The XL-mHG test for enrichment: Algorithms, bounds, and power

Florian Wagner[1,2,*]

[1]**Graduate Program in Computational Biology and Bioinformatics, Duke University, Durham, NC, USA**
[2]**Center for Genomic and Computational Biology, Duke University, Durham, NC, USA**
[*]**Email: florian.wagner@duke.edu**

ABSTRACT

The XL-mHG test is a semiparametric test for enrichment in ranked lists with boolean (0/1-valued) entries. It is a generalization of the nonparametric mHG test, designed to provide some control over the kind of enrichment that is being tested for, and to allow a flexible trade-off between the sensitivity and robustness of the test. Here, I describe an improved algorithm to efficiently calculate the XL-mHG p-value p, and discuss upper and lower bounds for p. Furthermore, I perform simulations to show that the mHG test is a significantly more powerful alternative to the Kolmogorov-Smirnov (KS) test for detecting enrichment in scenarios that are frequently encountered in biological applications. An open-source Python/Cython implementation of the XL-mHG test is provided in the `xlmhg` package, available from PyPI and GitHub (`https://github.com/flo-compbio/xlmhg`) under an OSI-approved license.

Keywords: enrichment, XL-mHG test, KS test, algorithms, hypothesis testing

INTRODUCTION

Given a ranked list with boolean (0/1-valued) entries, the XL-mHG test (Wagner 2015b) is a semiparametric hypothesis test to determine whether there exists statistically significant *enrichment* of 1's "at the top of the list". It is a generalization of the nonparametric mHG (*minimum hypergeometric*) test, which was originally proposed by Eden, Lipson, et al. (2007).

By not relying on a fixed cutoff to define "the top", the mHG test can simultaneously detect an usual accumulation of 1's among the first few elements, as well as a moderate enrichment within, say, the entire first half of the list. This flexibility makes the test a very attractive choice when it is not known *a priori* what part of the list might exhibit enrichment. The XL-mHG introduces two parameters, X and L, that specify the *minimum number of 1's required for enrichment*, and the textitlowest cutoff to be examined, respectively. (It is important to note that the L parameter was already proposed by Eden, Lipson, et al. (2007), under the name $n_{\max}$.) These parameters provide a certain level of control over the kind of enrichment that is being tested for, as well as a flexible trade-off between the sensitivity and robustness of the test. For $X = 1$ and $L = N$, the XL-mHG test reduces to the mHG test.

This manuscript describes three results concerning the XL-mHG test: First, it introduces a new algorithm for calculating the XL-mHG p-value. This algorithm relies heavily on the same dynamic programming approach proposed by Eden, Lipson, et al. (2007), and therefore has the same $\mathcal{O}(N^2)$ time complexity. However, it adopts a distinct strategy in calculating the p-value, and enjoys several advantages in terms of its actual runtime and numerical stability. Second, the manuscript discusses modifications to an upper bound for the mHG p-value (Eden, Lipson, et al. 2007) that take the X and L parameters into account. This includes the description of a modified $\mathcal{O}(1)$ bound, and an $\mathcal{O}(N)$ algorithm to calculate an even tighter upper bound. Finally, I present simulations that demonstrate that the mHG test is significantly more powerful than the Kolmogorov-Smirnov (KS) test in certain scenarios that are particularly relevant for biological applications.

Notation and definitions

We represent a ranked list with boolean entries as a column vector $\boldsymbol{v}$ of length N, with all elements being either 0 or 1:

$$\boldsymbol{v} = (v_1, v_2, \ldots, v_N)^T,\ v_i \in \{0, 1\}$$

We therefore also refer to list entries as "elements". We refer to the set of all elements for which $v_i = 0$ as "the 0's", and to the set of all other elements as "the 1's". We also say that v_1 represents the "topmost" element, and v_N the "bottommost" element of the list. We further let K and W denote the total number of 1's and 0's in the list, respectively ($K + W = N$). Throughout this article, we assume that N and K (and therefore W) are fixed, unless stated otherwise. We next define $\mathcal{V}^{(N,K)}$ to be the set of all lists of length N that contain exactly K 1's (there are $\binom{N}{K}$ distinct lists in $\mathcal{V}^{(N,K)}$).

Let $f(k;\ N, K, n)$ represent the probability mass function of the hypergeometric distribution:

$$f(k;\ N, K, n) = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}} \qquad \text{(Hypergeometric PMF)}$$

Then, let $p^{\text{HG}}(k;\ N, K, n)$ represent the hypergeometric p-value:

$$p^{\text{HG}}(k;\ N, K, n) = \sum_{j=k}^{\min(n,K)} f(j;\ N, K, n) \qquad \text{(Hypergeometric p-value)}$$

For any $\boldsymbol{v} \in \mathcal{V}^{(N,K)}$ and $n \in \{1, 2, ..., N\}$, let $k_n(\boldsymbol{v})$ represent the number of 1's among the first n elements of $\boldsymbol{v}$:

$$k_n(\boldsymbol{v}) = \sum_{i=1}^{n} v_i$$

Then, let $p_n^{\text{HG}}(\boldsymbol{v})$ represent the hypergeometric p-value for $\boldsymbol{v}$ using n as the "cutoff":

$$p_n^{\text{HG}}(\boldsymbol{v}) = p^{\text{HG}}(k_n(\boldsymbol{v});\ N, K, n)$$

The mHG test statistic $s^{\text{mHG}}(\boldsymbol{v})$ is then defined as follows (Eden, Lipson, et al. 2007):

$$s^{\text{mHG}}(\boldsymbol{v}) := \min p_n^{\text{HG}}(\boldsymbol{v}) \qquad \text{(mHG test statistic)}$$

Let V^0 be a random variable representing a list drawn uniformly at random from $\mathcal{V}^{(N,K)}$. Let $S^{\text{mHG},0}$ be the mHG test statistic of V^0. Then the mHG p-value $p^{\text{mHG}}(\boldsymbol{v})$ is defined as follows (Eden, Lipson, et al. 2007):

$$\begin{aligned} p^{\text{mHG}}(\boldsymbol{v}) &:= \Pr(S^{\text{mHG},0} \leq s^{\text{mHG}}(\boldsymbol{v})) \qquad \text{(mHG p-value)} \\ &= \mathbb{E}\left(\mathbb{1}[s^{\text{mHG}}(\boldsymbol{v}^0) \leq s^{\text{mHG}}(\boldsymbol{v}) \mid V^0]\right) \\ &= \left(\sum_{\boldsymbol{v}^0 \in \mathcal{V}^{(N,K)}} \mathbb{1}[s^{\text{mHG}}(\boldsymbol{v}^0) \leq s^{\text{mHG}}(\boldsymbol{v})]\right) / \left|\mathcal{V}^{(N,K)}\right| \end{aligned}$$

Given parameters X and L, both $\in \{1, 2, ..., N\}$, the XL-mHG test statistic $s_{\text{X,L}}^{\text{mHG}}(\boldsymbol{v})$ is defined as follows (Wagner 2015b):

$$s_{\text{X,L}}^{\text{mHG}}(\boldsymbol{v}) := \begin{cases} \min\limits_{\substack{k_n(\boldsymbol{v}) \geq X \\ n \leq L}} p_n^{\text{HG}}(\boldsymbol{v}) & \text{if } k_L(\boldsymbol{v}) \geq X, \\ 1 & \text{otherwise} \end{cases} \qquad \text{(XL-mHG test statistic)}$$

The XL-mHG p-value $p_{\text{X,L}}^{\text{mHG}}(\boldsymbol{v})$ is defined analogous to $p^{\text{mHG}}(\boldsymbol{v})$ (Wagner 2015b). Let $S_{\text{X,L}}^{\text{mHG},0}$ be the XL-mHG test statistic of V^0. Then:

$$\begin{aligned} p_{\text{X,L}}^{\text{mHG}}(\boldsymbol{v}) &:= \Pr(S_{\text{X,L}}^{\text{mHG},0} \leq s_{\text{X,L}}^{\text{mHG}}(\boldsymbol{v})) \qquad \text{(XL-mHG p-value)} \\ &= \mathbb{E}\left(\mathbb{1}[s_{\text{X,L}}^{\text{mHG}}(\boldsymbol{v}^0) \leq s_{\text{X,L}}^{\text{mHG}}(\boldsymbol{v}) \mid V^0]\right) \\ &= \left(\sum_{\boldsymbol{v}^0 \in \mathcal{V}^{(N,K)}} \mathbb{1}[s_{\text{X,L}}^{\text{mHG}}(\boldsymbol{v}^0) \leq s_{\text{X,L}}^{\text{mHG}}(\boldsymbol{v})]\right) / \left|\mathcal{V}^{(N,K)}\right| \end{aligned}$$

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1962v1 | CC-BY 4.0 Open Access | rec: 15 Apr 2016, publ: 15 Apr 2016

RESULTS

An improved algorithm for calculating the XL-mHG p-value

Eden, Lipson, et al. (2007) proposed a dynamic programming algorithm that calculates the exact mHG p-value in $\mathcal{O}(N^2)$. I have reviewed that algorithm in detail, and modified it to calculate exact p-values for the XL-mHG test (Wagner 2015b). I will henceforth refer to this modified algorithm as `PVAL1`.

Briefly, for given N, K, X, L, and $s^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})$, `PVAL1` determines the fraction of lists in $\mathcal{V}^{(N,K)}$ with an XL-mHG test statistic at least as good as (i.e., equal to or smaller than) $s^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})$. By definition, this is the XL-mHG p-value $p^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})$. The first key insight behind the approach developed by Eden, Lipson, et al. (2007) is that even though the number of lists in $\mathcal{V}^{(N,K)}$ grows extremely quickly with N (e.g., $|\mathcal{V}^{(100,20)}| \approx 5.4 \times 10^{20}$), there exist only $(K+1)*(W+1)$ unique "hypergeometric configurations" $\mu_{(n,k)} \in \mathcal{M}^{(N,K)}$ (with $W = N-K$), each associated with a hypergeometric p-value $p_{(n,k)}$. Any list $\boldsymbol{v} \in \mathcal{V}^{(N,K)}$ has a unique representation as a sequence of hypergeometric configurations $(\mu_1, \mu_2, ..., \mu_N)$, corresponding to all possible cutoffs $(1, ..., N)$. Eden, Lipson, et al. (2007) refer to this sequence of configurations as a *path* (through $\mathcal{M}^{(N,K)}$; see Figure 1). Let $\mathcal{R}_{\text{X,L}}(\boldsymbol{v})$ be the set of all configurations with $p_{(n,k)} \leq s^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})$. Then, each list whose path "enters" $\mathcal{R}_{\text{X,L}}(\boldsymbol{v})$ has a mHG test statistic of $s^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})$ or smaller. In this scheme, $p^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})$ therefore equals the fraction of lists whose paths enter $\mathcal{R}_{\text{X,L}}(\boldsymbol{v})$. Eden, Lipson, et al. (2007) showed that this problem exhibits *optimal substructure*, making it amenable to dynamic programming. First, the authors observed that each path that contains a configuration $\mu_{(n,k)}$ either also contains the configuration $\mu_{(n-1,k)}$ or $\mu_{(n-1,k-1)}$. In the grid representation of $\mathcal{M}^{(N,K)}$ shown in Figure 1, this means that a configuration (dot) is reached "from the left" or "from below", respectively. Furthermore, they proposed to calculate the fraction of paths $\pi(\boldsymbol{v})$ that do *not* enter $\mathcal{R}_{\text{X,L}}(\boldsymbol{v})$ (so that $p^{\text{mHG}}(\boldsymbol{v}) = 1 - \pi(\boldsymbol{v})$). The algorithm relies on the following recurrence relation for calculating the fraction of all paths (i.e., all $\boldsymbol{v} \in \mathcal{V}^{(N,K)}$) that do not enter $\mathcal{R}_{\text{X,L}}(\boldsymbol{v})$ before arriving at a given configuration $\mu_{(n,k)}$:

$$\pi_{(n,k)}(\boldsymbol{v}) = \begin{cases} 0, & \text{if } \mu_{(n,k)} \in \mathcal{R}_{\text{X,L}}(\boldsymbol{v}), \\ \pi_{(n-1,k)}(\boldsymbol{v})\frac{W-w+1}{N-n+1} + \pi_{(n-1,k-1)}(\boldsymbol{v})\frac{K-k+1}{N-n+1} & \text{otherwise} \end{cases}$$

(Recurrence relation for `PVAL1`)

Obviously, if $\mu_{(n,k)} \in \mathcal{R}_{\text{X,L}}(\boldsymbol{v})$, all paths arriving at $\mu_{(n,k)}$ have now entered $\mathcal{R}_{\text{X,L}}(\boldsymbol{v})$, and $\pi_{(n,k)}(\boldsymbol{v}) = 0$. The coefficients in the other case represent the fraction of lists with configuration $\mu_{(n-1,k)}$ that have a 0 in position n, and the proportion of lists with configuartion $\mu_{(n-1,k-1)}$ that have a 1 in position n, respectively. If $w = 0$, or if $k = 0$, the first or second term of the recurrence relation is omitted, respectively, for the case $\mu_{(n,k)} \notin \mathcal{R}_{\text{X,L}}(\boldsymbol{v})$. Together with the initial value $\pi_{(0,0)} = 1.0$ — at the beginning, none of the paths have entered $\mathcal{R}_{\text{X,L}}(\boldsymbol{v})$ —, and an efficient algorithm for determining whether $\mu_{(n,k)} \in \mathcal{R}_{\text{X,L}}(\boldsymbol{v})$ for all $\mu_{(n,k)}$, this allows the calculation of $\pi(\boldsymbol{v}) = \pi_{(N,K)}$ in $\mathcal{O}(N^2)$; see Wagner (2015b) for a more detailed discussion.

`PVAL1`, while mathematically accurate and computationally efficient, still has some drawbacks in practice. First, it always requires the calculation of *all* $\pi_{(n,k)}(\boldsymbol{v})$, even though in many cases, only a small fraction of configurations are in $\mathcal{R}_{\text{X,L}}(\boldsymbol{v})$. For example, when $L = N/10$, approx. 90% of all $\mu_{(n,k)}$ are excluded from $\mathcal{R}_{\text{X,L}}(\boldsymbol{v})$ by definition. Moreover, since $s^{\text{mHG}}(\boldsymbol{v})$ serves as a lower bound for $p^{\text{mHG}}(\boldsymbol{v})$, calculating the mHG p-value is mostly of interest when $s^{\text{mHG}}(\boldsymbol{v})$ is below a specific significance threshold α (e.g., $\alpha = 10^{-6}$). In these cases the number of configurations in $\mathcal{R}_{\text{X,L}}(\boldsymbol{v})$ can be expected to be very small as well. A second drawback arises from the fact that for technical reasons, computers typically do not represent decimal numbers as a string of (significant) digits. Instead, they use a *floating point* system which can only represent certain numbers from the real line. This can lead to inaccuracies when very small numbers are involved in addition or substraction. For example, in most computer programs, the expression $1.0 - 10^{-20}$ will surprisingly evaluate to (exactly) 1.0, because 1.0 is the closest representable number to $1.0 - 10^{-20}$ (see footnote[1]). For `PVAL1`, this means that when the true p-value is very small — say, smaller than 10^{-15} — , numerical inaccuracies start to occur in filling in the dynamic programming table (which relies on addition) and in the calculation of $p^{\text{mHG}}(\boldsymbol{v}) = 1 - \pi(\boldsymbol{v})$, resulting in an inaccurate p-value. More concretely, due to the lack of representable numbers between 1.0 and $1.0 - 10^{-16}$, the

[1] In the commonly used IEEE-754 *binary64* ("double-precision") system, the first representable number below 1.0 is approximately 0.9999999999999999 or 1.0 - 10^{-16}.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1962v1 | CC-BY 4.0 Open Access | rec: 15 Apr 2016, publ: 15 Apr 2016

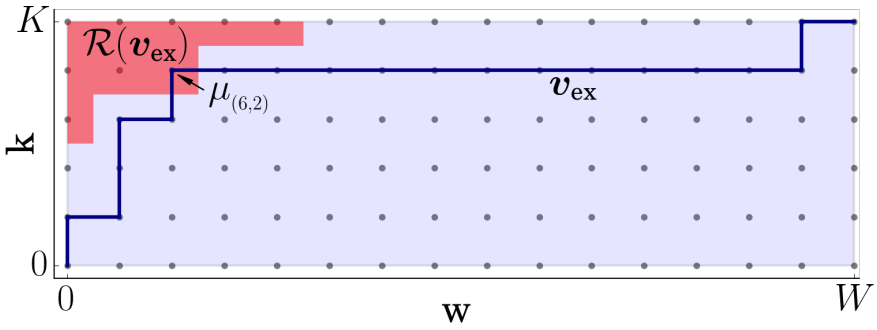

Figure 1. Representation of lists $\boldsymbol{v} \in \mathcal{V}^{(N,K)}$ as *paths* through $\mathcal{M}^{(N,K)}$ (Eden, Lipson, et al. 2007). Each gray dot represents a hypergeometric configuration $\mu_{(n,k)}$ (with $n = w + k$), and collectively, the dots in the $(K + 1) \times (W + 1)$ grid represent the set of all configurations in $\mathcal{M}^{(N,K)}$. In this example, $N = 20$ and $K = 5$. The path of the list $\boldsymbol{v}_{ex} = (1, 0, 1, 1, 0, 1, 0, ..., 0, 1, 0)^T$ is shown in navy blue. The mHG test statistic $s^{mHG}(\boldsymbol{v}_{ex})$ of this list is attained at the cutoff $n = 6$ (see arrow), for which $\boldsymbol{v}_{ex}$ has the configuration $\mu_{(6,4)}$. Shown in red is the space of all configurations in $\mathcal{R}(\boldsymbol{v}_{ex})$. These configurations are associated with an mHG test statistic equal to or smaller than $s^{mHG}(\boldsymbol{v}_{ex})$. The mHG p-value for $s^{mHG}(\boldsymbol{v}_{ex})$ is equal to the fraction of lists in $\mathcal{V}^{(20,5)}$ whose paths enter $\mathcal{R}(\boldsymbol{v}_{ex})$.

smallest non-zero p-value that can be obtained from `PVAL1` is $\approx 10^{-16}$ (see Figure 2**a**). When using an 80-bit "extended precision" data type, the smallest possible p-value is $\approx 10^{-19}$ (see Figure 2**b**).

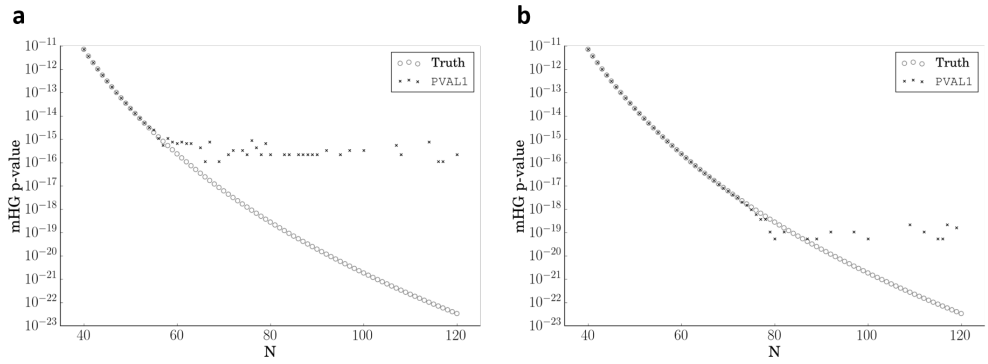

Figure 2. Numerical accuracy of `PVAL1`. Lists of varying length N ($N \in \{40, 41, ..., 120\}$), each consisting of exactly 20 1's followed by only 0's, were generated, and the mHG p-value for each list was calculated using `PVAL1`. Missing values correspond to cases where `PVAL1` returned a value of 0 or lower due to limited floating point accuracy. **a** Python implementation using the 64-bit "double-precision" data type. **b** Cython implementation using the 80-bit "extended precision" data type.

Motivated by these limitations, I sought to design an algorithm for calculating the XL-mHG p-value $p^{mHG}_{X,L}(\boldsymbol{v})$ that would not require filling in the entire dynamic programming table, and avoid numerical inaccuracies in cases where the true p-value is very small. I realized that both of these limitations result from the fact that `PVAL1` requires the computation of $\pi(\boldsymbol{v})$. If we could directly count the fraction of paths traversing $\mathcal{R}(\boldsymbol{v})$ (instead of calculating the opposite, and then substracting that number from 1), this would allow us to stop the algorithm once we are confident that we have discovered all configurations in $\mathcal{R}(\boldsymbol{v})$, and it would avoid substracting a very small number from 1.0 for highly significant tests (instead, we would add several small numbers that are close to 0, where the density of representable numbers is much higher). I first made the following observation: In the visual representation of $\mathcal{M}^{(N,K)}$ as a $(K + 1) \times (W + 1)$ grid (see Figure 1), paths can only enter $\mathcal{R}(\boldsymbol{v})$ "from below". To see this, we first introduce the following lemma:

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1962v1 | CC-BY 4.0 Open Access | rec: 15 Apr 2016, publ: 15 Apr 2016

Lemma 1 (Monotonicity property of the hypergeometric p-value)**.** *For all* $n < N$ *and* $k \leq \min(\{n, K\})$, $p^{HG}(k;\, N, K, n) < p^{HG}(k;\, N, K, n+1)$.

Proof. $p^{HG}(k;\, N, K, n+1)$ is the probability of having k or more successes among $n+1$ draws. We can represent "k or more successes among $n+1$ draws" as the union of two mutually exclusive events A and B, so that $p^{HG}(k;\, N, K, n+1) = \Pr(\mathrm{A} \cup \mathrm{B}) = \Pr(\mathrm{A}) + \Pr(\mathrm{B})$. Event A: "$k$ or more successes among n draws". Event B: "a successful draw, conditional on exactly $k-1$ successes among n draws". We then have $\Pr(\mathrm{A}) = p^{HG}(k;\, N, K, n)$, and $\Pr(\mathrm{B}) > 0$. Therefore, $p^{HG}(k;\, N, K, n+1) > p^{HG}(k;\, N, K, n)$. □

Since $\mathcal{R}_{\mathrm{X,L}}(\boldsymbol{v})$ is defined as the set of all configurations whose hypergeometric p-value is equal to or smaller than fixed value (namely, $s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})$), we know from Lemma 1 that when a configuration $\mu_{(n,k)}$ is in $\mathcal{R}_{\mathrm{X,L}}(\boldsymbol{v})$, then so is $\mu_{(n-1,k)}$, its "left neighbor" in the grid representation. Therefore, the only way for a path to *enter* $\mathcal{R}_{\mathrm{X,L}}(\boldsymbol{v})$ is "from below". In this case, $\mu_{(n,k)} \in \mathcal{R}_{\mathrm{X,L}}(\boldsymbol{v})$, but $\mu_{(n-1,k-1)} \notin \mathcal{R}_{\mathrm{X,L}}(\boldsymbol{v})$. We can refer to configurations for which this is true as "entry points" into $\mathcal{R}_{\mathrm{X,L}}(\boldsymbol{v})$ (see Figure 3). The basis of our new algorithm is then to calculate what fraction of paths enter $\mathcal{R}_{\mathrm{X,L}}(\boldsymbol{v})$ from below at all entry points, and then report the sum of all these fractions as the (XL-)mHG pvalue. However, since paths can exit and re-enter $\mathcal{R}_{\mathrm{X,L}}(\boldsymbol{v})$, we need to ensure that we only count each path once, when it enters $\mathcal{R}_{\mathrm{X,L}}(\boldsymbol{v})$ for the first time. In other words, we must only consider paths that have never traversed $\mathcal{R}_{\mathrm{X,L}}(\boldsymbol{v})$ before. Coincidentally, this is the exact same quantity that `PVAL1` uses in order to calculate $\pi(\boldsymbol{v})$ (see above).

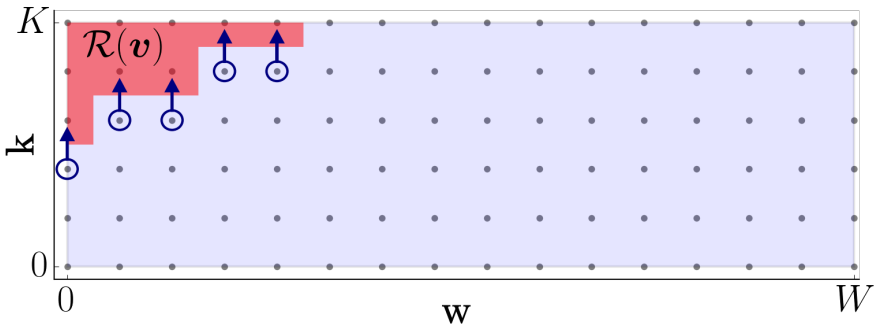

Figure 3. Idea behind `PVAL2`, illustrated using the example from Figure 1. At each "entry point" into $\mathcal{R}(\boldsymbol{v})$ (arrow tips), we calculate the fraction of paths entering from the configuration below (circles). However, in order to avoid counting paths more than once (some may exit and then re-enter $\mathcal{R}(\boldsymbol{v})$), we must base our calculation on only those paths that have not previously traversed $\mathcal{R}(\boldsymbol{v})$. This is the exact same quantity used by `PVAL1` to calculate $\pi(\boldsymbol{v})$. The (XL-)mHG p-value corresponds to total fraction of entering paths.

I refer to this new algorithm as `PVAL2`. Due to its reliance on the same recurrence relation as `PVAL1`, it requires only surprisingly small modifications to `PVAL1`. These are illustrated on a simplefied version of `PVAL2`, which relies on a separate routine to determine $\mathcal{R}(\boldsymbol{v})$ (see pseudocode below). The full algorithm is provided in Appendix A.

To test whether `PVAL2` exhibits better numerical stability than `PVAL1`, I repeated the experiment shown in Figure 2 for `PVAL2`. As can be seen in Figure 4, the new algorithm is able to calculate p-values much smaller than 10^{-16}, and numerical errors are no longer apparent.

To determine how the modifications introduced in `PVAL2` affect the runtime of the algorithm, I performed several benchmarks. As discussed above, I expected `PVAL2` to run significantly faster for lists containing significant enrichment, and for $L < N$. The benchmark results confirm this expectation, and show that in lists without enrichment and $L = N$, `PVAL2` runs only marginally faster than `PVAL1` (see Figure 5).

Bounds for the XL-mHG p-value

Eden, Lipson, et al. (2007) described a lower and an upper bound for the mHG p-value, both of which I review in Appendix B. The mHG test statistic $s^{\mathrm{mHG}}(\boldsymbol{v})$ itself serves as a lower bound for $p^{\mathrm{mHG}}(\boldsymbol{v})$ (see Theorem 1). I found that the lower bound applies unchanged to the XL-mHG p-value (see Theorem 4 in Appendix C).

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1962v1 | CC-BY 4.0 Open Access | rec: 15 Apr 2016, publ: 15 Apr 2016

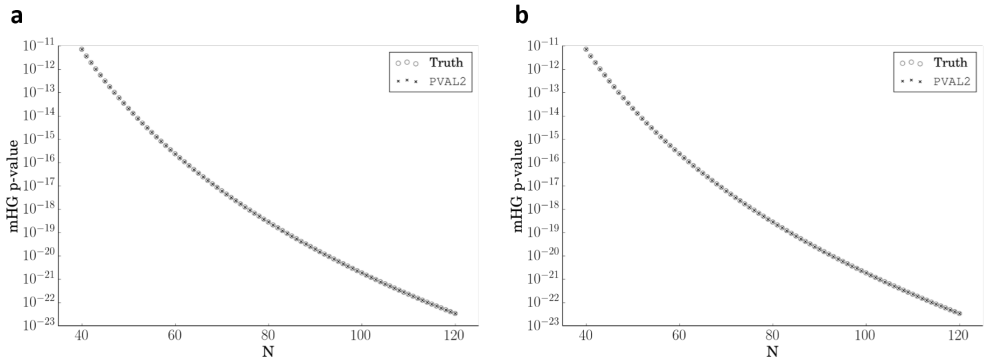

Figure 4. Numerical accuracy of `PVAL2`. Shown are results of an experiment as described in Figure 2, but conducted using `PVAL2`. **a** Python implementation using the 64-bit "double-precision" data type. **b** Cython implementation using the 80-bit "extended precision" data type.

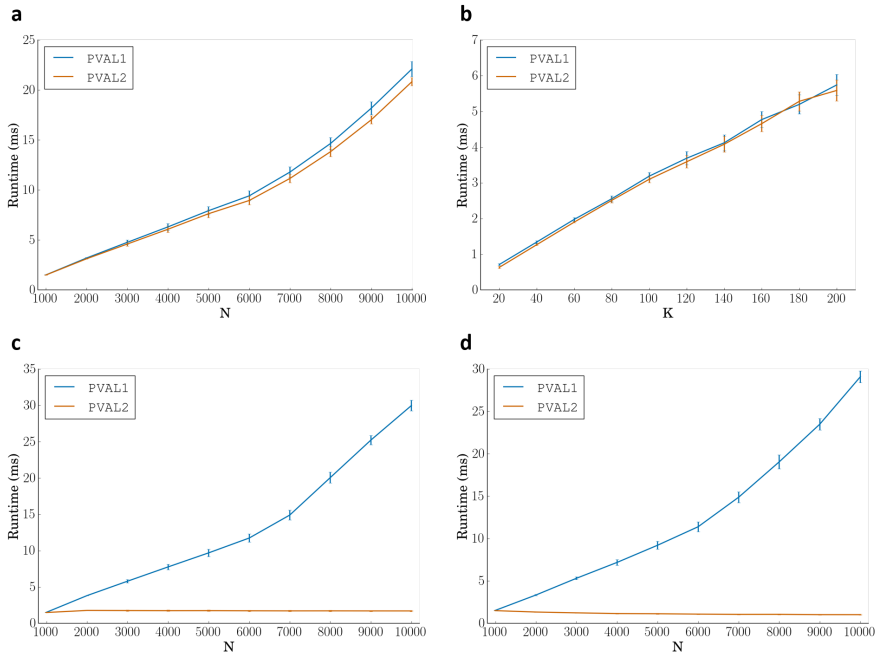

Figure 5. Comparison of runtimes of `PVAL1` and `PVAL2`. For each benchmark and each set of parameters, 100 lists were generated independently, and both algorithms were used to calculate the (XL)-mHG p-value for those lists. Shown are the means and standard deviations (error bars) over the 100 runs. All benchmarks were conducted using randomly generated lists where the positions of the 1's were sampled uniformly from all positions (except for **c**). **a** Benchmark using fixed K=100, for variable N (X=1; L=N). **b** Benchmark using fixed N=2,000, for variable K (X=1;L=N). **c** Benchmark for lists with enrichment, using fixed K=100 and variable N (X=1; L=N). The positions of the 1's were sampled uniformly from only the top 1,000 positions. **d** Benchmark using fixed K=100 and L=1,000, for variable N (X=1).

In the construction of their proof for the upper bound, Eden, Lipson, et al. (2007) introduced the notion of special cutoffs n_k, for $k \in \{1, ..., K\}$, that correspond to the lowest cutoffs so that $p^{\text{HG}}(k;\ N, K, n_k) \leq s^{\text{mHG}}(\boldsymbol{v})$. This allowed the authors to represent the mHG p-value as a union of K events, which correspond to observing a hypergeometric p-value equal to or smaller than $s^{\text{mHG}}(\boldsymbol{v})$ at the respective n_k. By applying a union bound, Eden, Lipson, et al. (2007) found that an upper bound for $p_n^{\text{HG}}(\boldsymbol{v})$ is given by $K * s^{\text{mHG}}(\boldsymbol{v})$ (see Theorem 3). Depending on the choice of the parameters X and L, not all k need to be considered in

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1962v1 | CC-BY 4.0 Open Access | rec: 15 Apr 2016, publ: 15 Apr 2016

Algorithm 1: `PVAL2-SIMPLE`, an improved algorithm to calculate $p^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})$ in $\mathcal{O}(N^2)$. This simplified version of `PVAL2` uses a separate routine to determine $\mathcal{R}(\boldsymbol{v})$, and does not handle comparisons of floating point variables properly. See Algorithm 5 in Appendix A for `PVAL2`.

Input: stat=$s^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})$, N, K, X, L
Output: pval=$p^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})$

```
R ← Algorithm 2 (stat, N, K, X, L) from Wagner (2015b)
pval ← 0.0
table ← empty (K + 1) × (W + 1) array of floats
table[0, 0] ← 1.0
W ← N-K
for n = 1 to L do
    k ← min(n,K)
    w = n-k
    // check whether we have seen all of R(v)
    if k = K and R[k, w] = 0 then
        break
    end if
    while k ≥ 0 and w ≤ W do
        if R[k, w] = 1 then
            table[k, w] ← 0.0
            // check if this is an entry point into R(v) (entering is only possible "from below")
            if k > 0 and R[k-1, w] = 0 then
                pval ← pval + (table[k-1, w] * (K-k+1)/(N-n+1))
            end if
        else if w > 0 and k > 0 then
            table[k, w] ← table[k, w-1] * (W-w+1)/(N-n+1) +
                  table[k-1, w] * (K-k+1)/(N-n+1)
        else if w > 0 then
            table[k, w] ← table[k, w-1] * (W-w+1)/(N-n+1)
        else if k > 0 then
            table[k,w] ← table[k-1, w] * (K-k+1)/(N-n+1)
        end if
        w ← w + 1
        k ← k - 1
    end while
end for
return pval
```

the corresponding expression for XL-mHG p-value, since it is required that $k \geq X$ and $k \leq \min\{K, L\}$. I refer to the set of k that need to be considered as $\mathcal{K}^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})$. Only the n_k for Therefore, some of the events in are by definition excluded from the union, which results in a tighter bound of $((\min\{K, L\}-X+1)s^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})$ (see Theorem 5 in Appendix C).

A closer examination of the proof for Theorem 5 suggests that depending on X, L, and $s^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})$, the actual number of events in the union of Equation (6) can be smaller than $(\min\{K, L\} - X + 1)$. This statement can be made more precise using the following two definitions:

$$k^{\min}_{\text{X,L}}(\boldsymbol{v}) := \min\{k : k \geq X, p^{\text{HG}}(k;\, N, K, k) \leq s^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})\}$$

$$k^{\max}_{\text{X,L}}(\boldsymbol{v}) := \begin{cases} \min\{k : n_k \geq L\}, & \text{if } n_K \geq L \\ K & \text{otherwise} \end{cases}$$

The number of unique events in Equation (6) is exactly $(k^{\max}_{\text{X,L}}(\boldsymbol{v}) - k^{\min}_{\text{X,L}}(\boldsymbol{v}) + 1)$, resulting in the following bound:

$$p^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v}) \leq (k^{\max}_{\text{X,L}}(\boldsymbol{v}) - k^{\min}_{\text{X,L}}(\boldsymbol{v}) + 1)s^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v}) \qquad (\mathcal{O}(N) \text{ upper bound for the XL-mHG p-value})$$

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1962v1 | CC-BY 4.0 Open Access | rec: 15 Apr 2016, publ: 15 Apr 2016

Let $b^{\text{mHG}}_{X,L}(\boldsymbol{v}) := (k^{\max}_{X,L}(\boldsymbol{v}) - k^{\min}_{X,L}(\boldsymbol{v}) + 1)s^{\text{mHG}}_{X,L}(\boldsymbol{v})$. It turns out that $k^{\min}_{X,L}(\boldsymbol{v})$ and $k^{\max}_{X,L}(\boldsymbol{v})$, and therefore $b^{\text{mHG}}_{X,L}(\boldsymbol{v})$, can be obtained in $\mathcal{O}(N)$. To do so, I designed the algorithm `PVAL-BOUND` (see Algorithm 6 in Appendix A). Therefore, in cases where we need to determine whether $p^{\text{mHG}}_{X,L}(\boldsymbol{v})$ is equal to or smaller than a pre-specified significance threshold α, we can first calculate the original upper bound in $\mathcal{O}(1)$. If this bound is larger than $p^{\text{mHG}}_{X,L}(\boldsymbol{v})$, we can invoke `PVAL-BOUND` to calculate a potentially tighter upper bound in $\mathcal{O}(N)$. Only if this value is still larger than $p^{\text{mHG}}_{X,L}(\boldsymbol{v})$ do we need to calculate the exact value of $p^{\text{mHG}}_{X,L}(\boldsymbol{v})$ in $\mathcal{O}(N^2)$ (using `PVAL2`). This procedure is summed up in `PVAL-THRESH` (see Algorithm 2).

Algorithm 2: `PVAL-THRESH`— Efficiently determine whether $p^{\text{mHG}}_{X,L}(\boldsymbol{v}) \leq \alpha$.

Input: thresh=α, stat=$s^{\text{mHG}}_{X,L}(\boldsymbol{v})$, N, K, X, L
Output: `TRUE` if $p^{\text{mHG}}_{X,L}(\boldsymbol{v}) \leq$ thresh, `FALSE` otherwise

```
if stat > α then
    // using lower bound
    return FALSE
else if (MIN(K,L) - X + 1) * s^mHG_{X,L}(v) < thresh then
    // using upper bound
    return TRUE
end if
// calculate tighter bound in O(N)
bound ← PVAL-BOUND(stat, N, K, X, L)
if bound ≤ thresh then
    return TRUE
end if
// calculate exact p-value in O(N^2)
pval ← PVAL2(stat, N, K, X, L)
if pval ≤ thresh then
    return TRUE
end if
return FALSE
```

The mHG test is much more powerful than Kolmogorov-Smirnov (KS) test in detecting certain types of enrichment

In biology, specifically in the field of genomics, both the Kolmogorov-Smirnov (KS) test (Subramanian et al. 2005) and the (XL-)mHG test (Eden, Navon, et al. 2009; Wagner 2015a) have been applied to detect enrichment of gene sets in ranked lists of genes. However, it is unclear which test is more powerful in the scenariors typically encountered in those analyses.

To address this question, I designed three simple experiments in which I simulated lists of length N=8,000, roughly corresponding to the number of genes typically expressed in a given cell type or tissue. In each experiment, I simulated varying levels of enrichment, corresponding to an overrepresentation of 1's among the first n elements of the list. For each simulated list, I applied both tests and asked whether it was significant at a significance level of $\alpha = 10^{-6}$, which approximately corresponds to a significance level of 0.05 after Bonferroni correction for multiple testing of thousands of gene sets (Wagner 2015a). The experiments differed by the choice of n parameter, as well as the total number of 1's in the list (K).

As shown in Figure 6, the mHG outperformed the KS test in all three experiments, with the differences being greatest in the first case, where n and K were very small. In that experiment, the mHG test achieved 100% power for 240-fold enrichment (corresponding to three out of five 1's being present among the first 20 elements of the list), whereas the KS test only achieved the same power for 400-fold enrichment (i.e., when all five 1's were present among the first 20 elements). In contrast, for large n, the difference was much smaller in terms of the absolute fold enrichment: The mHG test achieved 100% power for 2.2-fold enrichment, and the KS test for 2.6-fold enrichment.

DISCUSSION

The results presented here extend the work of Eden, Lipson, et al. (2007), who introduced the nonparametric mHG test statistic, developed the dynamic programming approach for calculating its p-value, and

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1962v1 | CC-BY 4.0 Open Access | rec: 15 Apr 2016, publ: 15 Apr 2016

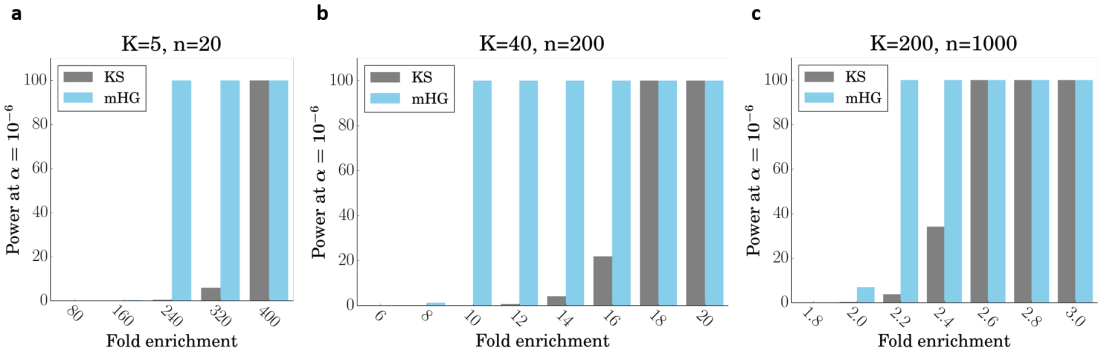

Figure 6. Power of the mHG test in comparison to the Komologorov-Smirnov (KS) test. Lists containing varying levels of fold enrichment within the "top of the list" (specified by the n parameter) were simulated. For each list, it was assessed whether the tests were significant at the level $\alpha = 10{-}6$. Plots show the estimated power (fraction of significant tests), calculated based on 1,000 simulations for each fold enrichment value. **a**-c show the results of three experiments for different choices of K and n, as indicated above each panel.

described both upper and lower bounds. This work is concerned with describing similar results for the XL-mHG test, which represents a semiparametric generalization of the mHG test (Wagner 2015b). I have shown that a small but significant modification to the approach proposed by Eden, Lipson, et al. (2007) for calculating p-values results in better numerical stability, and leads to significant speed-ups when enrichment is present, or when $L < N$. Furthermore, I have described lower and upper bounds for the XL-mHG p-value, and proposed an additional $\mathcal{O}(N)$-bound that is tighter than the $\mathcal{O}(1)$-bound. Finally, I have shown that the (XL-)mHG test is preferable to the KS test for detecting certain types of enrichment, which are of particular relevance in biological (genomic) applications.

The motivation for this work was to encourage the more widespread adoption of the XL-mHG test in biological and other applications; by providing a rigorous and transparent treatment of the statistical and algorithmic aspects of the XL-mHG test, and by providing an efficient, tested, and free open-source implementation in the form of the `xlmhg` Python/Cython package (see https://github.com/flo-compbio/xlmhg). There are multiple features that can make the XL-mHG an attractive choice: The semiparametric nature of the test, i.e., the nonparametric approach of the mHG test in combination with the X and L parameters, provide an efficient way to tailor the test to the kind of enrichment that is of interest in a particular application. In certain scenarios, the XL-mHG test is much more sensitive than the KS test, but the X parameter provides a means for trading off some of the sensitivity for increased robustness. Through its reliance on the hypergeometric distribution, the XL-mHG test also has the property that the exact distribution of 1's below n^*, the cutoff giving rise to the value of the test statistic, is not important. In other words, the test is robust to outliers, which is especially desirable when some of the 1's are expected to represent "false positives". Finally, efficient algorithms and implementations allow an individual test to be performed in only a few milliseconds, even for large values of N.

METHODS

Implementation of `PVAL1` and `PVAL2`

The `PVAL1` and `PVAL2` algorithms were implemented twice, once in Python and once in Cython. The Cython programming language is a superset of Python that compiles to C code. When type declarations are added, the generated C code can avoid (slow) calls to the Python C-API, resulting in speeds comparable to that of native C programs. At the same time, results (in this case, XL-mHG p-values) can easily be passed back into Python code. The Cython implementation uses the `long double` variable type for all floating point operations. Most compilers implement this type using 80-bit "extended precision", with the notable exception of the Microsoft Visual C++ compiler[2]. Therefore, the Cython implementation is much faster and more accurate compared to the Python implementation. However, the Python implementation

[2]see https://en.wikipedia.org/wiki/Long_double

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1962v1 | CC-BY 4.0 Open Access | rec: 15 Apr 2016, publ: 15 Apr 2016

does not require compilation. By default, all implementations use a relative tolerance of 10^{-12} (see Algorithm 3), which was found to give accurate results.

Testing `PVAL2` and `PVAL-BOUND` for correctness

Since the algorithms proposed here are not entirely trivial, it can be difficult to establish their correctness. I therefore implemented test procedures for the Cython implementations of `PVAL2` and `PVAL-BOUND` that rely on alternative algorithms for calculating the XL-mHG p-value and the $\mathcal{O}(N)$-bound, respectively. I then compared the results of those alternative algorithms to those obtained with `PVAL2` and `PVAL-BOUND`. I found that the results were identical for all cases tested, which led me to conclude that both algorithms are in fact correct. The tests were implemented and executed as *unit tests* within the framework provided by the `pytest` Python package (version 2.8.5), and are included in the `xlmhg` Python/Cython package, under `tests/test_correct_pval.py` and `tests/test_correct_bound.py` (see https://github.com/flo-compbio/xlmhg).

More specifically, to test the correctness of `PVAL2`, I chose $N = 50$ and $K = 10$, and generated a reference table of hypergeometric p-values $p_{(n,k)}$, for all possible hypergeometric configurations (i.e., for all possible n and k), using the `scipy.stats.hypergeom.sf` function from the `scipy` Python package (version `0.17.0`). Then, for each possible combination of X and L ($X, L \in \{1, ..., N\}$), I used the reference table to obtain all possible values of the XL-mHG test statistic $s^{\text{mHG}}_{X,L}(\boldsymbol{v})$ (by setting $p_{(n,k)} = 1$ for all $k < X$ and $n > L$). For each value of the test statistic, I then calculated the XL-mHG p-value $p^{\text{mHG}}_{X,L}(\boldsymbol{v})$ using both `PVAL1` and `PVAL2`, and tested whether the output of both algorithms was identical, within a margin of error due to the numerical errors discussed in the results section. Specifically, I used the `IS_EQUAL` algorithm with a relative tolerance of 10^{-8} to determine if the two results were identical. In total, $56,400$ such comparisons were conducted, and the p-values were found to be identical in all cases.

To test the correctness of `PVAL-BOUND`, I implemented another testing procedure, again choosing $N = 50$ and $K = 10$. To obtain an alternative algorithm for calculating $b^{\text{mHG}}_{X,L}(\boldsymbol{v})$, I designed a simpler version of `PVAL-BOUND` that assumes that all $p_{(n,k)}$ are already known. In addition to testing whether both algorithms returned identical values for $b^{\text{mHG}}_{X,L}(\boldsymbol{v})$, I also tested whether those values were in fact equal to or larger than $p^{\text{mHG}}_{X,L}(\boldsymbol{v})$, and whether in all cases $b^{\text{mHG}}_{X,L}(\boldsymbol{v})$ was equal to or smaller than the $\mathcal{O}(1)$-bound (i.e., $(\min\{K, L\} - X + 1)s^{\text{mHG}}_{X,L}(\boldsymbol{v})$). Again, a total of $56,400$ tests were conducted, and all tests passed. Furthermore, in $34,858$ out of the $56,400$ cases, $b^{\text{mHG}}_{X,L}(\boldsymbol{v})$ was found to be strictly smaller than $(\min\{K, L\} - X + 1)s^{\text{mHG}}_{X,L}(\boldsymbol{v})$, indicating that the $\mathcal{O}(N)$-bound is indeed tighter than the $\mathcal{O}(1)$-bound.

Assessing the numerical stability of `PVAL1` and `PVAL2`

All lists tested in Figures 2 and 4 consisted of 20 1's, followed by a varying number of 0's. Obviously, we have $n^* = 20$ for all those lists. In other words, the best cutoff for all those lists is 20, so that the "top of the list" contains all 1's and no 0's, and the mHG test statistic is the hypergeometric p-value at that cutoff. Due to the special structure of those lists, calculation of the true mHG p-value $p^{\text{mHG}}(\boldsymbol{v})$ is trivial as well. Since for given N and K, no other list exhibits an equally good minimum hypergeomtric p-value, $p^{\text{mHG}}(\boldsymbol{v})$ corresponds to $1/|\mathcal{V}^{(N,K)}| = s^{\text{mHG}}(\boldsymbol{v})$.

Benchmarks of `PVAL1` and `PVAL2`

The benchmarks of `PVAL1` and `PVAL2` were carried out using the `repeat` function from Python's `timeit` module. For each randomly generated list, $s^{\text{mHG}}_{X,L}(\boldsymbol{v})$ was pre-calculated, and then the runtime of the functions `get_xlmhg_pval1` and `get_xlmhg_pval2` from the Cython module (`xlmhg.mhg_cython`) were measured. The measurements were taken for 10 identical calls of the function (number=10), and the minimum runtime over three tests (repeat=3) was recorded. To obtain the final runtime, this minimum was divided by the number of calls (10).

Power comparison between the mHG test and the KS test

For each experiment (i.e., each choice of K and n), and each fold change value f, I generated random lists as follows: First, I calculated the number of 1's within the "top of the list" (i.e., above the n'th cutoff) as $k = f * (n/N)$ (the fold enrichment values were chosen in a way that would result in integer numbers). I then used the `numpy.random.choice` function from the `numpy` Python package (version 1.10.4) to sample k ranks from $\{1, ..., n\}$ without replacement. I then set the elements at those ranks to 1. I then used the same function to sample $K - k$ ranks from $\{n + 1, ..., N\}$ without replacement,

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1962v1 | CC-BY 4.0 Open Access | rec: 15 Apr 2016, publ: 15 Apr 2016

and set the elements of at those ranks to 1. All elements were set to 0. I repeated this procedure 1,000 times, to generate 1,000 random lists. I then applied both the mHG test and the KS test to each list, and tested whether the p-values were equal to or smaller than 10^{-6}. For the KS test, I gave the list of cutoffs corresponding to the 1's to the `scipy.stats.kstest` function from the `scipy` Python package (version 0.17.0), and also specified the following arguments: `alternative='greater'`, `mode='approx'`, and `cdf=rank_cdf`, where `rank_cdf` is a function that returns n/N for each n. Jupyter notebooks with the code for the simulations will be made available in a separate GitHub repository, at https://github.com/flo-compbio/xlmhg-paper.

ACKNOWLEDGMENTS

I would like to thank Dr. Sandeep Dave for his support.

REFERENCES

Eden, Eran, Doron Lipson, et al. (2007). "Discovering motifs in ranked lists of DNA sequences". In: *PLoS Computational Biology* 3.3, e39. DOI: 10.1371/journal.pcbi.0030039.

Eden, Eran, Roy Navon, et al. (2009). "GOrilla: a tool for discovery and visualization of enriched GO terms in ranked gene lists". In: *BMC Bioinformatics* 10, p. 48. DOI: 10.1186/1471-2105-10-48. URL: http://www.ncbi.nlm.nih.gov/pubmed/19192299 (visited on 12/28/2011).

Subramanian, Aravind et al. (2005). "Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles". In: *Proceedings of the National Academy of Sciences of the United States of America* 102.43, pp. 15545–15550. DOI: 10.1073/pnas.0506580102.

Wagner, Florian (2015a). "GO-PCA: An Unsupervised Method to Explore Gene Expression Data Using Prior Knowledge". In: *PloS One* 10.11, e0143196. DOI: 10.1371/journal.pone.0143196.

— (2015b). "The XL-mHG Test For Enrichment: A Technical Report". In: *arXiv:1507.07905 [stat]*. arXiv: 1507.07905. URL: http://arxiv.org/abs/1507.07905 (visited on 08/04/2015).

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1962v1 | CC-BY 4.0 Open Access | rec: 15 Apr 2016, publ: 15 Apr 2016

Appendices

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1962v1 | CC-BY 4.0 Open Access | rec: 15 Apr 2016, publ: 15 Apr 2016

A ALGORITHMS

A.1 Pseudocode for PVAL2

We first describe two auxiliary algorithms, IS_EQUAL and HGP, and then describe PVAL2.

Algorithm 3: IS_EQUAL— Test whether two floating point numbers should be considered equal.

Input: a, b, tol (relative tolerance)
Output: TRUE or FALSE

```
if a = b or |a-b| ≤ tol * MAX(|a|, |b|) then
    return TRUE
else
    return FALSE
end if
```

Algorithm 4: HGP— Calculate hypergeometric p-value $p_n^{\text{HG}}(\boldsymbol{v})$ when $f(k;\ N, K, n)$ is already known.

Input: N, K, n, k, p=$f(k;\ N, K, n)$
Output: pval=$p_n^{\text{HG}}(\boldsymbol{v})$

```
pval ← p
while k < MIN(K, n) do
    p ← p * ((n-k)*(K-k)) / ((k+1)*(N-K-n+k+1))
    pval ← pval + p
    k ← k + 1
end while
return pval
```


PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1962v1 | CC-BY 4.0 Open Access | rec: 15 Apr 2016, publ: 15 Apr 2016

Algorithm 5: PVAL2— Improved algorithm to calculate $p^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})$ in $\mathcal{O}(N^2)$.

Input: N, K, X, L (X, L $\in \{1, ..., N\}$), stat=$s^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})$, tol (relative tolerance)
Output: pval=$p^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})$

```
pval ← 0
W ← N-K
table ← empty (K + 1) × (W + 1) array of floats
table[0, 0] ← 1
p_start ← 1
for n = 1 to L do
    if K ≥ n then
        k = n
        p_start = p_start * (K-n+1)/(N-n+1)
    else
        k = K
        p_start = p_start * n/(n-K)
    end if
    p = p_start
    hgp = p
    w = n-k
    if k = K and (hgp > s and not IS_EQUAL(hgp, s, tol)) then
        // we're not in R(v), even though k = K
        // this means we've seen all of R(v), so we're done
        break
    end if
    while k ≥ X and w ≤ W and (hgp < stat or IS_EQUAL(hgp, stat, tol)) do
        // we're in R(v), so π_(n,k)(v) = 0
        table[k, w] ← 0
        // check if this is an entry point into R(v) (entering is only possible "from below")
        if table[k-1, w] > 0 then
            // calculate the fraction of paths entering (only those that have never entered R(v) before),
            // then add that number to pval
            pval ← pval + (table[l-1, w] * (K-k+1) / (N-n+1))
        end if
        p ← p * (k*(N-K-n+k)) / ((n-k+1)*(K-k+1))
        hgp ← hgp + p
        w ← w + 1
        k ← k - 1
    end while
    // we left R(v), now calculate π_(n,k)(v) for the remaining configurations for cutoff n
    while k ≥ 0 w ≤ W do
        if k = 0 then
            table[k, w] ← table[k, w-1] * (W-w+1)/(N-n+1)
        else if w = 0 then
            table[k, w] ← table[k-1, w] * (K-k+1)/(N-n+1)
        else
            table[k, w] ← table[k, w-1] * (W-w+1)/(N-n+1) +
                  table[k-1, w] * (K-k+1)/(N-n+1)
        end if
        w ← w + 1
        k ← k - 1
    end while
end for
return pval
```


PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1962v1 | CC-BY 4.0 Open Access | rec: 15 Apr 2016, publ: 15 Apr 2016

A.2 Pseudocode for PVAL−BOUND

Algorithm 6: PVAL−BOUND— Calculate an upper bound for the XL-mHG p-value in $\mathcal{O}(N)$.

Input: N, K, X, L (X, L $\in \{1, ..., N\}$), stat=$s^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})$, tol (relative tolerance)
Output: $b^{\text{mHG}}_{\text{X,L}}(\boldsymbol{v})$

```
if stat = 1 then
    return 1
else if X > K or X > L then
    return 0
end if
min_KL ← MIN(K,L)
k_min ← 0
p ← 1.0
n = 1
while (n ≤ K or (p ≤ stat or IS_EQUAL(p, stat, tol)) and n ≤ L do
    if n ≤ K then
        k ← n
        p ← p * ((K-n+1) / (N-n+1))
        if k < X or (p > stat and not IS_EQUAL(p, stat, tol)) then
            k_min ← n
        end if
    else
        k ← K
        p ← p * (n / (n-K))
    end if
    n ← n + 1
end while
if k_min = min_KL then
    // R is empty (this never happens for valid s^mHG_X,L(v))
    return 0
end if
k_min ← k_min + 1
if n ≤ L or (n = L+1 and p ¿ stat and not IS_EQUAL(p, stat, tol)) then
    // we left R_X,L(v) at or before reaching the L'th cutoff ⟹ k^max_X,L(v) = K
    return MIN((K-k_min+1)*stat, 1)
end if
// we did not leave R_X,L(v) — "go down the diagonal" until we step out of R_X,L(v)
n ← n - 1
k ← MIN(n, K)
hgp ← p
while hgp ≤ stat or IS_EQUAL(hgp, stat, tol) do
    p ← p * ((k*(N-K-n+k)) / ((n-k+1)*(K-k+1)))
    hgp ← hgp + p
    k ← k - 1
end while
// now we left R_X,L(v)
k_max ← k+1
return MIN((k_max-k_min+1)*stat, 1)
```


PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1962v1 | CC-BY 4.0 Open Access | rec: 15 Apr 2016, publ: 15 Apr 2016

B REVIEW OF BOUNDS FOR THE MHG P-VALUE

In this section, I will review the bounds for the mHG p-value that were first described by Eden, Lipson, et al. (2007).

Theorem 1 (Lower bound for the mHG p-value)**.** *For any* $v \in \mathcal{V}^{(N,K)}$, $p^{mHG}(\boldsymbol{v}) \geq s^{mHG}(\boldsymbol{v})$.

Proof. Recall that V represents a list drawn uniformly at random from $\mathcal{V}^{(N,K)}$. Let $P_n^{\text{HG},0}$ be the hypergeometric p-value of V for the cutoff n. From the definition of the mHG test statistic, it follows that:

$$\begin{aligned} p^{\text{mHG}}(\boldsymbol{v}) &= \Pr\left(S^{\text{mHG},0} \leq s^{\text{mHG}}(\boldsymbol{v})\right) \\ &= \Pr\left(\bigcup_{n=1}^{N}\left(P_n^{\text{HG},0} \leq s^{\text{mHG}}(\boldsymbol{v})\right)\right) \end{aligned} \tag{1}$$

In other words, we know that $S^{\text{mHG},0} \leq s^{\text{mHG}}(\boldsymbol{v})$ whenever there exists at least one cutoff n for which $P_n^{\text{HG},0} \leq s^{\text{mHG}}(\boldsymbol{v})$. We also know that $s^{\text{mHG}}(\boldsymbol{v})$ is attained at some $n = n^*$. We therefore observe the following inequality:

$$\begin{aligned} p^{\text{mHG}}(\boldsymbol{v}) &= \Pr\left(\bigcup_{n=1}^{N}\left(P_n^{\text{HG},0} \leq s^{\text{mHG}}(\boldsymbol{v})\right)\right) \\ &\geq \Pr\left(P_{n^*}^{\text{HG},0} \leq s^{\text{mHG}}(\boldsymbol{v})\right) \end{aligned} \tag{2}$$

By definition of the hypergeometric p-value, $\Pr(P_{n^*}^{\text{HG},0} \leq s^{\text{mHG}}(\boldsymbol{v})) = s^{\text{mHG}}(\boldsymbol{v})$. The theorem therefore follows. □

Theorem 2 (Loose upper bound for the mHG p-value)**.** *For any* $v \in \mathcal{V}^{(N,K)}$, $p^{mHG}(\boldsymbol{v}) \leq N s^{mHG}(\boldsymbol{v})$.

Proof. When we apply a union bound to Equation (1), we have:

$$p^{\text{mHG}}(\boldsymbol{v}) \leq \sum_{n=1}^{N} \Pr(P_n^{\text{HG},0} \leq s^{\text{mHG}}(\boldsymbol{v})) \tag{3}$$

By definition of the hypergeometric p-value, $\Pr(P_n^{\text{HG},0} \leq s^{\text{mHG}}(\boldsymbol{v})) = s^{\text{mHG}}(\boldsymbol{v})$. The theorem then follows follows from Equation (3). □

For the proof of the next bound, we need the following monotonicity property of the mHG p-value.

Theorem 3 (Tighter upper bound for the mHG p-value; LIPSON bound)**.** *For any* $\boldsymbol{v} \in \mathcal{V}^{(N,K)}$, $p^{mHG}(\boldsymbol{v}) \leq K s^{mHG}(\boldsymbol{v})$.

Proof. Given $s^{\text{mHG}}(\boldsymbol{v})$, let $\mathcal{K}^{\text{mHG}}(\boldsymbol{v})$ be the set of all k for which $p^{\text{HG}}(k;\, N, K, k) \leq s^{\text{mHG}}(\boldsymbol{v})$. We know that $\mathcal{K}^{\text{mHG}}(\boldsymbol{v})$ is not empty, since $s^{\text{mHG}}(\boldsymbol{v})$ was attained for some $k = k_{n^*}(\boldsymbol{v})$. Then, for each $k \in \mathcal{K}^{\text{mHG}}(\boldsymbol{v})$, let n_k be the largest value of n for which $p^{\text{HG}}(k;\, N, K, n) \leq s^{\text{mHG}}(\boldsymbol{v})$. This definition makes sense because of the aforementioned monotonicity property (Lemma 1). Let $P_{n_k}^{\text{HG},0}$ be the hypergeometric p-value of V for the cutoff n_k. Then we can represent $p^{\text{mHG}}(\boldsymbol{v})$ as follows:

$$\begin{aligned} p^{\text{mHG}}(\boldsymbol{v}) &= \Pr\left(S^{\text{mHG},0} \leq s^{\text{mHG}}(\boldsymbol{v})\right) \\ &= \Pr\left(\bigcup_{k \in \mathcal{K}^{\text{mHG}}(\boldsymbol{v})}\left(P_{n_k}^{\text{HG},0} \leq s^{\text{mHG}}(\boldsymbol{v})\right)\right) \end{aligned} \tag{4}$$

In other words, we have $S^{\text{mHG},0} \leq s^{\text{mHG}}(\boldsymbol{v})$ whenever the hypergeometric p-value for at least one of the n_k is equal to or smaller than $s^{\text{mHG}}(\boldsymbol{v})$. We can then apply another union bound to Equation (4):

$$p^{\text{mHG}}(\boldsymbol{v}) \leq \sum_{k \in \mathcal{K}^{\text{mHG}}(\boldsymbol{v})} \Pr(P_{n_k}^{\text{HG},0} \leq s^{\text{mHG}}(\boldsymbol{v})) \tag{5}$$

Again, by definition of the hypergeometric p-value, $\Pr(P_n^{\text{HG},0} \leq s^{\text{mHG}}(\boldsymbol{v})) = s^{\text{mHG}}(\boldsymbol{v})$. We have $|\mathcal{K}^{\text{mHG}}(\boldsymbol{v})| \leq K$. The theorem therefore follows from Equation (5).

□

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1962v1 | CC-BY 4.0 Open Access | rec: 15 Apr 2016, publ: 15 Apr 2016

C BOUNDS FOR THE XL-MHG P-VALUE

Theorem 4 (Lower bound for the XL-mHG p-value)**.** *For any* $\boldsymbol{v} \in \mathcal{V}^{(N,K)}$, $p^{mHG}_{\mathrm{X,L}}(\boldsymbol{v}) \geq s^{mHG}_{\mathrm{X,L}}(\boldsymbol{v})$.

Proof. In the trivial case $s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v}) = 1$, we have $p^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v}) = 1$. In the remainder, we therefore treat the case $s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v}) < 1$. Let $s_n(\boldsymbol{v};\, X, L)$ represent the value "contributed" by the n'th cutoff in the calculation of the XL-mHG test statistic:

$$s_n(\boldsymbol{v};\, X, L) = \begin{cases} p^{\mathrm{HG}}_n(\boldsymbol{v}), & \text{if } k_n(\boldsymbol{v}) \geq X \text{and } n \leq L, \\ 1.0 & \text{otherwise} \end{cases}$$

Furthermore, let the random variable S^0_n represent the value of $s_n(\boldsymbol{v};\, X, L)$ for a list drawn uniformly at random from $\mathcal{V}^{(N,K)}$. We then have:

$$\begin{aligned} p^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v}) &= \Pr\left(S^{\mathrm{mHG},0}_{\mathrm{X,L}} \leq s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})\right) \\ &= \Pr\left(\bigcup_{n=1}^{N} \left(S^0_n \leq s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})\right)\right) \end{aligned}$$

Furthermore, we know that since $s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v}) < 1$, the test statistic was attained at some n^*; i.e., $s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v}) = s_{n^*}(\boldsymbol{v};\, X, L)$. Therefore, we have:

$$\Pr\left(\bigcup_{n=1}^{N} \left(S^0_n \leq s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})\right)\right) \geq Pr(S^0_{n^*} \leq s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v}))$$

Since $s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})$ was attained at n^*, we know that $n^* \leq L$. Furthermore, we know that $k_{n^*}(\boldsymbol{v}) \geq X$ and that $p^{\mathrm{HG}}(k;\, N, K, n^*) > s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})$ for any $k < k_{n^*}(\boldsymbol{v})$ (hypergeometric p-values strictly increase with smaller k). Therefore, we have $\Pr(S^0_{n^*} \leq s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})) = \Pr(P^{\mathrm{HG},0}_{n^*} \leq s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})) = s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})$, and therefore $p^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v}) \geq s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})$.

□

Theorem 5 (Upper bound for the XL-mHG p-value)**.** *For any* $\boldsymbol{v} \in \mathcal{V}^{(N,K)}$, $p^{mHG}_{\mathrm{X,L}}(\boldsymbol{v}) \leq (\min\{K, L\} - X + 1) s^{mHG}_{\mathrm{X,L}}(\boldsymbol{v})$.

Proof. In the trivial case $s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v}) = 1$, we have $p^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v}) = 1$. In the remainder, we therefore treat the case $s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v}) < 1$. Let $\mathcal{K}^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})$ be defined as follows:

$$\mathcal{K}^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v}) = \{k : p^{\mathrm{HG}}(k;\, N, K, k) \leq s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v}), k \geq X, k \leq L\}$$

Since $s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v}) < 1$, the test statistic was attained at some cutoff n^*, for some $k = k_{n^*}(\boldsymbol{v})$:

$$p^{\mathrm{HG}}(k_{n^*}(\boldsymbol{v});\, N, K, n^*) = s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})$$

Since $k_{n^*}(\boldsymbol{v}) \leq n^*$, we can rely on Lemma 1 to infer that $k_{n^*}(\boldsymbol{v}) \in \mathcal{K}^{\mathrm{mHG}}(\boldsymbol{v})$, so $\mathcal{K}^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})$ is not empty. We define n_k for all $k \in \mathcal{K}^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})$ as in the proof for Theorem 3 (see Appendix B), and then define and $n'_k = \min\{n_k, L\}$ for all n_k. We can then represent $p^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})$ as:

$$\begin{aligned} p^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v}) &= \Pr\left(S^{\mathrm{mHG},0}_{\mathrm{X,L}} \leq s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})\right) \\ &= \Pr\left(\bigcup_{k \in \mathcal{K}^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})} \left(P^{\mathrm{HG},0}_{n'_k} \leq s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})\right)\right) \end{aligned} \tag{6}$$

We apply a union bound to Equation (6) and observe, as in Theorem 3 (see Appendix B), that $\Pr(P^{\mathrm{HG},0}_{n'_k} \leq s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})) = s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})$. We have $|\mathcal{K}^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})| \leq \min\{K, L\} - X + 1$ events in the union, which means that $p^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v}) \leq (\min\{K, L\} - X + 1) s^{\mathrm{mHG}}_{\mathrm{X,L}}(\boldsymbol{v})$.

□

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1962v1 | CC-BY 4.0 Open Access | rec: 15 Apr 2016, publ: 15 Apr 2016