Exact Paired-Permutation Testing for Structured Test Statistics

Ran Zmigrod  Tim Vieira  Ryan Cotterell
University of Cambridge  Johns Hopkins University  ETH Zürich
rz279@cam.ac.uk  tim.f.vieira@gmail.com  ryan.cotterell@inf.ethz.ch

Abstract
Significance testing—especially the paired-permutation test—has played a vital role in developing NLP systems to provide confidence that the difference in performance between two systems (i.e., the test statistic) is not due to luck. However, practitioners rely on Monte Carlo approximation to perform this test due to a lack of a suitable exact algorithm. In this paper, we provide an efficient exact algorithm for the paired-permutation test for a family of structured test statistics. Our algorithm runs in \(O(GN(\log GN)(\log N))\) time where \(N\) is the dataset size and \(G\) is the range of the test statistic. We found that our exact algorithm was 10x faster than the Monte Carlo approximation with 20000 samples on a common dataset.

1 Introduction
How confident can we be that System \(U\) is more accurate than System \(V\)? Questions of this form are widespread in natural language processing (Dietterich, 1998; Koehn, 2004; Ojala and Garriga, 2010; Clark et al., 2011; Berg-Kirkpatrick et al., 2012; Dror et al., 2018) and statistical hypothesis testing provides answers (Lehmann and Romano, 2005). In this paper, we study the paired-permutation test (Good, 2000)—a commonly used hypothesis test in NLP because it makes no assumptions on the distribution of the data or the evaluation metric used to compare the two systems (Yeh, 2000; Dror et al., 2018, 2020; Deutsch et al., 2021). The paired-permutation test checks whether a test statistic is significant by evaluating the probability that a value at least as large as the observed statistic would occur if system outputs were randomly swapped. Thus, an exact algorithm for evaluating the paired-permutation test involves a summation over all \(2^N\) possible swaps. Without any assumptions on the test statistic, we can only exactly compute this sum in \(O(2^N)\) time. Thus, practitioners often resort to running a Monte Carlo (MC) approximation which replaces the summation with \(K \ll 2^N\) randomly sampled swaps. Although the MC approximation is often practical, it unfortunately introduces additional error when determining the significance of a test (Serlin, 2000; Koehler et al., 2009).

This paper proposes a family of additively structured, integer-valued test statistics. Test statistics of this form admit an efficient exact algorithm that leverages the fast Fourier transform (Cooley and Tukey, 1965; Cormen et al., 2022) to run in \(O(GN(\log GN)(\log N))\) time where \(N\) is the size of the dataset and \(G\) is the range of the test statistic. We compare the efficiency of our exact method to the MC approximation for comparing part-of-speech taggers on the Universal Dependency Dataset (Nivre et al., 2018). Surprisingly, our exact algorithm is faster than MC approximation: given 10000 sentences, our algorithm is 10x faster than MC with 20000 samples and 3x faster than MC with 5000 samples, taking \(\approx 0.1\) seconds.

Figure 1: Runtime comparison of exact_perm_test using convolve_dp and convolve_fft, and monte_carlo as a function of the number of entries in the dataset. See for §5 for experimental details.
2 Paired-Permutation Testing

The paired-permutation test (Good, 2000), the focus of this work, is a common null hypothesis significance test that has a natural application to many problems in NLP (Peyrard et al., 2021). The test attempts to reject the null hypothesis, described below, at significance level \( \alpha \); typically \( \alpha = 0.05 \).

**Preliminaries.** Suppose we want to compare the performances of two systems \( U \) and \( V \) where each system was evaluated on a dataset of the same \( N \) entries. We place the entries of \( U \) and \( V \) into a pair of arrays of length \( N \) denoted \( u \) and \( v \).

**The null hypothesis.** The goal of a paired-permutation test is to test whether the entries \( u \) and \( v \) are independent of the labels \( U \) and \( V \) themselves. The reason that this is the question we ought to care about is that, fundamentally, if the system label (in this case, \( U \) or \( V \)) provides no information (in the sense of mutual information) about the entry, then we should not prefer one system to another. And, from basic information theory, we know that two random variables (RVs) have no mutual information iff they are independent. So, independence of the system’s label and the system’s set of entries is the right thing to inquire about. In the language of frequentist testing, the hypothesis that a system’s labels and individual entries are independent is known as the **null hypothesis.** And, under a paired-permutation test, the goal is to ascertain whether the data (the observed entries \( u \) and \( v \)) provide enough evidence to reject the null hypothesis, i.e., to conclude that the label of a system shares information with the quality of its individual entries, and are indeed dependent.

**The null distribution.** Next, in order to attempt to reject the null hypothesis, we require a distribution over (hypothetical) pairs of entries \( u' \) and \( v' \) whose individual entries are independent of the system labels \( U \) and \( V \), which is achieved through the construction of RVs \( U^\circ \) and \( V^\circ \), whose joint distribution can be used to sample our hypothetical \( u' \) and \( v' \). Traditionally, \( \mathbb{P}[U^\circ, V^\circ] \) is referred to as the **null distribution.** A paired-permutation test provides a simple recipe for constructing such an RV pair. The first step is to make an entry-wise independence assumption: we define the joint probability \( \mathbb{P}[U^\circ, V^\circ] \equiv \prod_{n=1}^{N} \mathbb{P}[U_n^\circ, V_n^\circ] \). This means that the prediction a system makes for the \( n \)th entry is independent of the \( m \)th entry when \( n \neq m \). In the second step, we further define the entry-wise joint distribution as

\[
\mathbb{P}[U_n^\circ = u_n, V_n^\circ = v_n] \overset{\text{def}}{=} \frac{1}{2} \quad \text{(stay)} \quad (1a)
\]

\[
\mathbb{P}[U_n^\circ = v_n, V_n^\circ = u_n] \overset{\text{def}}{=} \frac{1}{2} \quad \text{(swap)} \quad (1b)
\]

In words, \( \mathbb{P}[U_n^\circ, V_n^\circ] \) is a uniform distribution over swapping \( U \) and \( V \)’s prediction for the \( n \)th entry. All in all, this definition of \( \mathbb{P}[U^\circ, V^\circ] \) for the null distribution gives us a uniform distribution over all \( 2^N \) ways swapping of the labels and the individual entries of the observed predictions \( u \) and \( v \). And, importantly, the joint distribution \( \mathbb{P}[U^\circ, V^\circ] \), encodes the fact that the sampled entries are independent of the system label.

**The test statistic and the p-value.** The final ingredient we need in a null hypothesis test is a **test statistic**, whose job it is to provide a summary of samples \( (u', v') \sim \mathbb{P}[U^\circ, V^\circ] \) and thereby facilitate comparison of samples from the null distribution \( \mathbb{P}[U^\circ, V^\circ] \) and the observed entries \( (u, v) \). In this work, we will define a test statistic as function \( t(u, v) \). In principle, we can choose any test statistic \( t \) that allows us to distinguish \( u \) and \( v \), i.e., we have have \( t(u, v) = 0 \iff u = v \). Now, given observed entries \( u \) and \( v \), the p-value is defined as

\[
p = \mathbb{P}[t(U^\circ, V^\circ) \geq \xi] \quad (2)
\]

where \( \xi \overset{\text{def}}{=} t(u, v) \) is the observed effect. In words, the p-value is the probability of observing a test statistic \( t(u', v') \) with a value as large as \( t(u, v) \) where \( (u', v') \sim \mathbb{P}[U^\circ, V^\circ] \) are sampled from the null distribution. Recall that the system labels and entries are independent under the null distribution by construction, so the p-value tells us, under the independence assumption, how likely such a large test statistic would have been observed. The test says that we have sufficient evidence to reject the null hypothesis when \( p < \alpha \). These concepts are depicted in Fig. 2.

**Figure 2:** Depiction of the pmf of the test statistic \( t \), the p-value, and the observed value \( \xi \).
3 Structured Test Statistics

We now discuss a common special case of the paired-permutation test where the test statistic has a particular structure. In §4, we show how to exploit this structure to develop an efficient algorithm to exactly compute the test. The specific assumption we make is that the test statistic is an integer-valued additively decomposable function. Formally, this assumption means that we can rewrite \( t(u, v) \) as follows

\[
t(u, v) \overset{\text{def}}{=} h(g(u, v))
\]

for any function \( h \) and additively decomposable function \( g(u, v) \overset{\text{def}}{=} \sum_{n=1}^{N} g(u_n, v_n) \) such that \( g \) is an integer-valued function with a range of size \( \mathcal{O}(G) \). The structure of (3) will allow us to derive an efficient algorithm for evaluating \( \mathbb{P}[t(U^g, V^g)] = \mathbb{P}[h(\sum_{n=1}^{N} g(u_n, v_n))] \).

We now dissect this equation. Each summand \( g(u_n, v_n) \) can take on one of two values \( \xi_n \overset{\text{def}}{=} g(u_n, v_n) \) and \( \xi_n \overset{\text{def}}{=} g(v_n, u_n) \) with equal probability. We rewrite the sum \( \sum_{n=1}^{N} g(u_n, v_n) \) as \( S \overset{\text{def}}{=} \sum_{n=1}^{N} Z_n \) where \( Z_n \) are uniform RVs over the set \( \{\xi_n, \xi_n\} \). Each \( Z_n \) has probability mass function (PMF)

\[
f_n(z) \overset{\text{def}}{=} \text{pmf}_Z(\xi_n, \xi_n)(z)
\]

The domain of each PMF, \( \text{dom}(f_n) \), contains at most two elements. Let \( S \overset{\text{def}}{=} \text{dom}(S) \). Clearly, \( |S| = \mathcal{O}(GN) \) as we have a sum over \( N \) RVs \( Z_n \) each with domain size \( \mathcal{O}(G) \). The following theorem shows that we can evaluate \( \mathbb{P}[t(U^g, V^g)] \) from the distribution of \( S \), which we will show in the next section is efficient to compute.

**Theorem 1.** For any test statistic \( t \) that factorizes as in (3) with \( h \) and \( g \), the distribution of the test statistic under the null distribution decomposes as

\[
\mathbb{P}[t(U^g, V^g)] = \mathbb{P}[h(S)]
\]

**Proof.**

\[
\mathbb{P}[t(U^g, V^g)] = \mathbb{P}
\left[
\begin{array}{c}
    h\left(\sum_{n=1}^{N} g(U_n, V_n^g)\right)
\end{array}
\right]
\]

(6a)

\[
= \mathbb{P}\left[h\left(\sum_{n=1}^{N} Z_n\right)\right]
\]

(6b)

\[
= \mathbb{P}[h(S)]
\]

(6c)

\[\blacksquare\]

```
1: def monte_carlo(u, v, g, h, K):
2:   for n = 1 to N:
3:     xi_n ← g(u_n, v_n) # Local effect (stay)
4:     xi_n ← g(v_n, u_n) # Local effect (swap)
5:   f_n ← pmf_z(xi_n, xi_n)
6:   bar xi ← h(∑_{n=1}^{N} xi_n) # Compute observed effect
7:   # Sample K random stay or swap actions from each local PMF f_n,
8:   z_n{k} ← f_n for n = 1 to N, k = 1 to K
9:   return \(\frac{1}{K} \sum_{k=1}^{K} \mathbb{1}\left[h\left(\sum_{n=1}^{N} z_n{k}\right) \geq \bar{\xi}\right]\)
```

Algorithm 1: Monte Carlo approximation algorithm for the paired-permutation test.

**Example.** A common test statistic is the difference in accuracy, in which each entry \( u_n \in \{1, ..., C\}^G \) where \( C \) is the number of classes and in this case, \( G \) is the maximum length of an entry sequence (or one if each entry has a binary accuracy value). Then \( g(u_n, v_n) \in \{-G, ..., G\} \) is the difference in the number of correct predictions between individual entries \( u_n \) and \( v_n \). We can additionally define the function \( h \) as either \( h(x) = x \) or \( h(x) = \lfloor x \rfloor \) depending on whether we want a one-tailed or two-tailed significance test.

A Monte Carlo paired-permutation test. The best of our knowledge, no practical exact algorithm for the paired-permutation test has been given in the literature. Thus, most practical implementations of the paired-permutation test use an MC approximation, whereby one randomly samples from \( S \) to approximate \( \mathbb{P}[U^g, V^g] \). We give this MC algorithm as monte_carlo in Alg 1 which runs in \( \mathcal{O}(KN) \) time where \( K \) is the number of samples taken.

4 An Exact Paired-Permutation Test

In this section, we describe two exact, efficient algorithms for computing the \( p \)-value under the paired-permutation test for any structured test statistic (see (3)). Our algorithms hinge on an important theorem in probability: The PMF of the sum of independent events is the convolution of their individual PMFs (Ross, 2008, p. 252). Let \( f_S \) denote the PMF

\[\text{In App. B, we extend our algorithms to work with test statistics that use } m \text{ additively decomposable scoring functions, e.g., difference in } F_1 \text{ scores.}\]
1: \textbf{def} exact_perm_test\((u,v,g,h)\):
2: \textbf{for} \(n=1\) to \(N\):
3: \(ξ_n ← g(u_n,v_n)\) \hspace{1em} \text{Local effect (stay)}
4: \(ξ_n ← g(v_n,u_n)\) \hspace{1em} \text{Local effect (swap)}
5: \(f_n ← \text{pmf}_Z(ξ_n,ξ_n)\)
6: \(ξ ← h\left(\sum_{n=1}^{N} ξ_n\right)\) \hspace{1em} \text{Compute observed effect}
7: \(f_S ← f_1 * \cdots * f_N\) \hspace{1em} \text{Convolves the } f_n\text{'s}
8: \textbf{return} \(\sum_{ξ∈S} f_S(ξ) 1[\text{dom}(h(ξ)) ≥ ξ]\)

Algorithm 2: Compute the exact \(p\) value for the paired-permutation test for structured test statistics.

Lemma 2. For any two entries, \(u\) and \(v\), and test statistic \(t\) that factorizes as in (3) with \(h\) and \(g\), exact_perm_test\((u,v,g,h)\) returns \(p\) in \(O(GN+r(G,N))\) time, \(O(N+s(G,N))\) space. We define \(r(G,N)\) and \(s(G,N)\) as the time and space complexities for constructing \(f_1 * \cdots * f_N\).

Proof. The correctness of exact_perm_test is by Theorem 1 and (7c). All lines except for Line 7 and Line 9 require at most \(O(N)\) time and space. Line 9 runs in \(O(GN)\) time and \(O(1)\) space. Thus, exact_perm_test runs in \(O(N+GN+r(G,N))\) time and \(O(N+s(G,N))\) space.

The computational question is then: What is the most efficient algorithm for evaluating \((f_1 * \cdots * f_N)(ξ)\)? In the following two subsections, we present \(O(GN^2)\) time and \(O(GN log GN)(log N))\) time algorithms for performing this \(N\)-fold convolution.

4.1 Convolution by Dynamic Programming

Our first approach builds a dynamic program (DP) that takes advantage of the sparsity of our RVs to efficiently construct the PMF \(f_S\). We do this by constructing a PMF array \(F_n ∈ \mathbb{R}^S\) for \(n ∈ \{0, ..., N\}\) (we use \(n = 0\) as an initialization base case) such that \(F_n(ξ) = (f_1 * \cdots * f_n)(ξ)\). As we apply each convolution, we know that \(f_n\) is only non-zero at \(ξ_n\) and \(ξ_{n-1}\), and so we can run each convolution in \(O(GN)\) time. The pseudocode for this approach is given as convolve\_DP in Alg 3.

Theorem 3. For any RVs \(Z_1, ..., Z_N\) with PMFs \(f_1, ..., f_N\), convolve\_DP\((f_1, ..., f_N)\) returns \(f_1 * \cdots * f_N\) in \(O(GN^2)\) time, \(O(GN)\) space.

Proof. The proof of correctness of convolve\_DP is given in App. A. Each \(F_n\) has \(O(GN)\) elements and so convolve\_DP clearly runs in \(O(GN^2)\) time. Furthermore, at any iteration, we only require \(F_n\) and \(F_{n-1}\) and so we can execute convolve\_DP in \(O(GN)\) space.

4.2 Convolution by FFT

Our second approach uses the fast Fourier transform (FFT; Cooley and Tukey, 1965; Cormen et al., 2022) to evaluate the convolutions. Using this method means that each convolution takes \(O(GN log GN)\) time \(O(GN)\) space. We further exploit the commutativity of convolution to perform the \(N\)-fold convolution in \(log N\) convolutions using a recursive program. The pseudocode for this approach is given as convolve\_FFT in Alg 4.
1: \textbf{def} \texttt{convolve\_FFT}(f_{1},...,f_{N}) :
2: \hspace{1em} \textbf{if} \ N = 1 : \textbf{return} f_{1}
3: \hspace{1em} \textbf{return} \texttt{convolve\_FFT}(f_{1},...,f_{N/2})
4: \hspace{2em} \ast \texttt{convolve\_FFT}(f_{N/2+1},...,f_{N})

Algorithm 4: Recursive algorithm to compute the pmf of S as the N-fold convolution of the pmfs \(f_{1},...,f_{N}\). Here the convolution operation \((\ast)\) runs in \(O(GN \log GN)\) time thanks to the FFT (Cooley and Tukey, 1965).

**Theorem 4.** For any RVs \(Z_{1},...,Z_{N}\) with PMFs \(f_{1},...,f_{N}\), \(\texttt{convolve\_FFT}(f_{1},...,f_{N})\) returns \(f_{1} \ast \cdots \ast f_{n}\) in \(O(GN(\log GN)(\log N))\) time, \(O(GN \log GN)\) space.

**Proof.** The correctness of \(\texttt{convolve\_FFT}\) is due to Cooley and Tukey (1965). The recursion of \(\texttt{convolve\_FFT}\) can be given as

\[
T(N) = 2T\left(\frac{N}{2}\right) + O(GN \log GN) \quad (9)
\]

Solving this recursion, we call \(T \mathcal{O}(\log N)\) times. Therefore, the time complexity of \(\texttt{convolve\_FFT}\) is \(O(GN(\log GN)(\log N))\). Similarly, each call requires \(O(GN)\) space and \(\texttt{convolve\_FFT}\) has a space complexity of \(O(GN \log N)\).

**Corollary 1.** For any two entries, \(u\) and \(v\), and test statistic \(t\) that factorizes as in (3) with \(h\) and \(g\), \(\texttt{exact\_perm\_test}(u,v,g,h)\) returns \(p\) in \(O(GN(\log GN)(\log N))\) time, \(O(GN)\) space.

**Proof.** The correctness and complexity bounds are due to Theorem 2 and Theorem 4. Specifically, Line 7 can be executed using \texttt{convolve\_FFT}. 

5 Experiments

We demonstrate the efficiency of our exact algorithms by simulating paired-permutation tests between the accuracy of two systems. In order to have some control over the \(p\)-value, \(N\), and \(G\) (maximum length of a sentence), we randomly generate our two system outputs from a measured distribution. Specifically, we will use the Stanza\(^{3}\) (Qi et al., 2020) part-of-speech tag accuracy statistics when evaluating on the English Universal Dependencies (UD) test set (Nivre et al., 2018). We sample our outputs from the normal distribution where the mean and standard deviation match the rates of Stanza’s observed accuracy. We further sample the length of each sample sentence according to the distribution of lengths in the test set. These distributions are provided in Tab. 1.

We show that, empirically, the exact test is more efficient than the MC approximation; this is evinced in Fig. 1 where we have compared the runtime of \(\texttt{exact\_perm\_test}\) using \(\texttt{convolve\_DP}\) and \(\texttt{convolve\_FFT}\) against \(\texttt{monte\_carlo}\) for various sample sizes \((K \in \{5000,10000,20000,40000\})\).\(^{4}\) We note that using \(\texttt{convolve\_DP}\) is already more efficient than running \(\texttt{monte\_carlo}\) with \(K = 40000\) and \(K = 20000\) (up to \(N \approx 8000\)).\(^{5}\) Furthermore, \(\texttt{convolve\_FFT}\) is much faster and we observe a speed-up between 3x and 30x, depending on the number of samples \(K\). Indeed, using \(\texttt{convolve\_FFT}\) allows us to perform an exact paired-permutation test for \(N = 10000\) in approximately one-tenth of a second.

6 Conclusion

We presented an algorithm to compute the exact \(p\)-value of a paired-permutation test for the case of a family of structured test statistics, including the difference in accuracy. Our algorithm runs in \(O(GN(\log GN)(\log N))\) time and requires \(O(GN)\) space. We empirically show that our exact algorithm is faster than Monte Carlo approximation techniques. The theory of our work is extensible to a more general class of test statistics which we discuss in App. B. We hope that this work encourages the use of exact paired-permutation tests in future NLP research.

### Ethical Concerns

We foresee no ethical concerns in this work.

---

\(^{3}\)The code and pre-trained model are both freely accessible at https://github.com/stanfordnlp/stanza.

\(^{4}\)The experiment used an Apple M1 Max processor.

\(^{5}\)We note that the average test set size of the 129 UD treebanks we examined is just over 1000 sentences, and only three treebanks had more than 6000 sentences.
Acknowledgments

We would like to thank the reviewers for their invaluable feedback and time spent engaging with our work. The first author is supported by the University of Cambridge School of Technology Vice-Chancellor’s Scholarship as well as by the University of Cambridge Department of Computer Science and Technology’s EPSRC.

References

Taylor Berg-Kirkpatrick, David Burkett, and Dan Klein. 2012. An empirical investigation of statistical significance in NLP. In Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, EMNLP-CoNLL 2012, July 12-14, 2012, Jeju Island, Korea, pages 995–1005. ACL.

Jonathan H. Clark, Chris Dyer, Alon Lavie, and Noah A. Smith. 2011. Better hypothesis testing for statistical machine translation: Controlling for optimizer instability. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pages 176–181, Portland, Oregon, USA. Association for Computational Linguistics.

James W. Cooley and John W. Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation, 19(90):297–301.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2022. Introduction to Algorithms, 4 edition. MIT Press.

Daniel Deutsch, Rotem Dror, and Dan Roth. 2021. A statistical analysis of summarization evaluation metrics using resampling methods. Transactions of the Association for Computational Linguistics, 9:1132–1146.

Thomas G. Dietterich. 1998. Approximate statistical tests for comparing supervised classification learning algorithms. Neural Computation, 10(7):1895–1923.

Rotem Dror, Gili Baumer, Segev Shlomov, and Roi Reichart. 2018. The hitchhiker’s guide to testing statistical significance in natural language processing. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1383–1392, Melbourne, Australia. Association for Computational Linguistics.

Rotem Dror, Lotem Peled-Cohen, Segev Shlomov, and Roi Reichart. 2020. Statistical Significance Testing for Natural Language Processing. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

Phillip Good. 2000. Permutation Tests A Practical Guide to Resampling Methods for Testing Hypotheses. Springer.

Elizabeth Koehler, Elizabeth Brown, and Sebastien J. Haneuse. 2009. On the assessment of Monte Carlo error in simulation-based statistical analyses. The American Statistician, 63(2):155–162.

Philipp Koehn. 2004. Statistical significance tests for machine translation evaluation. In Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing, pages 388–395, Barcelona, Spain. Association for Computational Linguistics.

Erich Leo Lehmann and Joseph P. Romano. 2005. Testing Statistical Hypotheses. Springer.

Steven S. Muchnick. 1998. Advanced Compiler Design and Implementation. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Joakim Nivre, Mitchell Abrams, Željko Agić, Lars Ahrenberg, Lene Antonsen, Katya Aplonova, Maria Jesus Aranzabe, Gashaw Arutie, Masayuki Asahara, Luma Ateyah, Mohammed Atta, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbuc Mititelu, Victoria Basmov, John Bauer, Sandra Bellato, Kepa Bengoetxea, Yevgeny Berzak, Irshad Ahmad Bhat, Riyaz Ahmad Bhat, Erciica Biagetti, Eckhard Bick, Rogier Blokland, Victoria Bobicev, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Adriaine Boyd, Aljoscha Burghardt, Mariska Candito, Bernard Caron, Gauthier Caron, Gulsen Cebiroglu Eryigit, Flavio Massimiliano Cecchini, Giuseppe G. A. Celano, Slavomír Čeplo, Savas Cetin, Fabrizio Chalub, Jinho Choi, Yongseok Cho, Jayeol Chun, Silvie Cinková, Aurélie Collomb, Çağrı Çöltekin, Miriam Connor, Marine Courtin, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Aranza Diaz de Ilarraza, Carly Dickerson, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Ali Elkahky, Binyam Ephrem, Tomaž Erjavec, Aline Etienne, Richard Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Sebastian Garza, Kim Gerdes, Filip G¨unter, Iakes Goenaga, Koldo Gojenola, Memduh G¨ok¨umark, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzitis, Bruno Guillaume, Céline Guillot-Barbance, Nizar Habash, Jan Hajiˇc, Jan Hajiˇc jr., Linh Hà M ˜y, Na-Rae Han, Kim Harris, Dag Haug, Barbora Hladká, Jaroslava Hlaváˇcová, Florinoc Hociung, Petter Hohle, Jena Hwang, Radu Ion, Elenia Irinuma, Olajídé Ishola, Tomáš Jelínek, Andreas Johannsen, Fredrik Jørgensen, Húner Kajskarsa, Sylvain Kahane, Hiroshi Kanayama, Jenny Kanerva, Boris Katz, Tolga Kayadelean, Jessica Kenney, Václava Kettnerová, Jesse Kirchner, Kamil Kopacewicz, Natalia Kotsyba, Simon Krek, Sookyeong Kwak, Veronika Laippala, Lorenzo Lambertino, Lucia Lam, Tatiana Lando, Septina Dian Larasati,
Alexei Lavrentiev, John Lee, Phuong Lê Hồng, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, Kyung-Tae Lim, Nikola Ljubešić, Olga Loginova, Olga Lyeshevskaya, Teresa Lynn, Mackenzie McAuley, Aieb Makazhanov, Michael Mandl, Christopher Manning, Ruli Manurung, Catia Marandante, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Cátia Mátuling, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Keiko Morie, Shiinsuke Mori, Bjartur Mortsensen, Bohdan Moskalovskyi, Kadri Muschnek, Yugo Murawaki, Kaili Müürisep, Pinkey Nainwani, Juan Ignacio Navarro Horiaček, Anna Nedoluzhko, Guntas Nešpore-Bērziņa, Luong Nguyên Thi, Huyễn Nguyên Thị Minh, Vitaly Nikolaev, Rattima Nitsaroj, Hanna Nurmi, Stina Ojala, Adédayo Olatunji, Ma Omura, Petra Osenova, Robert Ostling, Lilja Øvrelid, Niko Partanen, Elena Pascaul, Marco Passarotti, Agnieszka Patejuk, Guilherme Paulino-Passos, Siyao Peng, Benoit Sagot, Shoval Sadde, Benoît Sagot, Shadi Saleh, Tanja Samardžić, Stephanie Samson, Manuela Sanguinetti, Baiba Saulēte, Vahe Salahi, Manolita Simanek, Nathan Schneider, Sebastian Schuster, Djamé Sedah, Wolfgang Secker, Moigan Seraji, Mo Shen, Atsuko Shimada, Muh Shohibussirri, Dmitry Sichenin, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkö, Mária Simkövá, Kirill Simov, Aaron Smith, Isabel Soares-Bastos, Carolyn Spadine, Antonio Stella, Milan Straka, Ana Stremjavá, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Yuta Takahashi, Takaaki Tanaka, Isabelle Tellier, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, Marat M. Yavrumyan, Zhuoran Yu, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, Manying Zhu. 2018. Universal dependencies 2.3. LIN-DAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Markus Ojala and Gemma C. Garriga. 2010. Permutation tests for studying classifier performance. *The Journal of Machine Learning Research*, 11:1833–1863.

Maxime Peyrard, Wei Zhao, Steffen Eger, and Robert West. 2021. Better than average: Paired evaluation of NLP systems. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers)*, Virtual Event, August 1-6, 2021, pages 2301–2315. Association for Computational Linguistics.

Peng Qi, Yuhao Zhang, YuHui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the Association for Computational Linguistics: System Demonstrations*.

Sheldon Ross. 2008. *A First Course in Probability*, 8th edition. Pearson.

Ronald C. Serlin. 2000. *Testing for robustness in Monte Carlo studies*. *Psychological Methods*, 5(2):230.

Alexander Yeh. 2000. More accurate tests for the statistical significance of result differences. In *COLING 2000 Volume 2: The 18th International Conference on Computational Linguistics*.
A Proof of Correctness of convolve_DP

We prove the correctness of convolve_DP using the following lemma.

Lemma 1. For any \(N\) RVs \(Z_1, \ldots, Z_N\) with PMFs \(f_1, \ldots, f_N\) respectively and \(n \in \{1, \ldots, N\}\), convolve_DP\((f_1, \ldots, f_N)\) constructs \(F_n\) such that for any \(\xi \in \mathcal{S}\),

\[
F_n(\xi) = f_n(\xi) \overset{\text{def}}{=} (f_1 \ast \cdots \ast f_n)(\xi) \tag{10}
\]

Proof. We prove this by induction on \(N\).

Base case: \(N = 1\). We have that \(F_0(0) = 1\) and \(F_0(\xi) = 0\) for all \(\xi \in \mathcal{S} \setminus \{0\}\). Therefore, \(F_1(\xi_0) = F_1(\xi_1) = \frac{1}{2}\) and \(F_1(\xi) = 0\) for all \(\xi \in \mathcal{S} \setminus \{\xi_0, \xi_1\}\) as expected.

Inductive step: Assume (10) holds for \(N = n - 1\). Let \(N = n\) and consider \(f_{(n-1)} \ast f_n\).

\[
(f_{(n-1)} \ast f_n)(\xi) = \sum_{\xi' \in \mathcal{S}} f_{(n-1)}(\xi') f_n(\xi - \xi') = \sum_{\xi' \in \text{dom}(f_n)} f_{n-1}(\xi + \xi') f_n(\xi') \tag{11}
\]

This is exactly the construction in the for-loop between Line 7 and Line 8. Therefore, \(F_n(\xi) = f_n(\xi)\). \(\blacksquare\)

As \(F_N\) will contain the \(N\)-fold convolution \(f_1 \ast \cdots \ast f_N\), convolve_DP is correct by definition.

B Paired-Permutation Test for Higher-order Test Statistics

In this section, we extend our approach for the paired-permutation test to test statistics that are functions of \(m\) additively decomposable functions. In symbols, this assumption means that we can rewrite \(t\) as follows

\[
t(u, v) \overset{\text{def}}{=} h(g_1(u, v), \ldots, g_m(u, v)) \tag{12}
\]

for any function \(h\) and integer-valued, additive decomposable functions \(g_i\) (i.e., \(g_i(u, v) \overset{\text{def}}{=} \sum_{n=1}^{N} g_i(u_n, v_n)\)). We now define \(\xi_n^u\) and \(\xi_n^v\) as \(m\)-tuples,

\[
\xi_n^u \overset{\text{def}}{=} (g_1(u, v), \ldots, g_m(u, v)) \tag{13}
\]

\[
\xi_n^v \overset{\text{def}}{=} (g_1(v, u), \ldots, g_m(v, u)) \tag{14}
\]

And so each RV \(Z_n\) has the same PMF as in (4b). We can then define an analogous function to exact_perm_test for the case of \(m\) additively decomposable functions. We give pseudocode for this as exact_perm_test\(_m\) in Alg 5. The convolution algorithms, convolve_DP and convolve_FFT, can both be used to perform for convolution step in Line 7.

Theorem 5. For any two entries, \(u\) and \(v\), and test statistic \(t\) that factorizes as in (12) with \(h\) and \(g_1\) to \(g_m\), exact_perm_test\(_m\)(\(u, v, g_1, \ldots, g_m, h\)) returns \(p\) in \(\mathcal{O}(G^m N^m (\log GN)(\log N))\) time and \(\mathcal{O}(G^m N^m \log N)\) space.

Proof. The proof of correctness for exact_perm_test\(_m\) is the same as Theorem 2. The expensive operation in the algorithm is the convolution step (Line 7). We can perform a FFT \(m\)-dimensional convolution in \(\mathcal{O}(G^m N^m \log GN)\) time and \(\mathcal{O}(G^m N^m)\) space. As we require \(\mathcal{O}(\log N)\) convolution steps, exact_perm_test\(_m\) runs in \(\mathcal{O}(G^m N^m (\log GN)(\log N))\) time and \(\mathcal{O}(G^m N^m \log N)\) space. \(\blacksquare\)

Example. A common example for a test statistic that requires multiple additively decomposable functions is the difference in \(F_1\) scores. Similar to accuracy, each entry \(u_n \in \{1, \ldots, C\}\) and \(G\) is the maximum length of an entry sequence. Let \(tp(u_n)\) and \(in(u_n)\) be the number of true positive and incorrect predictions made in entry \(u_n\) respectively. Then the difference in \(F_1\) scores can be given as

\[
t(u, v) \overset{\text{def}}{=} \frac{\sum_{n=1}^{N} tp(u_n)}{\sum_{n=1}^{N} tp(u_n) + \frac{1}{2} \sum_{n=1}^{N} in(u_n)} - \frac{\sum_{n=1}^{N} tp(v_n)}{\sum_{n=1}^{N} tp(v_n) + \frac{1}{2} \sum_{n=1}^{N} in(v_n)} \tag{15}
\]
Algorithm 5: Pseudocode to find exact $p$ value for the paired-permutation test for test statistics comprised of $m$ additively decomposable functions.

We can therefore use four-additively decomposable functions, $g_1$ to $g_4$, that decompose such that $g_1(u_n, v_n) = g_3(v_n, u_n) = \text{tp}(u_n)$ and $g_2(u_n, v_n) = g_4(v_n, u_n) = \text{in}(u_n)$. Our $h$ function then takes four arguments and can be defined as

$$h(x_1, x_2, x_3, x_4) \equiv \frac{x_1}{x_1 + \frac{1}{2} x_2} - \frac{x_3}{x_3 + \frac{1}{2} x_4} \quad (16)$$

We can additionally apply an absolute value to $h$ to check for the absolute difference in $F_1$ scores; doing this would make the significance test two-tailed rather than one-tailed.