FWER CONTROL CLOSURE ALGORITHMS FOR E-VALUES

WILL HARTOG AND LIHUA LEI

Abstract. The closure principle is a standard tool for achieving family-wise error rate control in multiple testing settings, given a valid local test of a subset of hypotheses. Working in the setting of the graphical approach, we focus on the weighted average e-value local test, a method of combining hypotheses unique to e-values, which arise naturally in settings of sequential testing. We develop efficient dynamic programming algorithms for computing the closure of this local test using dynamic programming for any directed acyclic graph procedure, for which the closure is computed in $O(n)$. We also give an efficient algorithm for computing this closure for the full graph (weighted Bonferroni), and further extend our results to an additional limited class of cyclic graphs.

1. Introduction and Preliminaries

In analyses that test multiple hypotheses, it is necessary to consider the problem of multiple testing, and control for some notion of false rejection. One such approach is to control the Family-wise Error Rate (FWER), the probability of any false rejection, at a given level $\alpha \in (0, 1)$.

1.1. Closed Testing.

Given null hypotheses $H_1, ..., H_n$, a local test for index subset $I \subset [n]$ tests the hypothesis $H_I = \cap_{i \in I} H_i$ that all the nulls H_i in I are true.

Given a family of valid local tests $\{\phi_I\}_{I \subset [n]}$, i.e. $\mathbb{P}_{H_I}(\phi_I = 1) \leq \alpha$ for all $I \subset [n]$, the "closure" of $\{\phi_I\}_{I \subset [n]}$ rejects H_i, the individual hypothesis, if $\phi_I = 1$ for all $I \ni i$. Any admissible procedure which controls FWER is a closed test [4].

In the p-value literature, the most common local test is weighted Bonferroni, which given p-values $p_1, ..., p_n$, and a set of valid weights for any I, i.e. $\{w_i(I)\}$ with $\sum_{i \in I} w_i(I) \leq 1$, computes $\phi_I = 1$ iff $\min_{i \in I} \frac{p_i}{w_i(I)} \leq \alpha$, which is valid as

$$\mathbb{P}_{H_I}(\phi_I = 1) = \mathbb{P}_{H_I}(\cup_{i \in I} p_i \leq \alpha w_i(I)) \leq \sum_{i \in I} \mathbb{P}_{H_I}(p_i \leq \alpha w_i(I)) \leq \sum_{i \in I} \alpha w_i(I) \leq \alpha$$

by a union bound.

The challenge becomes simplifying the computation from the $2^n - 1$ local tests to an efficient number. The literature includes many such procedures including Holm's procedure, which uses the local test of unweighted Bonferroni.

A larger class of local tests, in which Holm's procedure belongs, is that of the graphical approach, introduced by Bretz et al.[1], in which each hypothesis H_i is assigned an initial α-budget α_i, and the weights $w_i(I)$ are determined by a directed graph with associated transition probabilities. In particular, $w_i(I) = \frac{\alpha_i(I)}{\alpha}$ where $\alpha_i(I) = \alpha_i + \sum_{j \notin I} \alpha_j p_{ji}(I)$, with $p_{ji}(I)$ the probability that the random walk on the given weighted graph starting at j hits i first out of the elements of I.

Date: June 4, 2024.

Key words and phrases. e-values, multiple testing, graphical approach.

Holm's procedure is an example of the graphical approach, with the complete graph and equal weights. The Fallback procedure (source) is an example of the graphical approach with a chain graph. In general, any graph can be used for the purposes of the graphical approach, where its closure can be efficiently computed, with a simple greedy algorithm for determining the rejection set.

1.2. **e-values.** e-values are a recent alternative to p-values as a tool for hypothesis testing and quantifying evidence against the null. For a null hypothesis H_0, an e-value for H_0 is a realization of an e-variable e which has the property that $\mathbb{E}_{H_0}[e] \leq 1$. By Markov's inequality, the test $\phi = 1\{e \geq 1/\alpha\}$ is valid at level α, since by Markov's inequality $\mathbb{P}(e \geq 1/\alpha) \leq \alpha$.

e-values arise naturally in settings, especially in always-valid inference, where the practitioner would like the flexibility of having valid inference at any stopping time. This framework is very relevant for the online data collection in the tech setting, including the setting discussed by Johari et al. [3]. In general, e-values offer more robustness to get valid inference in the presence of various concerns about data-dependent experimental decisions, such as a post-hoc choice of α [2].

An appealing feature of e-values is that they combine easily; the weighted average of e-values is an e-value, as noted by Vovk and Wang [5]. In the setting of devising a local test for closed testing, $e_I = \sum_{i \in I} w_i e_i$ for $\sum_{i \in I} w_i \leq 1$ has that

$$\mathbb{E}_{H_I}[e_I] = \sum_{i \in I} w_i \mathbb{E}_{H_i}[e_i] \leq \sum_{i \in I} w_i \leq 1.$$

Rejecting H_I when $e_I \geq 1/\alpha$ is a valid local test, called weighted e-Bonferroni, so the family of such local tests with weights $w_i(I)$ for subset I gives an alternative closed testing procedure for the same weights as the corresponding procedure with p-values.

In particular, when comparing these two approaches with e-values $e_1, ..., e_n$ and the e-to-p p-values $1/e_1, ..., 1/e_n$, we see that e-Bonferroni is strictly better than p-Bonferroni, since if $\min_{i \in I} p_i/w_i(I) \leq \alpha$, then $w_i(I)e_i \geq 1/\alpha$ for some $i \in I$, and thus $\sum_{i \in I} w_i(I)e_i \geq 1/\alpha$.

In the context of closed testing, given a valid family of local tests, we reject H_i when $\min_{I \ni i}\{\sum_{i \in I} w_i(I)e_i\} \geq 1/\alpha$, so we define as a notion of evidence for H_i when adjusted for multiple hypotheses the adjusted e-value $e_i^* = \min_{I \ni i}\{\sum_{i \in I} w_i(I)e_i\}$. These are easy to report and function analogously to adjusted p-values, which are standard to report.

1.3. **Contributions.** We produce efficient polynomial time algorithms for the closure of the e-Bonferroni local test, with weights determined in various different ways. We compute the adjusted e-values as a notion of evidence for an alternative hypothesis after accounting for multiple hypotheses.

In Section 2 we give the e-value version of Holm's test, which gives each index of the local test equal weight. In Section 3 we give the e-value version of the Fallback procedure, which apportions weight to the indices of a local test based on the transitions of a chain graph, according to the graphical approach. We also extend the result to the graphical approach with a tree graph.

In Section B we generalize to obtain a procedure for the closure of the graphical approach with any DAG.

In Section B we discuss the future potential of producing results for non-DAGs, and extend the DAG results to select non-DAGs which display local DAG structure.

2. e-Holm

We have e-values $\{e_i\}_{i=1}^n$, where we consider our test for the global null on a subset $I \subset [n]$. We reject H_I at level α if $\sum_{i\in I} e_i \geq \frac{|I|}{\alpha}$. This is valid as the variable $\frac{1}{|I|}\sum_{i\in I} e_i$ is a weighted average of e-values and thus is an e-value itself. This is the e-value version of Bonferroni which would be rejecting if the most significant e-value is significant at the α level.

In the language of the rest of this paper, this corresponds to the graphical approach with the complete graph and equal α-budget $\alpha_i = \frac{1}{n}$ for each i.

Thus we define our algorithm as the closure of this subset test.

$$H_i \text{ rejected iff } \sum_{j\in I} e_j \geq \frac{|I|}{\alpha} \text{ for all } I \ni i. \tag{2.1}$$

We sort our e-values $e_{(1)}, ..., e_{(n)}$ in decreasing order (to match the order of the associated p-values), and call $H_{(i)}$ the hypothesis associated with the e-value $e_{(i)}$. We notice that for any $k \leq n-j$, the minimum value of $\sum_{j\in I} e_{(j)}$ with $i \in I$ and $|I| = k+1$ is $I = \{j\} \cup \{n-k+1, n-k+2, ..., n\}$, since otherwise for any $j \neq i < n-k+1$ in our subset we could replace $e_{(j)}$ with $e_{(j')} \leq e_{(j)}$ with $n-k+1 \leq j'$. Thus, for any fixed size $|I|$ we only need to consider the $|I|$ largest e-values, if

For any $|I| = k+1$, we need to check that

$$e_{(i)} + \sum_{j\in I\setminus\{i\}} e_{(j)} \geq \frac{k+1}{\alpha} \Leftrightarrow e_{(i)} \geq \frac{1}{\alpha} + \sum_{j\in I\setminus\{i\}} \left(\frac{1}{\alpha} - e_{(j)}\right).$$

Thus for any i we need to find

$$\max_{I\subset[n]\setminus\{i\}} \left[\frac{1}{\alpha} + \sum_{j\in I} \left(\frac{1}{\alpha} - e_{(j)}\right)\right],$$

in order to find the appropriate threshold to test $e_{(i)}$ at. We see that this is maximized by taking

$$I = \{j \neq i : e_{(j)} < \frac{1}{\alpha}\}.$$

This leads us to the critical index $j' = \min\{j \in [n] : e_{(j)} < \frac{1}{\alpha}\}$. Then we can simplify our test to the following:

$$H_{(i)} \text{ rejected iff } e_{(i)} \geq \frac{1}{\alpha} + \sum_{j\geq j'} \left(\frac{1}{\alpha} - e_{(j)}\right). \tag{2.2}$$

Notice that we do not need to include the condition that $j \neq i$ because if $i \geq j'$ then we know automatically that $e_{(i)} < \frac{1}{\alpha}$ so by 2.1 with $I = \{i\}$ we know that $H_{(i)}$ is not rejected. This algorithm takes $O(n)$ time to compute j' and in so doing compute $C = \sum_{j\geq j'} \left(\frac{1}{\alpha} - e_{(j)}\right)$, after which point it takes another $O(n)$ time to check the lower $i < j'$ hypotheses. Once we reject one hypothesis we know the others below are also rejected. [I can formalize this algorithm in the paper].

This result is nice and useful for quickly determining whether or not the hypotheses are rejected. However it does not compute the adjusted e-value $e_i^* = \min_{I \ni i} \frac{1}{|I|} \sum_{j \in I} e_{(j)}$, which does not depend on the level α as our method does.

To compute the adjusted e-value, we know for any subset size the exact minimum test e-value, so the worst-case runtime of such an algorithm is $O(n^2)$, over each index i and each subset size. Namely, for any number of other elements k, the minimum value is

$$\min_{I \ni i, |I|=k+1} \frac{1}{|I|} \sum_{j \in I} e_{(j)} = \begin{cases} \frac{e_{(i)} + \sum_{j=n-k+1}^{n} e_{(j)}}{k+1} & k \leq n-i \\ \frac{\sum_{j=n-k}^{n} e_{(j)}}{k+1} & k > n-i \end{cases}. \tag{2.3}$$

This value comes from the k lowest non-i e-values, plus $e_{(i)}$, and replacing any of these e-values with another cannot decrease the average. In addition, we know that we can rule out $|I| > n-i+1$, since for any $|I| > n-i+1$ we can take $I' = I \cap \{i, i+1, ..., n\}$ and get $\frac{1}{|I|} \sum_{j \in I} e_{(j)} \geq \frac{1}{|I'|} \sum_{j \in I'} e_{(j)}$, since adding an e-value ($\ell < i$) with $e_{(\ell)} \geq \frac{1}{|I'|} \sum_{j \in I'} e_{(j)}$ cannot decrease the average. Thus, we define $E_k = \sum_{j=n-k+1}^{n} e_{(j)}$ and give the following algorithm which gives the adjusted e-values.

Algorithm 1: e-Holm Adjusted e-values

Input: Vectors $\{e_i\}_{i=1}^n$;
Sort: $\{e_i\}_{i=1}^n$ from highest to lowest as $\{e_{(i)}\}_{i=1}^n$;
Compute: $E_1 = e_{(n)}$, $E_k = E_{k-1} + e_{n-k+1}$ for $k = 2$ to $k = n$;
Initialize: $e_{(n)}^* = e_{(n)}$, $e^* = e_{(n)}, k = 1$;
For: $i = n-1$ to 1,
Do: $e^* = e^* + \frac{e_{(i)} - e_{(i+1)}}{1+k}$;
While $(e^* > e_{(n-k)})$: $e^* = \frac{1+k}{2+k} e^* + \frac{1}{2+k} e_{(n-k)}$; $k+ = 1$.
$e_{(i)}^* = e^*$.
Output: $\{e_{(i)}^*\}_{j=i}^n$.

Because $e_{(i)} \geq e_{(i+1)}$, the minimizer $k_i =_{k<n-i} \{\frac{e_{(i)} + E_k}{1+k}\}$ is decreasing in i, so we know that we can start our search for the minimizer k_i from k_{i+1}, which gives us Algorithm 1.

This algorithm correctly computes our minima as earlier argued, and has runtime $O(n \log n)$, from the sorting. The computation only changes k's value n times, and changes e^* a maximum of $2n$ times.

One interpretation of e-Holm is in the *effective number of hypotheses* for testing each individual hypothesis. For p-Holm, in which the ith smallest p-value is compared to the threshold $\alpha/(m-i+1)$, the effective number of hypotheses is $m-i+1$. However, for e-Holm, this number is always smaller, given by the final value of k in Algorithm 1 before assigning $e_{(i)^*}$.

3. e-Fallback

The Fallback procedure is defined by a fixed sequence of hypotheses, $H_1 \to H_2 \to \cdots \to H_n$ with associated e-values, and α budget $\{\alpha_i\}_{i=1}^n$ with $\sum_{i=1}^n \alpha_i = \alpha$. This is a special case of the later discussed graphical case with a chain graph. A special case of the fallback procedure is the fixed sequence test, which has $\alpha_1 = \alpha$ and all

others $\alpha_i = 0$ for $i > 1$. A valid e-value subset null test is as follows: for $|I| = k$ with $I = \{i_1, ..., i_k\}$ and $i_1 < i_2 < \cdots < i_k$:

$$H_I \text{ rejected if } \sum_{\ell=1}^{k} \left(\sum_{i=i_{\ell-1}+1}^{i_\ell} \alpha_i \right) e_{i_\ell} \geq 1. \tag{3.1}$$

We take $i_0 = 0$ for notational simplicity. This is equivalent to reassigning the α-budget of all intermediate indices before i_ℓ to i_ℓ and then doing a α significance the weighted average e-value with weight α'_{i_ℓ}/α. We give a dynamic programming algorithm with $O(n^2)$ runtime to compute the minimum over $I \ni j$ for each $j \in [n]$.

Algorithm 2: e-Fallback

Input: Vectors $\{e_i\}_{i=1}^n, \{\alpha_i\}_{i=1}^n$;
Initialize: $m_0 = 0$;
Iterate: $m_i = \min_{0 \leq j < i} \{m_j + e_i \sum_{k=j+1}^{i} \alpha_k\}$;
Output: $\{m_i\}_{i=1}^n$.

Then $\{m_i/\alpha\}_{i=1}^n$ are the adjusted e-values where rejecting H_i for $m_i/\alpha \geq 1/\alpha$ controls the FWER at level α.

We now prove that our computation of m_i gives the minimum value of $\alpha e_I = \sum_{\ell=1}^{k} \left(\sum_{i=i_{\ell-1}+1}^{i_\ell} \alpha_i \right) e_{i_\ell}$ over $i \in I$.

Theorem 3.1. *For each i, we have that $m_i = \min_{i \in I \subset [n]} \sum_{\ell=1}^{k} \left(\sum_{i=i_{\ell-1}+1}^{i_\ell} \alpha_i \right) e_{i_\ell} = \min_{i \in I \subset [n]} \alpha e_I$, where $I = \{i_1, ..., i_k\}$.*

Proof. First we note that for any $I \ni i$, $J = I \cap [i]$ has $\alpha e_I \leq \alpha e_J$, since the α-budget for $j > i$ gets assigned nowhere. Taking $i = i_{\ell'}$ WLOG,

$$\alpha e_I - \alpha e_J = \sum_{\ell=\ell'+1}^{k} \left(\sum_{i=i_{\ell-1}+1}^{i_\ell} \alpha_i \right) e_{i_\ell} \geq 0.$$

Thus we only need to show that $m_i = \min_{i \in I \subset [i]} \alpha e_I$. For each $i = i_k$, we consider the various options for i_{k-1}, which can be any element of $[i-1]$ or $k = 1$ so $I = \{i\}$. Thus we can rewrite

$$\min_{i \in I \subset [i]} \alpha e_I = \min_{0 \leq j < i} \left[\min_{j \in J \subset [j]} \alpha e_J + e_i \sum_{k=j+1}^{i} \alpha_k \right],$$

where the minimum for $j = 0$ is 0 by convention. Then we use induction, with our base case of $m_1 = \alpha_1 e_1$, where we assume that $\min_{j \in J \subset [j]} \alpha e_J = m_j$ for all $j \leq i - 1$, in which case we get that $\min_{i \in I \subset [i]} \alpha e_I = m_i$, as intended.

□

This dynamic programming approach performs a minimization over at most n values for each of n indices so has $O(n^2)$ runtime. However, as we will see the next two subsections, we can achieve an $O(n)$ runtime with some algorithmic shortcuts.

3.1. **Reverse Search Algorithm.** In this subsection, we show that there is an algorithm to perform the optimization of e-Fallback without searching all i previous hypotheses each time we compute m_i. This algorithm will serve as a stepping stone to our $O(n)$ stack-based algorithm in Section 3.2, as well as the generalized DAG algorithm in Section B.

Consider $I \ni i$. Again we note that we only need to consider subsets $i \in I \subset [n]$ with $i = \max_{k \in I} k$, since the later indices are not ancestors of i.

Our framework for analyzing this graph in a new way is to, for each $j \leq i$, consider the e-value which will be assigned to its α-budget α_j. We know that this budget will not be lost, and because we have a chain, some e-value e_k with $j \leq k \leq i$ will be assigned to this budget (by having $e_j, ..., e_{k-1}$ excluded from the set I). We will propose a greedy strategy for computing e_i^* which recursively goes backwards along the chain, choosing whether or not to include the node j. Locally, we note that between $I = [i]$ and $J = I \setminus \{j\}$ for $j < i$, $\alpha e_I - \alpha e_J = \alpha_j(e_j - e_{j+1})$. Thus, whether or not we should include j in this local comparison depends on whether $e_j \leq e_{j+1}$. Our algorithm utilizes these local comparisons:

Algorithm 3: Reverse Search for e-Fallback

Input: Vectors $\{e_i\}_{i=1}^n, \{\alpha_i\}_{i=1}^n$;
Initialize: $m_i = 0$ for all $i = 0...n$;
Iterate: For $i = 1$ to n:
Do: Set $j = i - 1$ and repeat until $e_j \leq e_i$:
$m_i += \alpha_j e_i$, $j -= 1$
$m_i += m_j$.
Output: $\{m_i\}_{i=1}^n$.

Again $\{m_i/\alpha\}_{i=1}^n$ is our set of adjusted e-values. The idea behind this algorithm is that, starting with α_{i-1}, we should include the node $i-1$ if its e-value is lower than e_i. Otherwise, we should assign its budget to the e-value e_i, so we create a variable $e_{i-1}^{(i)}$ which is the optimal assignment of e-value to the budget α_i. Then for $i-2$, we should assign to its budget whatever e-value, between it and the optimal assignment for $i-1$, is lower. Note, however, that we can combine this into a dynamic programming framework, where once we have included our first node j, we are then optimizing for the best subset which includes i, given to us by $m_j = \min_{j \in J} \alpha e_J$. This algorithm, for each i, takes as many reverse steps as it takes to reach the next e-value that is $\leq e_i$, which could be as high as n, meaning a naĂŻve runtime of $O(n^2)$. However, we see that this worst case occurs only if the e-values $\{e_i\}_{i=1}^n$ are increasing. If, for example, we assume that the e-values are continuous and exchangeable, then the expected runtime is $O_P(n)$. We state this as Theorem C.1, which we prove in Section C

Theorem 3.2. *The output of Algorithm 3 has the property that* $m_i = \min_{i \in I \subset [n]} \alpha e_I$ *for all* $i \in [n]$.

Proof. We already know that we only need to consider $I \subset [i]$, so we'll follow an inductive approach. First we see that $m_1 = \alpha_1 e_1$ is correct. Then for any $i \in [n]$, we assume that for all $j < i$, $m_j = \min_{j \in J \subset [n]} \alpha e_J = \min_{j \in J \subset [j]} \alpha e_J$. Then we set $j^* = \max\{j < i : e_j \leq e_i\}$ (this is the index in the algorithm for which we add m_j at the conclusion of the search loop), and consider any $I \subset [i]$, and $J = I \setminus \{j^* + 1, ..., i-1\} \cup \{j^*\}$. Then we claim $\alpha e_J \leq \alpha e_I$. To show this we compute

$$\alpha e_I - \alpha e_J = \alpha_{\to j^*}(e_{j^*}^{(I)} - e_{j^*}) + \sum_{j=j^*+1}^{i-1} \alpha_j (e_j^{(I)} - e_i), \tag{3.2}$$

where $e_j^{(I)} = e_k$ for $k = \min\{\ell \in I | \ell > j\}$, the e-value that j's budget is assigned to, and $\alpha_{\to j^*} = \sum_{k_{j^*} < k \leq j^*} \alpha_k$, where $k_{j^*} = \max\{k \in I | j^* > k\}$, which is the α-budget flowing into node j^*. Then we note that each $e_j^{(I)}$ for $j \geq j^*$ is in the set $\{e_{j^*+1}, ..., e_i\}$, which all are $\geq e_i$, so for all $j \geq j^*$ we have $e_j^{(I)} \geq e_i \geq e_{j^*}$, so each term in our expression (3.2) is nonnegative, meaning that for any $i \in I \subset [n]$, including the index j^* and excluding $j^* + 1, ..., i - 1$ is at least as good, after which point we simply need to minimize over $j^* \in J^* \subset [j^*]$, a value which we have and know is correct, m_{j^*}, so we conclude that

$$m_i = m_{j^*} + (\sum_{j=j^*+1}^{i} \alpha_j) e_i = \min_{i \in I \subset [n]} \alpha e_I, \tag{3.3}$$

concluding our induction and our proof.

□

This algorithm gives a significant shortcut, as the previous smaller e-value e_j is likely close to e_i, but in the worst case, the runtime is still $O(n^2)$. A simple addition to the algorithm can account for this and produce an $O(n)$ runtime.

3.2. **Stack Optimization.** The key observation is that for any i, there is a list of candidates for the most recent smaller e-value, and once we

Algorithm 4: Stack Search for e-Fallback

Input: Vectors $\{e_i\}_{i=1}^n, \{\alpha_i\}_{i=1}^n$;
Initialize: $m_i = 0$ for all $i = 0...n$, stack $s = \{\}$;
Iterate: For $i = 1$ to n:
Do: Set $\alpha_i^* = \alpha_i$:
Pop (j, a_j^*) from s and add $\alpha_i^* = \alpha_i^* + \alpha_j^*$ until $e_j \leq e_i$ or $s = \{\}$
If stop because $e_j \leq e_i$: $m_i := \alpha_{ii}^e + m_j$, add $(j, a_j^*), (i, a_i^*)$ to s
If stop because $s = \{\}$: $m_i := \alpha_i e_i^*$, add (i, a_i^*) to s
Output: $\{m_i\}_{i=1}^n$.

To prove that this algorithm is correct, we prove that any time a node is removed permanently from the stack, it can never be the most recent smaller e-value for a new node, thereby ensuring that we will always end up with the same value as Algorithm 3, which we already proved is correct in Theorem 3.2.

Before the theorem of correctness and proof, we present a brief visualization for how each step of the algorithm works. In Figure 1, we see the candidate indices in the stack, which are the orange points, and the red line representing the cutoff given by the new e-value in line. We can see that an index is included if its e-value is smaller than all subsequent in the stack, and excluded if there is a later index with a smaller e-value.

In Figure 1, note that the set of candidate e-values is increasing in the index, and the most recent e-value is always included.

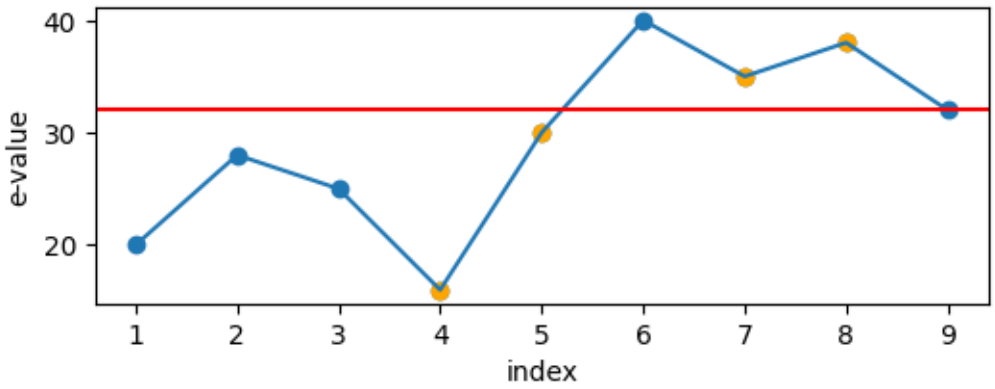

FIGURE 1. Visualization of stack e-Fallback Algorithm 4 step $i = 9$, with e-values on the y-axis. The last point is the new e-value, and the orange dots are the elements of the stack. In searching the stack, we see that $j = 5$ is the most recent smaller e-value.

Theorem 3.3. *The output of Algorithm 4 has the property that* $m_i = \min_{I \ni i} \alpha e_I$ *for all* $i \in [n]$.

Proof. We have already shown in Theorem 3.2 that Algorithm 3 is correct, so we show that these algorithms have the same output, namely that for each $i \in [n]$, the index j^* is included in the stack s at the start of step i, where $j^* = \max\{j < i : e_j \leq e_i\}$. We know that j^* was added to the stack after step j^* since the current index is always added. By the definition of j^*, $e^j > e_i$ for all $j^* < j < i$, meaning that in the iterative step for index j of Algorithm 4, either j^* was not searched, or it was the most recent smaller index. In either case, j^* remained in the stack after each such step j, and thus it is in the stack at step i. □

As desired, Algorithm 4 has runtime $O(n)$, since each index is added to the stack once initially, and again when it is the minimizing j^*, for $O(n)$ stack additions. Then each index is checked and removed from the stack only once; if it is checked and retained then it was the minimizing j^*, giving an $O(n)$ number of removals.

4. General Approach for DAGs

For a general DAG, we notice a local phenomenon that we claim will extend to an algorithm that computes the global minimum for any index i. We first establish the result for the simpler case of the Fallback procedure (which we will later extend to general DAGs).

In this section we develop the extension of the reverse search Fallback algorithm from Section 3.1 to the general DAG. The intuition is the same, that for any node i we can only look at the ancestor graph, and search backwards to decide whether or not to include each node.

Note that we assume that the nodes are already topologically ordered with respect to the input graph, which is a reasonable constraint given in practice a practitioner will be designing the graph themselves so will be able to design the data collection to have the nodes ordered properly. In practice we might write a line of code to check that the nodes are topologically ordered.

This output again has that $\{m_i/\alpha_i\}_{i=1}^n$ are the adjusted e-values $\{e_i^*\}_{i=1}^n$.

Algorithm 5: e-DAG Graphical Approach

Input: Vectors $\{e_i\}_{i=1}^n, \{\alpha_i\}_{i=1}^n$, transition matrix $w \in^{n\times n}$, where $[n]$ is a topological ordering with respect to the graph $G = (V, E)$;
Iterate: For $i \in [n]$:
Do: Initialize e-assignments $\{e_j^{(i)}\}_{j\in[i]}$, $m_i = 0$, compute A_i the set of ancestors of i, including i;
Initialize $e_i^{(i)} = e_i$, $m_i = \alpha_i e_i$:
For $i \neq j \in A_i$ decreasing $e_j^{(i)} = \min(e_j, \sum_{(j,k)\in E, k\in T_i} w_{jk} e_k^{(i)})$ and set $m_i += \alpha_j e_j^{(i)}$.
Output: $\{m_i\}_{i=1}^n$

Theorem 4.1. *The output of Algorithm 5 has the property that* $m_i = \min_{i\in I\subset[n]} \alpha e_I$ *for all* $i \in [n]$.

Proof. We introduce the notation of I_i^*, the optimal subset that is explicitly constructed in Algorithm 5. We have that

$$I_i^* = \{j \in A_i | e_j^{(i)} = e_j\}, \tag{4.1}$$

meaning I_i^* is the set of indices that we included in our recursive construction. We note again that we only need to consider those nodes in A_i, which are ancestors of i. Then we wish to show that for any $i \in [n], i \in I \subset A_i$, we have $\alpha e_I \geq \alpha e_{I_i^*}$, in which case we are done. To do this we note that in the computation of any αe_I, there is some value assigned to each α_j, which we call $e_j^{(I)}$, as in our proof of Theorem 3.2. Note that for I_i^*, $e_j^{(I_i^*)} = e_j^{(i)}$ from Algorithm 5. Then we wish to show that

$$\alpha e_I - \alpha e_{I_i^*} = \sum_{j\in[n]} \alpha_j (e_j^{(I)} - e_j^{(I_i^*)}) \tag{4.2}$$

is nonnegative for all $i \in I \subset [n]$, which we will do by showing that $e_j^{(I)} \geq e_j^{(i)}$ for all $j \in [n]$. We proceed by induction, first seeing that $e_i^{(I)} = e_i^{(I_i^*)} = e_i$. Then we assume that for all $j < k \leq i$, $e_k^{(I)} \geq e_k^{(I_i^*)}$. Then for j, if $j \notin A_i$ we have the inequality trivially as $e_j^{(I_i^*)} = 0$, and otherwise

$$e_j^{(I_i^*)} = \min(e_j, \sum_{(j,k)\in E} w_{jk} e_k^{(I_i^*)}) \leq e_j^{(I)}, \tag{4.3}$$

as

$$e_j^{(I)} = \begin{cases} e_j & j \in I \\ \sum_{(j,k)\in E} w_{jk} e_k^{(I)} & j \notin I \end{cases},$$

where by our inductive assumption and the topological ordering of $[n]$, $e_k^{(I_i^*)} \leq e_k^{(I)}$ for all $(j, k) \in E$. So in either case for whether $j \in I$, our constructed set does better. Then we are done.

□

5. Simulation Results

In this section we develop a framework for comparing the power of our e-value procedures to that of the corresponding p-value procedures. We work in the sequential testing setting, where we have a test martingale for many hypotheses, and can test either via e-values, 1/e p-values, or the always-valid p-values [3]. Since a primary purpose of sequential testing is to make a rejection once significance is reached, for some choice of hypothesis or hypotheses.

We present results for power in terms of how often the e-procedure improves on the p-procedures, as well as by what percentage in terms of ratio of stopping times. Both give useful information for gauging how meaningful the potential improvement of e-Holm is in the sequential setting.

We examine the Holm's procedure described in Section 2 and the graphical procedure described in Section B, specifically for a useful structuring of hypotheses in a factorial design.

5.1. **e-Holm Simulations.** Our setting is 20 hypotheses $H_i : \mu_i = 0$, for i.i.d. data $X_i \sim \mathcal{N}(\mu_i, 1)$, where the known alternative for each is μ_{alt}, which we vary on $\{0.5, 1, 1.5, 2\}$. Each hypothesis is tested with a test martingale from a sequential probability ratio test (SPRT) with known alternative, a la Johari et al.. For our simulation, there are 5 true alternatives. We define our data collection stopping time as the time at which at least one hypothesis can be rejected, after accounting for multiplicity (via each procedure). In terms of adjusted e-values $\{e^*_{it}\}_{i=1}^n$, this stopping time is $T_e = \min\{t : \max_{i\in[n]} e^*_{it} \geq 1/\alpha\}$, and we define T_{ep} and T_p analagously for the 1/e p-values and always-valid p-values.

For each value of μ_{alt}, we run 1000 Monte Carlo simulations, where we save the stopping times T_e, T_{ep}, T_p, all computed with the same sequence of simulated data. For practical purposes there is a maximum number of iterations of 2000, though no run reaches this threshold. Note that because we stop once the most significant hypothesis is rejected, its test martingale is at its maximum, and thus $T_e \leq T_p$ always, and $T_p = T_{ep}$. This follows from the discussion of the comparison between e-Bonferroni and p-Bonferroni in Section 1.2. Thus, we only compare T_e and T_p, and compute for $m = 1000$ iterations and empirical distribution $\mathbb{P}_m$:

- Comparison (probability of improving on p-Holm): $\mathbb{P}_m(T_e < T_p)$
- Ratio (conditional on there being a difference): $\mathbb{E}_m[T_e/T_p | T_e \neq T_p]$

For the former, shown in Figure 2, the e-Holm comparison is always non-negative, the greater the better. For the latter, shown in Figure 3, the e-Holm ratio is always no greater than 1, the lower the better. Note that both metrics are necessary to understand the efficiency of e-Holm, as a small ratio is meaningless with a small comparison.

The probability comparison in Figure 2 shows that the biggest improvement in likelihood of beating p-Holm comes for smaller μ_{alt}, which makes intuitive sense as the stopping times are larger, leaving more margin for improvement. This reasoning also applies to the results in Figure 3, which show that the lower the stopping time the lower the potential the ratio has to be, as low as 0.6 on average for $T_e \neq T_p$ when $\mu_{alt} = 2$. This intuitive rationalization aside, it is promising that all offer improvements of at least 0.05 in terms of $\mathbb{P}_m(T_e < T_p)$ and ≈ 0.9 for $\mathbb{E}_m[T_e/T_p | T_e \neq T_p]$.

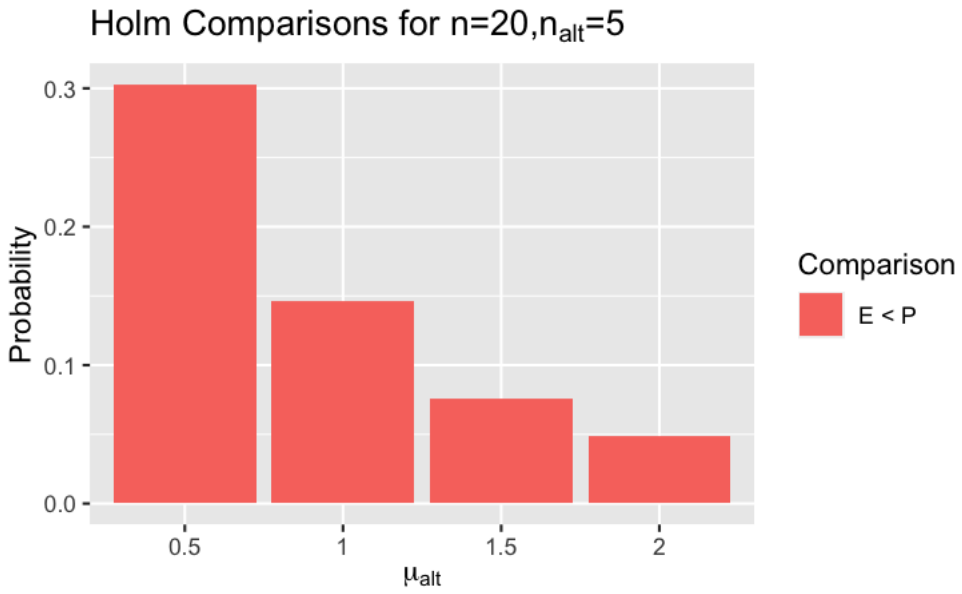

FIGURE 2. Empirical probability from $m = 1000$ iterations of e-Holm improving on p-Holm for a stopping time of first rejection.

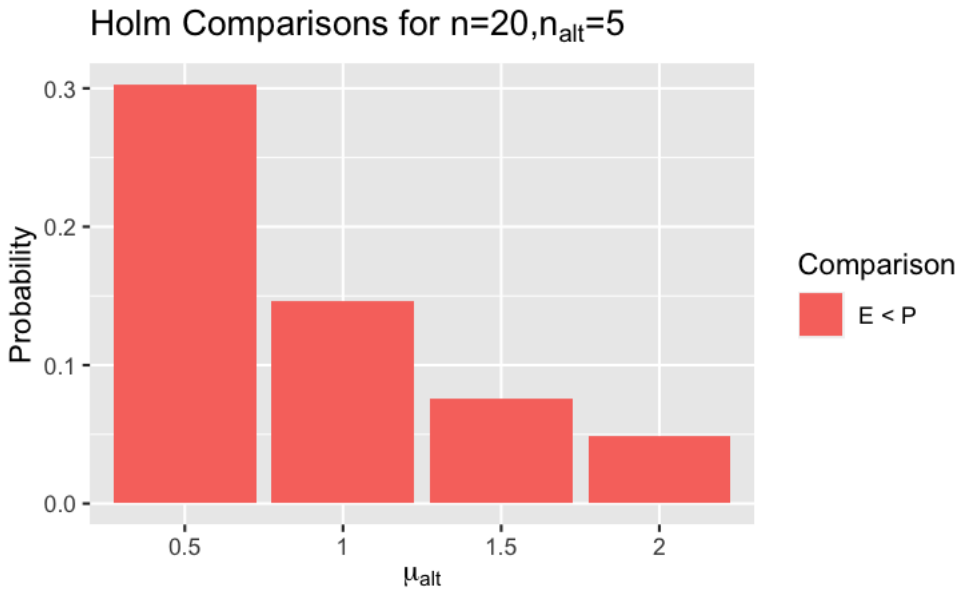

FIGURE 3. Empirical ratio from $m = 1000$ iterations of stopping time of first rejection for e-Holm and p-Holm, when they differ.

5.2. **e-DAG Simulations.** We continue our simulation study with a similar framework under a different model, this time with a structure to our hypotheses in comparison to the agnostic treatment of hypotheses in Section 5.1. We use the example of a factorial design, which is a useful framework for testing not just primary nodes, but also secondary, tertiary, etc. nodes. For example, one might be interested in the effect of a drug and the effect of gender on outcomes as primary node, but also the interaction effect of the drug for a particular gender, a secondary node.

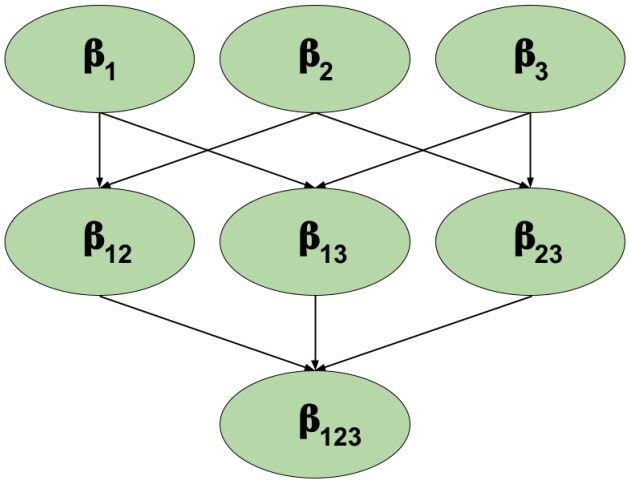

FIGURE 4. The transition graph for the graphical approach in a factorial design with three primary nodes.

To illustrate this framework, consider a simple three-factor model, say for A/B testing in a tech application, with three proposed treatments. The model is that data Y_i is drawn with treatments $X_i \in \{0,1\}^3$ according to

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \beta_3 X_{i3} \tag{5.1}$$

$$+ \beta_{12} X_{i1} X_{i2} + \beta_{23} X_{i2} X_{i3} + \beta_{13} X_{i1} X_{i3} + \beta_{123} X_{i1} X_{i2} X_{i3} + \epsilon_i \tag{5.2}$$

Where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ is an i.i.d. Gaussian error term. In practice one might use a more complicated model but for simplicity we take this linear model with fixed effects and i.i.d. errors with known variance (which we take to be 1). Here the hypotheses $H_j : \beta_j = 0$ correspond to primary nodes, $H_{jk} : \beta_{jk} = 0$ correspond to secondary interaction nulls, and $H_{123} : \beta_{123} = 0$ corresponds to the tertiary interaction null.

In an experiment, the practitioner likely cares mostly about the primary hypotheses, but would like to test the secondary (and higher order) hypotheses as well. An approach is to use the graphical procedure to define a graph in which each primary hypothesis is a root, whose children are the secondary hypotheses containing the primary hypothesis. The children of H_1 are H_{12} and H_{13} in our three-way example, which is visualized in Figure 4.

In the context of the graphical approach, it is most common to take the transition probabilities to be uniform, and the starting α budget to be α/p for each of the p primary endpoints, in our case $\alpha/3$ for each of H_1, H_2, H_3. The result of this choice is that, in terms of rejecting any primary hypothesis, we use Bonferroni to determine if any node can be rejected. Once one of these nodes can be rejected, its budget transfers to the corresponding secondary hypotheses, which might be rejected, though they require a higher level of significance. Note that the ancestor graph of H_i is just H_i, so we can only reject H_i if its standalone test martingale can reject at level $\alpha/3$.

The upside is that rejecting, for example, H_{13}, becomes potentially easier because we can use e-Bonferroni as the local test, as if H_1 is rejected, we now can boost out

adjusted e-value for H_{13} via the value of H_3, if it remains unrejected. However, it is unclear a priori whether this, or the fact that the always-valid p-value gets to use the max value of the test martingale, will win out in various settings.

In order to investigate this trade-off, we perform a similar simulation study to Section 5.1, with our three-way factorial design model. It is the same in that we have $m = 1000$ iterations, and we vary $\mu_{alt} \in \{0.5, 1, 1.5, 2\}$. Our stopping time is the rejection of H_{13}, where $\beta_1, \beta_3, \beta_{13}$ are the only nonzero parameters. We fix $\beta_1 = \beta_3 = 0.5$, with this alternative being known, and β_{13} being unknown.

We consider a process in which at time t, we receive $2^3 = 8$ measurements corresponding to draws of our model with each of $X \in \{0, 1\}^3$, which we will notate using binary. We can then isolate each coefficient (plus mean zero noise) from the data by taking combinations of our time t data. For example, $Y_{t100} - Y_{t000} = \beta_1 + \mathcal{N}(0, 2)$, and similar for the other primary endpoints. Then to isolate β_{13}, we can take

$$Y_{t101} - Y_{t100} - Y_{t001} + Y_{t000} = \beta_{13} + \mathcal{N}(0, 4). \tag{5.3}$$

Note that because we are adding and subtracting more datapoints, the variance increases. This procedure generalizes via a pattern that follows the inclusion-exclusion principle. Then for each of these test statistics, we can construct a test martingale via a SPRT for an alternative mean, which for us is known except for H_{13}, for which we estimate the alternative mean for time t as a predictable function of $\{Y_s\}_{s=1}^{t-1}$ which is just the sample average (equivalent to linear regression), with a burn-in time of 10 batches.

With these test martingales defined, we define our stopping time to be the time that H_{13} is rejected, after accounting for multiplicity via e-DAG/ep-DAG/p-DAG, i.e. $T_e = \min\{t : e_t^{13*} \geq 1/\alpha\}$, for e_t^{13*} the adjusted e-value for H_{13}, and analagously defined for T_p and T_{ep}. Note that each of these procedures only depends on the test martingales for H_1, H_3, H_{13}.

As in Section 5.1, we record the probability of T_e improving on T_p, but now it could be worse, and T_p, T_{ep} could differ, so we record three comparisons: $\mathbb{P}_m(T_e < T_p), \mathbb{P}_m(T_e > T_p)$, and $\mathbb{P}_m(T_e < T_{ep})$. Since e-values have other documented pros besides our procedure (see ...), knowing the latter is relevant. We also record the ratio conditional on a difference, both for T_p and T_{ep}, because they might differ. In notation, these quantities are $\mathbb{E}_m[T_e/T_p | T_e \neq T_p]$ and $\mathbb{E}_m[T_e/T_{ep} T_e \neq T_{ep}]$.

In general, power tends to more favorable for e-DAG for higher values of μ_{alt}, with the only favorable ratio occurring for $\mu_{alt} = 2$. However, the results show that the gains over ep-DAG are consistent and substantial, with settings in which e-DAG is able to improve upon p-DAG.

This first set of plots (Figures 5 and 6) are for primary budget, meaning each of H_1, H_2, H_3 is assigned budget $\alpha/3$, and no budget for the secondary and tertiary hypotheses. We also perform the same analysis for equal budget among all seven hypotheses, which allows H_{13}, the subject of the simulation study, to be rejected even if H_1 and H_3 are both unrejected. This might be desirable if the secondary hypothesis is of interest separately from the primary ones.

Just like the simulations show, the equal budget experiment generally follow similar trends to plots for primary budget, but with improvements across the board. Most notably, the probability of p-DAG beating e-DAG becomes very low for $\mu_{alt} = 2$, with a corresponding ratio of less than 0.95. Both simulation settings (primary and equal budget for factorial design) show that e-DAG can consistently be a notable

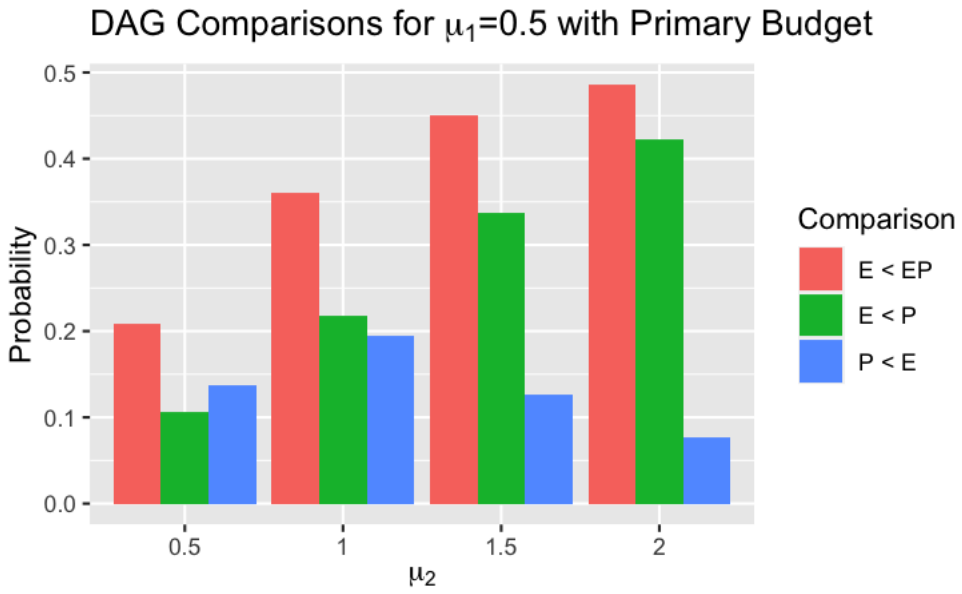

FIGURE 5. Empirical probability from $m = 1000$ iterations of e-DAG differing from p-DAG or ep-DAG for a stopping time of rejecting a specific secondary hypothesis in a factorial design graphical model with evenly distributed primary budget.

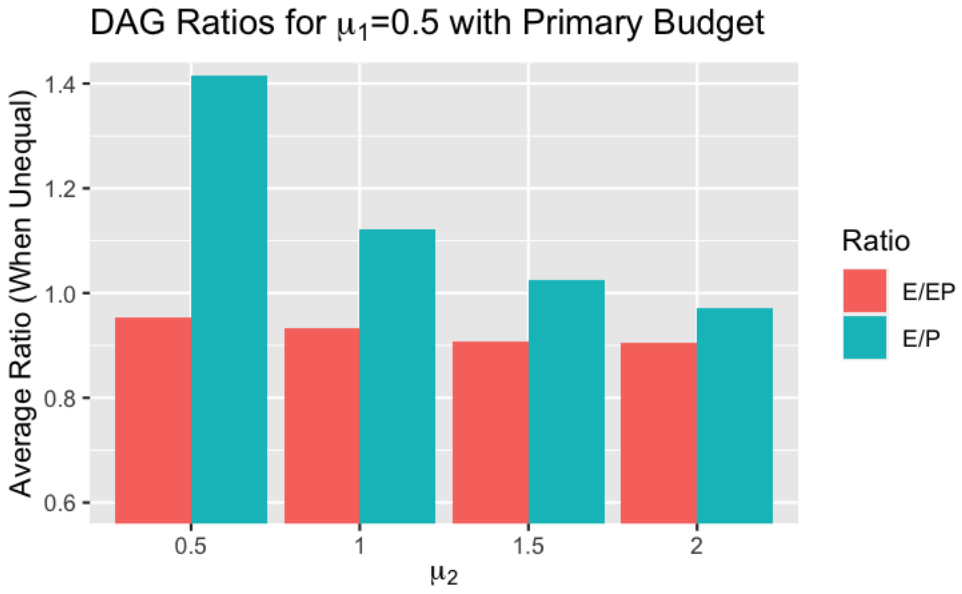

FIGURE 6. Empirical ratio from $m = 1000$ iterations of the stopping time from Figure 5 for e-DAG to p-DAG and ep-DAG.

improvement on ep-DAG, while having mixed results compared to p-DAG, albeit with many promising settings.

A. Extension to Trees

The approach for trees boils down to a re-application of the dynamic programming approach we used for the Fallback method. We consider $G = (V, E)$ a directed tree

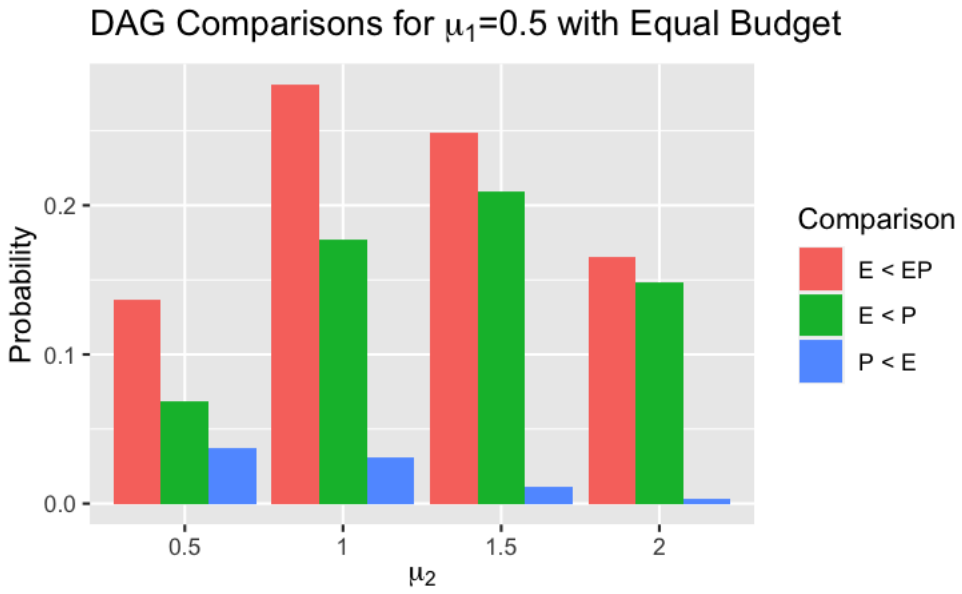

FIGURE 7. Empirical probability from $m = 1000$ iterations of e-DAG differing from p-DAG or ep-DAG for a stopping time of rejecting a specific secondary hypothesis in a factorial design graphical model with equal budget across seven hypotheses.

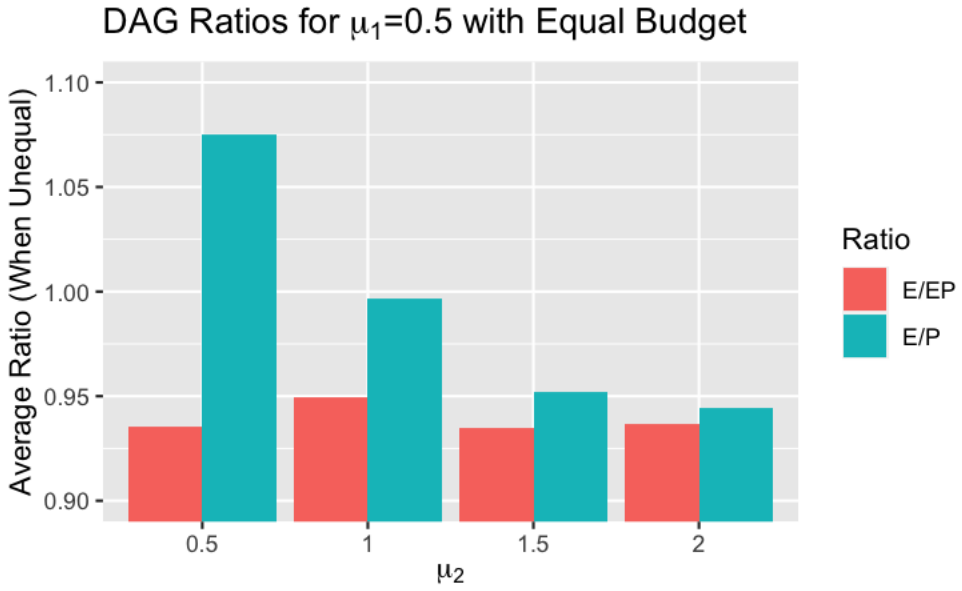

FIGURE 8. Empirical ratio from $m = 1000$ iterations of the stopping time from Figure 7 for e-DAG to p-DAG and ep-DAG.

graph, with one node, labelled 1, having no parents, and each other node having exactly one parent. Labelling our $n = |V|$ vertices, we have associated e-values and α-budgets $\{(e_i, \alpha_i)\}_{i=1}^n$. Like fallback, for any subset $I \subset [n]$ of the vertices, we define the global test e-value e_I by recursively giving the budget of each excluded node to its children and then updating the graph. If there are no children to give budget to then that budget disappears. For example, the e-value for the full graph

$[n]$ is $e_I = \sum_{i=1}^{n} \frac{\alpha_i}{\alpha} e_i$, the weighted average of the e_i's according to weights α_i/α. We equivalently notate this as $\alpha e_I = \sum_{i=1}^{n} \alpha_i e_i$. Each e-value e_i will have an adjusted e-value e_i^* which is the value obtained from the closure of the local subset test:

$$e_i^* = \min_{I \ni i} e_I. \tag{A.1}$$

By the closure principle these adjusted e-values control the FWER at level α. To see how we can approach computing these adjusted e-values, we, for any index i, and subset $i \in I \subset [n]$, define the set $J_i = \{j \in I | j \text{ is an ancestor of } i\}$, where $i \in J_i$, and write $I = J_i \cup K$ with $K \cap J_i = \emptyset$. We see immediately that

$$e_I \geq e_{J_i} \forall I \ni i, \tag{A.2}$$

since $\alpha e_I = \sum_{j \in J_i} \alpha_j^I e_j + \sum_{k \in K} \alpha_k^I e_k$, and $\alpha e_{J_i} = \sum_{j \in J_i} \alpha_j^{J_i} e_j$ with $\alpha_j^{J_i} = \alpha_j^I$, since all the indices in K are not ancestors of i and thus cannot be ancestors of any $j \in J_i$.

One way to imagine this is by recursively removing non-i leaves from the graph of I one by one until the only leaf is i. Each removal results in its leaf's budget disappearing from the graph, leaving the budget of i and its ancestors untouched, but decreasing the subset e-value. Therefore, we conclude that for each i, we can define $I_i = \{j \in [n] | j \text{ is an ancestor of } i\}$, and compute $e_i^* = \min_{I \subset I_i} e_I$, a much simpler computation. This observation holds for any graph, but specifically on a tree, the graph G restricted to the index set I_i is exactly the Fallback graph, so we can effectively use the algorithm from Section 3, modified with a depth-first search to reduce runtime. For notational simplicity we assume index 1 is the root of the tree.

Algorithm 6: e-Tree Graphical Approach

Input: Vectors $\{e_i\}_{i=1}^n$, Adjacency Matrix $G \in \{0,1\}^{n \times n}$;
Initialize: $m_1 = \alpha_1 e_1$, i=1;
Iterate: For j such that $G_{ij} = 1$, compute m_j and iterate on j;
Output: $\{m_j\}_{j=1}^n$.

Then, as in our e-Fallback algorithm, $\{m_i/\alpha\}_{i=1}^n$ are our adjusted e-values $\{e_i\}_{i=1}^n$. The proof that these are correct follows from (A.2) and Theorem 3.1. For runtime, the algorithm at index i does a minimization over d_i values for d_i the depth of index i, so our runtime is $O(\sum_{i=1}^n d_i)$, which for a tree with 1 root and $n-1$ leaves gives $O(n)$ and for a tree with one leaf (i.e. the chain defining Fallback), gives $O(n^2)$ as per Section 3. For a binary tree of depth d and $n = 2^d - 1$ nodes, we get $O(n \log n)$ runtime.

B. Extensions to Select non-DAGs

In this appendix we extend our result from Section to a larger class of graphs which we call (tentatively) Index-Local DAGs. The key motivating observation is that our argument in Section relies on the fact that for every index i, the graph over which we have to optimize is a DAG, with i as its only leaf. Thus, we define:

Definition B.1. A graph $G = (V, E)$ is an *Index-Local DAG (ILDAG)* if for every i, the graph $G^{(i)}$ formed by removing $j \in V$ such that there is no path from j to i, as well as removing any $(i, k) \in E$, is a DAG.

The motivation for this is that when computing e_i^*, we have shown we only need to consider $I \subset I_i$, and at that only subsets containing i. Since $i \in I$, we know the α-budget at i will never be redistributed, so we can remove the edges from i and have the same minimization problem. At this point, if the remaining graph is a DAG then we have by the result of Section B that the DAG algorithm, for each i, will produce correct adjusted e-values $\{e_i^*\}_{i=1}^n$. This can be done by performing the DAG algorithm's backwards search for each i on its ancestor graph A_i, and ignoring any reverse edges $i \to j$.

The result is somewhat limited, but there are at least two interesting cases, being the cyclical Fallback graph, which is identical but for an additional edge $n \to 1$, ensuring that we are always working with the full α-budget. This graph clearly satisfies the requirement to be an ILDAG, since removing any one edge $(i \to i+1)$ breaks the cycle and forms a Fallback chain starting at $i+1$ and ending at i.

The other example is the gatekeeper procedure for two endpoints with a cycle between them, a visualization of which is in Figure X.

Because we rerun Algorithm 5 for each $i \in [n]$, with potentially all nodes included in A_i, the runtime is worst-case $O(n^2|E|)$.

C. Expected Runtime of e-Fallback Algorithm 3

Theorem C.1. *In the context of Algorithm 3:*

1. *If the e-values $\{e_i\}_{i=1}^n$ are exchangeable, then the runtime is $O_P(n)$.*
2. *If there are k non-null e-values, and the $n-k$ null e-values are exchangeable, then the runtime is $O_P(kn)$.*

Proof. First we show part (1), by denoting t_i as the number of comparisons to e_i queried, i.e. $t_i = i - j^*$ in the notation of the previous proof. Then we see that the runtime is $O(\sum_{i=1}^n t_i)$, where in expectation we compute

$$\mathbb{E}_P t_i = \sum_{k=1}^{\infty} \mathbb{P}(t_i \geq k) = \sum_{k=1}^{i-1} \mathbb{P}(t_i \geq k), \tag{C.1}$$

where $\mathbb{P}(t_i \geq k) = \mathbb{P}(\min(\{e_{i-k+1}, ..., e_i\} = e_i) \leq \frac{1}{k!}$ with equality if the distribution of e_i is continuous, a result following from order statistics of exchangeable collections of random variables. By the Darth Vader Rule, $\mathbb{E}_P t_i \leq \sum_{k=1}^{i-1} \frac{1}{k!} \leq e - 1$, so we conclude that our runtime is $O_P((e-1)n) = O_P(n)$.

For part (2), we use a similar framework, but lower bound $t_i \leq n$ for e_i a non-null, and $t_i \leq k + t_i'$ where t_i' is the number of nulls preceding i until we reach a null e-value with $e_j \leq e_i$. We enumerate the nulls by indices $\{i_j\}_{j=1}^{n-k}$, and define for each $i = i_\ell$ $j_0 := \max\{j < \ell | e_{i_j} \leq e_i\}$, the corresponding index for just the nulls. Then defining $t_i' = \ell - j_0$ we have $\mathbb{E}_P t_i' \leq e - 1$ again, and our upper bound $t_i \leq k + e - 1$. Putting these together for k non-nulls and $n-k$ nulls, we have runtime

$$\begin{aligned} O_P(k(n) + (n-k)(k+e-1)) &= O_P(nk + nk + (e-1)n - k^2 - (e-1)k) \\ &= O_P(nk + n), \end{aligned}$$

so $O_P(nk)$ for $k > 0$.

□

This result is nice but uses exchangeability assumptions that are unrealistic. However, it gives a sense of why we expect that this algorithm will provide a significant

runtime improvement. Our treatment of the non-nulls in particular is a worst-case perspective in which $e_j > e_i$ for any non-null j and null i, which is likely not always true. We present a few simulation settings in which the runtime improvements become clear.

References

[1] Bretz, F., Posch, M., Glimm, E., Klinglmueller, F., Maurer, W., Rohmeyer, K. (2011). Graphical approaches for multiple comparison procedures using weighted Bonferroni, Simes, or parametric tests. Biometrical Journal, 53 (6), 894-913.

[2] GrĂźnwald, P. (2024). Beyond Neyman-Pearson: e-values enable hypothesis testing with a data-driven alpha. arXiv: https://arxiv.org/abs/2205.00901.

[3] Johari, R., Koomen, P., Pekelis, L., Walsh, D. (2022). Always Valid Inference: Continuous Monitoring of A/B Tests. Operations Research, 70 (3), 1806-1821.

[4] Sonnemann, E. and Finner, H. (1988). VollstandigkeitssÄ¤tze fĂźr multiple Testprobleme. Multiple Hypothesenprufung. Springer, 121-135.

[5] Vovk, V. and Wang, R. (2021). E-values: Calibration, combination and applications. Annals of Statistics 49 (3), 1736-1754.

Stanford University, Department of Statistics, Stanford CA 94305

Stanford University, Graduate School of Business, Stanford CA 94305
Email address: whartog@stanford.edu