Fast and Memory-Efficient Significant Pattern Mining via Permutation Testing

Felipe Llinares-López
D-BSSE, ETH Zürich
felipe.llinares@bsse.ethz.ch

Mahito Sugiyama
ISIR, Osaka University
JST, PRESTO
mahito@ar.sanken.osaka-u.ac.jp

Laetitia Papaxanthos
D-BSSE, ETH Zürich
laetitia.papaxanthos@bsse.ethz.ch

Karsten M. Borgwardt
D-BSSE, ETH Zürich
karsten.borgwardt@bsse.ethz.ch

ABSTRACT

We present a novel algorithm for significant pattern mining, *Westfall-Young light*. The target patterns are statistically significantly enriched in one of two classes of objects. Our method corrects for multiple hypothesis testing and correlations between patterns via the *Westfall-Young permutation procedure*, which empirically estimates the null distribution of pattern frequencies in each class via permutations.

In our experiments, *Westfall-Young light* dramatically outperforms the current state-of-the-art approach, both in terms of runtime and memory efficiency on popular real-world benchmark datasets for pattern mining. The key to this efficiency is that, unlike all existing methods, our algorithm does not need to solve the underlying frequent pattern mining problem anew for each permutation and does not need to store the occurrence list of all frequent patterns. *Westfall-Young light* opens the door to significant pattern mining on large datasets that previously involved prohibitive runtime or memory costs.

Our code is available from `http://www.bsse.ethz.ch/mlcb/research/machine-learning/wylight.html`

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data mining

Keywords

Significant pattern mining; p-value; Multiple hypothesis testing; Westfall-Young permutation

1. INTRODUCTION

Frequent pattern mining is a fundamental problem in data mining, in particular in association rule mining [2]. In its most popular instance, frequent itemset mining, a user is given a database of transactions, each of which includes a set of items. In its original application [1], each transaction represents the set of items purchased by a customer in a supermarket. A *frequent itemset* is then a set of items that co-occur in different transactions more often than a predefined frequency threshold. Another prominent instance is frequent subgraph mining in graph databases [19].

Significant pattern mining (or discriminative pattern mining) extends the classic problem of frequent pattern mining to two classes of transactions, such as cases and controls in clinical studies [8]. The interest is in finding those sets of items that occur statistically significantly more often in one class than in the other.

This problem is fundamentally important in many applications, ranging from marketing to health care. For instance, while frequent itemset mining tries to find products that are co-bought by all customers, significant pattern mining tries to detect products that are co-bought significantly more often by elderly customers than by younger customers. Similarly, frequent itemset mining may screen electronic health records for combinations of drugs that have frequently been administered to the same patients. In significant pattern mining, however, we may be interested in combinations of drugs that have been administered to patients who show severe side effects significantly more often than to patients who show no side effects [14].

A critical problem in significant pattern mining is the multiple testing problem, which arises due to the fact that the number of patterns tested for association with class membership is often in the millions or billions. If not properly corrected for, significant pattern mining will retrieve a huge number of false positives; that is, patterns deemed to be significantly associated with class membership by mistake. In most applications domains of data mining, from the natural sciences to the social sciences, these patterns are often explored further in a time-consuming and cost-consuming experimental validation. Therefore, a high number of a false positives is an enormous waste of resources in these fields.

In light of this multiple testing problem, current approaches to significant pattern mining face at least one of the following problems:

1. Many methods for finding patterns that are *associated* with class membership do not correct for multiple testing at all ([4, 10, 20, 35], see [36] for a comprehensive survey), even those that report a measure of statistical significance for this association [3, 5, 13, 32].

KDD'15, August 10-13, 2015, Sydney, NSW, Australia.
ACM 978-1-4503-3664-2/15/08.
DOI: http://dx.doi.org/10.1145/2783258.2783363 .

2. Other methods have corrected for multiple testing using a naive Bonferroni correction, which leads to algorithms that are unable to retain statistical power if the whole search space is to be explored [27, 29, 28]. Therefore, those methods require that arbitrary limits be set on the maximum pattern size in order to keep the number of patterns being tested for association small.
3. Recent approaches to correct for multiple testing in frequent itemset mining [24, 18, 22], based on the seminal work by Tarone [23], can work with arbitrary pattern sizes, but do not consider the dependence structure between patterns, which is very important in pattern mining due to the subset/superset relationship between patterns. Ignoring this dependence leads to a loss of statistical power; that is, the ability to detect truly associated patterns.
4. The single approach to pattern mining that corrects for multiple testing and takes pattern dependence into account [25] is expensive, either in terms of memory or runtime, and is limited in its applicability to larger datasets.

Our goal in this article is to present an approach to significant pattern mining that corrects for multiple testing, takes the dependence between patterns into account, and is efficient in terms of memory and runtime.

Below we will describe the multiple hypothesis testing problem in detail and discuss the current state-of-the-art in significant pattern mining in Section 2. Section 3 introduces our novel algorithm and the improvements it achieves. The experiments in Section 4 show that our method outperforms existing techniques in two instances of pattern mining (itemset mining and subgraph mining) across several datasets. Section 5 summarizes our key findings.

2. BACKGROUND

2.1 Problem Statement

Let $\mathcal{T} = \{t_1, t_2, t_3, ..., t_n\}$ be a set of transactions defined in a universe of m *items*. Each transaction (or sample) t_i can be described by m binary features that indicate whether or not the corresponding items are present in the transaction. Thus, the transaction database $\mathcal{T}$ can be encoded as a $n \times m$ binary matrix $\mathbf{T}$. Furthermore, each transaction is also tagged with a binary-class label attribute $C \in \{c_0, c_1\}$. In the following, we call *pattern* a set of items $\mathcal{S} = \{l_1, l_2, \ldots, l_k\}$, $l_j \in \{1, \ldots, m\}$ of size k, and define the binary random variable $G(\mathcal{S}, t_i) = \mathbf{T}_{i,l_1} \wedge \mathbf{T}_{i,l_2} \wedge \ldots \wedge \mathbf{T}_{i,l_k}$ such that $G(\mathcal{S}, t_i) = 1$ if the pattern $\mathcal{S}$ is contained in transaction t_i and $G(\mathcal{S}, t_i) = 0$ otherwise. In other words, $G(\mathcal{S}, t_i)$ is simply an indicator binary variable that takes value 1 if pattern $\mathcal{S}$ is included in transaction t_i and value 0 otherwise.

For each pattern $\mathcal{S}$, we evaluate $G(\mathcal{S}, t_i)$ for each transaction t_i, $i = 1, \ldots, n$ and build the following 2×2 contingency table:

Variables	$G(\mathcal{S}, t) = 1$	$G(\mathcal{S}, t) = 0$	Row totals
$C = c_1$	$a_\mathcal{S}$	$n_1 - a_\mathcal{S}$	n_1
$C = c_0$	$x_\mathcal{S} - a_\mathcal{S}$	$n - n_1 + a_\mathcal{S} - x_\mathcal{S}$	$n - n_1$
Col totals	$x_\mathcal{S}$	$n - x_\mathcal{S}$	n

n denotes the total number of transactions, n_1 the number of transactions with class label $C = c_1$; $x_\mathcal{S}$ is the number of transactions that include pattern $\mathcal{S}$, i.e. the *support* of pattern $\mathcal{S}$ and, finally, $a_\mathcal{S}$ is the number of transactions in class c_1 that include pattern $\mathcal{S}$; that is, the support of pattern $\mathcal{S}$ among transactions of class c_1.

Intuitively, our objective is to decide if the observed value of $a_\mathcal{S}$ indicates that pattern $\mathcal{S}$ is over-represented in one of the two classes, which would the occurrence of the pattern $\mathcal{S}$ within a transaction and the class labels dependent random variables. In the next section, we describe how to carry out that assessment in a statistically rigorous manner.

2.2 Statistical Association Testing

In *statistical association testing*, the goal is to determine whether two random variables, such as $G(\mathcal{S}, t)$ and C in our setting, are statistically dependent or *associated*. Statistical association testing is conservative, in that the *null hypothesis* of no association or independence between the two random variables is always assumed. Only when the data provides very strong evidence against that assumption will the null hypothesis be *rejected* and the random variables declared to be associated.

When the two random variables to be tested for association are binary, as is the case with $G(\mathcal{S}, t)$ and C, *Fisher's exact test* [11] is one of the most popular approaches. Fisher's exact test considers the margins $(x_\mathcal{S}, n_1, n)$ of the 2×2 contingency table to be fixed. It can be shown that, under the null hypothesis of independence between $G(\mathcal{S}, t)$ and C, the cell count $a_\mathcal{S}$ follows a hypergeometric distribution $P(. \mid x_\mathcal{S}, n_1, n)$:

$$P(a_\mathcal{S} = a \mid x_\mathcal{S}, n_1, n) = \binom{n_1}{a}\binom{n - n_1}{x_\mathcal{S} - a} \Big/ \binom{n}{x_\mathcal{S}}$$

In order to quantify the evidence provided by the data against the null hypothesis of independence, statistical association testing uses the concept of *p-values*. A p-value is defined as the probability of measuring an association that is at least as extreme as the one observed in the data when the null hypothesis of independence holds. Intuitively, the smaller the p-value is, the less plausible the data appears to be with the null hypothesis of independence.

In the context of Fisher's exact test, extreme values of association refer directly to the likelihood of observing a given $a_\mathcal{S} = u$. In other words, all events more extreme than observing $a_\mathcal{S} = u$ are observing cell counts $a_\mathcal{S} = k$ that are less likely to occur under the hypergeometric distribution $P(. \mid x_\mathcal{S}, n_1, n)$. Mathematically, the p-value is given by:

$$p_\mathcal{S}(u) = \sum_{k \mid P(k \mid x_\mathcal{S}, n_1, n) \leq P(u \mid x_\mathcal{S}, n_1, n)} P(k \mid x_\mathcal{S}, n_1, n)$$

Thus, $p_\mathcal{S}(u)$ is the cumulative probability of all possible values of the cell count $a_\mathcal{S}$ that are more unlikely than $a_\mathcal{S} = u$.

In statistical association testing, an association between the two random variables is deemed significant when the p-value is smaller than the *significance threshold* α, which must be fixed a priori. That is, a pattern will be declared significant if $p_\mathcal{S}(u) \leq \alpha$.

It can be shown that the probability of finding a false association - that is, producing a *false positive* - will be bounded above by α. Thus, if α is set to a low value, the discovered significant patterns are more likely to be true associations. On the other hand, using low values of α as significant thresholds causes many true associations to be missed as well. In other words, there is a trade-off between Type I error (probability of discovering a false association) and Type II error (probability of missing a true association)

that is controlled by α. The choice $\alpha = 0.05$ is by far the most common, although it generally depends on the particular application.

2.3 Multiple Hypothesis Testing

If a large number d of statistical association tests at level α are performed as described in the previous section, the expected number of false positives will be approximately αd, which will lead to a large number of false discoveries. Therefore, rather than controlling the Type I error for each pattern individually, it is advisable to control the FWER (family-wise error rate), which is defined as the probability of producing at least one false positive, FWER $= P(\text{FP} > 0)$, with FP denoting the number of false positives. To guarantee that FWER $\leq \alpha$, one can apply a *multiple testing correction* by changing the rule $p_{\mathcal{S}}(u) \leq \alpha$ by $p_{\mathcal{S}}(u) \leq \delta$, where δ is said to be a *corrected* significance threshold. Ideally, the problem to be solved is:

$$\delta^* = \max\{\delta \,|\, \text{FWER}(\delta) \leq \alpha\}$$

Note that it is desirable to maximize δ because larger values of the corrected significance threshold imply higher power to discover truly associated patterns, as discussed before. However, it is commonly not possible to evaluate FWER(δ) in closed form.

Thus, the most popular way to choose δ, the *Bonferroni correction* [7], uses a much simpler, suboptimal scheme that substitutes FWER(δ) by a loose upper bound FWER(δ) $\leq \delta d$, leading to $\delta^*_{\text{bon}} = \alpha/d$. Note, however, that: (1) the Bonferroni approximation FWER(δ) $\leq \delta d$ assumes that all patterns are mutually independent, leading to an overly conservative procedure; and (2) the Bonferroni correction is ineffective in our settings, as unless an arbitrary, restrictive limit is imposed on the maximum pattern size, d can be an extremely large number and δ^*_{bon} effectively 0.

Permutation testing

Alternatively, one of the most popular approaches for estimating FWER(δ) is a permutation-based resampling scheme proposed by Westfall and Young [30]. The idea is as follows: by randomly permuting the class labels of the transactions, one can generate a new, resampled transaction database for which no pattern $\mathcal{S}$ in the dataset is truly associated with the permuted class labels. Thus, in this new dataset, one can check if false positives have occurred by computing $p_{\min} = \min_{\mathcal{S}} p_{\mathcal{S}}$ and checking whether $p_{\min} \leq \delta$, in which case at least one false positive occurred; that is, FP > 0. If the whole procedure is repeated a sufficient number j_{P} of times (that is, $j_{\text{P}} = 10^3$ or 10^4), yielding a set $\{\, p^{(j)}_{\min} \,\}_{j=1}^{j_{\text{P}}}$, a good estimator of FWER(δ) $= P(\text{FP} > 0)$ can be found as:

$$\text{FWER}(\delta) = \frac{1}{j_{\text{P}}} \sum_{j=1}^{j_{\text{P}}} \mathbf{1}\left[p^{(j)}_{\min} \leq \delta\right]$$

where $\mathbf{1}[\bullet]$ is an indicator function that takes value 1 if its argument is true and 0 otherwise. It is then possible to accurately estimate δ^* as the α-quantile of $\{\, p^{(j)}_{\min} \,\}_{j=1}^{j_{\text{P}}}$; that is, choose δ^* such that a proportion α of the values in $\{\, p^{(j)}_{\min} \,\}_{j=1}^{j_{\text{P}}}$ are below δ^* and the remaining values are above it.

Westfall-Young permutation testing compensates for the dependence structure between patterns by directly estimating the joint null distribution of all test statistics, leading to largely increased statistical power with respect to a Bonferroni correction. The standard proof of FWER control for Westfall-Young permutation testing relies on a sufficient, yet not necessary, technical condition that is often hard to verify in practice; the *subset pivotality condition*. Nonetheless, permutation-based testing approaches are extensively used in practice and have produced numerous meaningful discoveries in such fields as computational biology [34, 33, 17].

However, Westfall-Young permutation testing can be extremely computationally demanding. Generating a single sample $p^{(j)}_{\min}$ naively requires the enumeration of all patterns and the computation of all their corresponding p-values, which is infeasible except in toy problems. More importantly, in order to obtain a reliable estimate of FWER(δ), a number of permutations in the order of $j_{\text{P}} = 10^3$ or $j_{\text{P}} = 10^4$ are required. Thus, a priori, applying permutation-based testing to significant pattern mining is a challenging problem. Circumventing this computational limitation is our main goal in this article.

2.4 The FastWY Algorithm

To the best of our knowledge, the work of Terada et al. in [25] has been the only previous attempt to make permutation testing in significant pattern mining tractable. Those authors proposed FastWY, an algorithm that uses inherent properties of discrete test statistics and succeeds in reducing the computational burden that the Westfall-Young permutation-based procedure entails. We introduce FastWY below, since we share the same problem setting and the key concept of the *minimum attainable p-value*. FastWY will be used as a baseline in our experiments.

The p-value for a given 2×2 contingency table in Fisher's exact test, or any other test statistic that assumes the margins $x_{\mathcal{S}}$, n_1 and n fixed, is a function only of the cell count $a_{\mathcal{S}}$. Since 2×2 contingency tables are discrete objects, $a_{\mathcal{S}}$ can only take a finite number of values; that is, $a_s \in [\![a_{\mathcal{S},min}, a_{\mathcal{S},max}]\!]$ with $a_{\mathcal{S},min} = \max(0, x_{\mathcal{S}} - (n - n_1))$ and $a_{\mathcal{S},max} = \min(x_{\mathcal{S}}, n_1)$. Thus, there exists a *minimum attainable p-value* $\Psi(x_{\mathcal{S}})$[1] strictly greater than 0:

$$\Psi(x_{\mathcal{S}}) = \min\{\, p_{\mathcal{S}}(u) \mid a_{\mathcal{S},\min} \leq u \leq a_{\mathcal{S},\max} \,\}$$

For Fisher's exact test, as the p-value is a sum of positive terms, the minimum attainable p-value $\Psi(x_{\mathcal{S}})$ is reached when $a_{\mathcal{S}} = a_{\mathcal{S},min}$ or $a_{\mathcal{S}} = a_{\mathcal{S},max}$ and can be computed as a simple function of the pattern support $x_{\mathcal{S}}$. Related to the minimum attainable p-value, we also introduce the *set of testable patterns at significance level* δ, $\mathcal{I}_T(\delta) = \{\, \mathcal{S} \mid \Psi(x_{\mathcal{S}}) \leq \delta \,\}$. The word *testable* refers to the fact that, by definition of $\Psi(x_{\mathcal{S}})$, it is impossible for patterns not in $\mathcal{I}_T(\delta)$ to be significant at level δ.

In [24], Terada et al. introduced a monotonically decreasing lower bound $\hat{\Psi}(x_{\mathcal{S}})$ on the true minimum attainable p-value $\Psi(x_{\mathcal{S}})$:

$$\hat{\Psi}(x_{\mathcal{S}}) = \begin{cases} \Psi(x_{\mathcal{S}}) & 0 \leq x_S \leq n_1, \\ 1/\binom{n}{n_1} & n_1 < x_{\mathcal{S}} \leq n \end{cases}$$

and define $\hat{\mathcal{I}}_T(\delta) = \{\, \mathcal{S} \,|\, \hat{\Psi}(x_{\mathcal{S}}) \leq \delta \,\}$, which always satisfies $\mathcal{I}_T(\delta) \subset \hat{\mathcal{I}}_T(\delta)$. While this introduces some untestable patterns in the surrogate set $\hat{\mathcal{I}}_T(\delta)$, one can rewrite $\hat{\mathcal{I}}_T(\delta) = \{\, \mathcal{S} \,|\, x_{\mathcal{S}} \geq \sigma(\delta) \,\}$ with $\sigma(\delta) = \hat{\Psi}^{-1}(\delta)$ and $\hat{\Psi}^{-1}(\delta)$ well-defined due to monotonicity. This is an important observation, as it links retrieval of the sets $\hat{\mathcal{I}}_T(\delta)$ to an instance

[1] $\Psi(x_{\mathcal{S}})$ also depends on the margins n_1 and n but, since those are the same for all patterns we drop them to simplify the notation.

of *frequent pattern mining*, leading to a tractable scheme to enumerate testable patterns.

FastWY also exploits this concept. It is based on a decremental search scheme, which starts with the support $\sigma = n_1$. For each σ, a frequent pattern miner is first used as a black box to retrieve the set $\hat{\mathcal{I}}_T(\sigma)$. The p-values $p_{\mathcal{S}}$ are then computed for all $\mathcal{S} \in \hat{\mathcal{I}}_T(\sigma)$ and $p'_{\min} = \min\{\, p_{\mathcal{S}} \mid \mathcal{S} \in \hat{\mathcal{I}}_T(\sigma) \,\}$ is evaluated. If $p'_{\min} \leq \hat{\Psi}(\sigma)$, no other pattern can make $p'_{\min}$ smaller and, therefore, $p'_{\min} = p_{\min}$. Otherwise if $p'_{\min} > \hat{\Psi}(\sigma)$, σ is decreased by one and the whole procedure is repeated until the condition $p'_{\min} \leq \hat{\Psi}(\sigma)$ is satisfied.

If j_{p} permutations are needed to empirically estimate the FWER, the procedure must be to repeated j_{p} times. This includes the sequence of frequent mining problems needed to retrieve the sets $\hat{\mathcal{I}}_T(\sigma)$ for each support value σ used throughout the decremental search. Given the usual range of values for j_{p}, such an approach is as infeasible in practice as the original brute force approach.

Inspection of the code, which the authors kindly shared on their website, reveals that the actual implementation of the algorithm is different from the description in [25]. Indeed, to avoid repeating the whole frequent pattern mining process j_{p} times, the authors resort to storing in memory the realizations of the variables $G(\mathcal{S}, t_i)$ for all $\mathcal{S} \in \hat{\mathcal{I}}_T(\sigma), i = 1, \ldots, n$ and every value of σ explored during the decremental search. This decision corresponds to a drastic trade-off between runtime and memory usage, leading to severe scalability limitations when the method is applied to even mid-sized datasets, but it is an effective way to have acceptable runtime for small-sized problems like the ones considered in [25].

Moreover, the algorithm requires computing all $\{\, p^{(j)}_{\min} \,\}_{j=1}^{j_{\mathrm{p}}}$ exactly, even if only the $\lceil \alpha j_{\mathrm{p}} \rceil$ smallest values in that set are actually involved in the computation of δ^* as the α-quantile of the set. This is problematic since $\hat{\Psi}^{-1}(\delta)$ is a monotonically decreasing function, which means that the larger $p^{(j)}_{\min}$ is, the smaller the minimum support for frequent pattern mining will be, requiring more enumeration runtime. Furthermore, if j_{p} is large, there is a high probability that some of the $p^{(j)}_{\min}$ will be rather large, creating a bottleneck in the algorithm. Nonetheless, a priori, it seems unclear how the $\lceil \alpha j_{\mathrm{p}} \rceil$ smallest values in $\{\, p^{(j)}_{\min} \,\}_{j=1}^{j_{\mathrm{p}}}$ could be identified without first evaluating them all.

In the next section, we propose a novel algorithm, Westfall-Young light, which overcomes all the scalability limitations of FastWY, leading to a large-scale, permutation-based significant pattern miner with FWER control.

3. WESTFALL-YOUNG LIGHT

In this section we present our contribution, the Westfall-Young light algorithm. Subsection 3.1 describes the method, starting from its pseudocode and then explaining the different steps in detail. Subsection 3.2 discusses the theoretical foundations of Westfall-Young light, proving that it provides exactly the same solution as Westfall-Young permutation-testing. Finally, in Subsection 3.3 we analyze all the improvements that Westfall-Young light provides over the current state-of-the-art method, FastWY.

3.1 The Algorithm: Westfall-Young Light

The pseudocode of Westfall-Young shown Algorithm 1 is composed of two parts: (1) the initialization part, function `Westfall-Young Light`; and (2) the core function `ProcessNext`, which processes each enumerated pattern and continues the enumeration recursively in a depth-first manner.

Algorithm 1 Westfall-Young light

```
1: Input: Transaction database T, class labels c, number of
   permutations j_p, and target FWER α
2: Output: Corrected significance threshold δ*
3: function Westfall-Young Light(α, j_p, T, c)
4:     for j = 1, ..., j_p do
5:         c^(j) ← randperm(c)
6:         p^(j)_min ← 1
7:     end for
8:     σ ← 1, δ ← Ψ̂(σ)
9:     ProcessNext(root, n)
10:    Return α-quantile of {p^(j)_min}_{j=1}^{j_p}
11: end function
12: function ProcessNext(S, x_S)
13:    Compute p-values p_S(u) for all u ∈ [a_{S,min}, a_{S,max}]
14:    for j = 1, ..., j_p do
15:        Compute a^(j)_S
16:        p^(j)_min ← min{ p^(j)_min, p_S(a^(j)_S) }
17:    end for
18:    FWER(δ) ← (1/j_p) Σ_{j=1}^{j_p} 1[p^(j)_min ≤ δ]
19:    while FWER(δ) > α do
20:        σ ← σ + 1, δ ← Ψ̂(σ)
21:        FWER(δ) ← (1/j_p) Σ_{j=1}^{j_p} 1[p^(j)_min ≤ δ]
22:    end while
23:    for S' ∈ Children(S) do
24:        Compute x_{S'}
25:        if x_{S'} ≥ σ then
26:            ProcessNext(S', x_{S'})
27:        end if
28:    end for
29: end function
```

Westfall-Young Light function

First, in Lines 4-7, we precompute all j_{p} permuted class label vectors $\mathbf{c}$, which can be stored as a binary matrix of size $n \times j_{\mathrm{p}}$. The set of minimum p-values for each of the j_{p} permutations, $\{\, p^{(j)}_{\min} \,\}_{j=1}^{j_{\mathrm{p}}}$, is initialized to 1; that is, the maximum value a p-value can take. Next, Lines 8 and 9 initialize the minimum support σ to 1 and compute the corresponding corrected significance threshold δ as the minimum attainable p-value for patterns with support σ. Note that the choice $\sigma = 0$ is trivial, as the corresponding δ would be $\delta = 1$, deeming all patterns both testable and significant.

After initialization, we must start the pattern enumeration procedure. Patterns are enumerated as nodes of a tree such that children $\mathcal{S}'$ of a pattern $\mathcal{S}$ have supports $x_{\mathcal{S}'} \leq x_{\mathcal{S}}$. This is a common assumption for classical pattern mining problems such as itemset, subgraph, or string mining [24, 22, 17]. Line 12 begins the enumeration process at the root of this tree by calling the `ProcessNext` function, which will continue enumerating patterns by exploring the tree with a depth-first strategy. Note that if the pattern tree has no clear root, one can always define a dummy root as an empty itemset without loss of generality.

ProcessNext function

The `ProcessNext` function processes every enumerated pattern $\mathcal{S}$, one at a time.

First, in Line 13, we precompute all possible p-values for a hypergeometric random variable with parameters $x_{\mathcal{S}}$, n_1 and n. This precomputation technique is one runtime improve-

ment in our algorithm. It is a consequence of the fact that, for fixed $x_{\mathcal{S}}$, n_1 and n, the computational complexity of evaluating Fisher's exact test p-values $p_{\mathcal{S}}(a_{\mathcal{S}})$ for a single value of $a_{\mathcal{S}}$ or for all $a_{\mathcal{S}} \in [a_{\mathcal{S},\min}, a_{\mathcal{S},\max}]$ is the same and equal to $O(\min\{x_{\mathcal{S}}, n_1\})$. This property can be readily checked from the definition of Fisher's exact test, since the main computational burden is evaluating the probability mass of the hypergeometric random variable with parameters $x_{\mathcal{S}}$, n_1 and n, a computation that can be shared across all j_{p} permutations. This reduces the computational complexity of this step from $O(j_{\mathrm{p}} \min\{x_{\mathcal{S}}, n_1\})$ to $O(\min\{x_{\mathcal{S}}, n_1\})$, with $j_{\mathrm{p}} \approx 10^4$. Thus, this novel "trick" makes the time complexity of evaluating Fisher's exact test negligible. Such an optimization is only possible if all j_{p} permutations are processed at the same time for each pattern $\mathcal{S}$; therefore, it is not feasible with the decremental scheme of FastWY.

In Lines 14-17 we compute the cell counts $a^j_{\mathcal{S}}$ for all permutations $j = 1, \ldots, j_{\mathrm{p}}$ and fetch the corresponding p-values $p^j_{\mathcal{S}} = p_{\mathcal{S}}(a^{(j)}_{\mathcal{S}})$, updating $p^{(j)}_{\min}$ if $p^j_{\mathcal{S}} < p^{(j)}_{\min}$ if needed. We then also update the current estimate of the FWER at level δ in Line 18. Next, between Lines 19-22, we check if the current estimate of the FWER is too large; that is, above the target FWER α. If it is, we must decrease the current threshold δ by increasing the minimum support σ until the empirical FWER is again below α. Note that every time σ is increased, δ decreases, which means that the empirical estimate of the FWER must be updated.

Finally, between Lines 23-28, the pattern enumeration process continues; the pattern tree is explored in a depth-first manner along every child of pattern $\mathcal{S}$ that is frequent at the current support σ.

This combination of frequent pattern enumeration and adaptive threshold adjustment continues until all patterns for a certain minimum support σ_f have been enumerated. That is, σ_f is the minimum support when Westfall-Young light finishes. Then, the original call to `ProcessNext` in Line 9 is completed and the algorithm terminates by returning the solution δ^* as the α-quantile of the set $\{\, p^{(j)}_{\min} \,\}_{j=1}^{j_{\mathrm{p}}}$.

3.2 Correctness of Westfall-Young Light

In this subsection, we will prove that Westfall-Young light correctly obtains the optimal δ^* based on the Westfall-Young FWER estimator.

THEOREM 1. (Correctness of Westfall-Young Light) *The Westfall-Young light algorithm returns the exact solution to* $\delta^* = \max\{\delta \,|\, \mathrm{FWER}(\delta) \leq \alpha\}$, *where* $\mathrm{FWER}(\delta)$ *is the empirical Westfall-Young FWER estimator* $\mathrm{FWER}(\delta) = \frac{1}{j_{\mathrm{p}}} \sum_{j=1}^{j_{\mathrm{p}}} \mathbf{1}\left[p^{(j)}_{\min} \leq \delta\right]$ *described in Section 2.3.*

The proof of Theorem 1 is based on exploiting the following properties of the FWER:

PROPOSITION 1. *For a fixed δ, processing a new pattern $\mathcal{S}$ can never make the empirical FWER estimate decrease.*

PROOF. Let $\mathcal{M}$ be the current set of patterns and $\mathcal{S}$ be a new pattern. We have $\min\{\, p_{\mathcal{R}} \,|\, \mathcal{R} \in \mathcal{M} \cup \{\mathcal{S}\}\,\} \leq \min\{p_{\mathcal{R}} \mid \mathcal{R} \in \mathcal{M}\}$ and hence $\mathbf{1}[\,\min\{p_{\mathcal{R}} \mid \mathcal{R} \in \mathcal{M}\} \leq \delta\,] \leq \mathbf{1}[\,\min\{p_{\mathcal{R}} \mid \mathcal{R} \in \mathcal{M} \cup \{\mathcal{S}\}\} \leq \delta\,]$. □

PROPOSITION 2. $\mathrm{FWER}(\delta)$ *with $\delta < \hat{\Psi}(\sigma-1)$ can be evaluated exactly using only the p-values of patterns with support $x_{\mathcal{S}} \geq \sigma$.*

PROOF. First note that $x_{\mathcal{S}} \geq \sigma \Leftrightarrow \mathcal{S} \in \hat{\mathcal{I}}_T(\hat{\Psi}(\sigma))$. Let $p'_{\min} = \min\{\, p_{\mathcal{S}} \mid \mathcal{S} \in \hat{\mathcal{I}}_T(\hat{\Psi}(\sigma))\,\}$. If $p'_{\min} > \delta$ then $p_{\min} > \delta$, because $p_{\mathcal{S}} \geq \hat{\Psi}(\sigma-1) > \delta \;\forall\, \mathcal{S} \notin \mathcal{I}_T(\hat{\Psi}(\sigma))$ by definition of $\hat{\mathcal{I}}_T(\bullet)$. Thus, $\mathbf{1}[\,p_{\min} \leq \delta\,] = \mathbf{1}[\,p'_{\min} \leq \delta\,]$. □

PROOF OF THEOREM 1. Proposition 1 guarantees that every time $\mathrm{FWER}(\delta) > \alpha$ in Line 20 of Algorithm 1, we know that $\delta^* < \delta$ and must therefore decrease the current threshold δ. This is because even if the current estimate $\mathrm{FWER}(\delta)$ is still noisy due to the many remaining patterns not yet processed, we know that processing further patterns will only make $\mathrm{FWER}(\delta)$ larger.

Finally, Proposition 2 guarantees that, at convergence, Westfall-Young light has all the information it needs to compute δ^*. Let σ_f be the minimum support at convergence and $\delta_f = \hat{\Psi}(\sigma_f)$ the associated corrected significance threshold. Similarly, let $\delta_{f,\mathrm{prev}} = \hat{\Psi}(\sigma_f - 1)$ be the immediately preceding corrected significance threshold.

By construction of the algorithm, $\mathrm{FWER}(\delta_f) \leq \alpha$ and $\mathrm{FWER}(\delta_{f,\mathrm{prev}}) > \alpha$. Thus, $\delta^* \in [\delta_f, \delta_{f,\mathrm{prev}})$. By Proposition 2, we can then only evaluate exactly $\mathrm{FWER}(\delta)$ for all $\delta < \delta_{f,\mathrm{prev}}$, but since $\delta^* < \delta_{f,\mathrm{prev}}$, that is all we need. In fact, this implies that our algorithm does not waste time computing exactly the values of $\{\, p^{(j)}_{\min} \,\}_{j=1}^{j_{\mathrm{p}}}$ that are above δ^*, translating into a much larger final minimum support σ_f for pattern enumeration. □

3.3 Improvements over FastWY

Compared to the current state-of-the-art FastWY, our proposal improves the following scalability aspects:

(I1) Instead of a decremental search strategy, it is based on an incremental search strategy, which recent studies have shown to be more efficient than decremental search [18, 22]. The incremental search is implemented in the initialization of the support σ in Line 8 of Algorithm 1 and when σ is updated in Line 20.

(I2) It computes the solution δ^* enumerating the frequent patterns only once, instead of j_{p} times. Moreover, it does so without any non-scalable memory overheads such as storing $G(\mathcal{S}, t_i)$ for all $\mathcal{S} \in \hat{\mathcal{I}}_T(\sigma), i = 1, \ldots, n$. Indeed, Line 25 prunes the patterns according to the support $x_{\mathcal{S}}$. Non-pruned patterns are processed using the function `ProcessNext`, which computes the j_{p} p-values corresponding to all j_{p} permutations simultaneously. There is no longer any need to repeat pattern mining once per permutation or store occurrence lists $G(\mathcal{S}, t_i)$ in memory.

(I3) It does not need to compute the $\lfloor (1-\alpha) j_{\mathrm{p}} \rfloor$ largest values of $\{\, p^{(j)}_{\min} \,\}_{j=1}^{j_{\mathrm{p}}}$ exactly, which greatly increases the minimum support for pattern enumeration and thus reduces the runtime. It also decreases the number of cell counts that need to be evaluated, further reducing the overall runtime. This is a consequence of the incremental search strategy described in the previous point and the natural stopping criteria of the method via ending of the depth-first enumeration recursion of function `ProcessNext`.

(I4) It uses an efficient scheme to share the computation of p-values across permutations, reducing the corresponding runtime for that task by a factor of j_{p}, with j_{p} in the order of 10^4. Therefore, the runtime for p-value evaluation in Westfall-Young light is negligible for all practical purposes. This is implemented via the precomputation of p-values in Line 13, which involves time complexity $O(\min\{n_1, x_{\mathcal{S}}\})$ independent of j_{p} and neg-

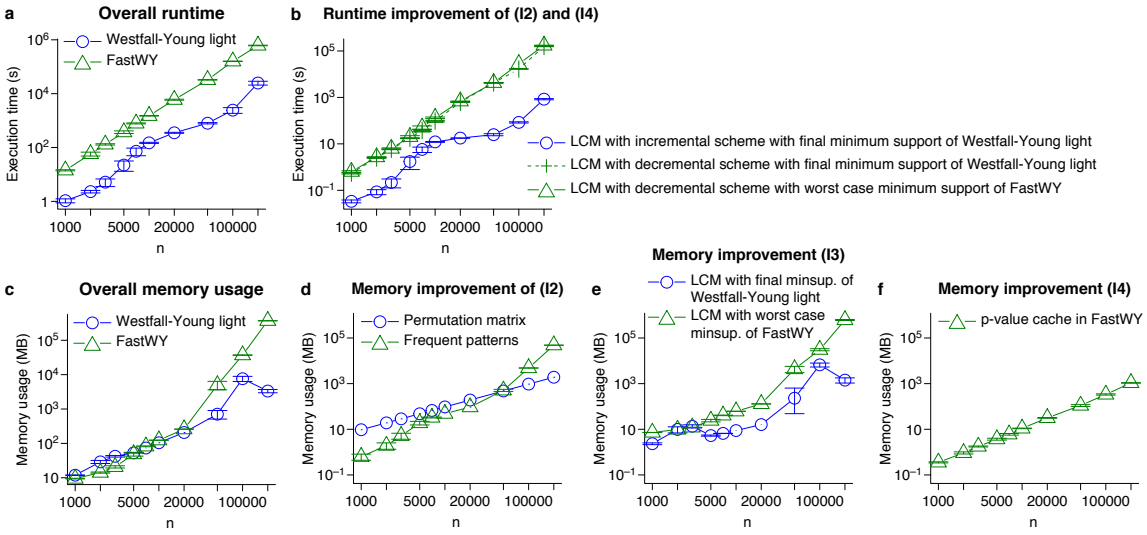


Figure 1: Simulation results. Runtimes beyond 12 hours are extrapolated. Note that both the x- and y-axes have logarithmic scale. Data show means±SD in 10 trials.

ligible storage complexity. Then, for each permutation j, we simply fetch the corresponding p-value from the precomputed ones. In contrast, FastWY caches every p-value computation in memory, creating an additional memory overhead.

In the next section, we empirically demonstrate that, as a consequence of improvements 1-4, Westfall-Young light has much better memory-usage scaling and can be up to three orders of magnitude faster in real-world data than FastWY.

4. EXPERIMENTS

We evaluate our proposed method Westfall-Young light on a wide range of synthetic and real-world datasets in two representative data mining problems: *significant itemset mining* and *significant subgraph mining*, compared to the current state-of-the-art FastWY.

4.1 Experimental Setup

Both Westfall-Young light and FastWY were written in C/C++. Since FastWY was implemented in Python by its authors, we reimplemented it in C/C++ for a fair comparison. This new implementation of FastWY used as a baseline is about two or three orders of magnitude faster and reduces the amount of RAM used by one or two orders of magnitude.

As a frequent closed itemset miner, we chose LCM version 3 [26] for both Westfall-Young light and FastWY. LCM has been shown to exhibit state-of-the-art performance in a large number of datasets and won the FIMI'04 frequent itemset mining competition [12]. The code was compiled using Intel C++ compiler version 14.0.1 and executed on a single 2.7 GHz Intel Xeon CPU with 256 GB of memory. In subgraph mining, we employed Gaston [19] as a frequent subgraph miner[2] because it is reported to be one of the fastest algorithms [31]. The code was compiled with `gcc` 4.8.2 and run on a single 2.5 GHz Intel Xeon CPU with 256 GB of memory.

[2]Code available from `http://www.liacs.nl/~snijssen/gaston/iccs.html`

4.2 Evaluation on Synthetic Data

First we analyze Westfall-Young light on synthetic data to evaluate the improvement of overall runtime and memory usage compared to FastWY and the contribution of each improvement from (I1) to (I4) in Section 3. Note that (I1) and (I3) contribute to the runtime efficiency and (I2), (I3), and (I4) to the memory efficiency.

We randomly generated a set of transactions with $n/n_1 = 2$ and $m = 10^3$, where the size of each transaction is 50, and we varied n from 10^3 to 10^5. Then 10 true causal itemsets $\mathcal{S}_1, \ldots, \mathcal{S}_{10}$ with $|\mathcal{S}_i| = 5$ were prepared, which were also randomly generated from the same domain of items, and each transaction with the label 1 contains one of those itemsets.

Results are plotted in Figure 1. The overall runtime in Figure 1**a** clearly shows that Westfall-Young light is one to two orders of magnitude faster than FastWY, and the difference gets larger as n increases. To explore the reason for this difference, we solely ran LCM with the decremental or incremental scheme, omitting the computational effort for permutation testing that is shared between methods. In addition, to separate the improvement (I1) and (I3), we also ran LCM with the decremental scheme by using the final minimum support obtained by our method. In our results, plotted in Figure 1**b**, the difference between a green line and a green dashed line shows the contribution of (I3), and the difference between a green dashed line and a blue line shows the contribution of (I1). Therefore, our improvement (I1) of employing the incremental search strategy is crucial for the efficiency.

The peak memory usage of our method is larger than that of FastWY if n is small, but gets smaller once n exceeds 10^4 and is an order of magnitude smaller if $n \geq 5 \cdot 10^4$ (Figure 1**c**). This comes from the aggregation of our three improvements (I2), (I3), and (I4), the contributions of which are demonstrated in Figures 1**d**, **e**, and **f**, respectively. Figure 1**d** shows the memory usage for a permutation matrix in our method (green) and storing all frequent patterns in FastWY (blue). We can see that the amount of memory for storing frequent patterns finally exceeds that for a permutation matrix. This is due to the number of frequent patterns exponentially increasing with n, while the amount

Table 1: Characteristics of itemset mining datasets. n and n_1 are the number of transactions in total and in the minor class, respectively, m is the number of items and $|t|/n$ the average transaction size. n/n_1 is shown only for labeled datasets.

Property	TicTacToe	Chess	Inetads	Mushroom	Breast cancer	Pumsb-star	Connect	BmsWebview	Retail	T10I4D100K	T40I10D100K	Bmspos
n	958	3196	3279	8124	12773	49046	67557	77512	88162	100000	100000	515597
n/n_1	2.89	–	7.14	2.08	11.31	–	–	–	–	–	–	–
m	18	75	1554	117	1129	7117	129	3340	16470	870	942	1657
$\|t\|/n$	6.93	37.00	12.00	22.00	6.70	50.48	43.00	4.62	10.31	10.10	39.61	6.53

Table 2: Characteristics of subgraph mining datasets, where $|V|$ and $|E|$ denote the number of vertices and edges, respectively.

Property	PTC (MR)	PTC (FR)	PTC (MM)	PTC (FM)	MUTAG	ENZYMES	D&D	NCI1	NCI41	NCI109	NCI167	NCI220
n	584	584	576	563	188	600	1178	4208	27965	4256	80581	900
n/n_1	3.23	3.74	3.18	3.15	2.98	2.00	2.42	2.00	17.23	2.00	8.38	3.10
$\max\|V\|$	181	181	181	181	28	126	5748	462	462	462	482	239
$\max\|E\|$	181	181	181	181	66	149	14267	468	468	468	478	255

of memory for storing a permutation matrix scales linearly. Moreover, the improvement (I3) results in less memory usage in LCM (shown in Figure 1**e**) due to a larger minimum support, and (I4) saves the extra amount of memory for the p-value cache in FastWY in Figure 1**f**. Interestingly, the overall memory usage decreases if $n = 2{\cdot}10^5$. This is because the final minimum support rises, resulting in less memory usage in LCM as shown in Figure 1**e**.

4.3 Evaluation on Real Data

Itemset Mining Datasets: We used four labeled datasets: TicTacToe[3], Inetads[4], Mushroom, and Breast cancer. The first three are commonly studied datasets taken from the UCI repository and Breast cancer is described in [25].

In addition, we used eight unlabeled datasets from the well-known public benchmark datasets for frequent itemset mining [12]: Bmspos, BmsWebview, Retail, T10I4D100K, T40-I10D100K, Chess, Connect and Pumsb-star. Since labels only affect the algorithm via n and n_1 as far as finding the corrected significance threshold δ^* is concerned, we considered two representative cases: $n/n_1 = 2$ or 10. Higher ratios n/n_1 are usually more computationally demanding. This results in a total of 20 different cases to be tested. The main properties of each dataset are summarized in Table 1.

Subgraph Mining Datasets: We used 12 labeled graph datasets: four PTC (Predictive Toxicology Challenge) datasets[5], MUTAG, ENZYMES, D&D[6], and four NCI (National Cancer Institute) datasets[7], where ENZYMES and D&D are proteins and others are chemical compounds. These datasets are popular benchmarks and have been frequently used in previous studies (e.g. [16, 21]). Graph nodes are labeled in all datasets and edges are also labeled except for ENZYMES and D&D. In the four PTC datasets, graphs labeled as CE, SE, or P were treated as positive and those of NE or N as negative, the same setting as in [15]. The properties of these datasets are summarized in Table 2.

Note that the number of nodes in subgraphs is bounded by 15 in NCI1, NCI109, and NCI220, 10 in MUTAG, NCI41, and NCI167, and 8 in ENZYMES such that the comparison partner, FastWY, can finish in a reasonable time, allowing us to check its peak memory consumption. For example, in ENZYMES with the maximum subgraph size 10, our method takes 3.6 hours while FastWY did not stop after two weeks. In D&D and the four PTC datasets, the size of subgraphs is unlimited.

Runtime and memory usage

As the main result, we compare the runtime and memory usage of our method Westfall-Young light and the comparison partner FastWY for all 20 itemset mining cases and 12 subgraph mining datasets. In both algorithms, the number j_{p} of permutations is the only parameter and is set $j_{\mathrm{p}} = 10^4$. We defer the discussion about the effect ofj_{p} to the end of this section. The target FWER α is never to be treated as a parameter, but as a user requirement fixed a priori before seeing the data. Here we use $\alpha = 0.05$, which is the most standard choice across different scientific disciplines.

The results for significant itemset mining are summarized in Figure 2, and those of significant subgraph mining are summarized in Figure 3. Overall, our algorithm Westfall-Young Light tends to be two to three orders of magnitude faster than FastWY in itemset mining and one to two orders of magnitude faster in subgraph mining. The exact difference in runtime is dataset-dependent, but there is a clear trend showing that, for large-scale datasets, the runtime gap between FastWY and Westfall-Young light increases sharply.

Even more importantly, six out of 20 significant itemset datasets correspond to the datasets Chess, Pumsb-star and Connect, for which FastWY crashed due to excessive memory requirements. Indeed, as far as memory usage is concerned, we see two different situations: (1) in 17 out of 32 datasets, both methods use approximately the same amount of memory (up to the order of magnitude); (2) in the others, the peak memory usage of FastWY is much larger than

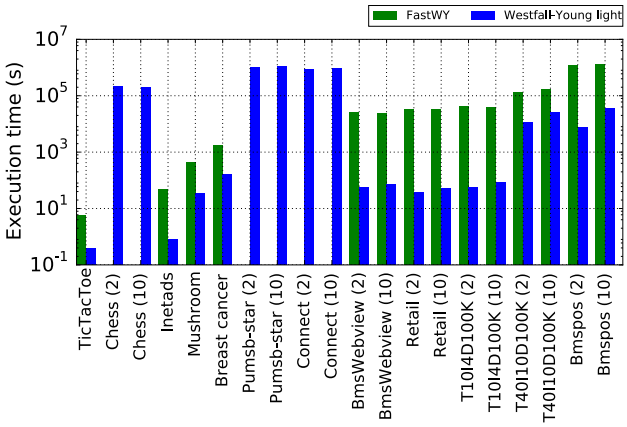
[3] `https://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame`

[4] `https://archive.ics.uci.edu/ml/datasets/Internet+Advertisements`

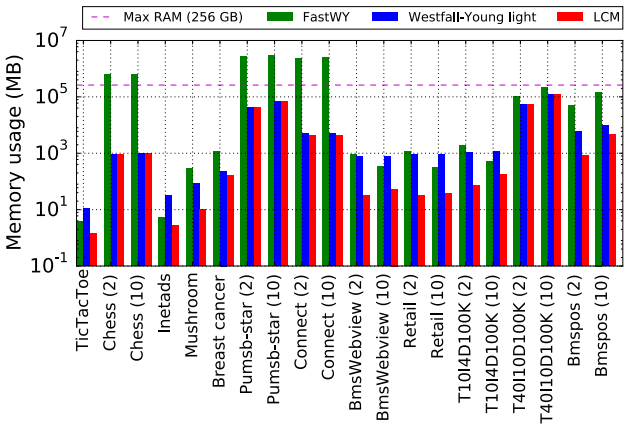
[5] `http://www.predictive-toxicology.org/ptc/`

[6] MUTAG, ENZYMES, and D&D are obtained from `http://mlcb.is.tuebingen.mpg.de/Mitarbeiter/Nino/Graphkernels/data.zip`

[7] `https://pubchem.ncbi.nlm.nih.gov/`



(a) Runtime

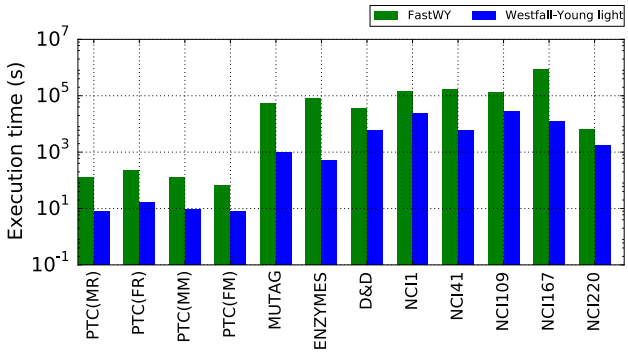


(b) Peak memory usage

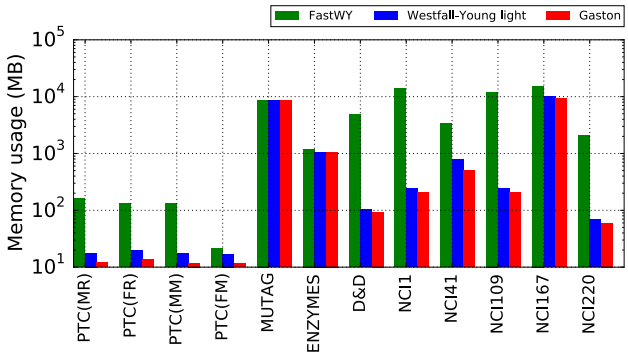
Figure 2: Performance comparison between Westfall-Young light and FastWY in itemset mining with $j_p = 10^4$. Numbers attached to dataset names denote the class ratio n/n_1.

that of Westfall-Young light, in some cases soaring up to the point in which the algorithm simply breaks down. It should also be noted that for most of the 17 datasets in which both methods have a similar memory footprint, the frequent pattern miner (LCM or Gaston) is responsible for almost the entire memory usage, which means there is little room for improvement. In general, the trend for the memory usage gap is similar to that for the runtime gap: FastWY only works properly in small-sized problems and shows poor scaling characteristics in larger datasets. With respect to memory usage, this is clearly due to the need to store $G(\mathcal{S}, t_i)$ for all $\mathcal{S} \in \hat{\mathcal{I}}_T(\sigma), i = 1, \ldots, n$. As a consequence, the peak memory overhead of FastWY scales with the number of frequent patterns enumerated by the algorithm, which tends to increase exponentially with the database size and density. In contrast, Westfall-Young light has a much smaller memory overhead that does not increase with the number of frequent patterns; it only increases linearly with the number of transactions n.

The actual memory usage of FastWY for the six out of 20 significant itemset mining datasets in which it crashed is actually only lower-bounded in Figure 2b; the numbers we plotted are a conservative lower bound on what the actual memory usage would have been, obtained simply by counting the number of testable patterns processed by Westfall-



(a) Runtime



(b) Peak memory usage

Figure 3: Performance comparison between Westfall-Young light and FastWY in subgraph mining with $j_p = 10^4$ permutations. The memory usage due to Gaston is included.

Young light and computing the amount of memory needed to store $G(\mathcal{S}, t_i)$ for all $\mathcal{S} \in \hat{\mathcal{I}}_T(\sigma), i = 1, \ldots, n$. This neglects: (1) all memory needed by LCM itself; (2) the fact that FastWY produces many more testable patterns due to computing the entire set $\{ p^{(j)}_{\min} \}_{j=1}^{j_p}$ exactly.

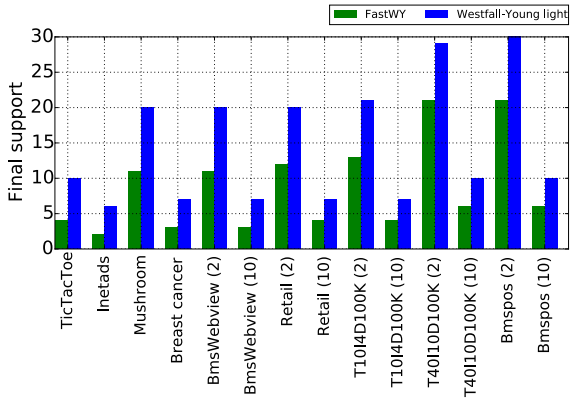
Final minimum support for frequent pattern mining

Since FastWY computes all values $\{ p^{(j)}_{\min} \}_{j=1}^{j_p}$ exactly, the final minimum support for frequent pattern mining is determined by $\max_j p^{(j)}_{\min}$. In contrast, Westfall-Young light has its final minimum support determined by the $\lceil \alpha j_p \rceil$ largest values only. The impact on the final support is shown quantitatively in Figure 4. It can be seen from the figure that, in most datasets, the final minimum support of FastWY is considerably larger than that of Westfall-Young light.

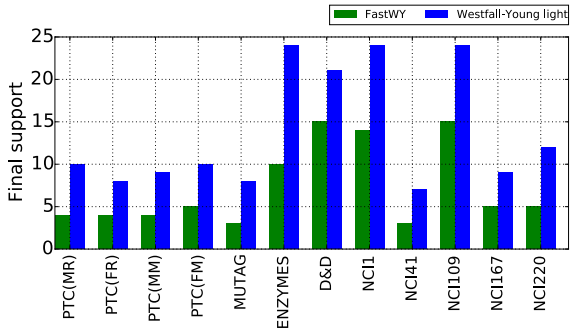
Most databases obey a power-law distribution, which makes patterns with low supports much more abundant than those with large supports. This effect is particularly harmful for FastWY as the memory overhead needed to store $G(\mathcal{S}, t_i)$ for all $\mathcal{S} \in \hat{\mathcal{I}}_T(\sigma), i = 1, \ldots, n$ scales with the number of frequent patterns at the final minimum support.

Statistical power of permutation-testing

Measuring the resulting FWER is a powerful proxy to compare the statistical power of several FWER-controlling methods. Optimal schemes should achieve a FWER as close as possible to α without it ever being larger. If FWER $\ll \alpha$, the method is overly conservative, and will therefore have a higher probability of missing true positives. In this section,



(a) Itemset mining.



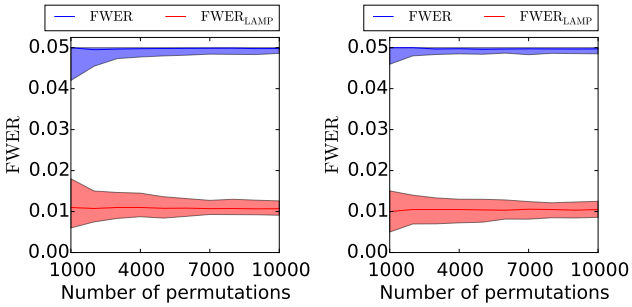
(b) Subgraph mining.

Figure 4: Comparison of final minimum support between Westfall-Young light and FastWY. Datasets for which FastWY crashed due to memory limitations were excluded.

we compare the resulting FWER of Westfall-Young permutation testing-based methods to that of LAMP, δ_{LAMP} [24], which is the first prominent method that controls the FWER in significant pattern mining. LAMP applies an improved Bonferroni correction that also exploits the concept of minimum attainable p-value $\Psi(x_{\mathcal{S}})$, but does not correct for the dependence between patterns, thus making it a baseline to measure the statistical power of Westfall-Young permutation testing approaches. Note that the resulting FWER of FastWY and Westfall-Young light is identical since both use the same underlying statistical procedure.

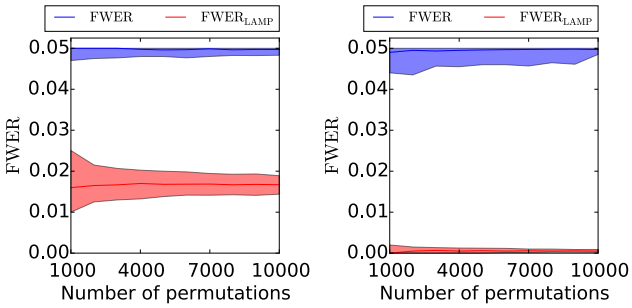
As far as parameters are concerned, we must only deal with the number j_{P} of permutations. Intuitively, the trade-off involved when setting j_{P} is clear: the larger j_{P}, the more precise the estimation of δ^* will be at the expense of increased runtime. To illustrate the effect of changes in the number j_{P} of permutations, we executed Westfall-Young light for 10 different values of j_{P} between $j_{\text{P}} = 10^3$ and $j_{\text{P}} = 10^4$ in steps of $\Delta j_{\text{P}} = 10^3$. For each pair (dataset, j_{P}) we repeat the execution 100 times and show the median empirical FWER as a function of j_{P}, along with the corresponding 5%–95% confidence interval.

We depict the results for four sample datasets due to space considerations: two datasets in itemset mining (BmsWebview and T40I10D100K) in Figure 5 and two datasets in subgraph mining (ENZYMES and NCI220) in Figure 6. As the figure shows, the median FWER appears to be fairly stable to changes in j_{P} and, more importantly, the spread of the empirical FWER saturates at about $j_{\text{P}} = 10^4$. Therefore, we believe $j_{\text{P}} = 10^4$ to be the safest parameter choice. Using



(a) BmsWebview ($n/n_1 = 2$) (b) T40I10D100K ($n/n_1 = 2$)

Figure 5: Empirical FWER versus j_{P} for two representative itemset mining databases.



(a) ENZYMES. (b) NCI220.

Figure 6: Empirical FWER versus j_{P} for two representative subgraph mining databases.

$j_{\text{P}} = 10^3$ might still lead to good performance while reducing the runtime by approximately one order of magnitude.

Finally, our results also clearly demonstrate that LAMP, while much more effective than a standard Bonferroni correction (which would have a FWER extremely close to 0), is still an overly conservative algorithm. It tends to yield an empirical FWER that oscillates between $\alpha/2$ and $\alpha/100$ depending on the dataset. Even just halving the target FWER can have drastic consequences and cause a large amount of significant patterns to be lost as a consequence of over-controlling the FWER; this point justifies the need to use permutation-testing based approaches if optimal statistical power is required.

5. CONCLUSIONS

In this paper, we have described a novel algorithm for mining statistically significant patterns, called *Westfall-Young light*, which allows users to adjust the probability of having false discoveries. The algorithm estimates the null distribution of the test statistics via Westfall-Young permutations, and succeeds in overcoming the massive computational cost of permutation testing in large databases by exploiting a set of computational tricks.

Empirically, our Westfall-Young light algorithm drastically improves upon the state of the art. The runtime decreases by up to three orders of magnitude and the peak memory usage by up to two orders of magnitude in several itemset and subgraph mining benchmarks. We also show that the peak memory usage of Westfall-Young light scales gently with the complexity of the database. In contrast, the peak memory usage of the state-of-the-art algorithm soars as

the databases become large and dense, thus breaking down in large-scale problems.

Several interesting challenges still remain to be addressed. In domains such as computational biology, there is increasing interest in less conservative statistical testing procedures that enjoy increased statistical power, such as FDR control [6]. Another critical problem is how to correct for confounders [9]. Those are predictive features that are correlated to both the target response and some of the patterns, artificially inflating the resulting p-values. Extending the framework in either of those directions would represent a valuable contribution.

6. ACKNOWLEDGMENTS

This work was funded in part by the Alfried Krupp von Bohlen und Halbach-Stiftung (KB), the SNSF Starting Grant 'Significant Pattern Mining' (KB), a Grant-in-Aid for Scientific Research (Research Activity Start-up) 26880013 (MS) and the Marie Curie Initial Training Network MLPM2012, Grant No. 316861 (FLL, KB).

7. REFERENCES

[1] C. C. Aggarwal and J. Han, editors. *Frequent Pattern Mining*. Springer, 2014.

[2] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, pages 207–216, 1993.

[3] A. Arora, M. Sachan, and A. Bhattacharya. Mining statistically significant connected subgraphs in vertex labeled graphs. In *SIGMOD*, pages 1003–1014, 2014.

[4] M. Atzmueller and F. Puppe. SD-Map – A fast algorithm for exhaustive subgroup discovery. In *PKDD*, volume 4213 of *LNCS*, pages 6–17, 2006.

[5] S. D. Bay and M. Pazzani. Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery*, 5(3):213–246, 2001.

[6] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B*, 57(1):289–300, 1995.

[7] C. E. Bonferroni. Teoria statistica delle classi e calcolo delle probabilità. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze*, 8:3–62, 1936.

[8] G. Dong and J. Bailey. *Contrast Data Mining: Concepts, Algorithms, and Applications*. Chapman&Hall/CRC, 2012.

[9] M. P. Epstein, R. Duncan, Y. Jiang, K. N. Conneely, A. S. Allen, and G. A. Satten. A permutation procedure to correct for confounders in case-control studies, including tests of rare variation. *Am J Hum Genet*, 91:215–223, 2012.

[10] W. Fan, K. Zhang, H. Cheng, J. Gao, X. Yan, J. Han, P. S. Yu, and O. Verscheure. Direct mining of discriminative and essential frequent patterns via model-based search tree. In *SIGKDD*, pages 230–238, 2008.

[11] R. A. Fisher. On the Interpretation of χ^2 from Contingency Tables, and the Calculation of P. *Journal of the Royal Statistical Society*, 85(1):87–94, 1922.

[12] B. Goethals and M. J. Zaki. Frequent itemset mining dataset repository (FIMI'04). `http://fimi.ua.ac.be/data/`, 2004.

[13] W. Hämäläinen. StatApriori: an efficient algorithm for searching statistically significant association rules. *Knowledge and Information Systems*, 23(3):373–399, 2010.

[14] J. Hopstadius and G. N. Norén. Robust discovery of local patterns: Subsets and stratification in adverse drug reaction surveillance. In *ACM SIGHIT*, pages 265–274, 2012.

[15] X. Kong and P. S. Yu. Semi-supervised feature selection for graph classification. In *SIGKDD*, pages 793–802, 2010.

[16] G. Li, M. Semerci, B. Yener, and M. J. Zaki. Effective graph classification based on topological and label attributes. *Statistical Analysis and Data Mining*, 5(4):265–283, 2012.

[17] F. Llinares-López, D. G. Grimm, D. A. Bodenham, U. Gieraths, M. Sugiyama, B. Rowan, and K. M. Borgwardt. Genome-wide detection of intervals of genetic heterogeneity associated with complex traits. *Bioinformatics, in press*, 2015.

[18] S. Minato, T. Uno, K. Tsuda, A. Terada, and J. Sese. A fast method of statistical assessment for combinatorial hypotheses based on frequent itemset enumeration. In *ECMLPKDD*, volume 8725 of *LNCS*, pages 422–436, 2014.

[19] S. Nijssen and J. N. Kok. A quickstart in frequent structure mining can make a difference. In *SIGKDD*, pages 647–652, 2004.

[20] P. K. Novak, N. Lavrač, and G. I. Webb. Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *JMLR*, 10:377–403, 2009.

[21] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-Lehman graph kernels. *JMLR*, 12:2359–2561, 2011.

[22] M. Sugiyama, F. Llinares-López, N. Kasenburg, and K. M. Borgwardt. Significant subgraph mining with multiple testing correction. In *SDM*, pages 37–45, 2015.

[23] R. E. Tarone. A modified bonferroni method for discrete data. *Biometrics*, 46(2):515–522, 1990.

[24] A. Terada, M. Okada-Hatakeyama, K. Tsuda, and J. Sese. Statistical significance of combinatorial regulations. *Proceedings of the National Academy of Sciences*, 110(32):12996–13001, 2013.

[25] A. Terada, K. Tsuda, and J. Sese. Fast westfall-young permutation procedure for combinatorial regulation discovery. In *IEEE International Conference on Bioinformatics and Biomedicine*, pages 153–158, 2013.

[26] T. Uno, T. Asai, Y. Uchida, and H. Arimura. An efficient algorithm for enumerating closed patterns in transaction databases. In *Discovery Science*, volume 3245 of *LNCS*, pages 16–31, 2004.

[27] G. Webb. Discovering significant rules. In L. Ungar, M. Craven, D. Gunopulos, and T. Eliassi-Rad, editors, *SIGKDD*, pages 434 – 443, New York, 2006. The Association for Computing Machinery.

[28] G. Webb. Layered critical values: A powerful direct-adjustment approach to discovering significant patterns. *Machine Learning*, 71(2-3):307–323, 2008.

[29] G. I. Webb. Discovering significant patterns. *Machine Learning*, 68(1):1–33, 2007.

[30] P. Westfall and S. S. Young. *Resampling-Based Multiple Testing: Examples and Methods for P-Value Adjustment*. Wiley, 1993.

[31] M. Wörlein, T. Meinl, I. Fischer, and M. Philippsen. A quantitative comparison of the subgraph miners MoFa, gSpan, FFSM, and Gaston. In *PKDD*, volume 3721 of *LNCS*, pages 392–403. Springer, 2005.

[32] X. Yan, H. Cheng, J. Han, and P. S. Yu. Mining significant graph patterns by leap search. In *SIGMOD*, pages 433–444, 2008.

[33] X. Zhang, S. Huang, F. Zou, and W. Wang. TEAM: Efficient two-locus epistasis tests in human genome-wide association study. *Bioinformatics*, 26(12):i217–i227, 2010.

[34] X. Zhang, F. Zou, and W. Wang. FastANOVA: an efficient algorithm for genome-wide association study. In *SIGKDD*, pages 821–829, 2008.

[35] A. Zimmermann, B. Bringmann, and U. Rückert. Fast, effective molecular feature mining by local optimization. In *ECMLPKDD*, volume 6323 of *LNCS*, pages 563–578, 2010.

[36] A. Zimmermann and S. Nijssen. *Supervised Pattern Mining and Applications to Classification*, pages 425–442. Springer, 2014.