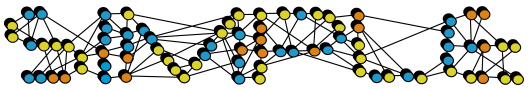
The  manual

"Keep It SIMPLE, Stupid!"

(Kelly Johnson, lead engineer at the Lockheed Skunk Works, coined the famous KISS principle stating that systems work best if they are kept simple rather than made complex, therefore simplicity should be a key goal in design and unnecessary complexity should be avoided.)

"Everything should be made as SIMPLE as possible, but no simpler"

(A. Einstein)

About SIMPLE

Single-particle IMage Processing Linux Engine (SIMPLE) does *ab initio* 3D reconstruction (programs **cluster** & **origami**), heterogeneity analysis (programs **cluster**, **origami** & **cycler**), and high-resolution refinement (programs **align, reconstruct, automask** & **cycler**). The SIMPLE back-end consists of an object-oriented numerical library with a single external dependency—the Fastest Fourier Transform in the West (FFTW) (Frigo and Johnson, 2005). The SIMPLE front-end consists of a few standalone, interoperable components developed according to the 'Unix toolkit philosophy'.

The SIMPLE codes prioritize computational efficiency, robustness, parameter-free, and empiric design over mathematical elegance. One of the SIMPLE development goals is to move cryo-EM image processing from the distributed computing environments to multicore desktop computers. Smaller data sets (<10,000 images) can feasibly be processed on current generation laptops. The SIMPLE algorithms are based on new developments inspired by methods used in the fields of operations research, artificial intelligence, machine learning, and coherent diffractive imaging, in addition to concepts and ideas shaped by decades of research in the single-particle electron microscopy community. SIMPLE is written in modern Fortran, combining the high performance of compiled code with the ease of

development via object oriented design paradigms. The modular design of the SIMPLE suite invites advanced users development of new alignment and 3D reconstruction algorithms for single-particle imaging. Most of the SIMPLE algorithms are parallelized for shared-memory architectures using the OpenMP protocol.

Cryo-EM overview

Structure determination of large, dynamic macromolecular complexes may provide molecular-level insights into fundamental biological processes. Cryo-electron microscopy (cryo-EM) of freestanding molecules (single-particles) can be used to obtain molecular density maps at atomic (in favorable cases) to intermediate resolution (8-30 Å). Many (>10,000) noisy projection images need to be aligned in three-dimensional (3D) space and have their compositional or conformational state assigned. Density maps are then obtained from aligned images using tomographic reconstruction techniques. Images are acquired using low electron dose in order to avoid excessive radiation damage. This results in extremely noisy images, with the sought detailed structural features intact. After correction for the effects of the contrast transfer function (CTF) of the electron microscope, the primary sources of image variations in a single-particle data set are due to differences in orientation of the imaged molecules and noise. Biological preparations of large macromolecules are often heterogeneous in composition as well as in conformation. Additional image variations may therefore be present due to the inherent 'heterogeneity' of large macromolecular complexes. It is critical to resolve these variations in order to obtain faithful 3D reconstructions. It is also of biological interest to characterize different conformations of biological macromolecules structurally. Variations due to 'heterogeneity' are generally subtler than variations due to differences in orientation, making them difficult to distinguish from noise-dependent variations.

Installation of SIMPLE

```
>>> mv Downloads/simple_mac_Dec18_2011.tar.gz /usr/local

>>> gunzip simple_mac_Dec18_2011.tar.gz

>>> tar -xvf simple_mac_Dec18_2011.tar

>>> cd simple

>>> ./simple_config.pl

Installation on hopper/local/biox2?
```

Hopper and Biox2 are clusters that we use for distributed SIMPLE execution. If you want a distributed SIMPLE version you will have to modify the **cluster_exec.pm** module in the **lib** folder (see **Distributed Execution** below for details).

```
>>> local
```

The files 'source_tcsh' and 'source_bash' contains the two lines you need to add to your .cshrc or .bashrc file, respectively, in order to have prompt access to the SIMPLE programs. If you are compiling SIMPLE from source, the compiler options used by the compile script **simple_compile.pl** are valid for the Intel Fortran compiler, assumed to be executed from the prompt with the command **ifort**. Currently, no other compilers are supported. Execute the compile script to get instructions.

Program descriptions

SIMPLE is not a standalone suite for single-particle reconstruction, but complements existing developments. It is assumed that the windowed single-particle images represent 2D projections, and therefore, that the projection slice theorem applies (Bracewell, 1956). This requires correction of the micrographs due to the contrast transfer function (CTF) of the electron microscope and particle windowing using other software. In addition, the windowed projections should be roughly centered in the box. Many program suites are available for dealing with these and other matters (Frank et al., 1996; Hohn et al., 2007; Ludtke et al., 1999; Sorzano et al., 2004). In EMAN, the envelope component of the CTF, describing the fall-off of the signal with resolution, is parameterized and then used in the class averaging procedure in the refinement. Some implementations only phase-corrects the micrographs, ignoring the envelope function initially. Instead, the final refined maps are B-factor sharpened. SIMPLE has so far been use in combination with the latter approach, assuming the same resolution fall-off for all micrographs. The SIMPLE workflow is divided into four phases:

(1) CTF correction and particle windowing.
(2) *Ab initio* reconstruction.
(3) Heterogeneity analysis.
(4) Refinement.

We usually execute phase (1) by parameterizing the CTF with Ctffid3 (Mindell and Grigorieff, 2003), doing CTF phase correction of the micrographs in Spider (Frank et al., 1996), and using EMAN's (Ludtke et al., 1999) program boxer for particle windowing. EMAN's graphical user interface is used for quick inspection of volumes and class averages. UCSF Chimera is used for more detailed analysis of the reconstructed volumes (Pettersen et al., 2004). Phases (2)-(4) are executed with SIMPLE, as detailed below.

The SIMPLE program instructions are listed in the order the programs are normally executed during the reconstruction process. Each paragraph begins with

an overview description of the algorithms used by the program, followed by a paragraph describing how to execute the program, and ending with a paragraph of comments describing alternative execution routes and parameter tweaking strategies. All SIMPLE programs are executed using the command line. Input and output consists of SIMPLE 2D Fourier transform stacks, Spider image stacks, Spider volumes, Spider document files, and .dat text files. The output is written directly to the working directory, giving you freedom to organize a directory structure that suits your needs. Remember that another round of execution in the same directory overwrites your old files. The notation <what> denotes input input parameter "what". <this|that> denotes alternative input parameters "this" and "that", [<noway>] denotes the optional parameter "noway", and {val} denotes the suggested or default value "val". Command line arguments are passed as *key*=<value>, where "*key*" is the key in the hash used to store the parameters. For example, the following instruction:

```
>>> program param=<what{20-40}> either=<this|that>
[itisanumber=<noway{0}>]
```

can be executed like this:

```
>>> program param=30 either=this
```

or like this:

```
>>> program param=35 either=that itisanumber=500
```

The command line interface is similar to the interface used by the *proc2d* and *proc3d* programs in EMAN, so you should feel right at home if you are familiar with EMAN. You are encouraged to contact the author of this manual and bark at him if the instructions contain errors (e-mail: hael@stanford.edu). Each program prints a short version of the description presented here when executed without command line arguments.

Command line dictionary

angres	angular resolution in the plane (in degrees)
box	image size in pixels (image assumed to be a square *box*box* array)
clsdoc	Spider format clustering document
doalign	do alignment or not.
dopca	do PCA or not.
debug	debug mode or not
fromp	from particle index
fstk	SIMPLE Fourier stack name (*.fim suffix required)
hp	high-pass limit (in Å)
lp	low-pass limit (in Å)
maxits	maximum number of iterations

maxp	maximum number of particle images in a cluster
minp	minimum number of particle images in a cluster
mode	mode of operation for certain multitasking programs
msk	circular or spherical mask radius (in pixels)
mw	molecular weight (in kD)
nbest	population size for the optimization
ncls	number of clusters
nptcls	number of particle images
nran	number of images in random sample
nrnds	number of restart rounds
nspace	number of projection directions in search space
nthr	number of openMP threads
nvars	number of eigenvectors
oritab	SIMPLE text file with orientations and state assignments
outbdy	body of output files, if file==“boring.job”, then “boring”==bdy
outfile	SIMPLE output text file
outstk	output spider image stack (*.spi suffix required)
pgrp	point-group symmetry
ring1	inner ring (used to generate mask for correlation calculation)
ring2	outer ring (used to generate mask for correlation calculation & PCA)
smpd	sampling distance (in Å)
stk	spider image stack name (*.spi suffix required)
to	to particle index
trs	origin shift search range parameter, search range is [-*trs*,*trs*]
vol1	spider volume name (*.spi suffix required)

Program: cluster

cluster is a program for image clustering based on reference-free in-plane alignment (Penczek et al., 1992) and probabilistic principal component analysis (PCA) for generation of feature vectors (Tipping and Bishop, 1999). Agglomerative hierarchical clustering (HAC) is used for grouping of feature vectors (Murtagh, 1983). Refinement of the clustering solution is done with the center-based k-means clustering. **Cluster** in-plane aligns the input image stack. Bicubic interpolation is used for shifting and rotating the stack before extraction of the pixels within the circular mask defined by mask radius *ring2*. Next, the probabilistic PCA method generates feature vectors from the vectors of extracted pixels. The ‘cluster.log’ file describes what was done in each step and lists the output files. The minimum cluster population (*minp*) prevents clusters below population *minp* to be represented by an output average. Clean up the class averages stack before generation of a Fourier stack and input to program **origami**, described below.

Usage:

```
>>> cluster stk=ptclstk.spi box=<box size(in pixels)>
nptcls=<nr of images in stack> smpd=<sampling distance(in
Å)> [ring1=<first ring(in pixels){5}>] [ring2=<second
ring(in pixels){box/2}>] [ncls=<nr clusters{500}>]
[minp=<minimum nr ptcls cluster{10}>] [nvars=<nr
eigenvectors{30/60}>] [nthr=<nr openMP threads{1}>]
[nran=<size of random sample{nptcls}>] [oritab=<SIMPLE
alignment doc>] [clsdoc=<Spider clustering doc>]
[realspace=<yes|no>] [dopca=<yes|no>] [doalign=<yes|no>]
[debug=<yes|no>]
```

Comments:

The setup allows for quick testing of the number of clusters. One pass produces the file 'pdfile.bin' containing the matrix of all pair-wise feature vector distances. Using the optional parameter *dopca* set to 'no' in a second round of execution from the same directory will make the program read the previously generated distances and re-do the clustering using whatever settings inputted for parameters *ncls* & *minp*. The clustering is primarily used as means for reducing the computations due to common lines-based search in ***origami.*** The general advice is to keep the number of clusters < 1500. In future SIMPLE releases we will provide nonlinear feature extraction that is better than PCA, which may allow for automatic determination of the number of clusters in the data. If you want top try a different clustering algorithm from another package, input *ncls*=1 to indicate that you only want cluster to produce an in-plane aligned stack for you and not do clustering. The optional parameter *oritab* is used to provide in-plane parameters for the clustering (provided by programs **align** or **cycler**, described below). This option is used for generating class averages that are going to be subjected to heterogeneity analysis by program **origami**, described below. The default setting uses 30 eigenvectors if you are not inputting in-plane parameters via optional parameter *oritab,* and 60 eigenvectors if you do input in-plane parameters. Note that the distance matrix is kept in RAM, so for large data sets you need LOTS of internal memory. This quirk can be addressed by using a random sample of the data for initial clustering by HAC. This is done by setting *nran* to some number < *nptcls*. In this setting, the HAC centers generated from the random sample are used to extend the clustering to the entire data set with k-means. This overcomes the well-known initialization problem of k-means and enables clustering of many hundreds of thousands of particle images. SIMPLE has been used to cluster 300,000 images with a box size of 100 using a random subset of 60,000 images on a machine with 96 GB RAM.

Program: spi_to_fim

spi_to_fim is a program for generating and stacking 2D Fourier transforms using a spider stack of real images as input. EMAN, Spider, and **cluster** generates such image stacks. Output consists of the files *outbdy.fim* (stack of transforms) and *outbdy.hed* (header) that are used by virtually all other SIMPLE programs.

Usage:

```
>>> spi_to_fim stk=spistackin.spi box=<box size(in pixels)>
nptcls=<nr of images in stack> smpd=<sampling distance(in
Å)> outbdy=<body of output files> [fromp=<start ptcl>]
[top=<stop ptcl>] [msk=<mask radius(in pixels)>]
[pad=<yes|no>] [debug=<yes|no>]
```

Comments:

Beware that the mask parameter is optional despite that the images subjected to Fourier transformation need to be masked with a soft mask to avoid edge-induced artifacts. If the input images to **spi_to_fim** are not masked beforehand, you must apply a soft mask here using the optional *msk* parameter. Setting the optional parameters *pad* to 'yes' will pad the images to twice the size before Fourier transformation. Padded Fourier stacks are used for reconstruction. It is indicated in the below described individual programs if the input Fourier stack is expected to be padded. It may be convenient for you to generate the padded stack now. Setting the optional parameter *debug* to 'yes' will not produce the full stack, but produce a spider stack (debug.spi) with one image with all transformations (masking, shifting & padding) applied to the first image of the input stack.

Program: fim_to_spi

Since there is a program spi_to_fim for generating stacks of Fourier transforms there is naturally also a program for back transformation of the Fourier stack: **fim_to_spi**. Output consists of a spider image stack.

Usage:

```
>>> fim_to_spi fstk=fprojs.fim outstk=rprojs.spi
[debug=<yes|no>]
```

Comments:

Set the optional parameter debug to "yes" if you want to print the stack header. This may be useful for checking that the stack has been generated using the intended input parameters.

Program: origami

origami is a program for *ab initio* 3D alignment and heterogeneity analysis of class averages using optimization of a spectral ensemble common line correlation coefficient (Elmlund et al., 2010; Elmlund et al., 2008). The coefficient is called 'spectral' because it uses adaptive low-pass filtering methods for improved search behavior and noise robustness. The term 'ensemble coefficient' was coined to distinguish the method from angular reconstitution, which uses an anchor set of only a few *ab initio* aligned class averages to orient the remaining ones (van Heel, 1987). Projection direction assignment is done in a discrete space using the

constraint that any two class averages are not allowed to occupy the same direction. Simulated annealing is used for the combinatorial optimization. The first discrete alignment is restarted three times on random starting configurations, and the best solution is written to the first alignment document. Finally, the additional origin shift and conformational state parameters are included in a greedy adaptive local search based scheme. This scheme steers a mixed continuous-discrete differential evolution optimizer operating over the six degrees of freedom of one Fourier plane. The initial volumes are refined with **cycler** (described below). Input to **origami** is a Fourier stack of class averages 'cavgstk.fim'. Output from **origami** consists of reconstructed volumes, with and without solvent flattening applied to reduce the background noise. The automatically generated 'origami.log' lists the output files.

Usage:

```
>>> origami fstk=cavgstk.fim lp=<low-pass limit(in Å){15-
30}> [froms=<number of states from{1}>] [tos=<number of
states to{1}>] [maxits=<nr of rounds{10}>] [msk=<mask
radius(in pixels)>] [mw=<molecular weight(in kD){0}>]
[frac=<fraction of ptcls to include{0.8}>] [amsklp=<auto
mask low-pass limit(in Å){50}>] [edge=<edge size for
softening of molecular envelope(in pixles){3}>] [nthr=<nr
of openMP threads{1}>] [trs=<origin shift(in pixels){3}>]
[hp=<high-pass limit(in A){100}>] [oritab=<input alignment
doc>] [pgrp=<cn|dn>] [doalign=<yes|no>] [debug=<yes|no>]
```

Comments:

The optional parameter *maxits* controls the number of cycler iterations. All cycler volumes are written to folders indicating which state distribution is being refined. You might be chocked to find 80 volumes in the *4_states* folder. Don't be. Just make sure that the refinement gracefully improved the starting volumes, store the volumes from the last iteration, and throw out the rest. The outcome of origami depends on the quality of the class averages. Hence, the low-pass limit should be set to reflect the information content in the averages. A limit of 20 Å works for most purposes. Origami can be executed in three modes:

(1) *Ab inito* reconstruction mode
(2) Early stage heterogeneity analysis mode
(3) Late stage heterogeneity analysis mode

In mode (1) class averages have been obtained in **cluster** using the reference-free 2D alignment algorithm (not inputting model-based alignment document), and execution might look like:

```
>>> origami fstk=cavgstk.fim lp=20 nthr=8 mw=500
```

In mode (2) you have generated model-based in-plane parameters (in **cycler** or **align**) and inputted them to generate class averages in **cluster.** Still, you are

somewhat concerned about bias to the low-resolution initial volume and want determine projection directions *ab initio.* Execution might look like:

```
>>> origami fstk=cavgstk.fim lp=20 nthr=8 mw=500 tos=3
```

In mode (3) you are confident with the alignment obtained by **cycler** or **align**, and you have generated beautiful class averages with **cluster** using these in-plane parameters as input. Now, use **align** or **cycler** to align the class averages back to the best so far reconstruction(s). Input the alignment document to origami and turn off the reference-free orientation refinement:

```
>>> origami fstk=cavgstk.fim lp=20 nthr=8 mw=500 tos=3
oritab=algndoc.dat doalign=no
```

Origami tests different number of states, from *froms* to *tos*. The number of states is a critical variable. When the number of states is increased, the number of parameters in the model is increased and the correlation will always improve. This is an example of an over fitting problem. In attempt to determine weather the increase from *s* to *s*+1 number of states gives rise to a significant correlation gain, the two distributions of particle correlations are compared using a Kolmogorov-Smirnov (K-S) statistical test. The K-S statistic and the K-S probability are printed to the 'cluster.log' file. If the K-S probability value is small and the K-S statistic is relatively large, the two distributions differ significantly. A binary heterogeneity will give rise to a significant change in correlation going from one to two state groups, whereas the change from two to three groups and three to four groups will be less significant. The K-S statistic may be informative, but it does not insure against lowly populated, poor quality state groups showing up. These groups are usually easy to identify and they should be excluded from further analysis.

Program: align

Align is a program for continuous reference-based 3D alignment of individual images, given input reference volume(s). The algorithm is based on advanced differential evolution (DE) for continuous global optimization (Elmlund and Elmlund, 2009). Previously, SIMPLE used a Fourier-based interpolation scheme that extracted common lines between reference central sections and the particle section. In this release, the Fourier-based interpolation scheme is reconciled with that used in Frealign. The entire reference section used for matching is now interpolated directly from the 3D Fourier volume, which requires little extra computation. The method of spectral self-adaptation is applied to estimate a particle dependent low-pass frequency limit.

Usage:

```
>>> align mode=<mode nr> fstk=<input Fourier stack(*.fim)>
vol1=<refvol_1.spi> [vol2=<refvol_2.spi> … etc.]
outfile=<output alignment doc> [msk=<spherical mask
```

```
radius(in pixels)>] [lp=<low-pass limit(in Å){30}>]
[trs=<origin shift(in pixels){3}>] [trsstep=<origin shift
stepsize{1}>] [nthr=<nr openMP threads{1}>] [oritab=<input
alignment doc>] [fromp=<start ptcl index>] [top=<stop ptcl
index>] [nspace=<nr of projectionns in discrete
search{500}>] [pgrp=<cn|dn>] [hp=<high-pass limit(in Å)>]
[debug=<yes|no>]
```

Comments:

Available modes are:

mode=20: multi-reference alignment with fixed low-pass limit (*lp*), no input orientations are required

mode=21: multi-reference alignment with spectral self-adaptation, input orientations are required

mode=23: for finding filtering threshold or do spectral scoring, input orientations are required

The strategy is to **begin with mode=20 alignment** on the starting volume(s) using a low-resolution low-pass limit (*lp* typically set to 30 Å). In the first rounds, the origin shift parameters will be far from optimal. If many solutions lie on the shift interval borders, the Fourier stack should be shifted (using program **shift_fim**, described below) and the mode=20 search re-run. This reduces the complexity of the optimization problem and improves the quality of the solutions obtained in later refinement rounds. After completing the shift alignment and the first orientation assignment, volumes are reconstructed with program **reconstruct** (described below). **In subsequent refinement rounds, mode=21-based alignment** is combined with reconstruction until quasi-convergence. The shift range should now be limited to around [-3,3]. **The mode=21 refinement should be run until convergence, as measured by the Fourier Shell Correlation (FSC) plot calculated between reconstructions from successive rounds.**

Program: reconstruct

Reconstruct is a program for reconstructing volumes from a padded SIMPLE Fourier transform stack (*.fim), given input orientations and state assignments (obtained by program **align**). The algorithm is based on Fourier gridding with a Gaussian window function. This window function reduces the real-space ripple artifacts associated with direct moving window sinc interpolation. The feature sought when implementing this algorithm was to enable quick, reliable reconstruction from aligned individual particle images. A bonus is that the gridding method is very easy to parallelize.

Usage:

```
>>> reconstruct fstkpd=fprojs.fim [oritab=algndoc.dat]
[msk=<mask radius(in pixels)>] [mw=<molecular weight(in
```

kD){0}>] [*frac*=<fraction of ptcls to include{0.8}>] [*lp*=<auto mask low-pass limit(in Å){40}>] [*nthr*=<nr of openMP threads{1}>] [*eo*=<yes|no>] [*pgrp*=<cn|dn>] [*debug*=<yes|no>] [*part*=<partition number>] [*fromp*=<start ptcl>] [*top*=<stop ptcl>] [*state*=<state to reconstruct>]

Comments:

Random orientations are used to generate the reconstruction if no orientations are inputted. The optional parameter *mw* is used to do solvent flattening of the output volumes to reduce the background noise. If you do not know the molecular weight of your complex or you study materials that are not protein (metallic nanoparticles, for example) then input 0 to indicate that the molecular weight is unknown. The defaiult *mw* value is set to 0. The optional parameter *eo* is used for generating even-odd reconstructions subjected to FSC analysis.

Program: automask

automask is a program for doing solvent flattening of an input spider volume. The algorithm for background removal is based on low-pass filtering and binarization. First, the volume is low-pass filtered to *lp*. A binary volume is then generated by assigning foreground pixels (=1) based on the volume calculated from the molecular weight, assuming a protein density of 1.43 g/mL. The edge of the resulting binary volume is softened by a real-space low-pass filter before multiplying it with the 'raw' input volume to generate the flattened map.

Usage:

>>> **automask** *vol1*=invol1.spi [*vol2*=invol2.spi etc.] *box*=<box size(in pixels)> *smpd*=<sampling distance(in Å)> *mw*=<molecular weight(in kD)> [*lp*=<low-pass limit(in Å){40}>] [msk=<mask radius(in pixels)>] [*edge*=<edge size for softening of the molecular envelope(in pixels){3}>] [debug=<yes|no>]

Comments:

No comments.

Program: cycler

cycler combines **align**, **reconstruct**, and **automask** into a single program for refinement that is suitable for shared-memory multi-processor architectures, since the parallelization is based on the OpenMP protocol. Not much else to say – the individual components are described above.

Usage:

>>> **cycler** *fstk*=<input Fourier stack(*.fim)> *fstkpd*=<input padded Fourier stack(*.fim)> *vol1*=<refvol_1.spi>

[*vol2*=<refvol_2.spi> … etc.] [*maxits*=<nr of rounds{50}>] [*msk*=<spherical mask radius(in pixels)>] [*mw*=<molecular weight(in kD){0}>] [*startit*=<start iteration nr{1}>] [*frac*=<fraction of ptcls to include{0.8}>] [*lp*=<low-pass limit(in Å){30}>] [*amsklp*=<automask low-pass limit(in Å){40}>] [*edge*=<edge size for softening of the molecular envelope(in pixels){3}>] [*trs*=<origin shift(in pixels){3}>] [*trsstep*=<origin shift stepsize{1}>] [*nthr*=<nr openMP threads{1}> [*oritab*=<input alignment doc>] [*pgrp*=<cn|dn>] [*specoff*=<yes|no>] [*hp*=<high-pass limit (in Å)>] [*mskfile*=<mask.spi>] [debug=<yes|no>]

Comments:

Cycler can be run on class averages or smaller data sets. If class averages are used, then turn off the spectral self-adaption (by setting specoff=yes) and set a fixed limit. For refinement of large data sets you need to do distributed execution in a cluster environment (see **Distributed Execution**, below).

Program: shift_fim

shift_fim is a program for shifting a SIMPLE Fourier stack according to the origin shifts printed in the alignment document produced by programs **align** or **cycle**. A linear phase shift is applied in Fourier space, so there are no interpolation errors associated with this operation. Happy shifting!

Usage:

>>> **shift_fim** *fstk*=fstack.fim *outfstk*=shifted_fstack.fim *oritab*=algndoc.dat [debug=<yes|no>]

Comments:

No comments.

Program: fish

fish is a program for fishing out good data. After a few rounds of bijective search, when the in-plane parameters are of reasonable quality, it is time to manually select good clusters and use this selection to extract good particle images. This is done by deleting junk averages (for example using EMAN) and making a stack of selected class averages (cavgs_sel.spi). The selected class averages are inputted together with all class averages (cavgs_all.spi), and the clustering document to fish. Fish then outputs the good particles (good_ptcls.spi), the good particles clustering (good_cls.spi) and a list of good average indices (good_avg_inds.spi).

Usage:

```
>>> fish stk=cavgs_all.spi stk2=cavgs_sel.spi
nptcls=<number of ptcl images> box=<image size(in pixels)>
clsdoc=<SPIDER classification doc> [nthr=<nr of openMP
threads{1}>] [debug=<yes|no>]'
```

Comments:

Use Spider to make the stacks that you need from the produced documents.

Technical stuff

Differential Evolution

The SMPLE optimizers include combinatorial optimization by simulated annealing and local search-based algorithms (Elmlund et al., 2010; Elmlund and Elmlund, 2009; Elmlund et al., 2009; Elmlund et al., 2008). A new differential evolution (DE) algorithm is used for continuous refinement of the candidate populations of orientations, both in the projection matching-based refinement and in the reference-free common lines-based 3D analysis (Das et al., 2009; Das and Suganthan, 2011). The simplest form of DE adds a scaled difference between two randomly selected population members of solution vectors to a third member (the *target* vector) to create a *donor* vector. A mechanism for blending components of the donor with the target and create a *trial* vector is introduced. The trial (or offspring) vector challenges the population vector with the same index. Once the last trial vector has been evaluated, the survivors of all the pair-wise competitions become parents for the next generation in the evolutionary cycle. Previously, a trial individual $\boldsymbol{V}_{i_g}$ was generated for each solution vector $\boldsymbol{Z}_{i_g}$ by weighted linear combination according to the below "mutation scheme" (Elmlund and Elmlund, 2009)

$$\boldsymbol{V}_{i_g} = \boldsymbol{Z}_{g\_best_g} + F \cdot \left(\boldsymbol{Z}_{\alpha_g} + \boldsymbol{Z}_{\beta_g} - \boldsymbol{Z}_{\gamma_g} - \boldsymbol{Z}_{\delta_g}\right)$$

with $\alpha \neq \beta \neq \gamma \neq \delta \neq i \in \{1,2,\dots,P\}$ being random indexes, different from the target solution index *i*. $\boldsymbol{Z}_{g\_best_g}$ is the individual in generation g with best correlation, and $F > 0$ is a scaling parameter. This mutation scheme has a tendency to converge prematurely to a local optimum due to lack of mechanisms for search diversification. To reduce the influence of the globally best solution vector on the population, and promote diversification of the search, a new neighborhood-based mutation model is introduced. The neighborhood of a solution vector is defined as the set of other vectors that it connects to. The graph of interconnections is called the neighborhood structure. The vector population is assumed to be organized on a ring-shaped neighborhood topology defined on the index graph of the solution vectors, such that vectors $\boldsymbol{Z}_{P_g}$ and $\boldsymbol{Z}_{2_g}$ are the immediate neighbors of vector $\boldsymbol{Z}_{1_g}$. For each vector in the population, a neighborhood of radius $r \in [1,(P-1)/2]$, consisting of the vectors $\boldsymbol{Z}_{i_g-r},\dots,\boldsymbol{Z}_{i_g},\dots,\boldsymbol{Z}_{i_g+r}$ is defined. Thus, the model maintains overlapping neighborhoods in a single population. Every individual has a local neighborhood that is relatively small, restricted, and defined by the ring-shaped topology. To maintain neighborhood diversity, the ring-shaped topology is initialized with random order of the solution vectors, and iteration over the index graph is done in randomized order. The local donor vector is defined as

$$\boldsymbol{L}_{i_g} = \boldsymbol{Z}_{i_g} + K_n \cdot \left(\boldsymbol{Z}_{n\_best_{i_g}} - \boldsymbol{Z}_{i_g}\right) + K_n \cdot F \cdot \left(\boldsymbol{Z}_{\alpha_g} + \boldsymbol{Z}_{\beta_g}\right)$$

where K_n is a uniform random number $\in [0,1]$, $F > 0$ is a scaling factor, the subscript $n\_best_{i_g}$ indicates the best vector in the neighborhood of $\boldsymbol{Z}_{i_g}$, and $\alpha \neq \beta \neq i \in [i-r, i+r]$ are random indices. The combination coefficient K_n replaces the discrete recombination operator used in many genetic and evolutionary algorithms for trial vector generation. The global donor vector is defined as

$$\boldsymbol{U}_{i_g} = \boldsymbol{Z}_{i_g} + K_g \cdot \left(\boldsymbol{Z}_{g\_best_{i_g}} - \boldsymbol{Z}_{i_g}\right) + K_g \cdot F \cdot \left(\boldsymbol{Z}_{\gamma_g} + \boldsymbol{Z}_{\delta_g} - \boldsymbol{Z}_{\varepsilon_g} - \boldsymbol{Z}_{\zeta_g}\right)$$

where K_g is a uniform random number $\in [0,1]$, $F > 0$ is a scaling factor, the subscript $g\_best_{i_g}$ indicates the globally best vector, and $\gamma \neq \delta \neq \varepsilon \neq \zeta \neq i \in [1, P]$ are random indices. The local and global donor vectors are combined using a linear weight $W \in (0,1)$ to form the final donor vector

$$\boldsymbol{V}_{i_g} = W \cdot \boldsymbol{U}_{i_g} + (1 - W) \cdot \boldsymbol{L}_{i_g}$$

$W = 0.5$ puts intensification of the search around the globally best vector (exploitation) and diversification of the search (exploration) on equal footing, whereas with $W < 0.5$ exploration is favored over exploitation. Unsupervised learning of W is discussed below. The Euler angles are treated as cyclic variables and the origin shift variable bounds are fulfilled by controlled randomization, in which violation of the left limit leads to the generation of a uniform random value in the left half of the interval. To decide whether the trial vector $\boldsymbol{V}_{i_g}$ should become a member of next generation, the greedy criterion is applied. If $\boldsymbol{V}_{i_g}$ gives a higher correlation than $\boldsymbol{Z}_{i_g}$ then $\boldsymbol{Z}_{i_{g+1}}$ is set to $\boldsymbol{V}_{i_g}$, otherwise the old vector $\boldsymbol{Z}_{i_g}$ is retained. The search continues until the best individual $\boldsymbol{Z}_{g\_best_{i_g}}$ converges. Analogous to the performance of simulated annealing, which depends on the control parameters that determine the annealing rate, the performance of the DE optimization depends on the scaling factor F, the weighting for balancing between intensification and diversification W, and the neighborhood size r. The control parameters are considered as parameters to be optimized (Brest et al., 2006; Das et al., 2009). Each population member is equipped with its own control parameters, expanding the dimensionality of the problem

$$p_{g,h} = \left\{ \left\{ \boldsymbol{Z}_{i_{g,h}}, F_{i_{g,h}}, W_{i_{g,h}} \right\}_{i_{g,h}=1}^{P} \right\}_{h=1}^{S}$$

W is initialized randomly $\in (0,1)$ and evolved similarly to the global donor vector

$$W_{i_{g+1}} = W_{i_g} + F_{i_g} \cdot \left(W_{g\_best_{i_g}} - W_{i_g}\right) + F_{i_g} \cdot \left(W_{\gamma_g} + W_{\delta_g} - W_{\varepsilon_g} - W_{\zeta_g}\right)$$

$F_{i_g} > 0$ is the self-adapting scaling factor and $\gamma \neq \delta \neq \varepsilon \neq \zeta \neq i \in [1, P]$ are the same random population indices that are used for generation of the global donor vector. The value of $W_{i_{g+1}}$ is restricted to the range [0.5,0.95] as follows

$$W_{i_{g+1}} = \begin{cases} 0.95 & if \quad W_{i_{g+1}} > 0.95 \\ 0.05 & if \quad W_{i_{g+1}} < 0.05 \end{cases}$$

The weight factor associated with a solution vector is changed only once in each generation. Furthermore, the mutation scheme is updated to include perturbation of the scaling factor according to

$$F_{i_{g+1}} = \begin{cases} 0.1 + 0.9r_1 & if \quad r_2 < 0.1 \\ F_{i_g} & if \quad r_2 \geq 0.1 \end{cases}$$

where r_1 and r_2 are uniform random numbers $\in [0,1]$. The new control parameter values are obtained before the mutation, which make them influence the trial vector generation and the selection. Only control parameter values that lead to better solutions influence the population by generation of competitive solutions. DE parameters that need to be externally defined are the neighborhood size, the population size, and the maximum number of generations to evolve. About ten times the number of degrees of freedom to be searched is a reasonable population size (Storn and Price, 1997). Self-adaptation of the control parameters adds an extra degree of freedom to the problem (Brest et al., 2006). The maximum number of generations should be selected to reflect the time required for convergence, and it puts a time-bound to the calculations. The maximum number of generations is hardcoded to 200, and the population size is set to 70.

Developer's corner

SIMPLE is written in modern Fortran—a language superior to C or C++ for expressing mathematical concepts. Modern Fortran has all the syntax elements required for encapsulation, inheritance, and polymorphism. A few object-oriented ideas are central in the SIMPLE design:

(1) code likely to change is separated from code expected to be static
(2) inheritance by composition is preferred over "classical" inheritance
(3) sub-problems are delegated to their own classes.

Instead of creating classes that inherit most of their talents from a superclass, the SIMPLE library assembles functionality from the bottom and up by equipping objects with references to other objects that supply the required functionality. In Fortran 95, inheritance by composition was the only alternative, whereas the 2003 standard provides the possibility to use inheritance by extending the type. We have selected to stay with inheritance by composition—a powerful alternative to inheritance by extension—that avoids strong entanglement between superclasses

and their subclasses. SIMPLE contains classes whose organization is driven primarily by I/O of data or desired functionality. I/O driven classes include:

simple_aligndata.f90	database of alignment information, Euler angles, state assignments etc., "pretty printing" functionality
simple_cmdline.f90	command line parser
simple_params.f90	for providing a global access point to all SIMPLE constants
simple_spidoc.f90	for I/O related to Spider document files
simple_stkspi.f90	for I/O related to Spider image stack files
simple_volspi.f90	for I/O related to Spider volume files

Interpolation of common lines, calculation of common line correlations, and reference-free common lines-based orientation search is executed by classes:

simple_comlin.f90	common line interpolation functionality
simple_comlin_corr.f90	common line correlation functionality
simple_rfree_search.90	high-level search routines

where the search class contains high-level routines for reference-free alignment in a discrete angular space (RAD), state assignment algorithms, and continuous orientation refinement. The **comlin_search** and **comlin_corr** singleton classes are organized so that the **comlin_search** using unit only needs to care about the search itself and the calculations of the various forms of correlations are hidden by the implementation. The construction also provides **comlin_corr** methods that can associate or nullify pointers to variables that will affect the functionality of the object. In object-oriented design, this construction is called the *decoration pattern*, because it decorates additional functionality to an object (Gamma et al., 1995). For example, an optimizer that is sampling state labeling solutions for aligned projections in order to resolve the different conformational states needs to calculate correlation coefficient over common lines between projections with the same state *only*. With the SIMPLE decoration construct, a pointer is associated to the integer array holding the state labeling solution by calling a **comlin_corr** method. This generates the desired functionality. To remove the decorated state-labeling functionality the pointer is nullified. The **rfree_search** class depends heavily on the classes responsible for optimization **simple_sa_opt.f90** and **simple_de_opt.f90**. Interpolation of central sections from 3D Fourier volumes, calculation of Fourier plane correlations, and reference-based orientation search by spectrally self-adaptive projection matching is executed by classes:

simple_fplane.f90	Fourier plane I/O, shift, rotation, low-pass filtering, etc.
simple_fplanes_corr.f90	Fourier plane correlation functionality
simple_rbased_search.90	Section matching-based search routines

where the search class contains the high-level routines for reference-based alignment and refinement. Advanced alignment and heterogeneity analysis algorithms can be built using the SIMPLE library. Allocation of memory in high-

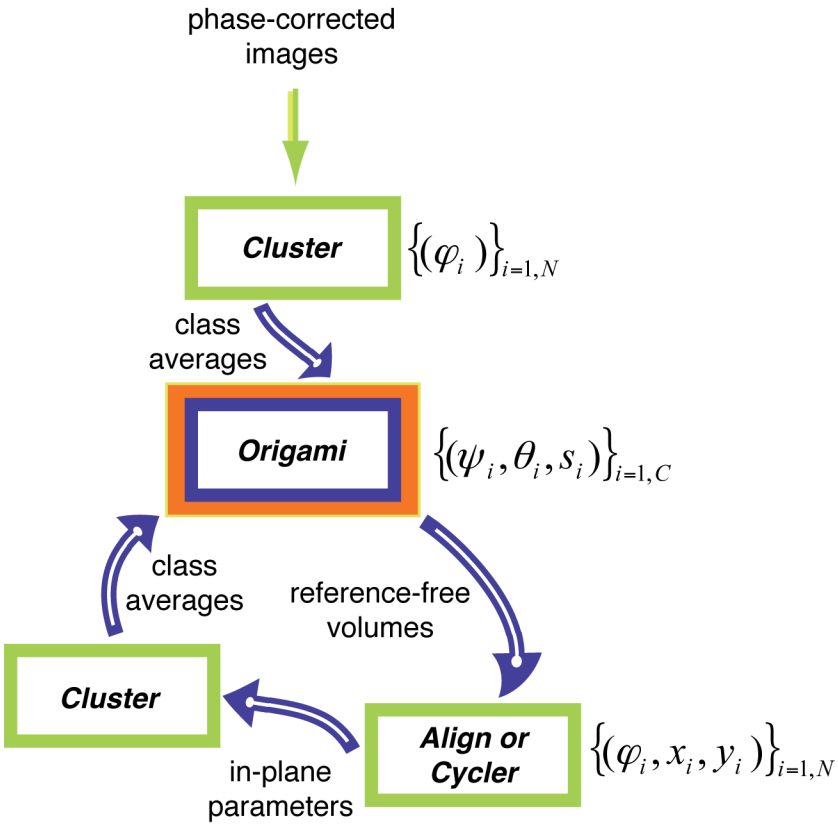
performance computing is time intensive. The physical memory of current generation computers is seldom limited. Therefore, code design that favors allocation of all memory immediately upon execution, avoiding excessive allocation and deallocation by subroutines and functions, is beneficial from a performance perspective. The complex object creation in SIMPLE applications relies on a builder. This refers to the builder pattern of object-oriented design (Gamma et al., 1995). The SIMPLE builder builds objects for calculating rotations, sorting, managing alignment data as well as complex arrays for 2D and 3D Fourier transforms. The SIMPLE applications are parameterized with respect to variables that are used for object creation, such as number of Fourier transforms, sampling distance, etc. All such input parameters are provided to the builder via the **simple_params** class. In addition, the builder and a few of the other classes are parameterized by a number of modes:

mode=02 unsupervised classification by program classify
mode=10 *ab initio* 3D analysis of class averages by origami
mode=20 multireference projection matching with fixed lowpass limit
mode=21 multireference projection matching with spectral self-adaptation
mode=23 for finding filtering tresholding/do spectral scoring
mode=30 reconstruction of volumes by Fourier gridding
mode=31 solvent flattening of a spider volume

Heterogeneity analysis

Two observations have guided the development of the SIMPLE heterogeneity analysis tool. First, accurate in-plane alignment is required for the clustering to resolve heterogeneity in 2D. Second, algorithms based on projection matching are not suitable for resolving heterogeneity in 3D because of the severe model bias (Elmlund, 2010). How do we design a method that provides high-quality in-plane parameters and overcomes the model bias? The two requirements appear to be at conflict, because high-quality in-plane parameters are difficult to obtain without applying 3D reference matching. The solution that SIMPLE provides is based on a division of the configuration space into two classes of parameters. The first class of parameters is composed of the in-plane degrees of freedom—the rotations and shifts in the plane of the projection. The second class of parameters consists of the state assignment parameters and the S^2 degrees of freedom—those responsible for movement of the normal vectors of the Fourier planes on the projection sphere. Projection matching refines the in-plane degrees of freedom. Reconstructions are generated and used as references for alignment in the usual manner, but these reconstructions only serve as abstract templates for obtaining refined in-plane parameters. The refined in-plane parameters obtained by programs **align** or **cycle** are inputted to program **cluster** to generate class averages with improved resolution. These class averages are subjected to *ab initio* alignment and heterogeneity analysis in program **origami,** which optimizes the spectral ensemble common line correlation over the S^2 degrees of freedom and state assignments. The crosstalk between the different SIMPLE components may be viewed as a method of

constraint propagation that reduces the search space of the common lines-based 3D analysis. Constraints are propagated from the template-based search via the in-plane degrees of freedom in order to force the clustering to resolve the heterogeneity in 2D. Next, constraints are propagated from the clustering to the 3D analysis by forbidding any two class averages from occupying the same projection direction in the discrete angular search. This constrained system reduces the complexity of the discrete 3D alignment with more than 100 orders of magnitude. Finally, a local search-based refinement scheme is applied that operates over all degrees of freedom in the neighborhood of the solution obtained so far. This final optimization is based on one-exchange greedy adaptive local search that steers a mixed continuous-discrete differential evolution optimizer operating over the six degrees of freedom of one Fourier plane. The below figure schematically illustrates the process. For details, see the open SIMPLE source code.



Ab initio **reconstruction and heterogeneity analysis via bijective orientation search.** Programs and flow of data, *N* is number of images, *C* is number of clusters. In the absence of any other in-plane alignment ***cluster*** uses a reference-free 2D alignment similar to (Penczek et al., 1992). The continuous, Fourier-based projection matching executed by ***align*** or ***cycler*** provides refined in-plane parameters. ***Align*** does projection matching given reference volumes; ***cycler*** combines projection matching, volume reconstruction, and solvent flattening of the volumes. Class averages are subjected to common lines-based assignment of projection directions and states in ***origami***. An approximate 3D alignment is obtained in a discrete angular space composed of projection directions *only*. The approximate solution is refined in an orientation continuum with differential evolution. Reference-free volumes are

Distributed execution

The module **cluster_exec.pm** in the SIMPLE **lib** folder is responsible for distributing **align** jobs in a Portable Batch System (PBS) environment. The executable **partition_master.pl** is used to distribute **align** jobs.

```
>>> partition_master.pl

run options:
     --refine
     --readme

>>> partition_master.pl --readme

Preparation of the .fim Fourier stack required for SIMPLE
refinement by program align is done by spi_to_fim.
Resolution boosting refers to mode=22-based alignment (see
the SIMPLE manual for details). The box size of the volumes
must be the same as the box size of the particle images
used to generate the Fourier stack. The first round should
be run with a fixed low-pass limit (30 A) to establish
initial orientations that are used to automatically
estimate the low-pass limit in the following round. The
resulting alignment documents need to be merged using
merge_algndocs.pl before reconstruction with program
reconstruct. Do not throw the merged alignment document. It
will be used to initiate the next round of refinement.
Happy Processing!

>>> partition_master.pl --refine

give to script:

1) spider reference volume(s) (multiple volumes separated
with commas, ex. vol1.spi,vol2.spi NO SPACES, '.spi'
extension required)
2) shift in pixels (ex. if given 3, then [-3,3] will be
searched)
3) point-group symmetry (cn/dn, where n=1..N)
4) stack with fourier transforms (ex. /PATH/fstack.fim,
where '.fim' is a SIMPLE format)
5) nr of transforms in the above stack
6) low-pass limit in Angstroms (note that this will now put
a lower resolution bound on the spectral self-adaption)
7) previous round's aligndata file or 0 if this is the
first round
8) use resolution boosting (y or n, note that boosting
should be used first after convergence has been achieved
without boosting)
```

```
9) nr of partitions
10) jobname
```

The **cluster_exec.pm** module is organized in functions responsible for calculating wall time, generating script headers, and generating script cores. These functions can be parameterized with respect to the system variable <SYSTEM> that is substituted by the **simple_config.pl** script so that the generated scripts fit the specifications of your system. Do not hesitate to contact me if you need help with modifying **cluster_exec.pm** for your system (hael@stanford.edu).

References

Bracewell, R.N. (1956). Strip integration in radio astronomy. Aust J Phys *9*, 198-217.

Brest, J., Greiner, S., Boskovic, B., Mernik, M., and Zumer, V. (2006). Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. Ieee Transactions on Evolutionary Computation *10*, 646-657.

Das, S., Abraham, A., Chakraborty, U.K., and Konar, A. (2009). Differential Evolution Using a Neighborhood-Based Mutation Operator. Ieee Transactions on Evolutionary Computation *13*, 526-553.

Das, S., and Suganthan, P.N. (2011). Differential Evolution: A Survey of the State-of-the-Art. Ieee Transactions on Evolutionary Computation *15*, 27-54.

Elmlund, D. (2010). Towards unbiased 3d reconstruction in single-particle cryo-electron microscopy. In The Royal Institutue of Technology (Palo Alto, The Royal Institutue of Technology), pp. 1-38.

Elmlund, D., Davis, R., and Elmlund, H. (2010). Ab Initio Structure Determination from Electron Microscopic Images of Single Molecules Coexisting in Different Functional States. Structure *18*, 777-786.

Elmlund, D., and Elmlund, H. (2009). High-resolution single-particle orientation refinement based on spectrally self-adapting common lines. Journal of Structural Biology *167*, 83-94.

Elmlund, H., Baraznenok, V., Linder, T., Szilagyi, Z., Rofougaran, R., Hofer, A., Hebert, H., Lindahl, M., and Gustafsson, C.M. (2009). Cryo-EM Reveals Promoter DNA Binding and Conformational Flexibility of the General Transcription Factor TFIID. Structure *17*, 1442-1452.

Elmlund, H., Lundqvist, J., Al-Karadaghi, S., Hansson, M., Hebert, H., and Lindahl, M. (2008). A new cryo-EM single-particle *ab initio* reconstruction method visualizes secondary structure elements in an ATP-fuelled AAA+ motor. J Mol Biol *375*, 934-947.

Frank, J., Radermacher, M., Penczek, P., Zhu, J., Li, Y.H., Ladjadj, M., and Leith, A. (1996). SPIDER and WEB: Processing and visualization of images in 3D electron microscopy and related fields. Journal of Structural Biology *116*, 190-199.

Frigo, M., and Johnson, S.G. (2005). The design and implementation of FFTW3. Proceedings of the IEEE *93*, 216-231.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). Design Patterns - Elements of Reusable Object-Oriented Software (Massachusetts, Addison-Wesley).

Hohn, M., Tang, G., Goodyear, G., Baldwin, P.R., Huang, Z., Penczek, P.A., Yang, C., Glaeser, R.M., Adams, P.D., and Ludtke, S.J. (2007). SPARX, a new environment for Cryo-EM image processing. Journal of Structural Biology *157*, 47-55.

Ludtke, S.J., Baldwin, P.R., and Chiu, W. (1999). EMAN: Semiautomated software for high-resolution single-particle reconstructions. Journal of Structural Biology *128*, 82-97.

Mindell, J.A., and Grigorieff, N. (2003). Accurate determination of local defocus and specimen tilt in electron microscopy. Journal of Structural Biology *142*, 334-347.

Murtagh, F. (1983). A surevy of recent advances in hierarchical clustering algorithms. Computer Journal *26*, 354-359.

Penczek, P., Radermacher, M., and Frank, J. (1992). 3-Dimensional Reconstruction of Single Particles Embedded in Ice. Ultramicroscopy *40*, 33-53.
Pettersen, E.F., Goddard, T.D., Huang, C.C., Couch, G.S., Greenblatt, D.M., Meng, E.C., and Ferrin, T.E. (2004). UCSF chimera - A visualization system for exploratory research and analysis. J Comput Chem *25*, 1605-1612.
Sorzano, C.O.S., Marabini, R., Velazquez-Muriel, J., Bilbao-Castro, J.R., Scheres, S.H.W., Carazo, J.M., and Pascual-Montano, A. (2004). XMIPP: a new generation of an open-source image processing package for electron microscopy. Journal of Structural Biology *148*, 194-204.
Storn, R., and Price, K. (1997). Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization *11*, 341-359.
Tipping, M.E., and Bishop, C.M. (1999). Probabilistic principal component analysis. Journal of the Royal Statistical Society Series B-Statistical Methodology *61*, 611-622.
van Heel, M. (1987). Angular Reconstitution - a Posteriori Assignment of Projection Directions for 3-D Reconstruction. Ultramicroscopy *21*, 111-123.