frontiers in
NEUROINFORMATICS

METHODS ARTICLE
published: 20 October 2009
doi: 10.3389/neuro.11.034.2009

Parallel workflows for data-driven structural equation modeling in functional neuroimaging

***Sarah Kenny*[1]*, Michael Andric*[2], *Steven M. Boker*[3], *Michael C. Neale*[4], *Michael Wilde*[1,6] *and Steven L. Small*[1,2,5]**

[1] *Computation Institute, The University of Chicago, Chicago, IL, USA*
[2] *Department of Psychology, The University of Chicago, Chicago, IL, USA*
[3] *Department of Psychology, University of Virginia, Charlottesville, VA, USA*
[4] *Department of Psychiatry, Virginia Commonwealth University, Richmond, VA USA*
[5] *Department of Neurology, The University of Chicago, Chicago, IL, USA*
[6] *Mathematics and Computer Science Division, Argonne National Laboratories, Argonne, IL, USA*

Edited by:
John Van Horn, University of California at Los Angeles, USA

Reviewed by:
Shantanu Joshi, University of California at Los Angeles, USA
John Van Horn, University of California at Los Angeles, USA

****Correspondence:***
Sarah Kenny, Computation Institute, University of Chicago, 5640 S Ellis Avenue, Chicago, IL 60637, USA.
e-mail: skenny@uchicago.edu

We present a computational framework suitable for a data-driven approach to structural equation modeling (SEM) and describe several workflows for modeling functional magnetic resonance imaging (fMRI) data within this framework. The Computational Neuroscience Applications Research Infrastructure (CNARI) employs a high-level scripting language called Swift, which is capable of spawning hundreds of thousands of simultaneous R processes (R Development Core Team, 2008), consisting of self-contained SEMs, on a high performance computing system (HPC). These self-contained R processing jobs are data objects generated by OpenMx, a plug-in for R, which can generate a single model object containing the matrices and algebraic information necessary to estimate parameters of the model. With such an infrastructure in place a structural modeler may begin to investigate exhaustive searches of the model space. Specific applications of the infrastructure, statistics related to model fit, and limitations are discussed in relation to exhaustive SEM. In particular, we discuss how workflow management techniques can help to solve large computational problems in neuroimaging.

Keywords: exhaustive search, OpenMx, SEM, swift, workflows

INTRODUCTION

CONCEPTS AND BACKGROUND ON CNARI AND DRIVING NEUROSCIENCE USAGE MODEL

In the past decade, there has been tremendous growth in the number and scope of functional brain imaging studies performed in the basic and applied neurosciences. These studies have been more complex than those of the past, often incorporating large numbers of participants, multiple physical sites, longitudinal follow-up, combinations of healthy groups and those with disease or injury, and/or additional types of behavioral or biological measurements. Although their numbers are increasing, the inherent complexity of data management and processing in such studies, particularly regarding anatomical and physiological data, represents a major stumbling block to their ultimate success. In studies of recovery from stroke, for example, medical data are stored in paper charts or in hospital medical information systems, behavioral and linguistic data are saved in spreadsheets on personal workstations, structural and metabolic magnetic resonance imaging (MRI) data are stored in manufacturer formats on scanners and/or with the functional MRI data in the file systems of data processing workstations. With these diverse representations of information, not even counting the possible addition of electrophysiological and other structurally unique data types, it is hard enough to perform single case studies that attempt to relate these data to each other, let alone studies that include statistically meaningful numbers of participants.

We have started building the Computational Neuroscience Applications Research Infrastructure (CNARI) to address these concerns (Stef-Praun et al., 2007; Hasson et al., 2008; Small et al., 2009). There are two basic components to CNARI: First is the use of relational database technology to represent the diverse data types of the study in a uniform representational framework that facilitates distributed data access, and permits powerful queries and data reductions to be performed significantly faster by parallelized statistical analysis procedures. Second is the use of "virtual data" grid computing, in which data and data processing are widely distributed on storage devices and computers, and where data transformation and analysis is specified in terms of abstract ("virtual") procedure descriptions. Together, these techniques enable a community of researchers to access *and share* data and perform data preparation and analysis without detailed knowledge of the internal workings of distributed computing and storage systems or of the network infrastructure that connects them.

Longitudinal functional brain imaging requires comparison of brain activation images within a single individual over time, and possibly also between single individuals and a group that represents some standard. For example, in a study of recovery from brain injury, the individual data might be compared to a normative (healthy) group. Although such comparisons can be performed using various scalar indices, we have recently begun to do this with entire activation networks. One way of modeling such networks of activation is with structural equation modeling (SEM), a method that uses known anatomy to augment the functional information with structural connectivity information, to create a model of both static and dynamic relationships (McIntosh and Gonzalez-Lima, 1994; Buchel and Friston, 1997; Horwitz et al., 1999). We have developed several such models (Solodkin et al., 2004; Skipper et al., 2007, 2009; Walsh et al., 2008), based on a combination of primate and human data (Ban et al., 1984, 1991; Petrides and Pandya, 1984, 1988, 1999; Rosa et al., 1993; Seltzer and Pandya, 1994; Rizzolatti et al., 1997, 1998; Hackett et al., 1999; Barbas, 2000). In one of these studies, we constructed a group network model for healthy right handed individuals performing bimanual movements, and compared this normative group model to two individuals with different biological states, two healthy left handed people and one individual with stroke. The fit between a strong left hander (i.e., one who used his left hand for everything) and the model was very tight if the hemispheres were flipped in the model. The fits between either the weak left hander (i.e., someone more ambidextrous) or the person with stroke and the group model were poor. These three examples were highly informative for understanding the neurobiology of bimanual movements (Walsh et al., 2008).

Building such models can be very complex and time consuming, requiring advanced anatomical knowledge and skill. Furthermore, while these previous methods have been useful for generating a set of possible models in the absence of exhaustive techniques, they are inherently flawed since they are based on anatomical connectivity data from non-human primates. In addition, the models created depend on the hypotheses being tested, and thus there is a large number of possible models for any particular set of fMRI activation data. To address these issues, we have embarked on an extension to CNARI that aims to facilitate a more objective type of data-driven SEM via highly parallelized workflows for generating and processing large numbers of models in a manner that is easily reconfigurable and replicable. The goal for this modeling approach is to explore as much as possible of the entire space of plausible models that account for the data. In this paper, we discuss the nature of this grid-enabled SEM, and describe how it can be used and applied to various research problems in brain imaging. One of the original purposes of CNARI was to facilitate the study of stroke recovery, with particular emphasis on natural recovery and treatment for language problems (aphasia). In our presentation, we will use specific examples from language processing, though the workflows presented are generalizable to a wide variety of other SEM problems.

CONCEPTS AND BACKGROUND ON SEM: THE MOTIVATION AND DESIGN OF OPENMX

Structural equation modeling (SEM) has a long history dating back to the development of path analysis by Wright (1921). SEM is a statistical tool for estimating a set of predicted covariances between variables that may be connected with either regression (asymmetric, directional) parameters or covariance (symmetric, non-directional) parameters (see Boker and McArdle, 2005, for a review). The advent of high speed computers and high level programming languages in the 1960s, together with advances in statistical methodology led to the development of software for fitting models to observed covariance matrices by maximum likelihood (Joreskog, 1967). This procedure is now commonly known as SEM (see e.g., Bollen, 1989; Loehlin, 1992, for introductions; see McIntosh and Gonzalez-Lima, 1994 for its use in neuroimaging). SEM is widely used for fitting statistical models to epidemiological, psychological, sociological and econometric data where there are multivariate outcomes and theoretical reasons to expect that linear or non-linear systems of equations may provide explanatory power in summarizing these large data sets. For instance, in an epidemiological study of heart disease, one may wish to control for a wide variety of possible behavioral covariates while simultaneously accounting for variance due to group membership or genetic variation. For such problems, SEM models represent state-of-the-art in statistical techniques. Neuroimaging data, is a prime candidate for modeling with SEM, given overlapping sources of variance both across space and time within individual as well as sources of variance across individuals due to group membership and other covariates.

SEM models can be described as a function of two model matrices, *A, S,* a filter matrix, *F* and a residual matrix *U*, such that the expected covariance between observed variables is:

$$R = F(I - \mathrm{A})^{-1}\,\mathrm{S}((\mathrm{I} - \mathrm{A})^{-1})'\mathrm{F}' + \mathrm{U}$$

where the model matrix *A* contains the asymmetric paths (regression coefficients), *S* contains the symmetric paths (covariance coefficients), and the filter matrix, *F*, strips the latent variables from the model matrices so that the result only contains expectations for the observed covariances (McArdle and McDonald, 1984; McArdle and Boker, 1990). One implementation of SEM is the software package Mx (Neale et al., 2003). The set of built-in functions that Mx can optimize includes maximum likelihood, generalized least squares, and full information maximum likelihood analysis of covariance matrices and/or observed means. In 2007, the OpenMx development project was started in order to rewrite Mx into open source, provide a scripting interface to the R statistical language (Ihaka and Gentleman, 1996) and provide a number of extensions to the software. Among these improvements was integrating the Mx SEM optimization engine into parallel workflow management software in order to be able to estimate parameters for large numbers of SEM models simultaneously. In this way, statistical resampling techniques such as bootstrapping, simulations to verify the performance of new models, and exhaustive search routines could make use of large-scale parallel computing resources. The current article describes the first application of the OpenMx software to a real-world exhaustive search problem.

WORKFLOW MANAGEMENT

BACKGROUND AND GOALS

The ability to submit a large number of processes simultaneously to multiple grid sites is a major computational challenge and cannot be accomplished without an evolved workflow management system. In a related research project, we have been developing a workflow system called Swift (Zhao, 2007), which has been our system of choice for submission and management of large-scale workflows for neuroimaging. Using Swift, individual researchers are able to map large amounts of input and output explicitly and make calls to the cataloged executables that sit on remote grid sites. We have been investigating ways to execute and manipulate exhaustive or partially pruned, data-driven SEM workflows using Swift to operate on covariance data derived from a relational fMRI experiment database. From the standpoint of parallel computing and workflow management this poses some interesting issues and also demonstrates, quite strikingly, the convenience (to the research scientist) of having an elegant, high-level means of expressing, reconfiguring and rerunning such workflows. Here we present several examples of such workflows and explain how they can be expressed and run using Swift, OpenMx and the computational resources of the TeraGrid (Catlett et al., 2007).

The availability of high performance computing systems (HPCs), ranging from multi-core workstations to clusters, grids, clouds, and now petascale supercomputers, creates opportunities to explore experimental datasets with SEM in ways never before possible. The availability of this computing power, however, can be difficult to harness, particularly for a neuroscientist not versed in high performance computing. For these researchers, it is undesirable to divert mental and manual effort from scientific exploration to the mechanics of large-scale parallel computing. At the same time, both the complexity and the scale of high performance environments makes it ever more challenging to assure the validity of scientific results obtained via such systems.

What scientists in general - and neuroscientists in particular - need, are ways to express the processing they want to perform in a compact, abstract, high-level notation that specifies only the logical nature of their computations, but which abstracts and automates all of the potential, varying details of implementing those computing abstractions across a wide range of computing platforms.

SWIFT AND CNARI

For the past two years our group at the University of Chicago Human Neuroscience Laboratory, in collaboration with the Computation Institute, has been developing and evaluating Swift, a parallel scripting language, for this purpose. Together with members of the OpenMx project described above, we have recently focused significant effort to create a library of Swift procedures for the flexible processing and analysis of data from fMRI and other neuroscience experiments.

We employ a programming model that "loosely couples" application programs. In this model, complete programs become our functions, and the arguments to, and results from these functions can be files, file-structured datasets, as well as database entries.

The goals of expressing data processing steps in an abstract notation are multifold: 1) to distill the computation down to the salient details and eliminate the mechanical details of file manipulation from the expression of the basic workflow steps; 2) to abstract data at a high level to relieve the programmer of concerns for the layout of the data on storage systems; 3) to enable the automatic parallelization of scripts in which independent streams of data are processed; and, 4) to enable the recording of all of the steps of a computation in an automatic, transparent manner. An overview of the scripting modules for SEM analysis, coded by the research scientist within the CNARI framework can be seen in **Figure 1**. The Swift programming language enables this model by providing the ability to represent application programs as procedures, and to define compound procedures that permit the user to create libraries of higher level processes that capture the essential protocols of an application's data preparation and analysis. The language's data model provides the ability to describe the datasets that are consumed and produced by the procedural abstractions by combining basic primitive data type definitions with a mapping mechanism of on-disk directory structures to form structures and arrays. These data objects are then automatically and transparently sent across distributed execution environments to remote and parallel Swift procedures.

The Swift language has a C-like syntax, but enforces many of the semantic aspects of a "functional" programming language. Procedures are expressed as functions, permitted to return multiple values; statements are executed in data-dependency order; variables (including array elements and structure members) are

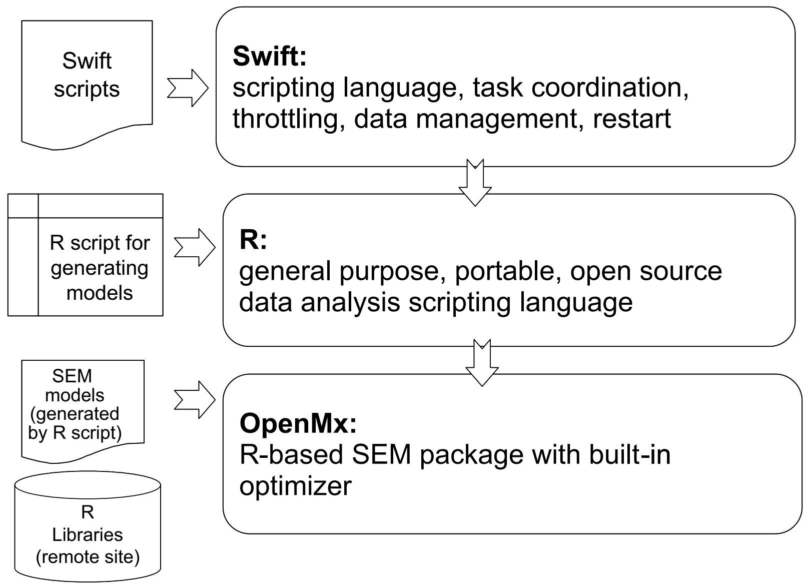
Swift scripts
Swift:
scripting language, task coordination, throttling, data management, restart
R script for generating models
R:
general purpose, portable, open source data analysis scripting language
SEM models (generated by R script)
R Libraries (remote site)
OpenMx:
R-based SEM package with built-in optimizer

FIGURE 1 | User Interface: Overview of the CNARI scripting modules for SEM workflows.

single-assignment, making it significantly simpler to determine independent operations and threads of control, and to execute these threads in parallel; a construct called "mapping" is provided to translate between the simple, clean regular abstract data model of Swift and the potentially messy, complex model of real-world directory structures and the file naming and structuring conventions expected by real-world applications.

The notation provides a simple set of flow-of-control statements, such as *if* and *switch* (case) statements. The primary way to express a set - potentially large - of parallel operations in swift is to utilize the *foreach()* statement. This statement iterates over a collection, assigning each member of the collection to a control variable, and then evaluating the body of the *foreach()* loop once for each value of the target collection. All iterations of a *foreach()* are potentially (and conceptually) performed in parallel; the runtime system provides appropriate "throttling" and scheduling of the potentially enormous number of parallel operations that this construct can generate and submit for processing.

Atomic procedures in Swift consist of wrappers around specifications that detail the invocation of application programs. In our SEM project, this mechanism is used by Swift to invoke the individual parallel model optimizations of the many thousands of models generated in an OpenMx SEM analysis workflow. "R" is the application program of execution. The invoking (master) program that calls the individual R programs creates a (potentially very long) list of evaluations, each of which is an R expression that embodies the OpenMx engine. The master program generates a large set of model calls and marshals the model's matrix into a text character stream.

The Swift model of data abstraction was to some degree inspired and motivated by the field of fMRI data analysis. In our earliest efforts to execute fMRI preprocessing workflows on computing grids we observed that the data model of the fMRI domain had a natural tree structure in which the vast number of files stored in traditional file system directories had somewhat similar patterns. These files included data from myriad experiments, test conditions and scans, and also included various types of lower level data such as anatomical and time series data represented in the image/header file pairs of the functional data format (e.g., Analyze or AFNI formats). This suggested to us that data definition constructs could be of significant benefit for scientific workflow scripting, such that data could be described in a "typed" fashion, much like the hierarchical model of "structs" in C or "classes" in Java. To enable an organization (or even a discipline, through community curation efforts such as those managed by collaborations like BIRN)[1] to define and standardize a uniform format for describing their common data elements, Swift provides the notion of data type and "mapping" of each type to a physical representation. The logical type is simple and abstract, and reflects only the logical level of the data; the "mapping" describes how each element of a structure is mapped onto the structure's physical representation on a file system. To some extent, Swift emulates the mapped filesystem structure on the remote resources where it instantiates processing. Generic mappers with a modest degree of representational flexibility are pre-defined in the swift system; but additional mappers can be created by users for their own communities and used throughout. **Figure 2** shows the Swift modules used for execution management once a user has mapped his files, and defined processing jobs within a Swift script.

Swift is easy for users to install, and its runtime system provides the client capabilities needed to use workstation, grid and cluster computing resources. From a single client computer, e.g., a modest workstation or personal laptop, the user can launch and control scripts that send parallel work for simultaneous execution on clusters, grids and supercomputers. The user can test the correct execution of the logical script workflow, just by executing directly

[1]http://www.loni.ucla.edu/BIRN/

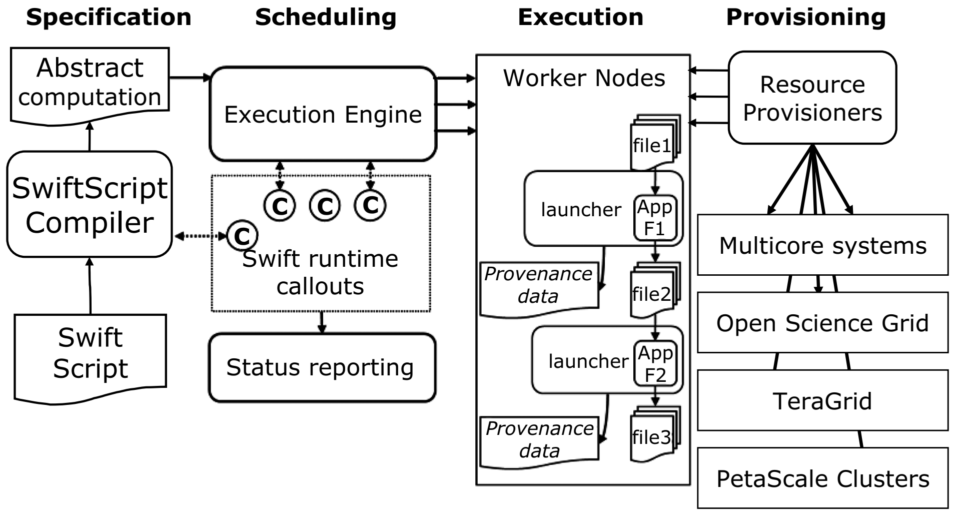

FIGURE 2 | Swift architecture: Managing workflow execution within CNARI. Specification and scheduling are implemented on the client side while execution is implemented on the remote computing resources.

on a local workstation. If the user's workstation has multiple cores, Swift can take advantage of those for modest but invaluable parallelism. And as the user's needs grow or the user is ready to scale up to increasingly large systems, Swift can readily expand to those systems with a single representation and a single client as we will show in our example workflows.

Swift scripts afford a highly productive way to produce and manage the software of neuroscience research units, whether they be local campus departments or international collaborations. In today's practice, organizations that need to process data from fMRI experiments typically develop and rely on locally produced sets of *ad hoc* scripts, usually written in a Linux "shell" language such as "c shell" (csh) or bash, or perhaps Perl, to organize the processing protocols and processes of the collaborations. In Swift, however, as all procedures are "typed" with a specific "signature" of data types for the input and output arguments, a more rigorous and less error-prone paradigm is imposed on the overall structure of the scripts. Thus Swift procedures serve as an interface-definition language for ordinary shell procedures. The overall higher-level process is then defined in a multilevel fashion, from top (highest) to bottom (lowest level) being:

- overall application (such as `multiNetworkSEM`)
- high-level scripts (such as `getCovariance()`)
- low level Swift interfaces (atomic procedures) such as `mxModelProcessor()`
- an external R wrapper script to do further argument manipulation (RInvoke.sh)
- the R tool itself (R CMD BATCH)

Special power and structure is afforded when the tool being run is not a "canned" compiled application, but rather itself a powerful data manipulation environment such as Perl or Python, or more specific to the model we describe in detail here, the R data analysis language with its vast package library of statistical and analytical procedures, including the OpenMx package used here. In this case, the actual script to be performed can be dynamically generated or selected from a template library, and sent to any computing site, which already has a suitable version of R and the OpenMx package installed.

DESCRIPTION OF THE fMRI EXPERIMENT DATA: THE EMBLEM DATABASE

We now give some concrete examples of how Swift can manipulate large datasets and enable novel analysis techniques by means of effective workflow management. The example framework we have employed our grid-enabled analysis techniques on is an fMRI investigation of the neural processing associated with emblematic gesture observation. Emblematic gestures ("emblems") are goal-directed, symbolic manual actions that, while expressed as culturally recognizable manual gestures, communicate a linguistically associable propositional meaning. Four experimental conditions were presented to participants in the MRI scanner: 1) *Emblem*, the symbolic manual gestures; 2) *Speech*, the spoken form of the linguistic propositions associated with the emblems; 3) *Emblem with Speech*, simultaneous presentations of the emblems with their verbalized linguistic associations; and 4) *Grasping*, observation of another type of goal-directed manual action, for which the neural regions associated with its processing have been well-characterized. Data were processed with AFNI (Cox, 1996) and mean normalized values of each of the hemodynamic response functions for every condition at every voxel in the brain were projected to 2-D cortical surface representations and spatially smoothed on the surfaces using SUMA (Saad et al., 2004). These surface values were then imported into MySQL database tables for relational indexing and further analyses.

SEM WORKFLOWS IN SWIFT

We have begun exploring extremely large, exhaustive SEM workflows as a means of investigating how efficient workflow tools can address computational problems that were previously considered unmanageable. Particularly, in using SEM for looking at functional connectivity many researchers are confined to hypothesis-driven approaches because they lack the tools to reliably implement data-driven methods; this situation can greatly impact mining and interpretation of datasets. In an attempt to address these issues, we are building an infrastructure that can be used by researchers to iterate over various parameters within these large sets in a reasonable amount of time and in a manner that is both dynamic and reliable. The following workflows were run on a TeraGrid HPC system known as Ranger. Ranger comprises 3,936 16-way SMP compute nodes providing 15,744 AMD Opteron™ processors for a total of 62,976 compute cores. The workflows were developed on and submitted (to Ranger) from a single-core Linux workstation running an Intel® Xeon™ 3.20 GHz CPU.

A model generator was developed for the OpenMx package and is designed explicitly to enable parallel execution of exhaustive or partially pruned sets of model objects. Given an *n* x *n* covariance matrix, it can generate the entire set of possible models with anywhere from 0 to n^2 connections; however, it can also take as input a single index from that set and it will generate and run a single model. What this means in the context of workflow design is that the generator can be controlled (and parallelized) easily by a Swift script. For example, using Swift as the interface to OpenMx we have these few lines of code:

WORKFLOW 1: 4-REGION EXHAUSTIVE SEM FOR A SINGLE EXPERIMENTAL CONDITION

```
app (mxModel min) mxModelProcessor(file
covMatrix, Rscript mxModProc, int modnum,
float initweight, string cond){
{
     RInvoke @filename(mxModProc) @
filename(covMatrix) modnum initweight cond;
}
file covMatrix<single_file_
mapper;file="speech.cov">;
Rscript mxScript<single_file_mapper;file="sin-
glemodels.R">;
int totalperms[] = [1:65536];
float initweight =.5;
foreach perm in totalperms{
   mxModel modmin<single_file_mapper; file=@
strcat(perm,".rdata")>;
11.    modmin = mxModelProcessor(covMatrix,
       mxScript, perm, initweight, "speech");
12. }
```

First, a covariance matrix containing activation data for 4 brain regions, over 8 time points, averaged over a group of subjects in the *Speech* condition was drawn from the experiment database and its location (in this example, on the local file system, though the file could be located anywhere) is mapped in line 5. Line 6 maps the R processing script and lines 1 through 4 define the atomic procedure for invoking R. Each iteration of the foreach loop maps its optimized model output file and calls `mxModelProcessor()` with the necessary parameters to generate and run a model. Each of these invocations of `mxModelProcessor()` is independent and is submitted for processing in parallel. Swift passes 5 variables for each invocation: (1) the covariance matrix; (2) the R script containing the call to OpenMx; (3) the permutation number, i.e., the index of the model; (4) the initialization weight for the free parameters of the given model; and (5) the experimental condition. Clearly, in this workflow all free parameters of the given model will have the same initialization weight as Swift is passing only one weight variable. When the job reaches a worker node on Ranger an R process is initialized, the generator creates the desired model by calculating where in the array that permutation of the model matrix falls. OpenMx then estimates the model parameters using a non-linear optimization algorithm called NPSOL (Gill et al., 1986), and the optimized model is returned and written out by Swift to the location specified in its mapping on line 10.

The above script completed in approximately 40 minutes. The script can then be altered to run over multiple experimental conditions by adding another outer loop:

WORKFLOW 2: 4-REGION EXHAUSTIVE SEM FOR 2 EXPERIMENTAL CONDITIONS

```
string conditions[] = ["emblem", "speech"];
int totalperms[] = [1:65536];
float initweight =.5;
foreach cond in conditions{
   foreach perm in totalperms{
    file covMatrix<single_file_mapper;file=@
    strcat(cond,".cov")>;
    mxModel modmin<single_file_mapper;file=@
    strcat(cond,perm,".rdata")>;
    modmin= mxModelProcessor(covMatrix,
    mxScript,perm, initweight, cond);
}
```

When the outer loop is added, the new workflow consists of 131,072 jobs since we are now running the entire set for two conditions. This workflow completed in approximately 2 hours (**Figure 3**).

WORKFLOW 3: 4-REGION EXHAUSTIVE SEM FOR MULTIPLE NETWORKS

In this workflow multiple 4-region networks are run for the *Emblem with Speech* experimental condition. The regions of interest (ROIs) designated are from FreeSurfer's[2] automatic parcellation of anatomical regions, based on the Duvernoy atlas (1991), and further manual subdivisions to delineate anterior and posterior extents of the superior temporal gyrus and sulcus, as well as superior and inferior segments of the precentral gyrus. Because *Emblem with Speech* involved subjects' perceiving simultaneously both spoken (audiovisual) and manual information, here we chose candidate regions expected to be involved in audiovisual recognition of speech and manual action: occipital pole (OP), middle occipital gyrus (MOG), anterior occipital sulcus (AOS), posterior superior temporal sulcus (STSp), posterior superior temporal gyrus (STGp), transverse temporal gyrus (TTG), and supramarginal gyrus (SMG). Covariance matrices of activation data for *Emblem with Speech* for several networks comprised of these ROIs were then queried from the database:

network 1: {OP, STGp, TTG, AOS}
network 2: {OP, MOG, AOS, STSp}
network 3: {TTG, STGp, SMG, STSp}

```
string conditions[] = ["emblemwithspeech"];
```

[2]http://surfer.nmr.mgh.harvard.edu/

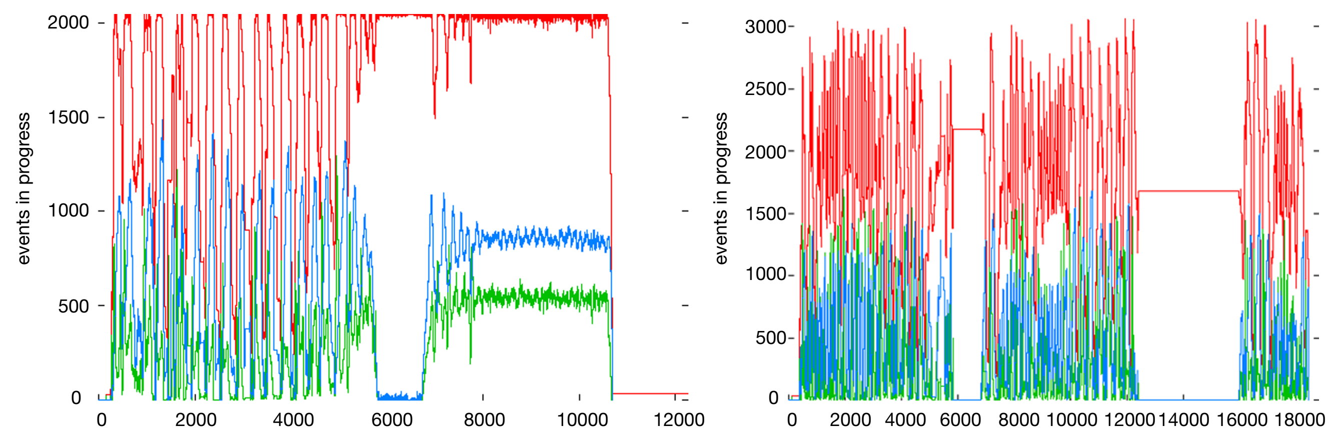

FIGURE 3 | Number of active processes during workflow execution: (left) Processing of the 4-region workflow over 2 experimental conditions. **(Right)** Processing of the 4-region workflow over multiple networks. The red line represents the execution of jobs on Ranger, while the blue and green represent the staging in and out of files respectively. Plots were generated by swift-plot-log, part of the Swift suite of tools.

```
int networks[] = [1:3];
int totalperms[] = [1:65536];
float initweight =.5;
foreach cond in conditions {
  foreach perm in totalperms {
   foreach n in networks {
    file covMatrix<single_file_mapper; file=@
    strcat("matrices/net",n,"_",cond,".
    cov")>;
    mxModel modmin<single_file_mapper; file=@
    strcat(n,"_",cond,"_",perm,".rdata")>;
    modmin = mxModelProcessor(covMatrix,mxScr-
    ipt,perm,initweight,
                             condition,@
                            strcat("net",n));
    }
  }
}
```

This results in a workflow containing 196,608 processing jobs (1 condition x 3 networks x 65536 models) and completed in approximately 5 hours on Ranger. For an example of how this might be used as part of a larger processing workflow see Section "Language Study Workflow in Swift" in the Appendix.

DISCUSSION AND FUTURE WORK

The workflows presented here do not result in a single "best" model representing connectivity amongst the four brain regions for the given conditions. Rather, their value lies in that they produce an exhaustive set of optimized models from which to begin searching for good-fitting models. Thus, a natural extension to this set of workflows might be a model-selection component based on a fit statistic (e.g., Bayesian information criterion, Akaike information criterion, RMSEA), an exploratory visualization component (see "Language Study Workflow in Swift" in the Appendix) or perhaps a combination of these methods. A "model-selection workflow" based on one or more fit statistics extending, for example, *workflow 1* would extract the desired fit statistic from each of the 65,536 optimized models and potentially keep or discard a given model based on whether or not it is above or below a selected threshold. It is worth noting that there is a good deal of controversy around which measures provide the most accurate model-selection (Bullmore et al., 2000) as well as some variation in how SEM software packages actually calculate those fit statistics (Clayton and Pett, 2008).

While the present workflows suggest new possibilities for exhaustive search and large-scale, parallel analysis techniques, their utility lies heavily in the ability to be easily replicated and reconfigured for use on varying datasets. Exhaustive search through a space of structural equation models is, *ab initio*, an exploratory technique. Thus, one cannot make statements concerning the probability that there are significant differences between models or that a selected parameter is significantly different from zero. The number of tested models is so great that any statistical argument concerning the likelihood of the data given a null hypothesis is overwhelmed by the number of comparisons made. In addition, one must be concerned about generalizability of results if a single data set was used—the exhaustive search may have overfit idiosyncrasies of the target data. Thus, it is imperative to cross-validate results from exhaustive search using other data sets.

On the other hand, an exhaustive search of the space of structural equation models for a particular data set does result in an empirical distribution of the fit statistics of the models. By plotting the log likelihood resulting from each fit against the number of degrees of freedom in its associated model, it is likely that clusters in the fit statistics will be observed. In this way, we may observe patterns of candidate models that are roughly equivalent given the data. Some of these models may be algebraically equivalent (vonOertzen, in press), and others may be empirically equivalent given the data. We intend the CNARI development effort to enable this type of data exploration.

Beginning with some basic pruning techniques, we can start to narrow down the space of models in the exhaustive set while leveraging Swift's ability to submit large numbers of processes, resulting in some powerful workflows. The first reduction in the exhaustive set of models is elimination of any models that are unidentified, that is, models containing negative degrees of freedom due to the presence of more unconstrained than constrained variables. The degrees of freedom can be easily calculated using the following formula:

```
(n(n+1)/2)-k
```

where n is the number of brain regions in the model and k is the number of free parameters and if the result is negative, the model is underidentified (Bollen, 1989). Additionally, a model with two-way symmetric connections is likely to fail attempts at optimization. Such a connection represents a type of cycle. In fact, most models containing cycles will be difficult to optimize as they are not usually identified in the absence of, e.g., longitudinal data (Neale and Cardon, 1992; Heath, 1993; Neale et al., 1994). The size if the acyclic set is given by

$4^{((n*(n-1)/2)}$.

An algorithm exists for finding cycles (Boker et al., 2002) that could potentially be used to further prune the model set. In addition to pruning cyclic and underidentified models, the set may also be pruned for models containing variables that lack residual error. The fit function cannot be evaluated under these circumstances, because the predicted covariance matrix is singular; therefore its determinant is zero, which results in the division of a negative quantity by zero in the calculation of the multivariate normal distribution probability density function, so optimization cannot be performed.

As **Table 1** shows, with a moderate degree of pruning, the set for four regions becomes trivial to run in the present infrastructure. Furthermore, the five-region set, while still a large number of processing jobs, becomes much more manageable.

Table 1 | Number of models produced for exhaustive and partially pruned workflows.

Regions	Exhaustive set	Identified	Acyclic
4	65,536	50,642	4,096
5	33,554,431	26,434,915	1,048,576
6	68,719,476,736	54,802,674,727	1,073,741,824

CNARI has been developed with the aim of managing a broad range of diverse neuroscience datasets and performing efficient, reliable parallel analysis workflows on them. Here we have demonstrated workflows that fully exercise this capability by applying this framework to large computational problems; namely, exhaustive search SEM. The need for data-driven techniques in modeling connectivity has emerged not only in our own work in studying language and aphasia but in SEM in general (Bullmore et al., 2000; Marrelec et al., 2007), though there has been little discussion of workflow management and parallel computing as means of addressing this need. Researchers, faced with seemingly insurmountable computational problems when selecting appropriate models to test, are often forced to rely on less-than-satisfactory approximations not only due to the sheer amount of processing power required but because of the daunting task of distributing those processing tasks in a cohesive manner such that the results are useful and replicable. As CNARI continues to evolve, we hope to expand these large-scale, data-driven workflows as we use them to address the complex research questions facing us.

ACKNOWLEDGMENTS

This research is supported in part by NSF grant OCI-721939, NIH grants DC08638, DA024304-02, DA-18673, 1R21DA024304—01, R21/R33 DC008638 and R01 DC07488, the U.S. Dept. of Energy under Contract DE-AC02-06CH11357, the James S. McDonnell Foundation and the TeraGrid HPC resources of the Texas Advanced Computing Center. The authors thank Ben Clifford and Mihael Hategan for creating and supporting the Swift parallel scripting system, and Michael Spiegal, Jeffrey Spies, and Tim Brick for developing and supporting the OpenMx system.

APPENDIX

LANGUAGE STUDY WORKFLOW IN SWIFT

The following is a prototype using Swift and demonstrating how the above modules can be assembled into a larger exploratory workflow. Exhaustive search is run for the *Emblem with Speech* condition on several four-region networks, and the results of the optimized models are stored in a connectivity database for visualization, further analysis, and pattern detection.

For each of the selected networks `multiNetworkSEM` is called with configuration files for the user to access the databases, information on the network to be processed, and the total number of models in the exhaustive set. First, the covariance data is pulled from the experiment database. This is seen in the `runQuery` function, which is Swift's call to a python database interface (see Small et al., 2009 for a more detailed description of this mediator component). Then for each iteration of the loop in line 34, Swift invokes `mxModelProcessor`, assigning each process a model to generate and optimize in OpenMx. The instantiation of the OpenMx model object and the call to the optimizer are encapsulated in the R script mapped on line 33, which is also passed to mxModelProcessor. Each of these processes writes out a file containing the result of the optimization, and these results can be read and inserted into the connectivity database, which is done with `insertOptMod`. It should be noted that both `insertOptMod` and `getCovariance` operate on the same principle: the user assembles a query that the python DBI will submit to the database. If the user also passes an R script (as in line 62), it will process the query result with that R script. Each result file is read, and its contents are inserted into the connectivity database where they can be further analyzed. A call to `plotLogLik` can be used to plot of the minimum values obtained by OpenMx for each model allowing for identification of patterns or clusters within the set (**Figure 4**).

```
 #### MultiNetworkSEM.swift


1. type file;
2. type mxMin;
3. type Rscript;
4. type dbConnect;
5. type mxModel{
6. int modnum;
7. int dof;
8. string best;
9. }
10.# ----------- atomic procedures ----------- #
11.
12.app (file matrix) runQuery (dbConnect dbconn,
   string query, Rscript calcCov){
13.     }
14.     mysqlPythonDBI query @calcCov @dbconn;
15.    }
16.
17.app (external inserted) insertMxResult
   (dbConnect dbconn, string query, file
    datafile)
18.    {
19.     mysqlPythonDBI query @dbconn stdout=@
        filename(inserted) @datafile;
20.    }
21. app (file min) mxModelProcessor ( file
    cov, Rscript mxModProc, int modnum, float
    weight, string cond, int net)
22.      {
23.     RInvoke @mxModProc @filename(cov) modnum
        weight cond net;
24.    }
25.
26. # ------ user-defined SEM procedures ------ #
27.
28. multiNetworkSEM(string condition,dbConnect
    emblemdb, dbConnect semdb, int n, string net,
    int totalperms[])
29. {
30.  float initweight =.75;
31.  file covariance<single_file_mapper;file=@
     strcat("net",n,"/",condition,".cov")>;
32.  covariance = getCovariance(condition, n,
     net, emblemdb);
33.  Rscript mxModProc<single_file_
     mapper;file="scripts/singlemodels.R">;
34.  foreach perm in totalperms{
35.    file modmin<single_file_mapper;file=@
       strcat("net",n,"/",condition,"_",perm,".
       stat")>;
```

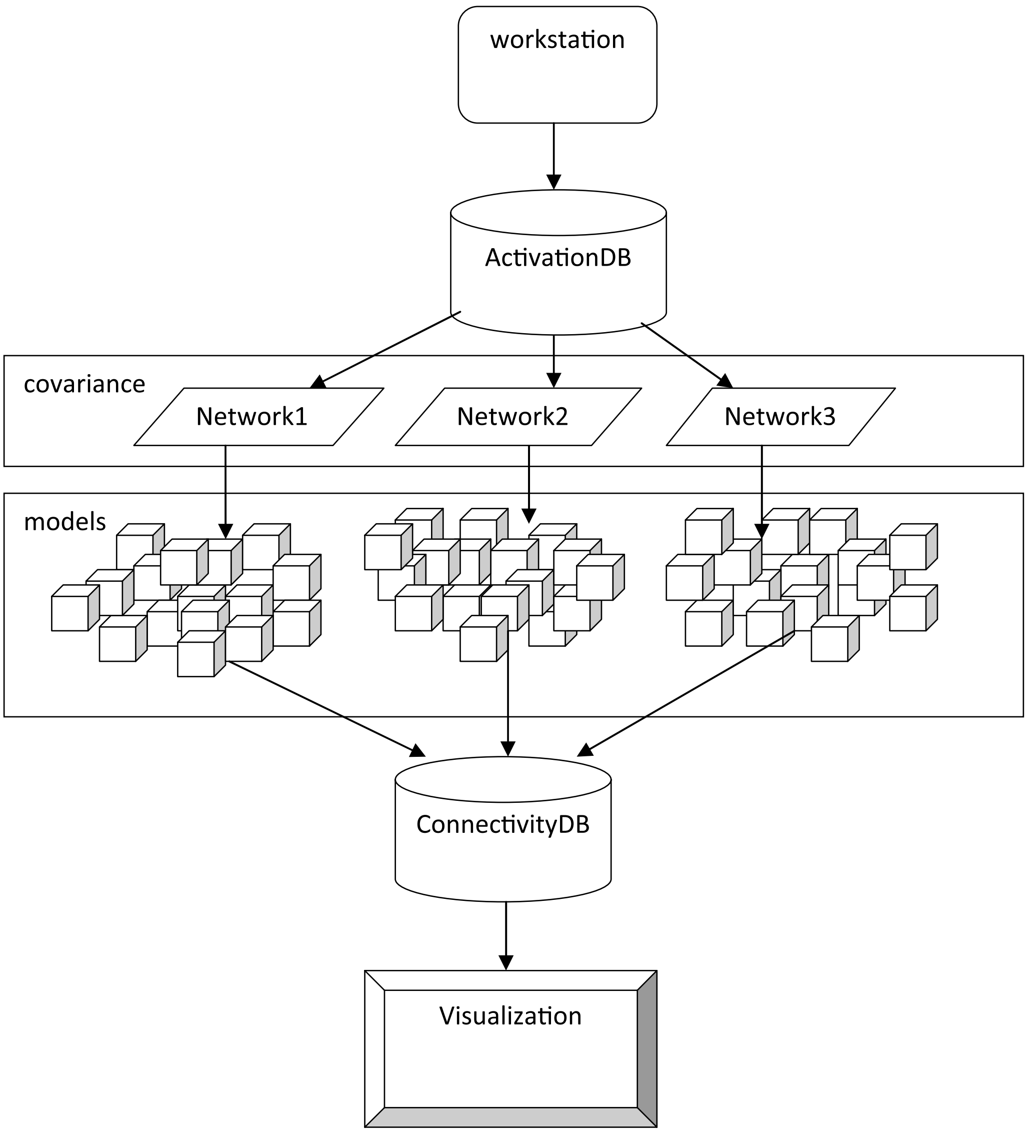

FIGURE 4 | Multinetwork Swift workflow for the *Emblem with Speech* condition.

```
   modmin = mxModelProcessor(covariance,mxMod
   Proc,perm,initweight,condition,n);
   external doneflag = insertOptMod(n, semdb,
   condition, modmin);
}

(external ins) insertOptMod(int net,
dbConnect dbconn, string cond, file modfile)
{
 string mysqlstr = @strcat("INSERT
 INTO optimized_models (network, deg_of_
 freedom, mx_minimum, modnum, cond) VALUES
 (",net,",DOF,BEST,MODNUM,",cond,");");
 string argList = @strcat(
 " --query ", mysqlstr,
 " --data ", @filename(modfile),
 " --conf ", @filename(dbconn));
 ins = insertMxResult(dbconn, argList,
 modfile);
}

(file covariance) getCovariance (string cond,
int net, string rois, dbConnect dbconn)
{
 string mysqlstr = @strcat("SELECT
 avg(",cond,"0B), avg(",cond,"1B),
 avg(",cond,"2B),",
"avg(",cond,"3B), avg(",cond,"4B),
 avg(",cond,"5B),",
"avg(",cond,"6B), avg(",cond,"7B),
 avg(",cond,"8B) ",
"FROM emblemfemlh where roi in (",rois,")
```

```
 group by roi ");
string argList = @strcat
" --conf ", "user.config",
" --query ", mysqlstr,
" --r_script ", "scripts/cov.R",
" --r_swift_args ", "matrices/net",net, "/", cond);
Rscript calcCov<single_file_mapper;file="scripts/cov.R">;
(covariance = runQuery(dbconn, argList, calcCov);
}
{
(file plotfile) plotLogLik(int net, string cond, dbConnect dbconn)

 Rscript rplot<single_file_mapper;file="scripts/plotloglik.R">;
 string mysqlstr = @strcat("SELECT deg_of_freedom,mx_minimum FROM optimized_models",
 " where network = ",net,";");
 string argList = @strcat(
 " --conf ", @filename(dbconn),
 " --query ", mysqlstr,
 " --r_script ", "scripts/plotloglik.R",
 " --r_swift_args ", @filename(plotfile));
 plotfile = runQuery(dbconn, argList, rplot);

 # ---------------- Main ----------------- #

string condition = "emblemwithspeech";
string networks[] = ["42, 34, 33, 60", "42, 15, 60, 80", "33, 34, 23, 80"];
dbConnect emblemdb <single_file_mapper; file="./user.config">;
dbConnect semdb <single_file_mapper; file="./user2.config">;
int totalperms[] = [1:65536];
foreach net,n in networks{
  multiNetworkSEM(condition,emblemdb,semdb,n,net,totalperms);
  }
```

REFERENCES

Ban, T., Naito, J., and Kawamura, K. (1984). Commissural afferents to the cortex surrounding the posterior part of the superior temporal sulcus in the monkey. *Neurosci. Lett.,* 49, 57–61.

Ban, T., Shiwa T., and Kawamura, K. (1991). Cortico-cortical projections from the prefrontal cortex to the superior temporal sulcal area (STs) in the monkey studied by means of HRP method. *Arch. Ital. Biol.*, 129, 259–272.

Barbas, H. (2000). Connections underlying the synthesis of cognition, memory, and emotion in primate prefrontal cortices. *Brain Res. Bull.* 52, 319–330.

Boker, S. M., and McArdle, J. J. (2005). Path analysis and path diagrams. In Encyclopedia of Statistics in Behavioral Science Vol. 3, B. Everitt and D. Howell, eds (New York, John Wiley & Sons), pp. 1529–1531.

Boker, S. M., McArdle, J. J., and Neale, M. (2002). An algorithm for the hierarchical organization of path diagrams and calculation of components of expected covariance. *Struct. Equ. Modeling,* 9, 174–194.

Bollen, K. A. (1989). Structural Equations with Latent Variables. New York, John Wiley & Sons.

Buchel, C., and Friston, K. J. (1997). Modulation of connectivity in visual pathways by attention: cortical interactions evaluated with structural equation modelling and fMRI. *Cereb. Cortex* 7, 768–778.

Bullmore, E., Horwitz, B., Honey, G., Brammer, M., Williams, S., and Sharma, T. (2000). How good is good enough in path analysis of fMRI data? *NeuroImage*, 11, 289–301.

Catlett, C. et al. (2007). TeraGrid: Analysis of Organization, System Architecture, and Middleware Enabling New Types of Applications, HPC and Grids in Action, L. Grandinetti, ed (Amsterdam, IOS Press, Advances in Parallel Computing Series).

Clayton, M. F., and Pett, M. A. (2008). AMOS versus LISREL: One data set, two analyses. *Nursing Res.*, 57, 283–292.

Cox, R. W. (1996). AFNI: software for analysis, and visualization of functional magnetic resonance neuroimages. *Comput. Biomed. Res.* 29, 162–173.

Duvernoy, H. M. (1991). The Human Brain: Surface, Three-dimensional Sectional Anatomy, and MRI. New York, Springer-Verlag.

Gill, P. E., Murray, W., Saunders, M. A., and Wright, M. H. (1986). User's Guide for NPSOL (Version 4.0): A FORTRAN package for nonlinear programming. Department of Operations Research, Stanford University, Stanford.

Hackett T. A., Stepniewska I., and Kaas J. H. (1999) Callosal connections of the parabelt auditory cortex in macaque monkeys. *Eur. J. Neurosci.,* 11, 856–866.

Hasson, U., Skipper, J. I., Wilde, M. J., Nusbaum, H. C., and Small, S. L. (2008). Improving the analysis, storage and sharing of neuroimaging data using relational databases and distributed computing. *Neuroimage,* 32, 693–706.

Horwitz, B., Tagamets, M. A., and McIntosh, A. R. (1999). Neural modeling, functional brain imaging, and cognition. *Trends Cogn. Sci. (Regul. Ed.)* 3, 91–98.

Ihaka, R., and Gentleman, R. (1996). R: A language for data analysis and graphics. *J. Comput. Graph. Stat.,* 5, 299–314.

Joreskog, K. G. (1967). Some contributions to maximum likelihood factor analysis. *Psychometrika,* 32, 443–482.

Loehlin, J. (1992). Latent Variable Models: An Introduction to Factor, Path, and Structural Analysis. Hillsdale, NJ: Lawrence Erlbaum Associates.

Marrelec, G., Horwitz, B., Kim, J., Pelegrini-Issac, M., Benali, H., and Doyon, J. (2007). Using partial correlation to enhance structural equation modeling of functional MRI data. *Magn. Reson. Imaging,* 25, 1181–1189.

McArdle, J. J., and Boker, S. M. (1990). Rampath. Hillsdale, NJ: Lawrence Erlbaum.

McArdle, J. J., and McDonald, R. P. (1984). Some algebraic properties of the reticular action model for moment structures. *Br. J. Math. Stat. Psychol.,* 87, 234–251.

McIntosh A. R., and Gonzalez-Lima, F. (1994) Structural equation modelling, and its application to network analysis in functional brain imaging. *Hum. Brain Mapp.*, 2, 2–22.

Neale, M. C., Boker, S. M., Xie, G., and Maes, H. H. (2003). Mx: Statistical modeling, 6th edn. Richmond, VA: Department of Psychiatry.

Neale, M. C., Eaves, L. J., and Kendler, K. S. (1994). The Power of the Classical Twin Study to Resolve Variation in Threshold Traits. Vol. 24, Netherlands: Springer.

Neale, M. C., and Cardon, L. R. (1992). Methodology for Genetic Studies of Twins and Families. NATO ASI Series. Vol. 67. Dordrecht, Kluwer Academic Publishers.

Petrides, M., and Pandya D. N. (1984). Projections to the frontal cortex from the posterior parietal region in the rhesus monkey. *J. Comp. Neurol.*, 228, 105–116.

Petrides, M., and Pandya D. N. (1988). Association fiber pathways to the frontal cortex from the superior temporal region in the rhesus monkey. *J. Comp. Neurol.* 273, 52–66.

Petrides, M., and Pandya, D. N. (1999). Dorsolateral prefrontal cortex: comparative cytoarchitectonic analysis in the human, and the macaque brain, and corticocortical connection patterns. *Eur. J. Neurosci.* 11, 1011–1036.

R Development Core Team (2008). R: A language and environment for statistical computing. R Foundation for Statistical Computing. Vienna, Austria. ISBN: 3-900051-07-0, Available at: http://www.R-project.org.

Rizzolatti, G., Luppino, G., and Matelli, M. (1998). The organization of the cortical motor system: new concepts.

Electroencephalogr. Clin. Neurophysiol., 106, 283–296.

Rizzolatti, G., Fogassi, L., and Gallese, V. (1997). Parietal cortex: from sight to action. *Curr. Opin. Neurobiol.*, 7, 562–567.

Rosa, M. G., Soares, J. G., Fiorani, M. Jr., and Gattass, R. (1993). Cortical afferents of visual area MT in the Cebus monkey: possible homologies between New and Old World monkeys. *Vis. Neurosci.*, 10, 827–855.

Saad, Z. S., Reynolds, R. C., Argall, B. D., Japee, S., and Cox, R. W. (2004). SUMA: An interface for surface-based intra- and inter-subject analysis with AFNI. Arlington, VA, IEEE International Symposium on Biomedical Imaging. pp. 1510–1513.

Seltzer, B., and Pandya, D. N. (1994). Parietal, temporal, and occipital projections to cortex of the superior temporal sulcus in the rhesus monkey: a retrograde tracer study. *J. Comp. Neurol.*, 343, 445–463.

Skipper, J. I., Godin-Meadow, S., Nusbaum, H. C., and Small, S. L. (2007). Speech-associated gestures, Broca's area, and the human mirror system. *Brain Lang.*, 101, 260–277.

Skipper, J. I., Goldin-Meadow, S., Nusbaum, H. C., and Small, S. L. (2009). Gestures orchestrate brain networks for language understanding. *Curr. Biol.* 19, 661–667.

Small, S. L., Wilde, M., Kenny, S., Andric, M., and Hasson, U. (2009). Database-managed Grid-enabled analysis of neuroimaging data: The CNARI framework. *Int. J. Psychophysiol.* 73, 62–72.

Solodkin, A., Hlustik, P., Chen, E. E., and Small, S. L. (2004). Fine modulation in network activation during motor execution and motor imagery. *Cereb. Cortex*, 14, 1246–1255.

Stef-Praun, I., Foster, U., Hasson, M., Hategan, S.L., and Wilde, S. M. (2007). Accelerating medical research using the Swift Workflow System. Paper Presented at the HealthGrid 2007, Geneva.

vonOertzen, T. (in press). Power equivalence in structural equation modeling. *Br. J. Math Stat. Psychol.*

Walsh, R. R., Small, S. L., Chen, E. E., and Solodkin, A. (2008). Network activation during bimanual movements in humans. *Neuroimage*, 43, 540–553.

Wright, S. (1921). Correlation and causation. *J. Agric. Res.*, 20, 557–585.

Zhao, H., Clifford, F., von, L., Nefedova, R., and Stef-Praun, W. (2007). Swift: Fast, Reliable, Loosely Coupled Parallel Computation. IEEE Congress on Services, pp. 199–206.

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Received: 11 April 2009; paper pending published: 10 July 2009; accepted: 09 September 2009; published online: 20 October 2009.

Citation: Kenny S, Andric M, Boker SM, Neale MC, Wilde M and Small SL (2009) Parallel workflows for data-driven structural equation modeling in functional neuroimaging. Front. Neuroinform. ***3****:34. doi: 10.3389/neuro.11.034.2009*