Accelerating Permutation Testing in Voxel-wise Analysis through Subspace Tracking: A new plugin for SnPM

Felipe Gutierrez-Barragan[a], Vamsi K. Ithapu[a], Chris Hinrichs[a], Camille Maumet[d], Sterling C. Johnson[c], Thomas E. Nichols[d], Vikas Singh[b,a], and the Alzheimer's Disease Neuroimaging Initiative [1]

[a]*Department of Computer Sciences, University of Wisconsin-Madison*
[b]*Department of Biostatistics & Med. Informatics, University of Wisconsin-Madison*
[c]*Department of Medicine, University of Wisconsin-Madison and William S. Middleton Veteran's Hospital*
[d]*Department of Statistics, The University of Warwick*
`http://felipegb94.github.io/RapidPT/`

Abstract

Permutation testing is a non-parametric method for obtaining the max null distribution used to compute corrected p-values to provide strong control of false positives. In neuroimaging, however, the computational burden of running such an algorithm can be significant. We find that by viewing the permutation testing procedure as the construction of a very large permutation testing matrix, $\mathbf{T}$, one can exploit structural properties derived from the data and the test statistics to reduce the runtime under certain conditions. In particular, we see that $\mathbf{T}$ has a low-rank plus a low-variance residual. This makes $\mathbf{T}$ a good candidate for low-rank matrix completion, where only a very small number of entries of $\mathbf{T}$ ($\sim 0.35\%$ of all entries in our experiments) have to be computed to obtain a good estimate. Based on this observation, we present RapidPT, an algorithm that is able to efficiently recover the max null distribution commonly obtained through regular permutation testing in voxel-wise analysis. We present an extensive validation on four varying sized datasets against two baselines: Statistical NonParametric Mapping (SnPM13) and a standard permutation testing implementation (referred as NaivePT). We find that RapidPT achieves its best runtime performance on medium sized datasets ($50 \leq n \leq 200$), with speedups of 1.5x - 38x (vs. SnPM13) and 20x-1000x (vs. NaivePT). For larger datasets ($n \geq 200$) RapidPT outperforms NaivePT (6x - 200x) on all datasets, and provides large speedups over SnPM13 when more than 10000 permutations (2x - 15x) are needed. The implementation is a standalone toolbox and also integrated within SnPM13, able to leverage multi-core architectures when available.

Keywords: Voxel-wise analysis, Hypothesis test, Permutation test, Low-rank matrix completion

Corresponding Author(s): Felipe Gutierrez Barragan (fgutierrez3@wisc.edu)

[1]Data used in preparation of this article was obtained from the Alzheimer's Disease Neuroimaging Initiative (ADNI) database (adni.loni.ucla.edu). As such, the investigators within the ADNI contributed to the design and implementation of ADNI and/or provided data but did not participate in analysis or writing of this report. A complete listing of ADNI investigators can be found at: `http://adni.loni.ucla.edu/wp-content/uploads/how_to_apply/ADNI_Acknowledgement_List.pdf`

1. Introduction

Nonparametric voxel-wise analysis, e.g., via permutation tests, are widely used in the brain image analysis literature. Permutation tests are often utilized to control the family-wise error rate (FWER) in voxel-wise hypothesis testing. As opposed to parametric hypothesis testing schemes K.J. Friston (1995); Worsley et al. (1992, 1996), nonparametric permutation tests Holmes et al. (1996); Nichols and Holmes (2002) can provide exact control of false positives while making minimal assumptions on the data. Further, despite the additional computational cost, permutation tests have been widely adopted in image analysis Arndt et al. (1996); M. Halber and Heiss (1997); Holmes et al. (1996); A.P. Holmes and Nichols (1998); Nichols and Holmes (2002); Nichols and Hayasaka (2003) via implementations in broadly used software libraries available in the community SnPM (2013); FSL (2012); Winkler et al. (2014).

Running time aspects of Permutation Testing. Despite the varied advantages of permutation tests, there is a general consensus that the computational cost of performing permutation tests in neuroimaging analysis can often be quite high. As we will describe in more detail shortly, high dimensional imaging datasets essentially mean that *for each permutation*, hundreds of thousands of test statistics need to be computed. Further, as imaging technologies continue to get better (leading to higher resolution imaging data) and the concurrent slowdown in the predicted increase of processor speeds (Moore's law), it is reasonable to assume that the associated runtime will continue to be a problem in the short to medium term. To alleviate these runtime costs, ideas that rely on code optimization and parallel computing have been explored Eklund et al. (2011); A. Eklund (2012, 2013). These are interesting strategies but any hardware-based approach will be limited by the amount of resources at hand. Clearly, significant gains may be possible if *more efficient schemes* that exploit the underlying structure of the imaging data were available. It seems likely that such algorithms can better exploit the resources (e.g., cloud or compute cluster) one has available as part of a study and may also gain from hardware/code improvements that are being reported in the literature.

Data acquired in many scientific studies, especially imaging and genomic data, are highly structured. Individual genes and/or individual voxels share a great deal of commonality with other genes and voxels. It seems reasonable that such correlation can be exploited towards better (or more efficient) statistical algorithms. For example, in genomics, Cheverud (2001) and Li and Ji (2005) used correlations in the data to estimate the effective number of independent tests in a genome sequence to appropriately threshold the test statistics. Also motivated by bioinformatics problems, Knijnenburg et al. (2009) approached the question of estimating the tail of the distribution of permutation values via an approximation by a generalized Pareto distribution (using fewer permutations). In the context of more general statistical analysis, the authors in Subramanian et al. (2005) proposed Gene Set Enrichment Analysis (GSEA) which exploits the underlying structure among the genes, to analyze gene-sets (e.g., where sets were obtained from biological pathways) instead of individual genes. If the genes within a gene-set have similar expression pattern, one may see improvements in statistical power. This idea of exploiting the "structure" towards efficiency (broadly construed) was more rigorously studied in Efron and Tibshirani (2007) and a nice non-parametric Bayesian perspective was offered in Dahl and Newton (2007). Within neuroimaging, a similar intuition drives Random Field theory based analysis Taylor and Worsley (2008), albeit the objective there is to obtain a less conservative correction, rather than computational efficiency. Recently, motivated by neuroimaging applications and computational issues, Gaonkar and Davatzikos (2013) derived an analytical approximation of statistical significance maps to reduce the computational burden imposed by permutation tests commonly used to identify which brain regions contribute to a Support Vector Machines (SVM) model. In summary, exploiting the structure of the data to obtain alternative efficient solutions is not new, but we find that in the context of permutation testing on *imaging data*, there is a great deal of voxel-to-voxel correlations that if leveraged properly can, in principle, yield interesting new algorithms.

1.1. Main Idea and Contributions

The starting point of our formulation is to analyze the entire permutation testing procedure via matrix algebra. In particular, the object of interest – $\mathbf{T}$ – is the matrix of voxel-wise statistics. Each row of $\mathbf{T}$ corresponds to the voxels in the brain imaging space, and each column is a specific labeling (or permutation of the labels) of the data. This perspective is not commonly used because a typical permutation test in neuroimaging rarely instantiates or even operates on this matrix of statistics. Apart from the fact that $\mathbf{T}$, in general, contains millions of entries, the reason for not working *directly* with it is because typically the goal is to derive the maximum null distribution to calculate a p-value for each test statistic. However, the central aspect of this work is to *exploit* the structure in $\mathbf{T}$ – the spatial correlation across different voxel-statistics. Such correlations are not atypical because the statistics are computed from anatomically correlated regions in the brain. Even far apart voxel neighbourhoods are inherently correlated because of the underlying biological structures. This idea drives the proposed novel permutation testing procedure. We first describe the contributions of this paper based on the observation that the permutation testing matrix is filled with related entries.

- **Classical permutation testing does redundant computations.** The fundamental claim we make in this work is – *classical permutation testing in high-dimensions is extremely redundant*. By classical, we mean the traditional resampling approach where *all* the voxel-wise statistics (i.e., all the entries of $\mathbf{T}$) are computed one-by-one. We formulate a novel multiple-testing framework based on this observation – we primarily justify this claim through extensive experiments beyond theoretical motivations. Based on this premise, a framework for estimating (or recovering) most of the statistics in $\mathbf{T}$ rather than "explicitly" computing them is proposed. We then provide two technical results for this framework. The first result justifies the modeling assumptions and several of the components involved in recovering $\mathbf{T}$. The second result shows that the error in the *global* maximum null distribution (i.e., the histogram of the maximum of voxel-wise statistics for a given permutation) is arbitrarily small.

- **A novel, fast and robust, multiple-hypothesis testing procedure.** Building upon the theoretical development, we propose a novel algorithm for permutation testing involving high-dimensional data. We refer to the algorithm as *RapidPT*, and we show that compared to existing state-of-the-art libraries for non-parametric testing, the proposed model achieves approximately 20$\times$ speed up over existing procedures. We further identify regimes where the speed up is even higher. RapidPT also is able to leverage serial and parallel computing environments seamlessly.

- **A plugin in SnPM (with stand-alone libraries).** Given the importance and the wide usage of permutation testing in neuroimaging (and other studies involving high-dimensional and multimodal data), we introduce a heavily tested implementation of RapidPT integrated as a plugin within the current development version of SnPM — a widely used non-parametric testing toolbox. Users can invoke RapidPT directly from within the SnPM graphical user interface and benefit from SnPM's familiar pre-processing and post-processing capabilities without a separate installation, bringing the performance promised by the theorems to a toolbox broadly utilized by the community in practice. Our extensive documentation Gutierrez-Barragan and Ithapu (2016) gives an overview of how to use RapidPT within SnPM.

We point out that the notion of exploiting correlations and the low-rank of $\mathbf{T}$ was introduced by our group at a conference Hinrichs et al. (2013). A few years later, the authors of Winkler et al. (2016) utilized this structure within several heuristics for speeding up permutations testing in GLMs. One of their approaches relies on our proposed notion of correlations between voxel-wise statistics. Although the present work shares some of the goals and motivation of Winkler et al. (2016) – specifically, utilizing the algebraic structure of T – there are substantial technical differences in the present approach, which we outline further below. First, unlike Winkler et al. (2016), we directly study permutation testing for images at a more fundamental level and seek to characterize mathematical properties of relabeling (i.e., permutation) procedures operating on high-dimensional imaging data. This is different from assessing whether the underlying operations of classical statistical testing procedures can be reformulated (based on the correlations) to reduce computational burden as in Winkler et al. (2016). Second, by exploiting celebrated technical results in random matrix theory, we provide theoretical guarantees for estimation and recovery of $\mathbf{T}$. Few such results were known. Note that empirically, our machinery avoids a large majority of the operations performed in Winkler et al. (2016). Third, several speed-up strategies proposed in Winkler et al. (2016) can be considered as special cases of our proposed algorithm — interestingly, if we were to increase the number 'actual' operations performed by RapidPT (from $\approx 1\%$, suggested by our experiments, to 50%), the computational workload begins approaching what is described in Winkler et al. (2016). Fourth, our experiments suggest that in regimes where we provide 15$\times$ or more speedup over classical permutation testing, the strategy in Winkler et al. (2016) reports a 4$\times$ slow down in runtime.

2. Permutation Testing in Neuroimaging

In this section, we first introduce some notations and basic concepts. Then, we give additional background on permutation testing for hypothesis testing in neuroimaging to motivate our formulation. Matrices and vectors will be denoted by bold upper-case and lower-case letters respectively, and scalars will be represented using non-bold letters. For a matrix $\mathbf{X}$, $\mathbf{X}[i,:]$ denotes the i^{th} row and $\mathbf{X}[i,j]$ denotes the element in i^{th} row and j^{th} column.

Permutation testing is a nonparametric procedure for estimating the empirical distribution of the global null Edgington (1969b,a); Edgington and Onghena (2007). For a two-sample univariate statistical test, a permutation test simply computes an *unbiased* estimate of the null distribution of the relevant univariate statistic (e.g., t or χ^2). Although univariate null distributions are, in general, well characterized, the sample maximum of the point-wise (voxel-wise) statistics usually does not have an analytical form due to strong correlations across voxels, as discussed in Section 1. Permutation testing is appropriate in this high-dimensional setting because it is non-parametric and does not impose any restriction on the correlations across the voxel-wise statistics. Indeed, when the test corresponds to group differences between samples based on a stratification variable, under the null hypothesis $\mathcal{H}_0$, the grouping labels given to the samples are artificial, i.e., they correspond to the set of all possible *relabellings* of the samples. In neuroimaging studies, typically the groups correspond to the presense or absence of an underlying disease condition (e.g., controls and diseased). Whenever $\mathcal{H}_0$ is true, the data sample representing a healthy subject is, roughly speaking, 'similar' to a diseased subject. Under this setting, in principle, *interchanging* the labels of the two instances will have no effect on the sampling distribution of the resulting voxel-wise statistics across all the dimensions (or covariates or features). So, if we randomly permute the labels of the data instances from both groups, under $\mathcal{H}_0$ the recomputed sets of voxel-wise statistics correspond to the same global null distribution. We denote the number of such relabellings (or permutations) by L. The histogram of all L maximum statistics i.e., the maximum across all voxel-wise statistics for a given permutation, is the empirical estimate of the *exact maximum* null distribution under $\mathcal{H}_0$. When given the true/real labeling (the labels for the samples in the two groups), to test for significant group-wise differences, one can simply compute the fraction of the max null that is more extreme than the maximum statistic computed across all voxels for this real labeling.

The case for strong null. Observe that when testing *multiple* sets of hypotheses there are two different 'family-wise control' procedures available. Our overall goal is to control the Type 1 error rate (FWER) incurred by evaluating the full set of hypotheses. There are two types of control of the FWER: weak and strong control Y. Hochberg (1987). A test is referred to as weak control for FWER whenever the Type 1 error rate is controlled only when all the hypotheses involved (here, the number of voxels being tested) are true. That is, $\mathcal{H}_0$ is true for (all) voxels. On the other hand, a test provides strong control for FWER whenever Type 1 error rate is controlled under any combination/proportion of the true hypotheses. It is known that the procedure described above (i.e., using the max null distribution calculated across all voxel-wise statistics) provides strong control of FWER Holmes et al. (1996). This is easy to verify because the maximum of all voxel-wise statistics is used to compute the corrected p-value, and so, the exact proportion of which hypotheses are true does not matter. Further, testing based on strong control will classify non-activated voxels as activated with a probability upper bounded by α, i.e., it has localizing power Holmes et al. (1996), a desirable property in neuroimaging studies in particular. For the remainder of this paper, we will focus on such strong control and restrict our presentation to the case of group difference analysis for two groups.

2.1. NaivePT*: The exhaustive Permutation Testing procedure*

Algorithm 1 summarizes a simple permutation testing algorithm. Given the data instances (or samples) from the two groups, $\mathbf{X}^1 \in \mathbb{R}^{v \times n_1}$ and $\mathbf{X}^2 \in \mathbb{R}^{v \times n_2}$, where n_1 and n_2 denote the number of subjects from the two groups respectively. Also, v denotes the number of voxels in the brain image, i.e., we are performing a total of v different statistical tests and computing the strong control FWER corrected p-values. Let $n = n_1 + n_2$ and $\mathbf{X} = [\mathbf{X}^1; \mathbf{X}^2]$ give the (row-wise) stacked data matrix ($\mathbf{X} \in \mathbb{R}^{v \times n}$). Note that a permutation of the columns of $\mathbf{X}$ corresponds to a group relabelling. The v distinct voxel-wise statistics are then computed for L such permutations in all which can be collected to construct the resultant *permutation testing matrix* $\mathbf{T} \in \mathbb{R}^{v \times L}$. The empirical estimate of the max null is simply the histogram of the maximum of each of the rows of $\mathbf{T}$ – denoted by h^L. Algorithm 1 is occasionally referred to as Monte Carlo permutation tests in the literature because of the random sampling involved in generating the statistics. This standard description of a permutation test will be used in the following sections to describe our proposed testing algorithm.

Algorithm 1 NaivePT The exhaustive permutation testing procedure.

```
Input: $\mathbf{X}^1$, $\mathbf{X}^2$, $L$, stat
Output: $\mathbf{T}$, $\mathrm{h}^L$
  $\mathbf{X} = [\mathbf{X}^1; \mathbf{X}^2]$, $n = n_1 + n_2$
  $m_1 \ldots, m_r \leftarrow [\varnothing]$
  for $i \in 1, \ldots, L$ do
    $j_1 \ldots, j_n \sim$ Permute$[1, n]$
    $\tilde{\mathbf{X}}^1 \leftarrow \mathbf{X}[:, j_1, \ldots, j_{n_1}]$, $\tilde{\mathbf{X}}^2 \leftarrow \mathbf{X}[:, j_{n_1+1}, \ldots, j_n]$
    $\mathbf{T}[:, i] \leftarrow \mathrm{test}(\tilde{\mathbf{X}}^1, \tilde{\mathbf{X}}^2)$
    $m_i \leftarrow$ Max$(\mathbf{T}[:, i])$
  end for
  $\mathrm{h}^L \leftarrow$ Histogram$(m_1, \ldots, m_L)$
```

2.2. The situation when L is large

Despite the exact (i.e., an unbiased estimation of the null) and non-parametric nature of permutation testing, it involves a significant computational footprint, making the exhaustive procedure from Algorithm 1 expensive in general. There are at least a few different reasons, partly due to a need to achieve highest possible power ($1 - \beta$, where β is the Type 2 error rate) while taking into account the correlation across multiple voxel statistics (which implies that the true FWER is more liberal compared to Bonferroni). We discuss these issues in some detail next.

1) Random sampling methods draw many samples from near the mode(s) of the distribution of interest, but fewer samples from the tail. To characterize the threshold for a small portion of the tail of a distribution, we must invariably draw a very large number of samples just so that the estimate converges. So, if we want an $\alpha = 0.01$ threshold from the max null distribution, we require many thousands of permutation samples — each of which entails randomizing the labels and recalculating all v test statistics (see Algorithm

1). This procedure is simple but computationally demanding when v is large (as is the case in neuroimaging).

2) Ideally, we want to ensure a low strong control FWER by first setting α to be very small, and then obtaining a very precise estimate of the corresponding threshold. The smallest possible p-value we can derive in this way is $1/L$, so to calculate very low p-values (essential in many applications), L must be very large.

3) A typical characteristic of brain imaging disorders, for instance in the *early* (e.g., preclinical) stages of Alzheimer's disease (AD) and other forms of dementia, is that the disease signature is subtle — for instance, in AD, the deposition of Amyloid load estimated via positron emission tomographic (PET) images or atrophy captured in longitudinal structural magnetic resonance images (MRI) image scans in the asymptomatic stage of the disease. The signal is weak in this setting and requires large sample size studies (i.e., large n) and a need for estimating the Type 1 error threshold with high confidence. The necessity for high confidence tail estimation implies that we need to sample many more relabelings, requiring a large number of permutations L.

3. A Convex Formulation to characterize the structure of $\boldsymbol{T}$

It turns out that many of the foregoing issues can be mitigated by exploiting the structural properties of the permutation testing matrix $\mathbf{T}$. This idea is at the heart of our proposed algorithm. Our strategy exploits the correlated (redundant) structure in the matrix of voxel-wise statistics using ideas from low-rank matrix completion, subspace tracking and random matrix theory. In this section, we describe this formulation in detail.

3.1. Low-Rank Matrix Completion

Given only a small fraction of the entries in a matrix, the problem of low-rank matrix completion (LRMC) Candès and Tao (2010) seeks to *recover* the missing entries of the entire matrix. Clearly, with no assumption on the properties of the matrix, such a recovery is ill-posed. Instead, if we *assume* that the column space of the matrix is low-rank and the observed entries are randomly sampled in a certain technical sense, then the authors of Candès and Tao (2010) and others have shown that, with sufficiently small number of entries, one can recover the orthogonal basis of the row space as well as the expansion coefficients for each column — that is, fully recover the missing entries of the matrix. Specifically, the number of entries required is roughly $r \log(d)$ where r is the column space's rank and d is the ambient dimension. By placing an ℓ_1-norm penalty on the eigenvalues of the recovered matrix, i.e., the nuclear norm Fazel et al. (2004); Recht et al. (2010), one optimizes a convex relaxation of an (non-convex) objective function which explicitly minimizes the rank. Alternatively, we can specify a rank r ahead of time, and estimate an orthogonal basis of that rank by following a gradient along the Grassmannian manifold Balzano et al. (2010); He et al. (2012). The LRMC problem has received a great deal of attention in the period after the Netflix Prize Bennett and Lanning (2007), and numerous applications in machine learning and computer vision have been investigated Ji et al. (2010). Details regarding existing algorithms and their analyses including strong recovery guarantees are available in Candès and Recht (2009); Recht (2011).

Formulation. Let us consider a matrix $\mathbf{T} \in \mathbb{R}^{L\times v}$. Denote the set of randomly subsampled entries of this matrix as Ω. This means that we have access to $\mathbf{T}_\Omega$, and our recovery (or imputation) task corresponds to estimating $\mathbf{T}_{\Omega^C}$ where Ω^C corresponds to the complement of the set Ω. Let us denote the estimate of the complete matrix be $\hat{\mathbf{T}}$. The completion problem can be written as the following optimization task,

$$\min_{\hat{\mathbf{T}}} \quad \|\mathbf{T}_\Omega - \hat{\mathbf{T}}_\Omega\|^2_{\text{Frob}} \tag{1}$$

$$\text{s.t.} \quad \hat{\mathbf{T}} = \mathbf{UW} \tag{2}$$

$$\mathbf{U} \text{ is orthogonal.} \tag{3}$$

where $\mathbf{U} \in \mathbb{R}^{v\times r}$ is the low-rank basis of $\mathbf{T}$ i.e., the columns of $\mathbf{U}$ correspond to the orthogonal basis vectors of the column space of $\mathbf{T}$. Here, Ω gives the measured entries and $\mathbf{W}$ is the matrix of coefficients that lets us reconstruct $\hat{\mathbf{T}}$. With these preliminaries, we first present an overview and then the actual algorithm.

3.2. Low-rank plus a long tail in T

Most datasets encountered in the real world (and especially in neuroimaging) have a dominant low-rank component. While the data may not be *exactly* characterized by a low-rank basis, the residual will not significantly alter the eigen-spectrum of the sample covariance in general. Strong correlations nearly always imply a skewed eigen-spectrum, because as the the eigen-spectrum becomes flat, the resulting covariance matrix tends to become sparser (the "uncertainty principle" between low-rank and sparse matrices Chandrasekaran et al. (2011)). Low-rank structure in the data is encountered even more frequently in neuroimaging — unlike natural images in computer vision, there is much stronger voxel-to-voxel homogeneity in a brain image.

While performing statistical hypothesis testing on these images, the low-rank structure described above carries through to $\mathbf{T}$ for purely linear statistics such as sample means, mean differences and so on. However, non-linearities in the test statistic calculation, e.g., normalizing by pooled variances, will perturb the eigen-spectrum of the original data, contributing a long tail of eigenvalues. This is the new set of a large number of significant singular values (large magnitudes) which need to be accounted for, if one intends to model $\mathbf{T}$ using low-rank structure. Ideally, we require that this long tail should either decay rapidly, or that it does not overlap with the dominant eigenvalues. This is equivalent to asking that the resulting non-linearities do not *decorrelate* the test statistics, to the point that the matrix $\mathbf{T}$ cannot be approximated by a low-rank matrix with high fidelity. For t-statistics, the non-linearities come from normalization by pooled variances, see for example a two-sample t-test shown in (4).

$$t = \frac{\mu_1 - \mu_2}{\sqrt{\frac{\sigma_1}{n_1} + \frac{\sigma_2}{n_2}}} \tag{4}$$

where (μ_1, σ_1) and (μ_2, σ_2) are the mean and standard deviations for the two groups respectively. Since the pooled variances calculated from correlated data X are unlikely to change very much from one permutation sample to another (except outliers), we expect that the spectrum of $\mathbf{T}$ will resemble that of the data (or sample) covariance, *with the addition of a long, exponentially decaying tail.* More generally, if the non-linearity does not decorrelate the test statistics too much, it will almost certainly preserve the low-rank structure.

3.3. Overview of Proposed method

If the low-rank structure dominates the long tail described above, then its contribution to $\mathbf{T}$ can be modeled as a low variance Gaussian I.I.D. residual. A Central Limit argument appeals to the number of independent eigenfunctions that contribute to this residual, and, the orthogonality of eigenfunctions implies that as more of them meaningfully contribute to each entry in the residual, the more independent those entries become. In other words, if this long tail begins at a low magnitude and decays slowly, then we can treat it as a Gaussian I.I.D. residual; and if it decays rapidly, then the residual will perhaps be less Gaussian, but also more negligible. Thus, our algorithm makes no direct assumption about these eigenvalues themselves, but rather that the residual corresponds to a low-variance I.I.D. Gaussian random matrix – its contribution to the covariance of test statistics will be *Wishart* distributed, and from this property, we can characterize its eigenvalues.

Why should we expect runtime improvements? The low-rank + long tail structure of the permutation testing matrix then translates to the following identity,

$$\mathbf{T} = \mathbf{U}\mathbf{W} + \mathbf{S} \tag{5}$$

where $\mathbf{S}$ is the *unknown* I.I.D. Gaussian random matrix. We do not restrict ourselves to one-sided tests here, and so, $\mathbf{S}$ is modeled to be zero-mean. Later in Section 4, we show that this apparent zero-mean assumption is addressed because of a post-processing step. The low-rank portion of $\mathbf{T}$ can be reconstructed by sub-sampling the matrix at Ω using the LRMC optimization from (1). Recall from the discussion in Section 3.1 that Ω corresponds to a subset of indices of the entries in $\mathbf{T}$, i.e., instead of computing all voxel-wise statistics for a given relabeling (a column of $\mathbf{T}$), only a small fraction η, referred to as the sub-sampling rate, are computed. Later in Sections 6 and 7, we will show that η is very small (on the orders of $< 1\%$). Therefore, the overall number of entries in Ω — *the number of statistics actually calculated to recover* $\mathbf{T}$ – is ηvL as opposed to vL for $\eta \ll 1$.

Since the core of the proposed method is to model $\mathbf{T}$ by accessing only a small subset of its entries Ω, we refer to it as a *rapid permutation testing* procedure – RapidPT. Observe that a large contributor to the running time of online subspace tracking algorithms, including the LRMC optimization from (1), is the module which updates the basis set $\mathbf{U}$; but once a good estimate for $\mathbf{U}$ has been found, this additional calculation is no longer needed. Second, the eventual goal of the testing procedure is to recover the max null as discussed earlier in Section 2, which then implies that the residual $\mathbf{S}$ should also be recovered with high fidelity. Exact recovery of $\mathbf{S}$ is not possible. Although, for our purposes, we only need its effect on the *distribution of the maximum* per permutation test. An estimate of the mean and variance of $\mathbf{S}$ then provides reasonably good estimates of the max null. We therefore divide the entire process into two steps: *training*, and *recovery* which is described in detail in the next section.

4. Rapid Permutation Testing – RapidPT

We discuss the specifics of the training and testing stages of RapidPT, and then present the complete algorithm followed by some theoretical guarantees regarding consistency and recovery of $\mathbf{T}$.

4.1. The training phase

The goal of the training phase is to estimate the basis $\mathbf{U}$. Here, we perform a small number of fully sampled permutation tests, i.e., for approximately a few hundred of the columns of $\mathbf{T}$ (each of which corresponds to a permutation) denoted by ℓ, all the v voxel-wise statistics are computed. This $v \times \ell$ "sub-matrix" of $\mathbf{T}$ is referred to as the *training set*, denoted by $\mathbf{T}_{ex}$. In our experiments, ℓ was selected to be either a fraction or a multiple of the total number of subjects n as described in section 6.2. From $\mathbf{T}_{ex}$, we estimate the basis $\mathbf{U}$ using sub-sampled matrix completion methods Balzano et al. (2010); He et al. (2012), making *multiple* passes over the training set with the (given) sub-sampling rate η, until convergence. This corresponds to initializing $\mathbf{U}$ as a random orthogonal matrix of a pre-determined rank r, and using the columns of $\mathbf{T}_{ex}$ repeatedly to iteratively update it until convergence (see Balzano et al. (2010); He et al. (2012) for details regarding subspace tracking). Once $\mathbf{U}$ is obtained in this manner, $\mathbf{W}_{ex}$ is obtained by running a simple least-squares procedure on $\mathbf{T}_{ex}$ and $\mathbf{U}$. The histogram of $\mathbf{T}_{ex} - \mathbf{U}\mathbf{W}_{ex}$ will then be an estimate of the empirical distribution of the residual $\mathbf{S}$ over the training set. We denote the standard deviation of these 'left over' entries as σ. We now discuss a few relevant aspects of this training phase.

Notice that in principle, one can estimate $\mathbf{U}$ directly from $\mathbf{T}_{ex}$ by simply computing the leading r principal components. This involves a brute-force approximation of $\mathbf{U}$ by computing the singular-value decomposition of a dense $v \times \ell$ matrix. Even for reasonably small v, this is a costly operation. Second, $\hat{\mathbf{T}}$, by definition, contains a non-trivial residual. We have no direct control on the structure of $\mathbf{S}$ except that it is I.I.D Gaussian. Clearly, the variance of entries of $\mathbf{S}$ will depend on the fidelity of the approximation provided by $\mathbf{U}$. Since the sub-sampling rate η (the size of the set Ω compared to vL) is known ahead of time, estimating $\mathbf{U}$ via a subspace-tracking procedure using η fraction of the entries of $\mathbf{T}_{ex}$ (where each column of $\mathbf{T}_{ex}$ modifies an existing estimate of $\mathbf{U}$, one-by-one, without requiring to store all the entries of $\mathbf{T}_{ex}$) directly provides an estimate of $\mathbf{S}$.

Bias-Variance Trade-off. When using a very sparse subsampling method i.e., sampling with small η, there is a bias-variance trade-off in estimating $\mathbf{S}$. Clearly, if we use the entire matrix $\mathbf{T}$ to estimate $\mathbf{U}$, $\mathbf{W}$ and $\mathbf{S}$, we will obtain reliable estimates of $\mathbf{S}$. But, there is an overfitting problem: the least-squares objective used in fitting $\mathbf{W}$ (in getting a good estimate of the max null) to such a small sample of entries is likely to grossly underestimate the variance of $\mathbf{S}$ compared to when we use the entire matrix; the sub-sampling problem is not nearly as over-constrained as it is for the full matrix. This sampling artifact reduces the apparent variance of $\mathbf{S}$, and induces a bias in the distribution of the sample maximum, because extreme values are found less frequently. This sampling artifact has the effect of "shifting" the distribution of the sample maximum towards zero. We refer to this as a bias-variance trade-off because, we can think of the need for shift as an outcome of the sub-optimality of the estimate of σ versus the deviation of the true max null from the estimated max null, We correct for this bias by estimating the amount of the shift during the training phase, and then shifting the recovered sample max distribution by this estimated amount. This shift is denoted by μ.

4.2. The recovery phase

In the recovery phase, we sub-sample a small fraction of the entries of each column of $\mathbf{T}$ successively, i.e., for each new relabeling, the voxel-wise statistics are computed over a small fraction of all the voxels to populate $\mathbf{T}_\Omega$. Using this $\mathbf{T}_\Omega$, and the pre-estimated $\mathbf{U}$, the reconstruction coefficients for this column $\mathbf{w} \in \mathbb{R}^{r\times 1}$ are computed. After adding the random residuals – I.I.D Gaussian with mean μ and standard deviation σ, to this $\mathbf{Uw}$, we have our estimate $\hat{\mathbf{T}}$ for this specific relabeling/permutation. Recall that $\mathbf{S}$ was originally modeled to be zero-mean (see (5)), but the presence of the shift μ suggests a $\mathcal{N}(\mu, \sigma^2)$ distribution instead. Overall, this entails recovering a total of v voxel-wise statistics from ηv of such entries where $\eta \ll 1$. This process repeats for all the remaining $L - \ell$ columns of $\mathbf{T}$, eventually providing $\hat{\mathbf{T}}$. Once $\hat{\mathbf{T}}$ has been estimated, we proceed exactly as in the NaivePT, to compute the max null and test for the significance of the true labeling.

4.3. The Algorithm

Algorithm 2 summarizes our RapidPT algorithm. The algorithm takes in the input data $\mathbf{X}$, the rank of the basis r, the sub-sampling rate η, the number of training columns ℓ and the total number of columns L as inputs, and returns the estimated permutation testing matrix $\mathbf{T}$ and the max null distribution h^L. As described earlier in Sections 4.1 and 4.2, Algorithm 2 first estimates $\mathbf{U}$, σ and the shift μ, which are then used to compute $\mathbf{W}$ and $\mathbf{S}$ for the L number of permutations.

5. Analysis of Recovery Guarantees

We now provide some analysis of Algorithm 2. We give two results which show that as long as the variance of the residual is below a certain level, we can recover the distribution of the sample maximum. Recall from (1) that for low-rank matrix completion methods to be applicable, we must assume that the permutation matrix $\mathbf{T}$ can be decomposed into a low-rank component plus a high-rank residual matrix $\mathbf{S}$: $\mathbf{T} = \mathbf{UW} + \mathbf{S}$. Here, $\mathbf{U}$ is a $v \times r$ orthogonal matrix that spans the $r \ll \min(v, L)$–dimensional column subspace of $\mathbf{T}$, and $\mathbf{W}$ is the corresponding coefficient matrix. Subsuming the shift into the coefficients, we can then treat the residual $\mathbf{S}$ as a random matrix whose entries are I.I.D zero-mean Gaussian with variance σ^2. We arrive at our first result by analyzing how the low-rank portion of $\mathbf{T}$'s singular values spectrum interlaces with the contribution coming from $\mathbf{S}$ by treating $\mathbf{T}$ as a low-rank perturbation of a random matrix. If this low-rank perturbation is sufficient to dominate the eigenvalues of the random matrix, then $\mathbf{T}$ can be recovered with high fidelity at a low sampling rate Balzano et al. (2010); He et al. (2012). Consequently, we can estimate the distribution of the maximum as well — this is shown by our second result.

Since the eigenvalues of $\mathbf{TT}^T$ are the squared singular values of $\mathbf{T}$, rather than analyzing the singular value spectrum of $\mathbf{T}$ directly, we can analyze the eigenvalues of $\mathbf{TT}^T$ using a recent result from Benaych-Georges and Nadakuditi (2011). This is important because in order to ensure recovery of $\mathbf{T}$, we require that its singular value spectrum should approximately retain the shape of $\mathbf{UW}$'s. More precisely, we require that for some $0 < \delta < 1$,

$$|\tilde{\phi}_i - \phi_i| < \delta\phi_i \qquad i = 1, \ldots, r; \qquad \tilde{\phi}_i < \delta\phi_r \qquad i = r+1, \ldots, v \tag{6}$$

where ϕ_i and $\tilde{\phi}_i$ are the singular values of $\mathbf{UW}$ and $\mathbf{T}$ respectively. Recall that in this analysis, $\mathbf{T}$ is considered to be a perturbation of $\mathbf{UW}$. Theorem 5.1 relates the rate at which eigenvalues are perturbed, δ, to the parameterization of $\mathbf{S}$ in terms of σ^2.

Algorithm 2 The RapidPT algorithm for permutation testing.

```
Input: $\mathbf{X}^1$, $X^2$, $r$, $\eta$, $L$, $\ell$, stat
Output: $\hat{\mathbf{T}}$, $\mathrm{h}^L$
  $\mathbf{X} = [\mathbf{X}^1; \mathbf{X}^2]$, $n = n_1 + n_2$
  TRAINING
  $\mathbf{U} \leftarrow$ Rand. Orth., $\mathbf{W}_{ex} = [\varnothing]$
  for $i \in 1, \dots, \ell$ do
    $j_1 \dots, j_n \sim$ Permute$[1, n]$
    $\tilde{\mathbf{X}}^1 \leftarrow \mathbf{X}[:, j_1, \dots, j_{n_1}]$
    $\tilde{\mathbf{X}}^2 \leftarrow \mathbf{X}[:, j_{n_1+1}, \dots, j_n]$
    $\mathbf{T}_{ex}[:, i] \leftarrow \mathrm{test}(\tilde{\mathbf{X}}^1, \tilde{\mathbf{X}}^2)$
    $k_1, \dots, k_{\lceil \eta v \rceil} \sim \mathrm{UNIF}[1, v]$
    $\tilde{\mathbf{T}} \leftarrow \mathbf{T}_{ex}[k_1, \dots, k_{\lceil \eta v \rceil}, i]$
    $\mathbf{U}, \mathbf{W}_{ex}[:, i] \leftarrow$ Subspace-Tracking$(r)$
  end for
  $\sigma \leftarrow$ Standard Deviation$\{\mathbf{T}_{ex} - \mathbf{U}\mathbf{W}_{ex}\}_\Omega$
  $\mu \leftarrow \sup_i$ Max$\{\mathbf{T}_{ex}[:, i] - \mathbf{U}\mathbf{W}_{ex}[:, i]\}$
  for $i \in 1, \dots, \ell$ do
    $\hat{\mathbf{T}}[:, i] \leftarrow \mathbf{T}[:, i]$
  end for
  RECOVERY
  for $i \in \ell + 1, \dots, L$ do
    $k_1, \dots, k_{\lceil \eta v \rceil} \sim \mathrm{UNIF}[1, v]$
    $j_1 \dots, j_n \sim$ Permute$[1, n]$
    $\tilde{\mathbf{X}}^1 \leftarrow \mathbf{X}[k_1, \dots, k_{\lceil \eta v \rceil}, j_1, \dots, j_{n_1}]$
    $\tilde{\mathbf{X}}^2 \leftarrow \mathbf{X}[k_1, \dots, k_{\lceil \eta v \rceil}, j_{n_1+1}, \dots, j_n]$
    $\tilde{\mathbf{T}} \leftarrow \mathrm{test}(\tilde{\mathbf{X}}^1, \tilde{\mathbf{X}}^2)$
    $\mathbf{W}[:, i] \leftarrow$ Complete$(\mathbf{U}, \tilde{\mathbf{T}}, k_1, \dots, k_{\lceil \eta v \rceil})$
    $\mathbf{s} \leftarrow \mathrm{i.i.d}\mathcal{N}^v(0, \sigma^2)$
    $\hat{\mathbf{T}}[:, i] \leftarrow \mathbf{U}\mathbf{W}[:, i] + \mathbf{s}$
  end for
  for $i \in 1, \dots, L$ do
    if $i \leq \ell$ then
      $m_i \leftarrow$ Max$(\hat{\mathbf{T}}[:, i])$
    else
      $m_i \leftarrow$ Max$(\hat{\mathbf{T}}[:, i]) + \mu$
    end if
  end for
  $\mathrm{h}^L \leftarrow$ Histogram$(m_1, \dots, m_L)$
```

The theorem's principal assumption also relates σ^2 inversely with the number of columns of the testing matrix $\mathbf{T}$ which is just the number of permutations L. Note however that the process may be split up between several matrices $\mathbf{T}_i$, and the results can then be combined.

For purposes of applying this result in practice we may then choose a number of columns L which gives the best bound. Theorem 5.1 also assumes that the number of permutations L is greater than the number of voxels v, which is a difficult regime to explore empirically. Thus, our numerical evaluations cover the case where $L < v$, while Theorem 5.1 covers the case where L is larger. From the definition of $\mathbf{T}$, we have,

$$\mathbf{T}\mathbf{T}^T = \mathbf{U}\mathbf{W}\mathbf{W}^T\mathbf{U}^T + \mathbf{S}\mathbf{S}^T + \mathbf{U}\mathbf{W}\mathbf{S}^T + \mathbf{S}\mathbf{W}^T\mathbf{U}^T \tag{7}$$

We first analyze the change in eigenvalue structure of $\mathbf{S}\mathbf{S}^T$ when perturbed by $\mathbf{U}\mathbf{W}\mathbf{W}^T\mathbf{U}^T$ (which has r non-zero eigenvalues). The influence of the cross-terms, $\mathbf{U}\mathbf{W}\mathbf{S}^T$ and $\mathbf{S}\mathbf{W}^T\mathbf{U}^T$, is addressed later. We have the following result,

Theorem 5.1 (Perturbation of eigenvalues). *Denote the r non-zero eigenvalues of $\mathbf{Q} = \mathbf{U}\mathbf{W}\mathbf{W}^T\mathbf{U}^T \in \mathbb{R}^{v\times v}$ by $\lambda_1 \geq \lambda_2 \geq, \ldots, \lambda_r > 0$; and let $\mathbf{S}$ be a $v \times t$ random matrix such that $\mathbf{S}_{i,j} \sim \mathcal{N}(0, \sigma^2)$, with unknown σ^2. As $v, L \to \infty$ such that $\frac{v}{L} \ll 1$, the eigenvalues $\tilde{\lambda_i}$ of the perturbed matrix $\mathbf{Q} + \mathbf{S}\mathbf{S}^T$ will satisfy*

$$|\tilde{\lambda_i} - \lambda_i| < \delta\lambda_i \qquad i = 1, \ldots, r; \qquad \tilde{\lambda_i} < \delta\lambda_r \qquad i = r+1, \ldots, v \tag{$\star$}$$

for some $0 < \delta < 1$, whenever $\sigma^2 < \frac{\delta\lambda_r}{L}$

Proof. (Sketch) The proof proceeds by constructing the asymptotic eigenvalues $\tilde{\lambda_i}$ (for $i = 1, \ldots, v$), and later bounding them to satisfy ($\star$). The construction of $\tilde{\lambda_i}$ is based on Theorem 2.1 from Benaych-Georges and Nadakuditi (2011). Firstly, an asymptotic spectral measure μ of $\frac{1}{L}\mathbf{S}\mathbf{S}^T$ is calculated, followed by its Cauchy transform $G_\mu(z)$. Using $G_\mu(z)$ and its functional inverse $G_\mu^{-1}(\theta)$, we get $\tilde{\lambda_i}$ in terms of λ_i, σ^2, v and L. Finally, the constraints in ($\star$) are applied to $\tilde{\lambda_i}$ to upper bound σ^2. The supplementary material includes the full proof. □

Note that the missing cross-terms will not change the result of Theorem 5.1 drastically, because $\mathbf{U}\mathbf{W}$ has r non-zero singular values and hence $\mathbf{U}\mathbf{W}\mathbf{S}^T$ is a low-rank projection of a low-variance random matrix, and this will clearly be dominated by either of the other terms. Having justified the model $\mathbf{T} = \mathbf{U}\mathbf{W} + \mathbf{S}$, the following theorem shows that the empirical distribution of the max null statistic approximates the true distribution.

Theorem 5.2. *Let $m_l = \max_i \mathbf{T}_{i,l}$ be the maximum observed test statistic at permutation trial l, and similarly let $\hat{m}_l = \max_i \hat{\mathbf{T}}_{i,l}$ be the maximum reconstructed test statistic. Further, let the maximum reconstruction error be ϵ, such that $|\mathbf{T}_{i,t} - \hat{\mathbf{T}}_{i,t}| \leq \epsilon$. Then we have,*

$$Pr\left[m_t - \hat{m}_t - (b - \hat{b}) > k\epsilon\right] < \frac{1}{k^2}$$

where b is the bias term described in Section 4, and $\hat{b}$ is its estimate from the training phase.

The result uses Chebyshev's bound. The full proof is given in the supplementary material.

6. Experimental Setup

To evaluate how RapidPT performs in practice, we performed an extensive set of experiments and compared its accuracy, runtime speedups and overall performance against two baselines. The first baseline we used was the latest release of the widely used MATLAB toolbox for nonparametric permutation testing in neuroimaging, Statistical NonParametric Mapping (SnPM) (SnPM (2013); Nichols and Holmes (2002)). The second baseline was a standard MATLAB implementation of algorithm 1, which we will call NaivePT. Both baselines serve to evaluate RapidPT's accuracy. Further, the very small differences between the results provided by SnPM and NaivePT offer a secondary reference point that tells us an acceptable range for RapidPT's results (in terms of differences). For runtime performance, SnPM acts as a state of the art baseline. On the other hand, NaivePT is used to evaluate how an unoptimized permutation testing implementation will perform on the datasets we use in our experiments. In this section, we describe the experimental data, the hyperparameters space evaluated, the methods used to quantify accuracy, and the environment where all experiments were run.

6.1. Data

The data used to evaluate RapidPT comes from the Alzheimer's disease Neuroimaging Initiative-II (ADNI2) dataset. The ADNI project was launched in 2003 by the National Institute on Aging, the National Institute of Biomedical Imaging and Bioengineering, the Food and Drug Administration, private pharmaceutical companies, and nonprofit organizations, as a $60 million, 5-year public-private partnership. The primary goal of ADNI is to test whether serial MRI, positron emission tomography (PET), other biological markers, and clinical and neuropsychological assessment can be combined to measure the progression of MCI and early AD.

For the experiments presented in this paper, we used gray matter tissue probability maps derived from T1-weighted magnetic resonance imaging (MRI) data. From this data, we constructed four varying sized datasets. We sampled n_1 and n_2 subjects from the CN and AD groups in the cohort, respectively. Table 1 shows a summary of the datasets used for our evaluations.

Dataset Size: n (n_1,n_2)			
50 (25,25)	100 (50,50)	200 (100,100)	400 (200,200)

Table 1: Dataset sizes used in our experiments. The table lists the total number of subjects (n) and how many of the participants were sampled from the CN (n_1) and AD groups (n_2).

6.1.1. Data Pre-processing

All images were pre-processed using voxel-based morphometry (VBM) toolbox in Statistical Parametric Mapping software (SPM, http://www.fil.ion.ucl.ac.uk/spm). After pre-processing, we obtain a data matrix $\mathbf{X}$ composed of n rows and v columns for each dataset shown in Table 1. Each row in $\mathbf{X}$ corresponds to a subject and each column is associated to a voxel that denotes approximately the same anatomical location across subjects (since the images are already co-registered). This pre-processing is commonly used in the literature and not specialized to our experiments.

6.2. Hyperparameters

As outlined in Algorithm 2, there are three high-level input parameters that will impact the performance of the procedure: the *number of training samples* (l), the *sub-sampling rate* (η), and the *number of permutations* (L). To demonstrate the robustness of the algorithm to these parameter settings, we explored and report on hundreds of combinations of these hyperparameters on each dataset. This also helps us identify the general scenarios under which RapidPT will be a much superior alternative to regular permutation testing. The baselines for a given combination of these hyperparameters are given by the max null distribution constructed by SnPM and NaivePT. The number of permutations used for the max null distributions of the baselines is the same as the number of permutations used by RapidPT for a given combination of hyperparameters.

- *Number of Training Samples:* The number of training samples, l, determines how many columns of $\mathbf{T}$ are calculated to estimate the basis of the subspace, and also how many training passes are performed to estimate the shift that corrects for the bias-variance tradeoff discussed in Section 4.1. We decided to use the total number of subjects n as a guide to pick a sensible l, the rationale is that the maximum possible rank of $\mathbf{T}$ is n (as discussed in Section 3). Further, l is also used to determine the number of passes performed to calculate the shift of the max null distribution. Calculating the shift is a cheap step, therefore it makes sense to use all the information available in $\mathbf{T}_{ex}$ to calculate the shift.

Number of Training Samples: l			
$\frac{n}{2}$	$\frac{3n}{4}$	n	$2n$

Table 2: Number of training samples used to evaluate RapidPT. The number n corresponds to the total number of subjects in the dataset. For instance, in For the 400 subject dataset the values for l used were 100, 200, 400, and 800

- *Sub-sampling rate:* The sub-sampling rate, η, is the percentage of all the enries of $\mathbf{T}$ that we will calculate (i.e., sample) when recovering the max null distribution. In the recovery phase, η determines how many voxel-wise test statistics will be calculated at each permutation to recover a column of $\bar{\mathbf{T}}$. For instance, if the data matrix has v columns (number of voxels) then instead of calculating v test statistics, we will sample only ηv (where $\eta \ll 1$) random columns and calculate test statistics only for those columns.

Sub-sampling rate: η					
0.1%	0.35%	0.5%	0.7%	1.5%	5%

Table 3: Sub-sampling rates used to evaluate RapidPT. η is the percentage of the total number of entries in $\mathbf{T}$ that will be calculated during the recovery phase.

- *Number of Permutations* The number of permutations, L, determines the total number of columns in $\mathbf{T}$. By varying L we are able to see how the size of $\mathbf{T}$ affects the accuracy of the algorithm and also how it scales compared to a standard permutation testing implementation with the same number of permutations (e.g., NaivePT or SnPM).

Number of Permutations: L						
$2,000$	$5,000$	$10,000$	$20,000$	$40,000$	$80,000$	$160,000$

Table 4: Number of permutations done to evaluate RapidPT.

6.3. Accuracy Benchmarks

In order to assess the accuracy and overall usefulness of the recovered max null distribution by RapidPT we used three different measures: Kullback-Leibler Divergence (KL-Divergence), t-thresholds/p-values and the resampling risk.

Kullback-Leibler Divergence: The KL-Divergence provides a measure of the difference between two probability distributions. One of the distributions represents the ground truth (SnPM or NaivePT) and the other an "approximation" (obtained via RapidPT). In this case, the distributions are the max null distributions (h^L). We use the KL-Divergence to identify under which circumstances (i.e., hyperparameters) RapidPT provides a good estimate of the overall max null distribution and if there are cases where the results are unsatisfactory.

T-Thresholds/p-values: Once we have evaluated whether all methods recover a similar max null distribution, we analyze if t-thresholds associated to a given p-value calculated from each max null distribution are also similar.

Resampling Risk: Two methods can recover a similar max null distribution and p-values, and yet partially disagree in *which voxels* should be classified as statistically significant (e.g., within a group difference analysis). The resampling risk is the probability that the decision of accepting/rejecting the null hypothesis differs between two methods (Jockel (1984)). Let v_1 and v_2 be the number of voxels whose null hypothesis was rejected according to the max null derived from Method 1 and 2, respectively. Further, let v_c be the number of voxels that are the (set) intersection of v_1 and v_2. The resampling risk can then be calculated as shown in (8).

$$\text{risk} = \frac{\frac{v_1 - v_c}{v_1} + \frac{v_2 - v_c}{v_2}}{2} \tag{8}$$

6.4. Implementation Environment and other details

All evaluation runs reported in this paper were performed on multiple machines with the same hardware configuration. The setup consisted of multiple 16 core machine with two Intel(R) Xeon(R) CPU E5-2670, each with 8 cores. This means that any MATLAB application will be able to run a maximum of 16 threads. To evaluate the runtime performance of RapidPT, we performed all experiments on two different setups which forced MATLAB to use a specified number of threads. First, we forced MATLAB to only use a single thread (single core) when running SnPM, RapidPT, and NaivePT. The performance results on a single threaded environment attempt to emulate a scenario where the application was running on an older laptop/workstation serially. In the second setup, we allow MATLAB to use all 16 threads available to demonstrate how RapidPT is also able to leverage a parallel computing environment to reduce its overall runtime.

Although all machines had the same hardware setup, to further ensure that we were making a fair runtime performance comparison, all measurements of a given figure shown in Section 7 were obtained from the same machine.

7. Results

The hyperparameter space explored through hundreds of runs allowed identifying specific scenarios where RapidPT is most effective. To demonstrate accuracy, we first show the impact of l and η on the recovery of the max null distribution by analyzing KL-Divergence. Then we focus on the comparison of the corrected p-values across methods and the resampling risk associated with those p-values. To demonstrate the runtime performance gains of RapidPT, we first calculate the speedup results across hyperparameters. We then focus on the hyperparameters that a user may use and look at how RapidPT, SnPM, and NaivePT scale with respect to the dataset and the number of permutations. Overall, the large hyperparameter space that was explored in these experiments produced hundreds of figures. In this section, we summarize the results of all figures within each subsection, but only present the figures that are interesting and we believe will convey the most important information about RapidPT. An extended results section is presented in the supplementary materials that accompany this paper.

7.1. Accuracy

Can we recover the max null distribution?
The left column of Figure 1 uses a colormap to summarize the KL-Divergence results obtained from comparing the max null distributions of a single run of SnPM versus multiple RapidPT runs with various hyperparameters. The right column of Figure 1 puts the numbers displayed in the colormaps into context by showing the actual max null distributions for a single combination of hyperparameters. Each row corresponds to each of the four datasets used in our evaluations.

The sub-sampling rate was the hyperparameter that had the most significant impact on the KL-Divergence. As shown in Figure 1, a sub-sampling rate of 0.1% led to high KL-Divergence, i.e., the max null distribution was not recovered in this case. For every other combination of hyperparameters RapidPT was able to sample at rates as low as 0.35% and still recover an accurate max null distribution. Most KL-Divergence values were in the $0.01 - 0.05$ range with some occassional values between $0.05 - 0.15$. However, using the max null distributions derived from only 2000 permutations leads to the resulting KL-Divergence to range mainly between $0.05 - 0.15$, as shown in the supplementary results.

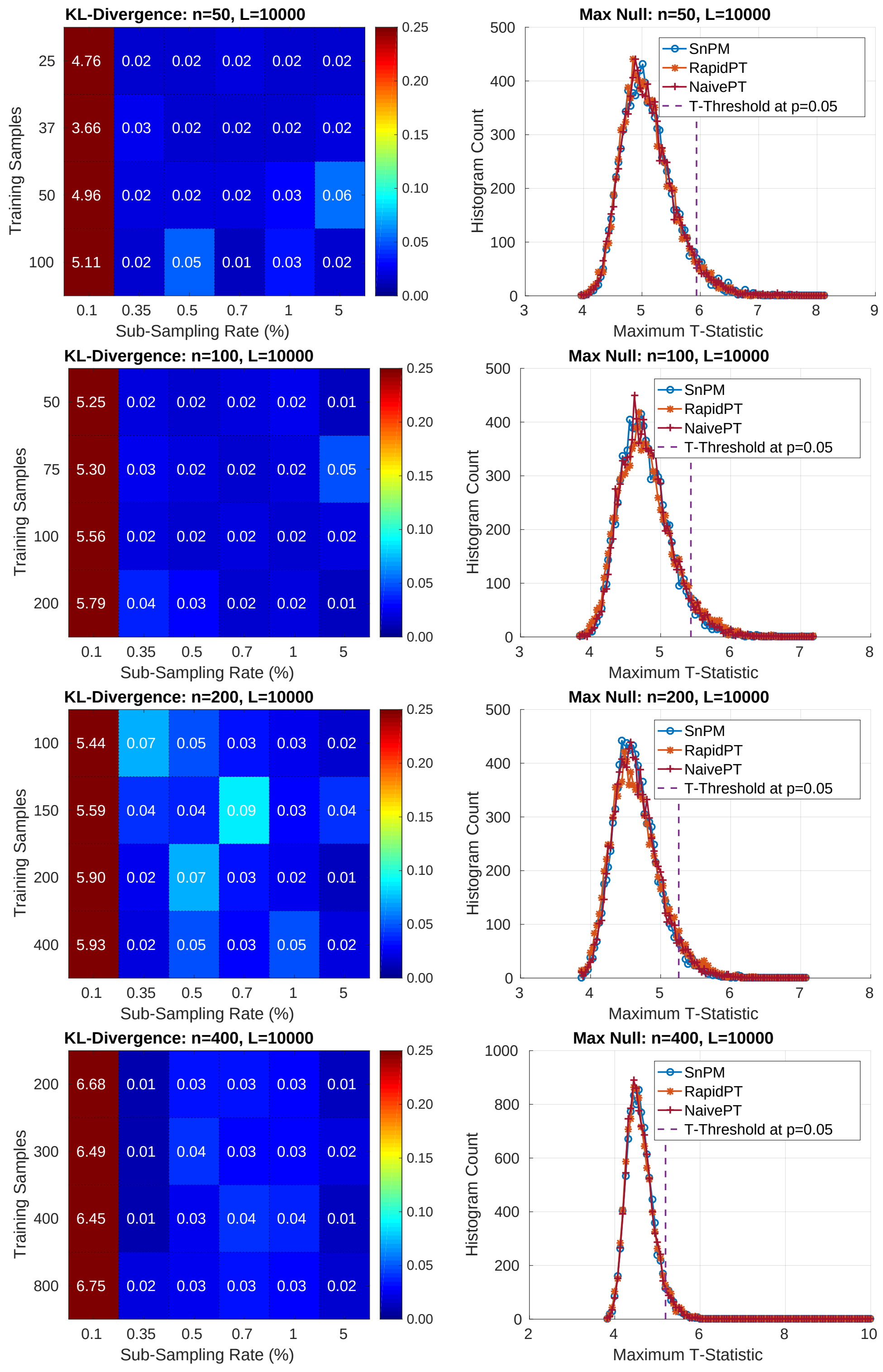


Figure 1: *Left:* Colormap of the KL-Divergence between the max null distributions of RapidPT and SnPM. Each colormap is associated to a run on one of the datasets and a fixed number of permutations. The resulting KL-Divergence from 24 hyperparameter combinations is displayed on each colormap. Rows 1, 2, 3, and 4 of this figure are associated to the 50, 100, 200, and 400 subject datasets respectively. *Right:* The max null distributions given by RapidPT, SnPM, and NaivePT for the hyperparameters: $L = 10000$, $l = n$, and $\eta = 0.35\%$.

Are we rejecting the correct null hypotheses?
The test statistics obtained using the original data labels whose value exceed the t-threshold associated to a given p-value will correspond to the null hypothesis rejected. Figure 2 shows the resultant mapping between t-threshold and p-values for the max null distribution for a given set of hyperparameters. It is evident that the difference across methods is very minimal. Moreover, Figure 2 shows that low p-values ($p < 0.1$), which are the main object of interest, show the lowest differences. However, despite the low percent differences between the p-values, in the larger datasets (100, 200, and 400 subjects) RapidPT consistently yields slightly more conservative p-values near the tails of the distribution. Nonetheless, Figure 3 shows that the resampling risk between RapidPT and the two baselines remains very close to the resampling risk between both baselines. In practice, these plots show that RapidPT will reject the null hypothesis for a slightly lower number of voxels than SnPM or NaivePT. However, the actual brain regions whose null hypotheses were rejected consistently match between both methods.

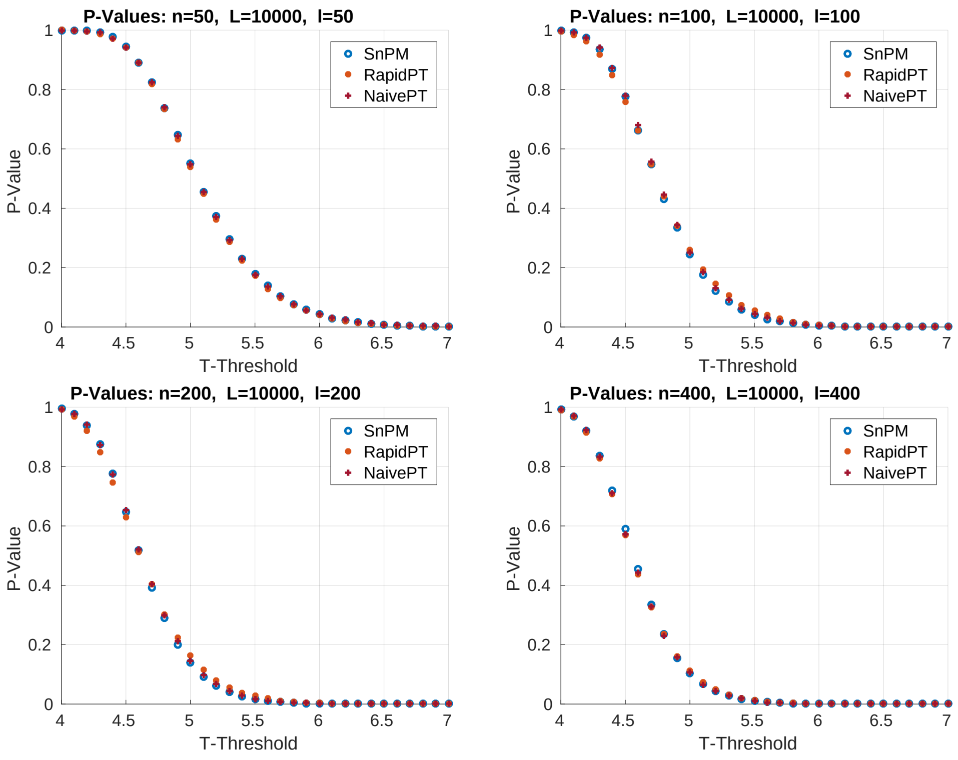


Figure 2: p-values for SnPM, RapidPT, and NaivePT. The hyperparameters used were: $\eta = 0.35\%$, $L = 10000$, and $l = n$. The results in this plot were obtained from the max null distributions shown in the right hand side of figure 1.

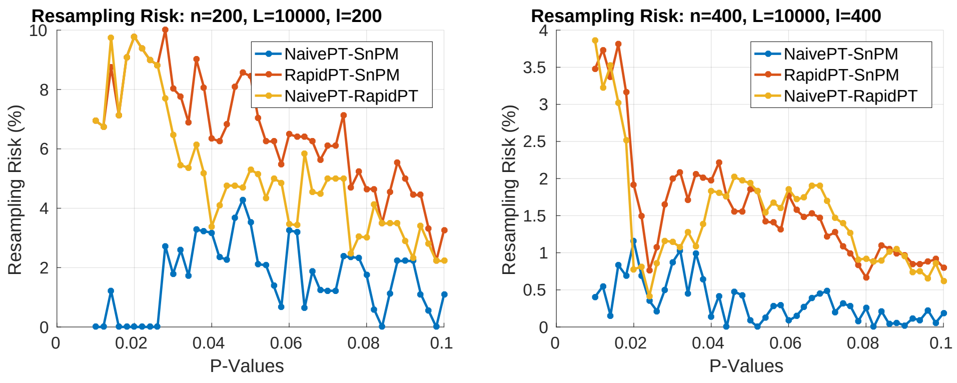


Figure 3: Resampling risk of NaivePT-SnPM, RapidPT-SnPM, and NaivePT-RapidPT. The hyperparameters used were: $\eta = 0.35\%$, $L = 10000$, and $l = n$.

7.2. Runtime Performance

7.2.1. Effect of hyperparameters on the speed of RapidPT

Figures 4 and 5 show the speedup gains of RapidPT over SnPM and NaivePT, respectively. Each column corresponds to a single dataset, and each row corresponds to a different number of permutations. The supplementary results include an exhaustive version of these results that show the speedup gains of RapidPT for many additional number of permutations.

As shown in Figure 4, RapidPT outperforms SnPM in most scenarios. With the exception of the $L = 2000$ and $L = 5000$ runs on the larger datasets ($n = 200$ and $n = 400$), the colormaps show that RapidPT is 1.5-30x faster than SnPM. As expected, a low η (0.35%,0.5%) and l ($\frac{n}{2}$,$\frac{3n}{4}$) leads to the best runtime performance without a noticeable accuracy tradeoff, as can be seen also in Fig. 1 earlier.

Figure 5 shows how RapidPT performs against a non-optimized permutation testing implementation. In this setup, RapidPT outperforms NaivePT in every single combination of the hyperparameters. The same speedup trends that were seen when comparing RapidPT and SnPM are seen between RapidPT and NaivePT but with a much larger magnitude. For the remainder of this section, the runtime results of NaivePT are no longer compared because the difference is too large.

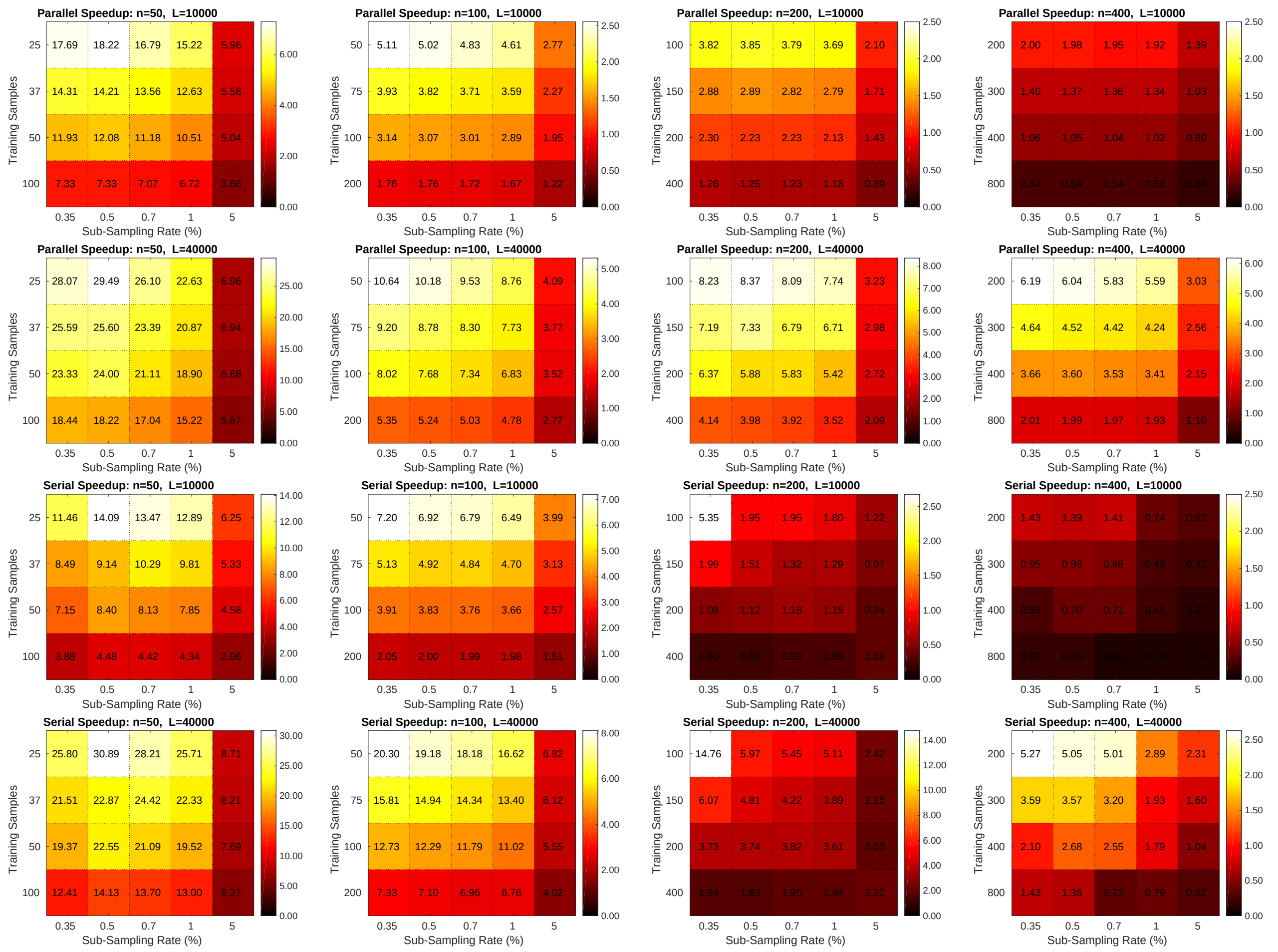


Figure 4: Colormaps of the speedup gains of RapidPT over SnPM in a serial and parallel computing environment. Each colormap corresponds to a run with a given dataset and a fixed number of permutations, and displays 20 different speedups resulting from different hyperparameter combinations. The first two rows correspond to the speedups obtained from the runs on 16 cores, and the last two columns from runs on a single core. Columns 1, 2, 3, and 4 of this figure are correspond to the 50, 100, 200, and 400 subject datasets, respectively.

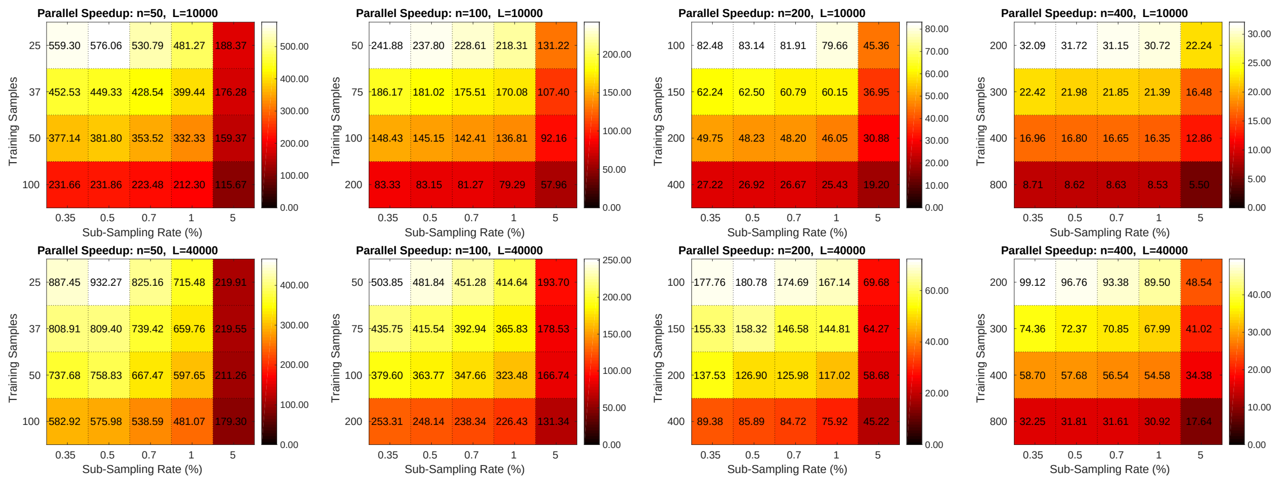


Figure 5: Colormap of the speedup gains of RapidPT over NaivePT. The organization of the colormaps and the information they display is the same as figure 4. These speedups correspond to both programs running on a MATLAB instance that could use all 16 available cores.

7.2.2. Scaling of RapidPT vs. SnPM

As opposed to η and l which only seem to have an impact on the runtime performance of RapidPT, the number of permutations and the size of the dataset have an impact on the runtime of both RapidPT and SnPM. In this section, we compare how RapidPT and SnPM scale as we vary these two parameters. Figures 6 and 7 show the runtime performance of RapidPT for $\eta = 0.35\%$ and $l = n$.

Number of Permutations: Figure 6 shows the super linear scaling of SnPM compared to RapidPT for all datasets as the number of permutation increases. Doubling the number of permutations in SnPM leads to an increase in the runtime by a factor of about two. On the other hand, doubling the number of permutations for RapidPT only affects the runtime of the recovery phase leading to small increases in the timing performance. The performance of both implementations is only comparable if we focus on the lower range of the number of permutations (5000) across datasets. As this number increases, as was shown in Figure 4, RapidPT outperforms the permutation testing implementation within SnPM.

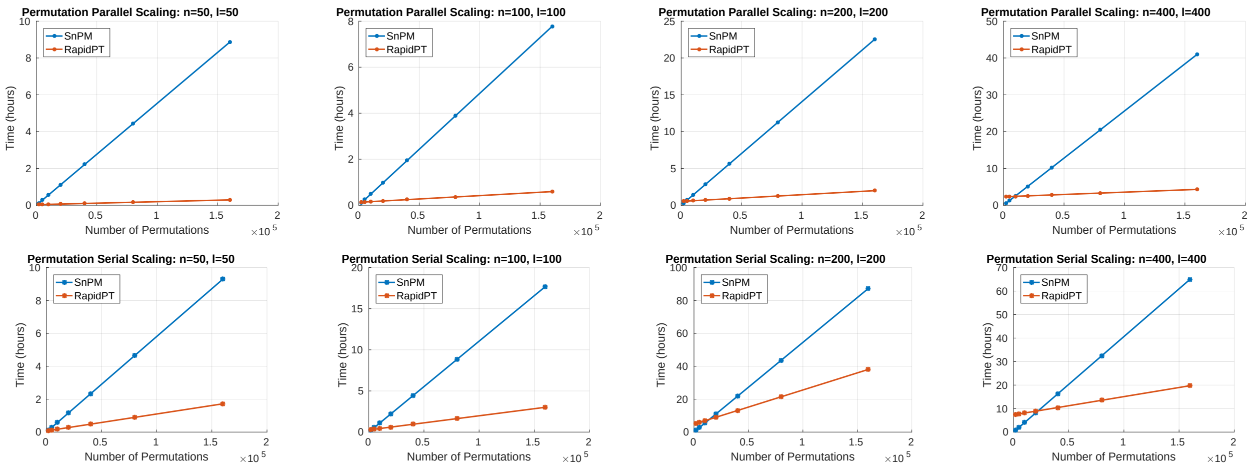


Figure 6: Effect of the number of permutations on the runtime performance of RapidPT and SnPM on 16 cores (first row) and on a single core (second row). The hyperparameters used were: $\eta = 0.35\%$ and $l = n$.

Dataset Size: Figure 7 shows the effect of the dataset size on the runtime of RapidPT and SnPM. In SnPM, as expected, increasing the dataset by a factor of two leads to an increase on the runtime by a factor of two. In RapidPT, however, increasing the dataset size has a variable effect on the runtime. The training phase ends up contributing more to the runtime as the dataset size increases, while the recovery phase runtime increases at much slower rate.

Overall, scaling the number of permutations have a stronger impact on the runtime performance of SnPM than RapidPT. On the other hand, scaling the dataset size has a more negative effect on the timing of RapidPT than in SnPM. Furthermore, if both parameters are increased at the same time the runtime of SnPM increases at a much faster rate than the runtime of RapidPT.

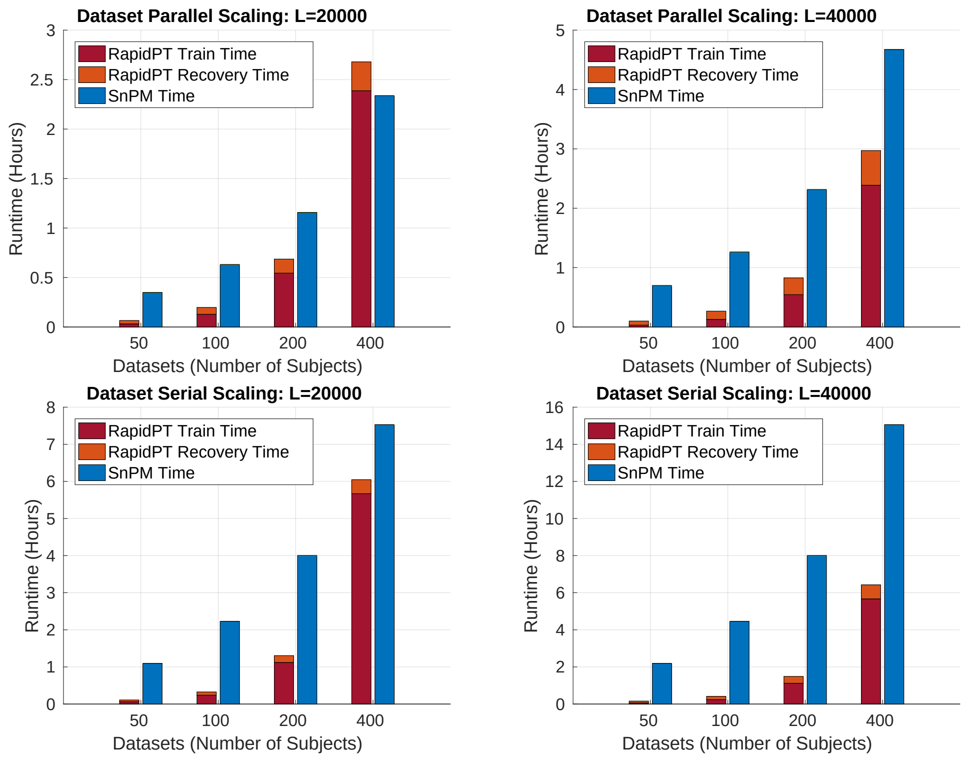


Figure 7: Effect of the dataset size on the performance of RapidPT and SnPM on 16 cores (first row) and on a single core (second row). The overall measured time of RapidPT is the result of the total time spent on the training phase and the recovery phase. The hyperparameters used were: $\eta = 0.35\%$, $L = 10000$, and $l = n$.

8. Discussion

8.1. Accuracy

8.1.1. Recovered Max null Distribution

Monte carlo permutation tests perform a randomization step where a random subset of the total number of permutations is chosen. This means that the constructed max null distribution from one run might slightly differ from another run, and as the number of permutations increases, this difference will decrease. In terms of KL-Divergence, this means that the KL-Divergence between two permutation testing runs on the same data will be small, but *not exactly zero*. The results show that given a good set of hyperparameters the KL-Divergence between a run of RapidPT versus a regular permutation testing run leads to a very low KL-Divergence which is the expected result, even if we run the *same permutation testing program twice*. The evaluated scenarios show that a sensible set of hyperparameters can be easily defined as long as the sub-sampling rate is sufficient for the recovery of the permutation testing matrix. The minimum number of sub-sampled entries needed to accurately recover $\mathbf{T}$ depends on the rank and dimensions of $\mathbf{T}$ as discussed in sections 3.1 and 5. In our experiments, $\eta \geq 0.35\%$ led to a large enough set of sub-sampled entries to obtain an accurate estimate of the max null distribution. For a brief discussion on how to pick a sensible η, please refer to section 3 of the supplementary material. Overall, once we have a sensible η, the resulting max null distribution constructed by RapidPT is consistent to the one recovered by regular permutation testing.

The number of permutations also has a significant impact on the KL-Divergence. As L increases, the KL-Divergence decreases. However, this is also true between any two permutation testing runs on the same dataset.

8.1.2. Evaluating p-values

As seen in the p-value spectrum plots, the p-values drawn from each max null distribution agree very well (given a good set of hyperparameters as discussed above). This follows from the low KL-Divergence, since the KL-Divergence is a measure of the overall difference between the distributions. The lowest differences, however, is located in the tails of the distributions. This means that the derived thresholds are expected to accept/reject almost the same null hypotheses.

8.1.3. Resampling Risk

Small signal datasets: Although the p-values agree across methods, a small difference may lead to slightly more or less null hypotheses to be rejected by a given method. This has a direct impact on the resampling risk between RapidPT and regular permutation testing. In datasets where there is a small signal difference between groups, we may see an elevated resampling risk. The reason is because a very small number of null hypotheses will be rejected. Therefore, at a given p-value, p, one of the methods will reject a small number of null hypotheses which the second method will not reject until the p-value is $p + \delta$, where $\delta << 1$. This slight difference in the number of rejected null hypotheses may have a significant impact in the resampling risk. For instance, in Figure 3, for $N = 200$ at $p = 0.05$, RapidPT rejected 59 (out of $\sim$ 568k statistics) null hypotheses and SnPM rejected 71: this led to a resampling risk of 8.45%. On the other hand, for $N = 400$ at $p = 0.05$ RapidPT rejected 2158 (out of $\sim$ 570k statistics) and SnPM rejected 2241 null hypotheses, resulting in a lower resampling risk of 1.85% even when there is a larger difference in the number of rejected hypotheses. This difference of RapidPT's and SnPM's rejections were among the boundary voxels of the rejection region (i.e., the mismatch is not at the center of the rejection/significant region). Similar elevated resampling risks resulted for the 50 and 100 subject datasets. Therefore, the resampling risk is a useful measure if it is presented together with the number of null hypothesis being rejected. Hence, RapidPT yields a slightly more conservative test, primarily because it is based on sub-sampling. Nevertheless, this sub-sampling is robust to locating the same voxel clusters that displayed group differences as the other methdods.

8.2. Runtime Performance

Our results show that under certain scenarios RapidPT provides substantial speedups over SnPM (state of the art) and NaivePT (simple implementation). From the runtime performance of NaivePT, it is evident that in practice it is more beneficial for the user to rely on a permutation testing toolbox such as SnPM. In the remainder of the discussion we will just consider the runtime performance of RapidPT and SnPM.

Dataset Size: Overall, the largest speedups in both the serial and parallel setups were obtained in the runs for the smallest dataset ($N = 50$). The number of training samples used to estimate the basis $\mathbf{U}$ is smaller and consequently the training phase time decreases.

As the dataset size increases, the training time introduces a considerable overhead which negatively impacts the speedup of RapidPT over SnPM.

Number of Permutations: The number of permutations have a linear impact on the runtime of both RapidPT and SnPM. But as shown in Figure 6, RapidPT runtime increases at a lower rate than SnPM's. This is expected since the number of permutations only impacts the runtime of the recovery phase. Therefore, the training phase time for a given setup is constant as the number of permutations changes. The results show that for datasets with less than or equal to 400 subjects between $5000-10000$ permutations is the threshold where despite the training overhead in RapidPT, the expected speedup is considerable to justify its use. In the larger datasets, when performing less than 5000 permutations, the training overhead becomes too large, as shown in the supplementary material.

8.2.1. Serial vs. Parallel Performance

The serial and parallel runs show very similar speedup trends across hyperparameters as shown in the results and supplementary material. However, in terms of actual runtime, both RapidPT and SnPM benefit from being able to run on a parallel environment. This is an essential feature for any software toolbox that will be running on modern workstations because multiple cores are available in nearly all computers shipped today.

8.3. Hyperparameter Recommendations

Sub-sampling Rate: The KL-Divergence colormaps show that as long as the sub-sampling rate is greater than or equal to 0.35%, RapidPT is able to accurately recover the max null distribution. Hence, the accuracy will not significantly improve as the sub-sampling rate increases. On the other hand, a low sub-sampling rate can significantly reduce the runtime and is therefore preferable.

Number of Training Samples: The number of training samples will ideally be the exact rank of $\mathbf{T}$. This is usually not known, however, we know that the rank is bounded by the number of subjects in the data matrix used to generate $\mathbf{T}$. Therefore, in our evaluations, we are able to accurately recover the max null distribution even when using as low as $\frac{n}{2}$ training samples. The recommended number of training samples in practice should be n.

Number of Permutations: When a large number of permutations is desired, RapidPT should be strongly considered due to its runtime gains. Note, however, that the user should also take into account the dataset size and look at the speedup colormaps to see the expected speedups.

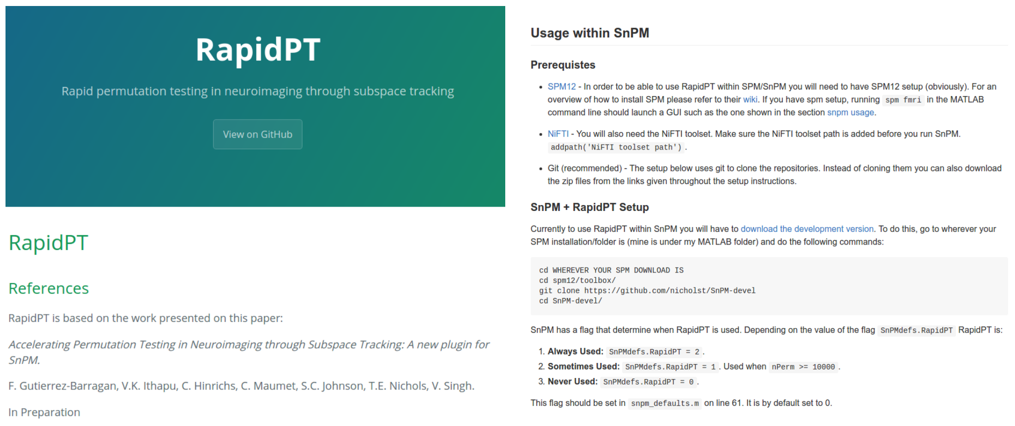
Dataset size: Although not a hyperparameter, as discussed above, the dataset size should play a role when the user selects which method to use. Large datasets ($n \geq 200$), RapidPT will be a good option if the user is planning to perform a large number of permutations. For medium-sized datasets ($50 \leq n \leq 200$), RapidPT will most likely lead to good speedup gains. For small datasets ($n \leq 50$), although RapidPT might lead to speedup gains over regular permutation testing, the total runtime will be on the order of minutes anyway.

As we briefly discussed in Section 1, Winkler et al. (2016) provides several strategies for reducing the runtime of permutation testing. But the authors do not report significant speedup gains against regular permutation testing on a 50 subject dataset with ≈ 200000 voxels. On the other hand, RapidPT is able to consistently outperform a state of the art permutation testing implementation (SnPM) on a 50 subject dataset with $\approx$ 540k voxels.

This boost in performance is, in large part, due to our low sub-sampling rates, much lower than what was proposed in Winkler et al. (2016). Our subspace tracking algorithm is, nonetheless, able to perform recovery in this sparse sampling setting.

8.4. SnPM Integration

SnPM is a toolbox that can be used within the software Statistical Parametric Mapping (SPM) SPM (2012). RapidPT has been integrated into the development version of SnPM SnPM (2013). This enables users to leverage the pre and post processing capabilities of SnPM. To use RapidPT within SnPM the user has to choose between 3 modes. The user sets a flag within SnPM that says to either *Always*, *Sometimes* or *Never* use RapidPT. This is described in the online documentation of RapidPT as shown in figure 8. Once this mode is specified the user can simply launch SPM and proceed with their normal SnPM workflow. Further discussion and walkthroughs of how to use SnPM and RapidPT within SnPM can be found in the documentation of both toolboxes Gutierrez-Barragan and Ithapu (2016); SnPM (2013).


RapidPT
Rapid permutation testing in neuroimaging through subspace tracking
View on GitHub

RapidPT

References

RapidPT is based on the work presented on this paper:

Accelerating Permutation Testing in Neuroimaging through Subspace Tracking: A new plugin for SnPM.

F. Gutierrez-Barragan, V.K. Ithapu, C. Hinrichs, C. Maumet, S.C. Johnson, T.E. Nichols, V. Singh.

In Preparation

Usage within SnPM

Prerequistes

- SPM12 - In order to be able to use RapidPT within SPM/SnPM you will need to have SPM12 setup (obviously). For an overview of how to install SPM please refer to their wiki. If you have spm setup, running `spm fmri` in the MATLAB command line should launch a GUI such as the one shown in the section snpm usage.
- NiFTI - You will also need the NiFTI toolset. Make sure the NiFTI toolset path is added before you run SnPM. `addpath('NiFTI toolset path')`.
- Git (recommended) - The setup below uses git to clone the repositories. Instead of cloning them you can also download the zip files from the links given throughout the setup instructions.

SnPM + RapidPT Setup

Currently to use RapidPT within SnPM you will have to download the development version. To do this, go to wherever your SPM installation/folder is (mine is under my MATLAB folder) and do the following commands:

```
cd WHEREVER YOUR SPM DOWNLOAD IS
cd spm12/toolbox/
git clone https://github.com/nicholst/SnPM-devel
cd SnPM-devel/
```

SnPM has a flag that determine when RapidPT is used. Depending on the value of the flag `SnPMdefs.RapidPT` RapidPT is:

1. **Always Used:** `SnPMdefs.RapidPT = 2`.
2. **Sometimes Used:** `SnPMdefs.RapidPT = 1`. Used when `nPerm >= 10000`.
3. **Never Used:** `SnPMdefs.RapidPT = 0`.

This flag should be set in `snpm_defaults.m` on line 61. It is by default set to 0.

Figure 8: Screenshots of the RapidPT website (left) and SnPM integration documentation (right).

9. Conclusion

In this paper, we have presented a new algorithmic framework that is able to efficiently approximate the max null distribution commonly obtained through permutation testing. By exploiting the structure of the permutation testing matrix, $\mathbf{T}$, and applying recent ideas from online matrix completion we show through our theoretical analysis and experimental evaluations that one can subsample entries of $\mathbf{T}$ at extremely low-rates and still construct a good estimate of $\mathbf{T}$. The algorithm first goes through a training phase where the basis and the distribution of the residual of $\mathbf{T}$ are estimated. Then it continues into the recovery phase where a small number of entries of each column of $\mathbf{T}$ are calculated and the rest are estimated through matrix completion. Finally, we obtain the max null distribution from the maximum value of each column in $\mathbf{T}$. Experiments on four varying sized datasets derived from the ADNI2 dataset showed that if we sub-sample at a high enough rate ($\eta \geq 0.35\%$) we can accurately recover the max null distribution at a fraction of the time in many scenarios. The implementation is available as a stand-alone open-source toolbox as well as a plugin for SnPM13 SnPM (2013), and is able to leverage multi-core architectures.

Acknowledgements. We thank Robert Nowak, Grace Wahba, Moo K. Chung and the anonymous reviewers for their helpful comments, and Jia Xu for helping with a preliminary implementation of the model. This work was supported in part by NIH R01 AG040396; NSF CAREER grant 1252725; NSF RI 1116584; Wisconsin Partnership Fund; UW ADRC P50 AG033514; UW ICTR 1UL1RR025011 and a Veterans Administration Merit Review Grant I01CX000165. Hinrichs was supported by a CIBM post-doctoral fellowship via NLM grant 2T15LM007359. The contents do not represent views of the Dept. of Veterans Affairs or the United States Government.

References

A. Eklund, P. Dufort, D. F. S. L., 2013. Medical image processing on the gpu - past, present and future. Medical Image Analysis 17, 1073–1094.

A. Eklund, M. Andersson, H. K., 2012. fmri analysis on the gpu-possibilities and challenges. Computer Methods and Programs in Biomedicine 105, 145–161.

A.P. Holmes, J. W., Nichols, T., 1998. Holmes and watson, on sherlock. Journal on Cerebral Blood Flow and Metabolism 18.

Arndt, S., Cizadlo, T., Andreasen, N., Heckel, D., Gold, S., O'Leary, D., 1996. Tests for comparing images based on randomization and permutation methods. Journal of Cerebral Blood Flow and Metaholism 16, 1271–1279.

Balzano, L., Nowak, R., Recht, B., Sept 2010. Online identification and tracking of subspaces from highly incomplete information. In: Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on. pp. 704–711.

Benaych-Georges, F., Nadakuditi, R. R., 2011. The eigenvalues and eigenvectors of finite, low rank perturbations of large random matrices. Advances in Mathematics 227 (1), 494–521.

Bennett, J., Lanning, S., 2007. The netflix prize. In: Proceedings of KDD cup and workshop. Vol. 2007. p. 35.

Candès, E., Tao, T., 2010. The power of convex relaxation: Near-optimal matrix completion. IEEE Trans. Inf. Theor. 56 (5), 2053–2080.
URL `http://dx.doi.org/10.1109/TIT.2010.2044061`

Candès, E. J., Recht, B., 2009. Exact matrix completion via convex optimization. Foundations of Computational mathematics 9 (6), 717–772.

Chandrasekaran, V., Sanghavi, S., Parrilo, P. A., Willsky, A. S., 2011. Rank-sparsity incoherence for matrix decomposition. SIAM Journal on Optimization 21 (2), 572–596.

Cheverud, J. M., 2001. A simple correction for multiple comparisons in interval mapping genome scans. Heredity 87 (1), 52–58.

Dahl, D. B., Newton, M. A., 2007. Multiple hypothesis testing by clustering treatment effects. Journal of the American Statistical Association 102 (478), 517–526.

Edgington, E., 1969a. Approximate randomization tests. The Journal of Psychology 72 (2), 143–149.

Edgington, E., 1969b. Statistical inference: The distribution-free approach.

Edgington, E., Onghena, P., 2007. Randomization tests. CRC Press.

Efron, B., Tibshirani, R., 2007. On testing the significance of sets of genes. The annals of applied statistics, 107–129.

Eklund, A., Andersson, M., Knutsson, H., 2011. Fast random permutation tests enable objective evaluation of methods for single subject fmri analysis. International Journal of Biomedical Imaging.

Fazel, M., Hindi, H., Boyd, S., 2004. Rank minimization and applications in system theory. In: American Control Conference, 2004. Proceedings of the 2004. Vol. 4. IEEE, pp. 3273–3278.

FSL, 2012. Fmrib software library (fsl). Available online at http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/.

Gaonkar, B., Davatzikos, C., 2013. Analytic estimation of statistical significance maps for support vector machine based multi-variate image analysis and classification. NeuroImage 78, 270 – 283.
URL `http://www.sciencedirect.com/science/article/pii/S1053811913003169`

Gutierrez-Barragan, F., Ithapu, V., 2016. RapidPT. Available online at https://github.com/felipegb94/RapidPT.

He, J., Balzano, L., Szlam, A., June 2012. Incremental gradient on the grassmannian for online foreground and background separation in subsampled video. In: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on. pp. 1568–1575.

Hinrichs, C., Ithapu, V., Sun, Q., Johnson, S., Singh, V., 2013. Speeding up permutation testing in neuroimaging. Neural Information Processing Systems (NIPS).

Holmes, A., Blair, R., Watson, J., Ford, I., 1996. Nonparametric analysis of statistic images from functional mapping experiments. Journal on Cerebral Blood Flow and Metabolism 16 (1), 7–22.

Ji, H., Liu, C., Shen, Z., Xu, Y., 2010. Robust video denoising using low rank matrix completion. In: CVPR. Citeseer, pp. 1791–1798.

Jockel, K.-H., 1984. Computational Aspects of Monte Carlo Tests. Physica-Verlag HD, Heidelberg.
URL `http://dx.doi.org/10.1007/978-3-642-51883-6_25`

K.J. Friston, A.P. Holmes, J.-B. P. P. G. S. W. R. F., 1995. Statistical parametric maps in functional imaging: A general linear approach. Human Brain Mapping 2, 189–210.

Knijnenburg, T. A., Wessels, L. F., Reinders, M. J., Shmulevich, I., 2009. Fewer permutations, more accurate p-values. Bioinformatics 25 (12), i161–i168.

Li, J., Ji, L., 2005. Adjusting multiple testing in multilocus analyses using the eigenvalues of a correlation matrix. Heredity 95 (3), 221–227.

M. Halber, K. Herholz, K. W. G. P., Heiss, W., 1997. Performance of a randomization test for single-subject 15o-water pet activation studies. Journal of Cerebral Blood Flow and Metabolism 17 (10), 1033–1039.

Nichols, T., Hayasaka, S., 2003. Controlling the familywise error rate in functional neuroimaging: a comparative review. Statistical Methods in Medical Research 12, 419–446.

Nichols, T. E., Holmes, A. P., 2002. Nonparametric permutation tests for functional neuroimaging: A primer with examples. Human Brain Map 15 (1), 1–25.

Recht, B., 2011. A simpler approach to matrix completion. The Journal of Machine Learning Research 12, 3413–3430.

Recht, B., Fazel, M., Parrilo, P. A., 2010. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. SIAM review 52 (3), 471–501.

SnPM, 2013. Statistical non parametric mapping toolbox (snpm). Available online at http://www2.warwick.ac.uk/fac/sci/statistics/staff/academic-research/nichols/software/snpm.

SPM, 2012. Statistical parametric mapping toolbox (spm). Available online at http://www.fil.ion.ucl.ac.uk/spm/.

Subramanian, A., Tamayo, P., Mootha, V. K., Mukherjee, S., Ebert, B. L., Gillette, M. A., Paulovich, A., Pomeroy, S. L., Golub, T. R., Lander, E. S., et al., 2005. Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. Proceedings of the National Academy of Sciences 102 (43), 15545–15550.

Taylor, J., Worsley, K., 2008. Random fields of multivariate test statistics, with applications to shape analysis. The Annals of Statistics 36 (1), 1–27.

Winkler, A., Ridgway, G., Douau, G., Nichols, T., Smith, S., 2016. Faster permutation inference in brain imaging. NeuroImage 141, 502–516.

Winkler, A. M., Ridgway, G. R., Webster, M. A., Smith, S. M., Nichols, T. E., 2014. Permutation inference for the general linear model. Neuroimage 92, 381–397.

Worsley, K., Evans, A., Marrett, S., Neelin, P., 1992. A three-dimensional statistical analysis for cbf activation studies in human brain. Journal of Cerebral Blood Flow Metabolism 12, 900–918.

Worsley, K., Marrett, S., Neelin, P., Vandal, A., Friston, K., Evans, A., 1996. A unified statistical approach for determining significant signals in images of cerebral activation. Human Brain Mapping 4, 58–73.

Y. Hochberg, A. T., 1987. Multiple Comparison Procedures, 1st Edition. Wiley.