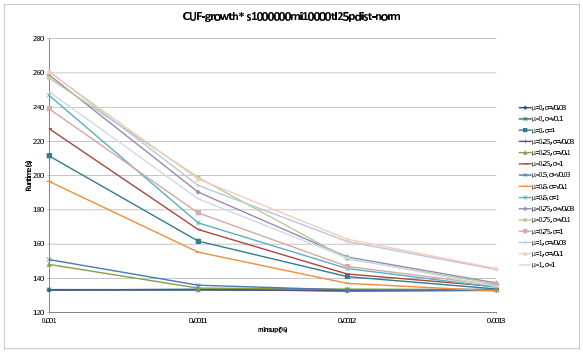
(a) 250,000 Transactions



(b) 500,000 Transactions

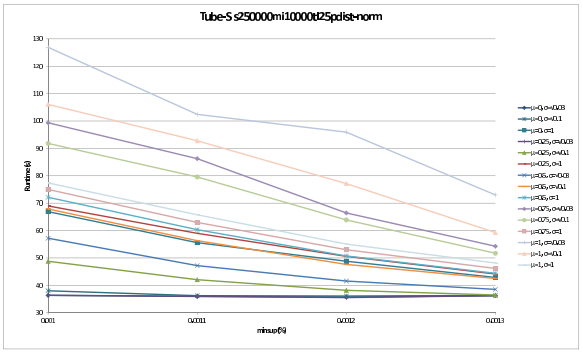


(c) 750,000 Transactions

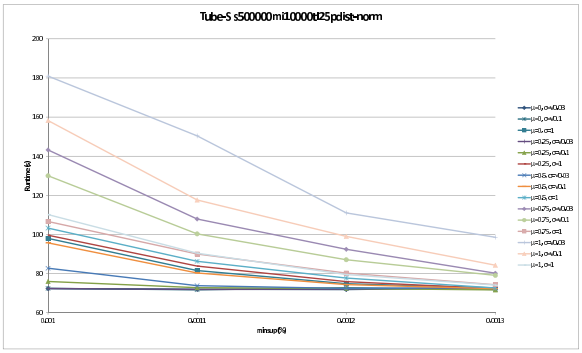


(d) 1,000,000 Transactions

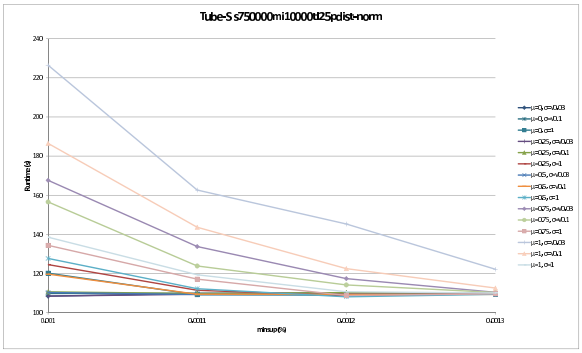
Figure B.43: Runtime: CUF-growth*, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 25$



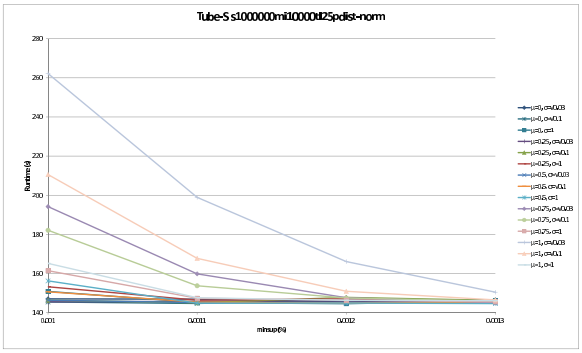
(a) 250,000 Transactions



(b) 500,000 Transactions

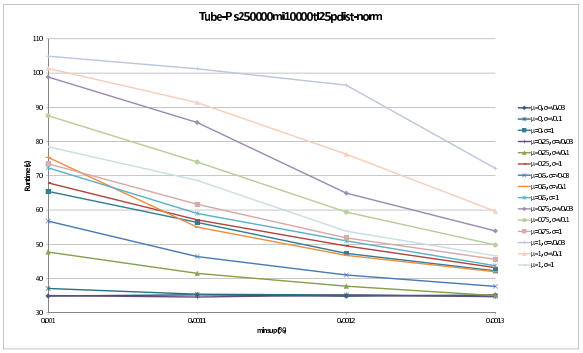


(c) 750,000 Transactions

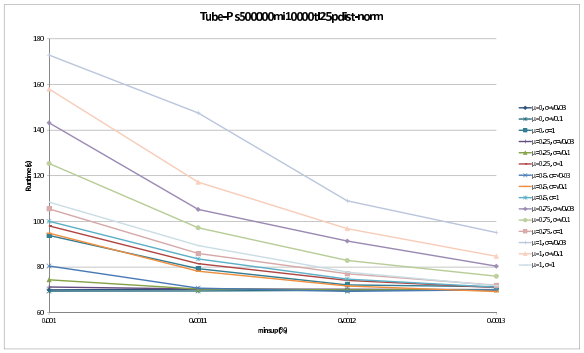


(d) 1,000,000 Transactions

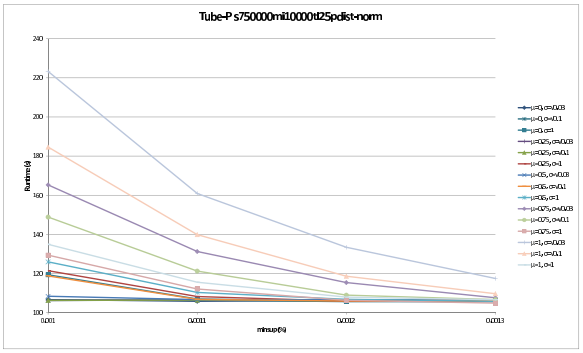
Figure B.44: Runtime: Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 25$



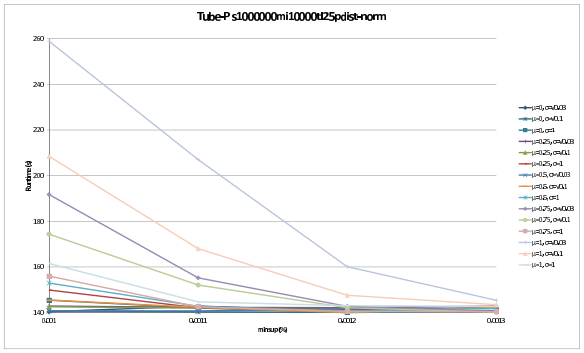
(a) 250,000 Transactions



(b) 500,000 Transactions



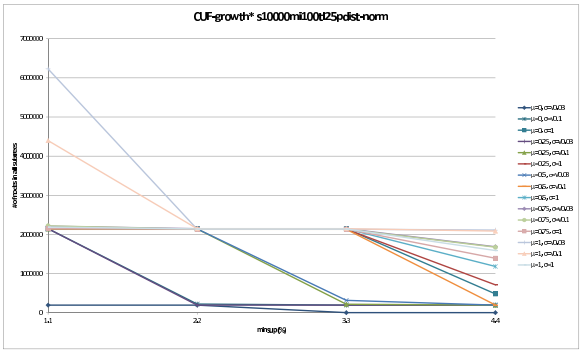
(c) 750,000 Transactions



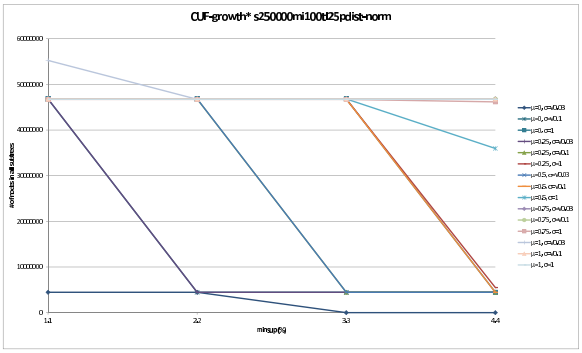
(d) 1,000,000 Transactions

Figure B.45: Runtime: Tube-P, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 25$

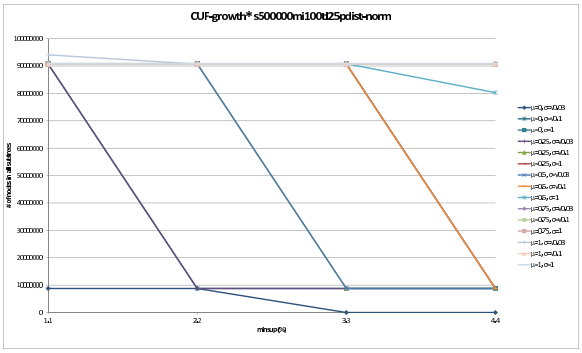
B.6 Transaction Length 25 Memory Usage



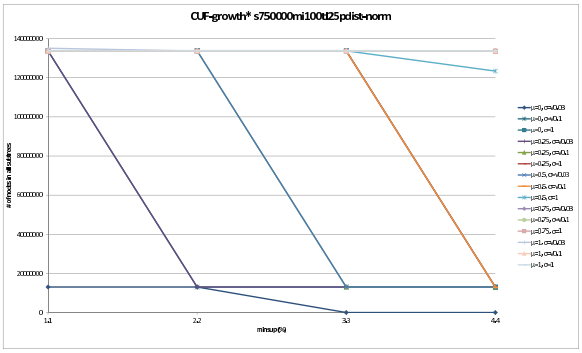
(a) 10,000 Transactions



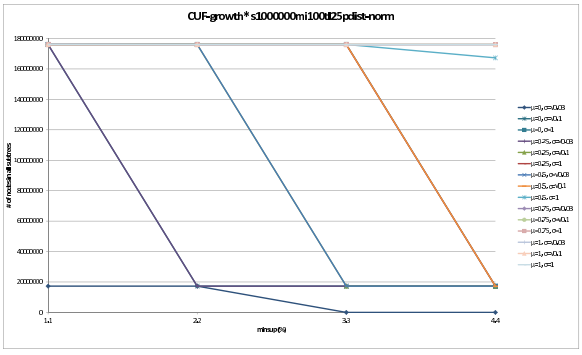
(b) 250,000 Transactions



(c) 500,000 Transactions

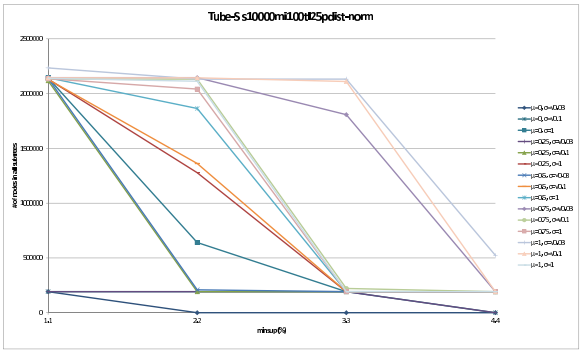


(d) 750,000 Transactions

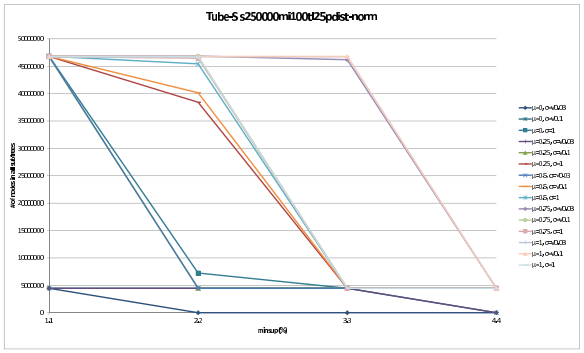


(e) 1,000,000 Transactions

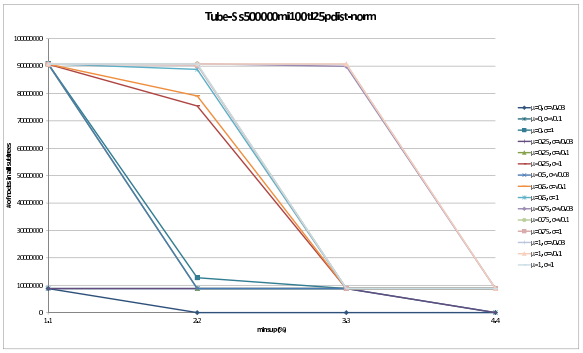
Figure B.46: Memory: CUF-growth*, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 25$



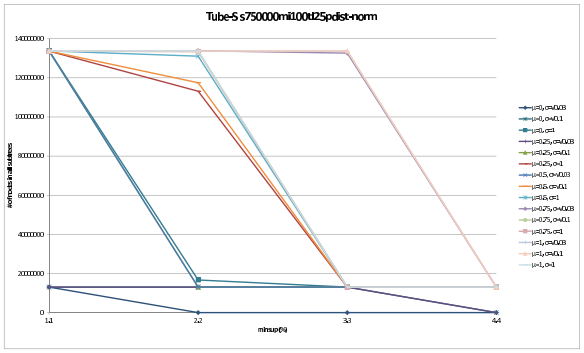
(a) 10,000 Transactions



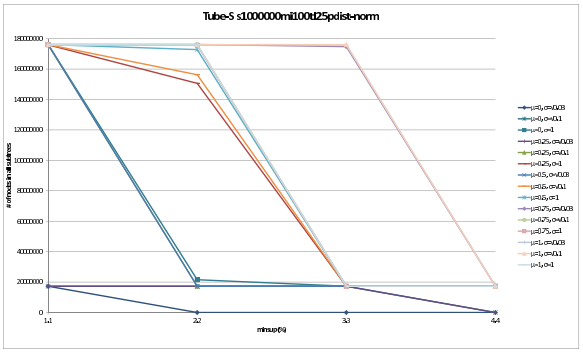
(b) 250,000 Transactions



(c) 500,000 Transactions

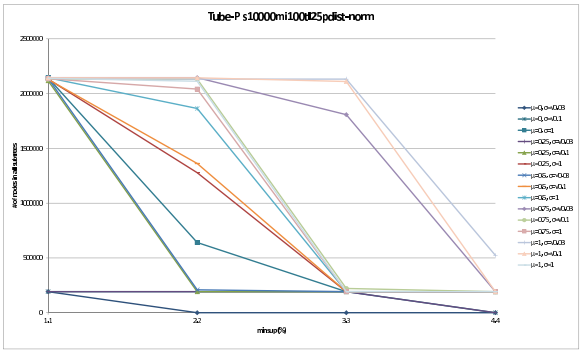


(d) 750,000 Transactions

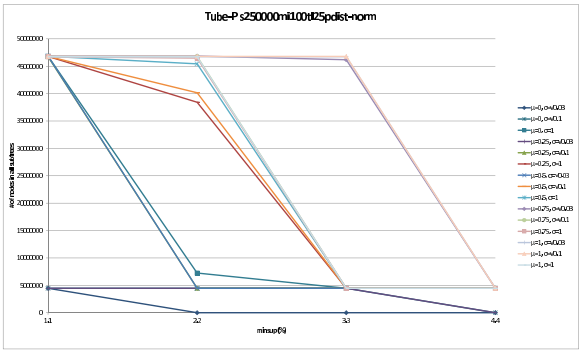


(e) 1,000,000 Transactions

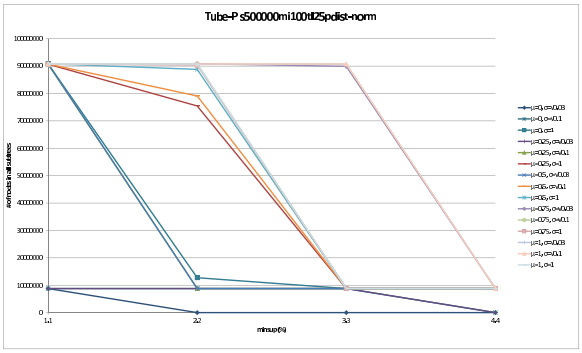
Figure B.47: Memory: Tube-S, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 25$



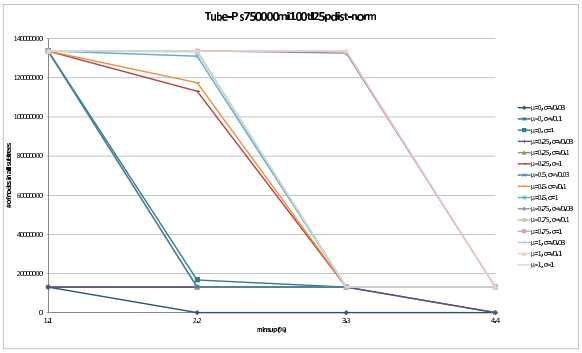
(a) 10,000 Transactions



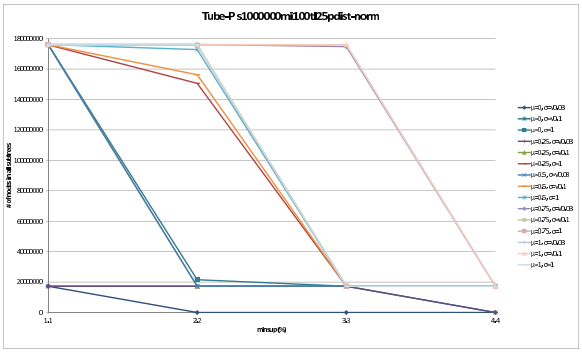
(b) 250,000 Transactions



(c) 500,000 Transactions

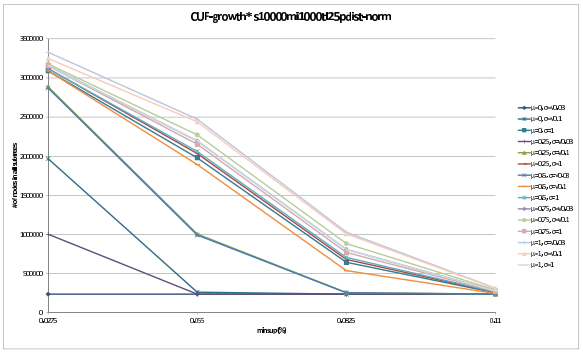


(d) 750,000 Transactions

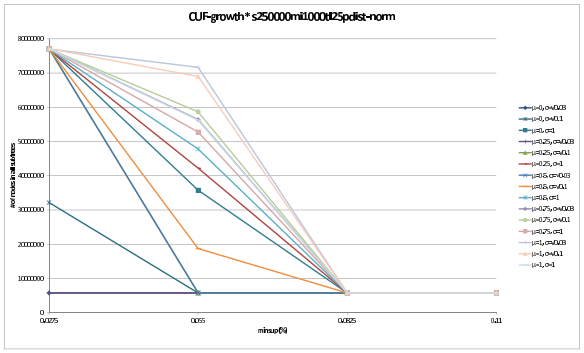


(e) 1,000,000 Transactions

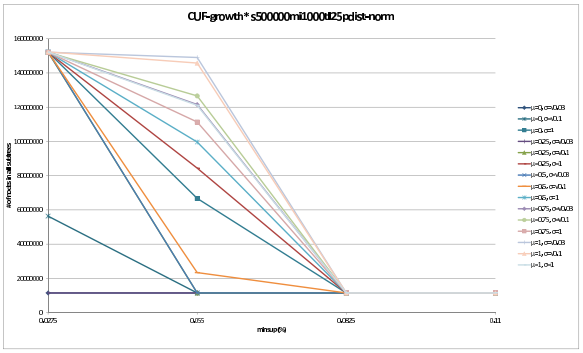
Figure B.48: Memory: Tube-P, $|\mathcal{I}| = 100$, $\overline{|t_j|} = 25$



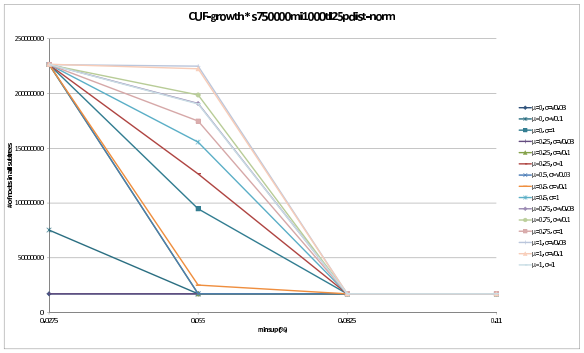
(a) 10,000 Transactions



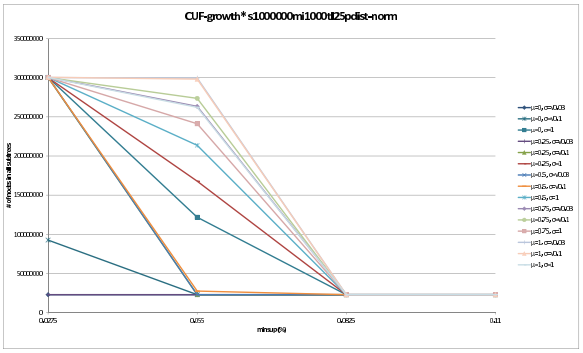
(b) 250,000 Transactions



(c) 500,000 Transactions

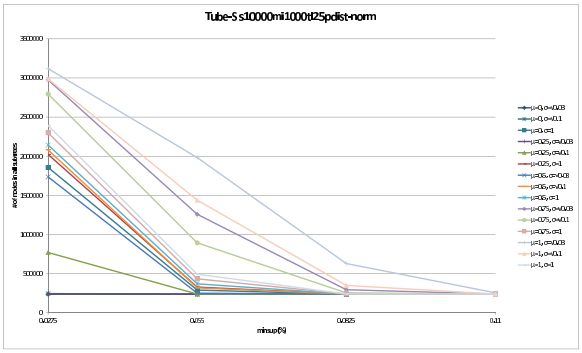


(d) 750,000 Transactions

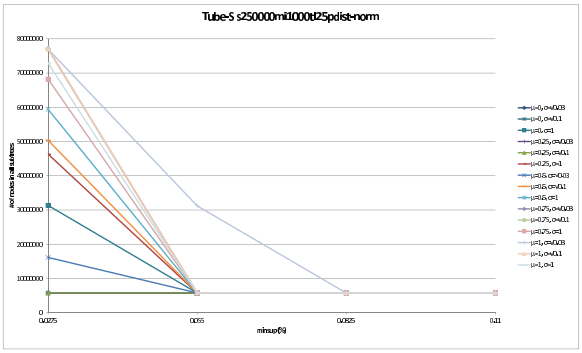


(e) 1,000,000 Transactions

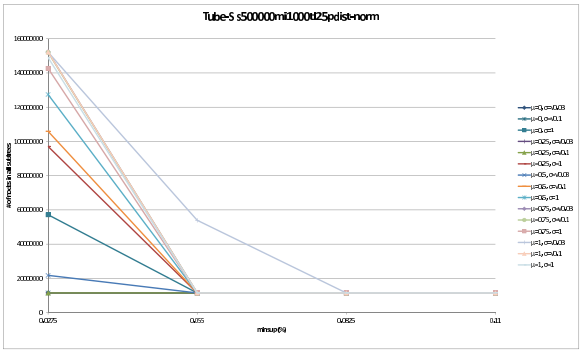
Figure B.49: Memory: CUF-growth*, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 25$



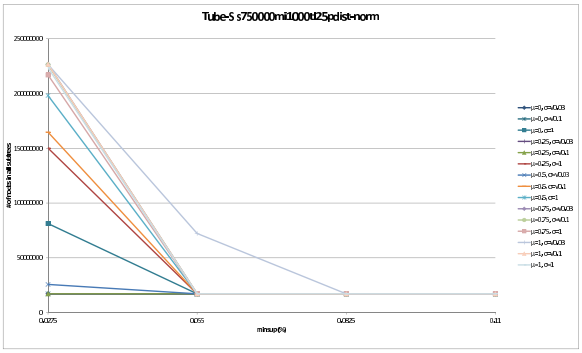
(a) 10,000 Transactions



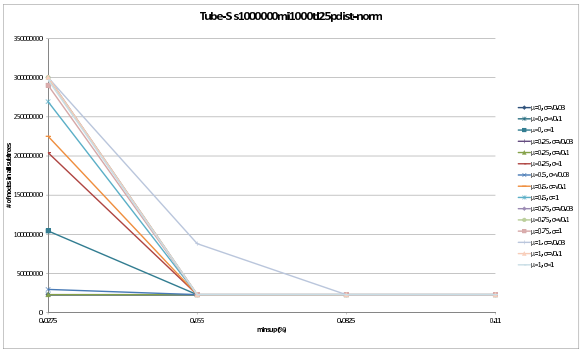
(b) 250,000 Transactions



(c) 500,000 Transactions

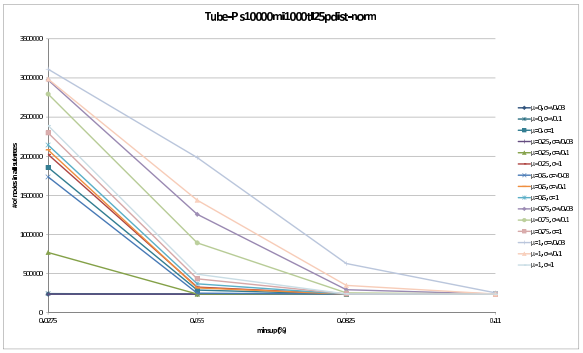


(d) 750,000 Transactions

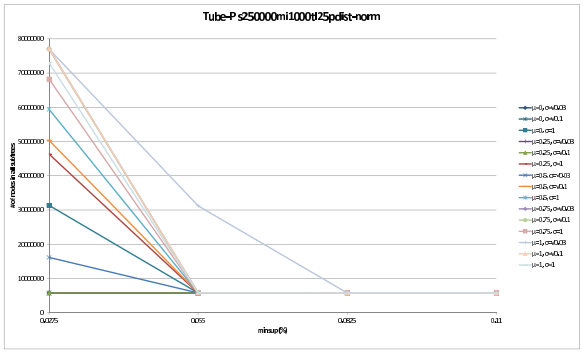


(e) 1,000,000 Transactions

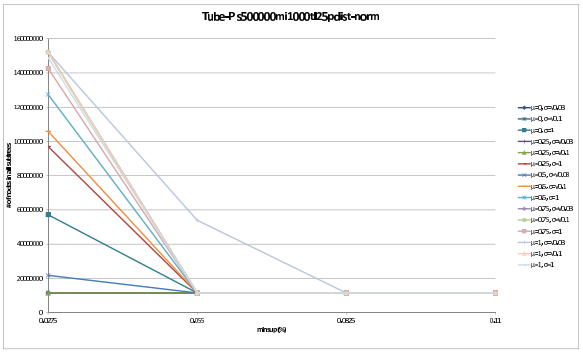
Figure B.50: Memory: Tube-S, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 25$



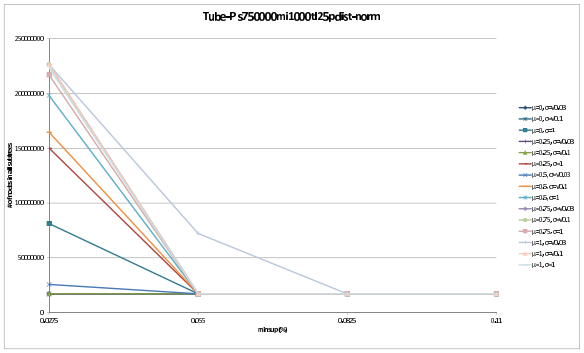
(a) 10,000 Transactions



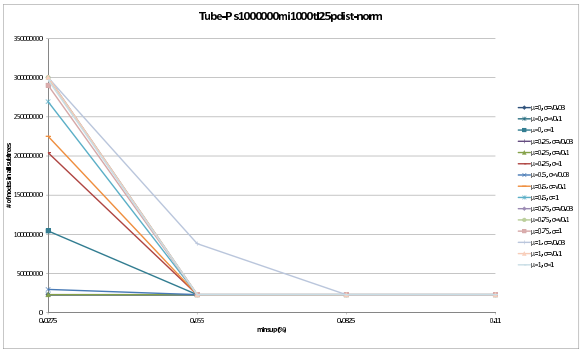
(b) 250,000 Transactions



(c) 500,000 Transactions

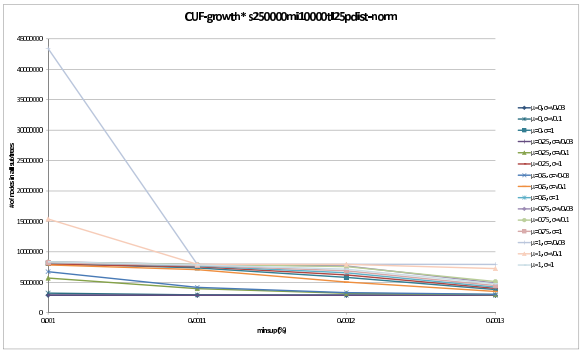


(d) 750,000 Transactions

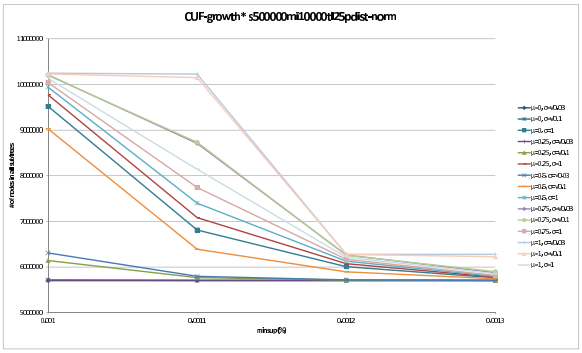


(e) 1,000,000 Transactions

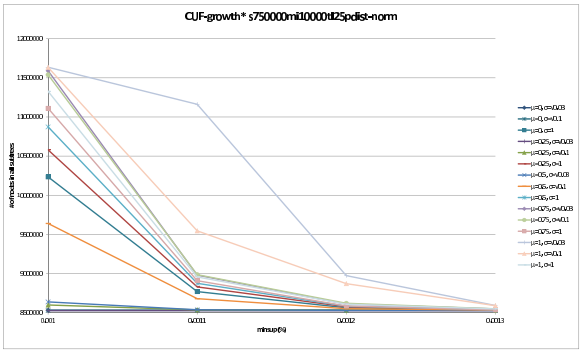
Figure B.51: Memory: Tube-P, $|\mathcal{I}| = 1000$, $\overline{|t_j|} = 25$



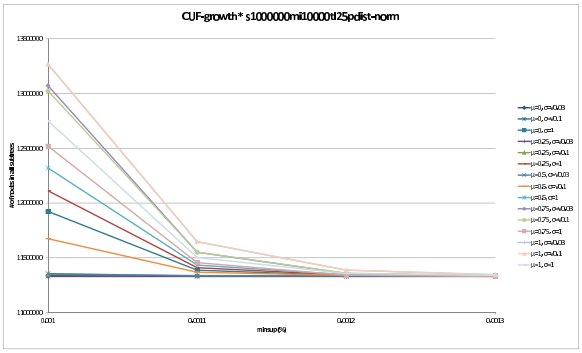
(a) 250,000 Transactions



(b) 500,000 Transactions

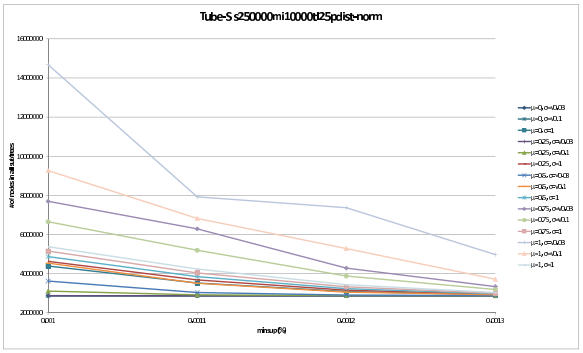


(c) 750,000 Transactions

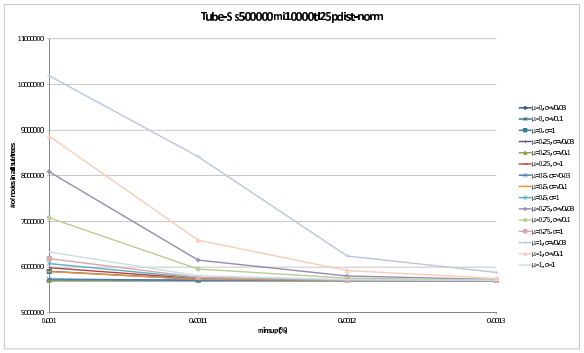


(d) 1,000,000 Transactions

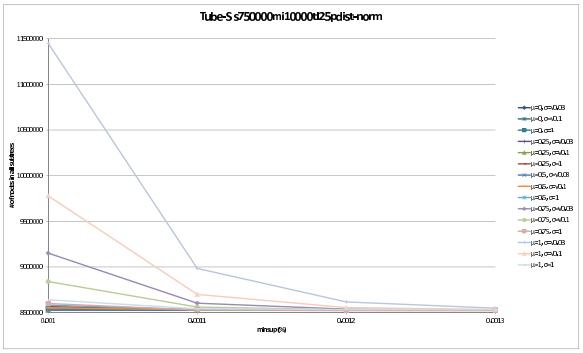
Figure B.52: Memory: CUF-growth*, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 25$



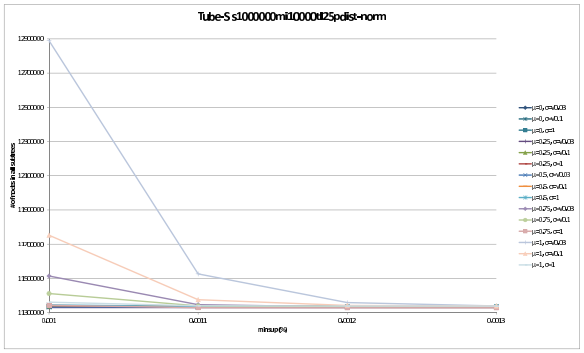
(a) 250,000 Transactions



(b) 500,000 Transactions

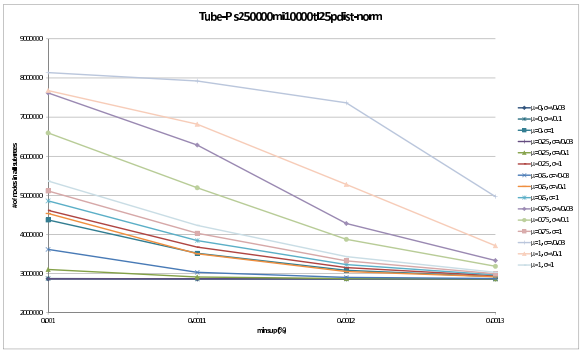


(c) 750,000 Transactions

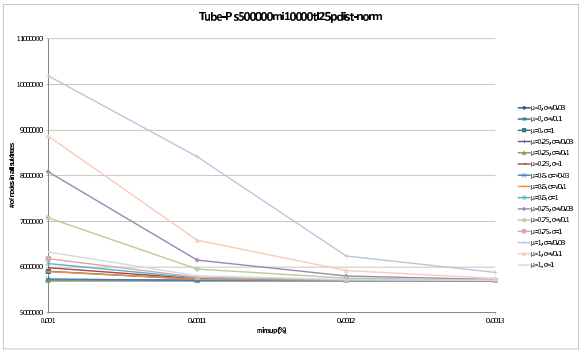


(d) 1,000,000 Transactions

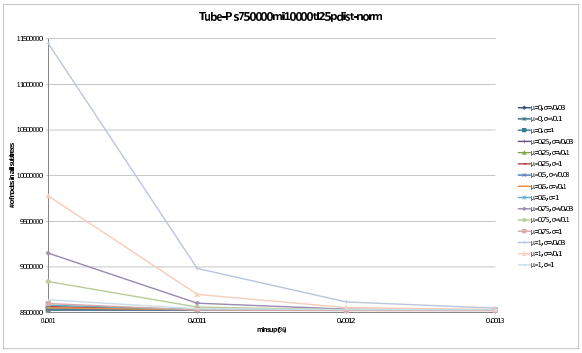
Figure B.53: Memory: Tube-S, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 25$



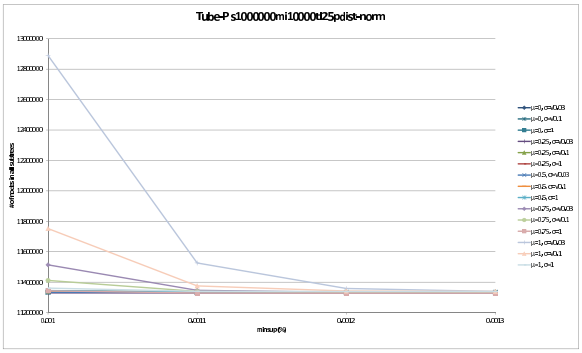
(a) 250,000 Transactions



(b) 500,000 Transactions



(c) 750,000 Transactions



(d) 1,000,000 Transactions

Figure B.54: Memory: Tube-P, $|\mathcal{I}| = 10000$, $\overline{|t_j|} = 25$

Bibliography

[1] Aggarwal, C. C., "Applications of frequent pattern mining," in *Frequent Pattern Mining*, Aggarwal, C. C. and Han, J., Eds. Springer International Publishing, 2014, pp. 443–467. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-07821-2_18

[2] Aggarwal, C. C., Li, Y., Wang, J., and Wang, J., "Frequent pattern mining with uncertain data," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09. New York, NY, USA: ACM, 2009, pp. 29–38. [Online]. Available: http://dx.doi.org/10.1145/1557019.1557030

[3] Aggarwal, C. C. and Yu, P. S., "A survey of uncertain data algorithms and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 5, pp. 609–623, May 2009. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2008.190

[4] Agrawal, R. and Srikant, R., "Fast algorithms for mining association rules in large databases," in *Proceedings of the VLDB 20th International Conference on Very Large Data Bases*, vol. 1215, 1994, pp. 487–499.

[5] Bernecker, T., Kriegel, H.-P., Renz, M., Verhein, F., and Züfle, A., "Probabilistic frequent itemset mining in uncertain databases," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09. New York, NY, USA: ACM, 2009, pp. 119–128. [Online]. Available: http://dx.doi.org/10.1145/1557019.1557039

[6] ——, "Probabilistic frequent pattern growth for itemset mining in uncertain databases," in *Scientific and Statistical Database Management*, ser. Lecture Notes in Computer Science, Ailamaki, A. and Bowers, S., Eds. Springer Berlin Heidelberg, 2012, vol. 7338, pp. 38–55. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31235-9_3

[7] Cheng, Y. and Church, G. M., "Biclustering of expression data," in *Intelligent Systems in Molecular Biology*, 2000, pp. 93–103. [Online]. Available: http://www.aaai.org/Papers/ISMB/2000/ISMB00-010.pdf

[8] Chui, C.-K. and Kao, B., "A decremental approach for mining frequent itemsets from uncertain data," in *Advances in Knowledge Discovery and Data Mining*, ser. Lecture Notes in Computer Science, Washio, T., Suzuki, E., Ting, K. M., and Inokuchi, A., Eds. Springer Berlin Heidelberg, 2008, vol. 5012, pp. 64–75. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-68125-0_8

[9] Chui, C.-K., Kao, B., and Hung, E., "Mining frequent itemsets from uncertain data," in *Advances in Knowledge Discovery and Data Mining*, ser. Lecture Notes in Computer Science, Zhou, Z.-H., Li, H., and Yang, Q., Eds.

Springer Berlin Heidelberg, 2007, vol. 4426, pp. 47–58. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-71701-0_8

[10] Cuzzocrea, A., Leung, C. K.-S., and MacKinnon, R. K., "Mining constrained frequent itemsets from distributed uncertain data," *Future Generation Computer Systems*, vol. 37, pp. 117–126, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.future.2013.10.026

[11] Do, T. D. T., Termier, A., Laurent, A., Negrevergne, B., Omidvar-Tehrani, B., and Amer-Yahia, S., "PGLCM: efficient parallel mining of closed frequent gradual itemsets," *Knowledge and Information Systems*, vol. 43, no. 3, pp. 497–527, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10115-014-0749-8

[12] Frawley, W., Piatetsky-Shapiro, G., and Matheus, C., "Knowledge discovery in databases: An overview," *AI Magazine*, vol. 13, no. 3, pp. 57–70, 1992. [Online]. Available: http://dx.doi.org/10.1609/aimag.v13i3.1011

[13] Han, J., Pei, J., and Yin, Y., "Mining frequent patterns without candidate generation," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '00. New York, NY, USA: ACM, 2000, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1145/342009.335372

[14] He, Z., Xu, X., Huang, Z. J., and Deng, S., "FP-outlier: Frequent pattern based outlier detection," *Computer Science and Information Systems*, vol. 2, no. 1, pp. 103–118, 2005. [Online]. Available: http://dx.doi.org/10.2298/CSIS0501103H

[15] Jabbour, S., Khiari, M., Sais, L., Salhi, Y., and Tabia, K., "Symmetry-based

pruning in itemset mining," in *Proceedings of the 2013 ICTAI IEEE International Conference on Tools with Artificial Intelligence*, Nov. 2013, pp. 483–490. [Online]. Available: http://dx.doi.org/10.1109/ICTAI.2013.78

[16] Jiang, F. and Leung, C. K.-S., "A business intelligence solution for frequent pattern mining on social networks," in *2014 IEEE International Conference on Data Mining Workshop (ICDMW)*, Dec. 2014, pp. 789–796. [Online]. Available: http://dx.doi.org/10.1109/ICDMW.2014.128

[17] Kamsu-Foguem, B., Rigal, F., and Mauget, F., "Mining association rules for the quality improvement of the production process," *Expert Systems with Applications*, vol. 40, no. 4, pp. 1034–1045, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2012.08.039

[18] Lee, G., Yun, U., and Ryu, K. H., "Sliding window based weighted maximal frequent pattern mining over data streams," *Expert Systems with Applications*, vol. 41, no. 2, pp. 694–708, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2013.07.094

[19] Leung, C. K.-S., "Uncertain frequent pattern mining," in *Frequent Pattern Mining*, Aggarwal, C. C. and Han, J., Eds. Springer International Publishing, 2014, pp. 339–367. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-07821-2_14

[20] Leung, C. K.-S. and Hayduk, Y., "Mining frequent patterns from uncertain data with mapreduce for big data analytics," in *Database Systems for Advanced Applications*, ser. Lecture Notes in Computer Science, Meng, W.,

Feng, L., Bressan, S., Winiwarter, W., and Song, W., Eds. Springer Berlin Heidelberg, Apr. 2013, vol. 7825, pp. 440–455. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37487-6_33

[21] Leung, C. K.-S. and MacKinnon, R. K., "BLIMP: A compact tree structure for uncertain frequent pattern mining," in *Data Warehousing and Knowledge Discovery*, ser. Lecture Notes in Computer Science, Bellatreche, L. and Mohania, M. K., Eds. Springer International Publishing, 2014, vol. 8646, pp. 115–123. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10160-6_11

[22] Leung, C. K.-S., MacKinnon, R. K., and Tanbeer, S. K., "Tightening upper bounds to the expected support for uncertain frequent pattern mining," *Procedia Computer Science*, vol. 35, pp. 328–337, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2014.08.113

[23] Leung, C. K.-S., Mateo, M. A. F., and Brajczuk, D. A., "A tree-based approach for frequent pattern mining from uncertain data," in *Advances in Knowledge Discovery and Data Mining*, ser. Lecture Notes in Computer Science, Washio, T., Suzuki, E., Ting, K., and Inokuchi, A., Eds. Springer Berlin Heidelberg, 2008, vol. 5012, pp. 653–661. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-68125-0_61

[24] Leung, C. K.-S. and Tanbeer, S. K., "Fast tree-based mining of frequent itemsets from uncertain data," in *Database Systems for Advanced Applications*, ser. Lecture Notes in Computer Science, goo Lee, S., Peng, Z., Zhou, X., Moon, Y.-S.,

Unland, R., and Yoo, J., Eds. Springer Berlin Heidelberg, 2012, vol. 7238, pp. 272–287. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-29038-1_21

[25] ——, "PUF-Tree: A compact tree structure for frequent pattern mining of uncertain data," in *Advances in Knowledge Discovery and Data Mining*, ser. Lecture Notes in Computer Science, Pei, J., Tseng, V. S., Cao, L., Motoda, H., and Xu, G., Eds. Springer Berlin Heidelberg, 2013, vol. 7818, pp. 13–25. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37453-1_2

[26] Li, W., Han, J., and Pei, J., "CMAR: Accurate and efficient classification based on multiple class-association rules," in *Proceedings of the 2001 ICDM IEEE International Conference on Data Mining*, 2001, pp. 369–376. [Online]. Available: http://dx.doi.org/10.1109/ICDM.2001.989541

[27] Lin, C.-W. and Hong, T.-P., "A new mining approach for uncertain databases using CUFP trees," *Expert Systems with Applications*, vol. 39, no. 4, pp. 4084–4093, Mar. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2011.09.087

[28] Liu, B., Hsu, W., and Ma, Y., "Integrating classification and association rule mining," in *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, 1998, pp. 80–86. [Online]. Available: http://www.aaai.org/Papers/KDD/1998/KDD98-012.pdf

[29] Liu, Y., Zhao, Y., Chen, L., Pei, J., and Han, J., "Mining frequent trajectory patterns for activity monitoring using radio frequency tag arrays," *IEEE*

Transactions on Parallel and Distributed Systems, vol. 23, no. 11, pp. 2138–2149, Nov. 2012. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2011.307

[30] Lu, E. H.-C., Lee, W.-C., and Tseng, V. S., "A framework for personal mobile commerce pattern mining and prediction," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 5, pp. 769–782, May 2012. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2011.65

[31] MacKinnon, R. K., Strauss, T. D., and Leung, C. K.-S., "DISC: Efficient uncertain frequent pattern mining with tightened upper bounds," in *2014 IEEE International Conference on Data Mining Workshop (ICDMW)*, Dec. 2014, pp. 1038–1045. [Online]. Available: http://dx.doi.org/10.1109/ICDMW.2014.129

[32] Pei, J., Han, J., Lu, H., Shojiro, N., Tang, S., and Yang, D., "H-mine: hyper-structure mining of frequent patterns in large databases," in *Proceedings of the 2001 ICDM IEEE International Conference on Data Mining*, 2001, pp. 441–448. [Online]. Available: http://dx.doi.org/10.1109/ICDM.2001.989550

[33] Tong, Y., Chen, L., Cheng, Y., and Yu, P. S., "Mining frequent itemsets over uncertain databases," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1650–1661, 2012.

[34] Tsai, M.-F., Tzeng, C.-W., Lin, Z.-L., and Chen, A. L. P., "Discovering leaders from social network by action cascade," *Social Network Analysis and Mining*, vol. 4, no. 1, 2014. [Online]. Available: http://dx.doi.org/10.1007/s13278-014-0165-9

[35] Wang, J. and Karypis, G., "HARMONY: Efficiently mining the best rules for classification," in *Proceedings of the 2005 Fifth SIAM International Conference on Data Mining*, 2005, pp. 205–216. [Online]. Available: http://dx.doi.org/10.1137/1.9781611972757.19

[36] Wang, K., Tang, L., Han, J., and Liu, J., "Top down FP-Growth for association rule mining," in *Advances in Knowledge Discovery and Data Mining*, ser. Lecture Notes in Computer Science, Chen, M.-S., Yu, P. S., and Liu, B., Eds. Springer Berlin Heidelberg, 2002, vol. 2336, pp. 334–340. [Online]. Available: http://dx.doi.org/10.1007/3-540-47887-6_34

[37] Wang, K., Xu, C., and Liu, B., "Clustering transactions using large items," in *Proceedings of the Eighth International Conference on Information and Knowledge Management*, ser. CIKM '99. New York, NY, USA: ACM, 1999, pp. 483–490. [Online]. Available: http://dx.doi.org/10.1145/319950.320054

[38] Xu, J., Li, N., Mao, X.-J., and Yang, Y.-B., "Efficient probabilistic frequent itemset mining in big sparse uncertain data," in *PRICAI 2014: Trends in Artificial Intelligence*, ser. Lecture Notes in Computer Science, Pham, D.-N. and Park, S.-B., Eds. Springer International Publishing, Dec. 2014, vol. 8862, pp. 235–247. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-13560-1_19

[39] Yang, Y. and Padmanabhan, B., "GHIC: a hierarchical pattern-based clustering algorithm for grouping web transactions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 9, pp. 1300–1304, Sept. 2005. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2005.145

[40] Zhang, Z., Teo, A., Ooi, B. C., and Tan, K.-L., "Mining deterministic biclusters in gene expression data," in *Proceedings of the 2004 BIBE IEEE Symposium on Bioinformatics and Bioengineering*, May 2004, pp. 283–290. [Online]. Available: http://dx.doi.org/10.1109/BIBE.2004.1317355

[41] Zhou, L., Zhong, Z., Chang, J., Li, J., Huang, J., and Feng, S., "Balanced parallel fp-growth with mapreduce," in *2010 IEEE Youth Conference on Information Computing and Telecommunications (YC-ICT)*, Nov. 2010, pp. 243–246. [Online]. Available: http://dx.doi.org/10.1109/YCICT.2010.5713090