SAINT 2 Operation and Design

(Draft v4.0 – David Fisher, 29/03/2011).

This document describes the operation and design of SAINT 2 (Sequential Algorithm Initiated at the Nitrogen Terminus – version 2), a protein structure prediction program designed to model cotranslational and non-cotranslational folding.

The main sections are:

- Introduction – aims and limitations of SAINT 2.
- Operation – using SAINT 2.
- Design – how SAINT 2 works.
- Scoring – the scoring function at the heart of SAINT 2.
- Reflections and further work.
- Appendices.

1. Introduction

Aims of SAINT 2

SAINT 2 was designed as a successor to SAINT, which is described in the paper:

Directionality in protein fold prediction, Ellis et al, 2010

Both SAINT and SAINT 2 were created to model cotranslational protein folding.

SAINT was implemented as a set of scripts that called the Rosetta protein structure prediction program to do the actual work. The aim of SAINT 2 was to develop a program that was more flexible and less opaque in its operations than SAINT.

SAINT 2 consists of a program written in C++ (`saint2`) and a set of supporting scripts. It does not depend on any third party software, other than `TMscore` for comparing structures (used during the clustering stage) and `scwrl4` for inserting side chains (after the final structure is selected).

Overview

The main input to SAINT 2 is an amino acid sequence, and the output is a set of predicted structures for the sequence.

Like Rosetta, SAINT 2 uses fragment replacement to generate candidate structures. Each move, a random position is chosen, and then a random fragment at that position (fragments are usually nine residues long). The current torsion and bond angles are then replaced with the angles from the fragment.

Replacing a fragment creates a new potential structure. The score for the new structure is calculated and compared to the current structure's score, and then a test is performed to see if the new structure should replace the old one, depending on the change in score. The test used depends on the parameters specified by the user.

If cotranslational folding is being performed, the chain starts with only a few residues extruded; more residues are added after a certain number of moves, until the chain is "full grown", and then the remaining moves are performed.

After the run is completed, the program outputs the structure with the lowest score found.

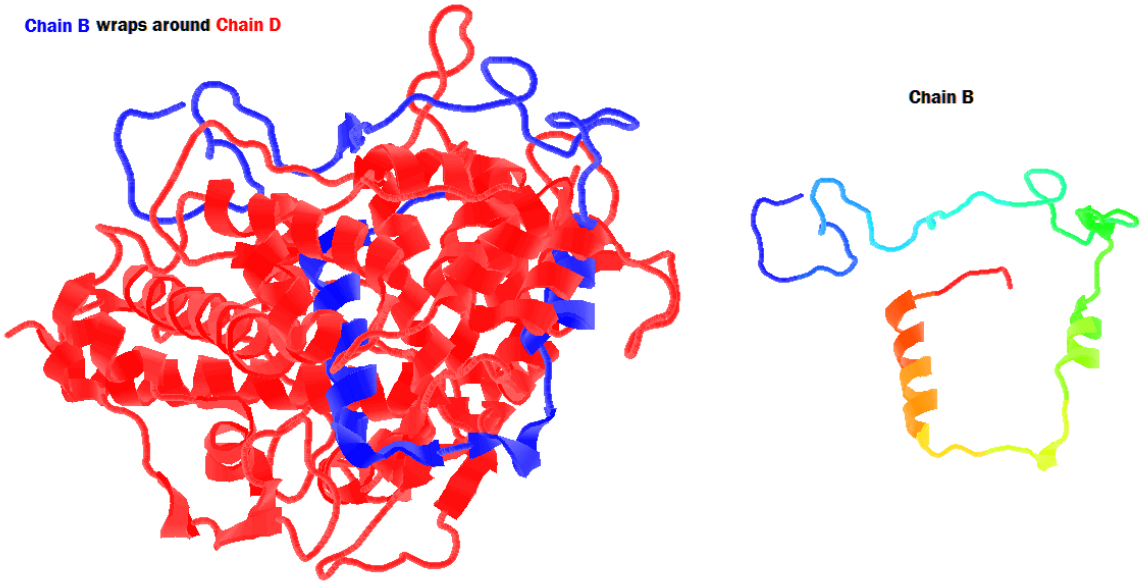
When all of the runs have been completed, the structures are clustered, and the most central structure in each cluster is output. The user must decide which structure is most likely to be the native one.

Since the structures generated by SAINT 2 contain only backbone atoms, a program such as `scwrl4` must be run on the final model to add side chain atoms.

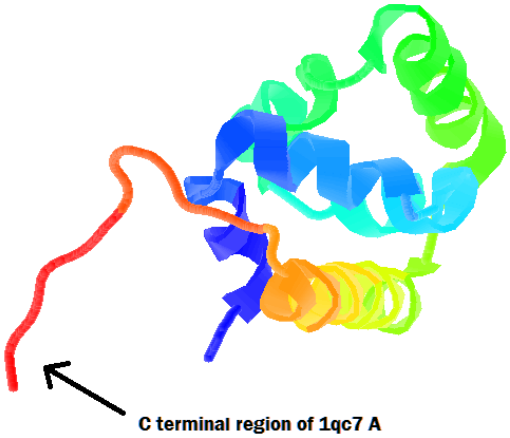
Target Structures

SAINT 2 was designed to fold a single, independent chain or domain. If the target structure is known, it is wise to verify by visual inspection that the chain really does fold independently.

An example of a chain which does not fold independently is chain B of protein 1d7w; it depends on the presence of the rest of the protein (chain D in particular) to obtain its shape:



Also note that the Protein Data Bank contains many chains which are just domains (sections of chains which fold relatively independently). Sometimes these contain extraneous segments at the beginning or end of the chain; for example, the C terminal region of 1qc7, chain A. The presence of such segments can lead to incorrect structure predictions.



2. Operation

Running SAINT 2

The simplest way to use SAINT 2 is with the `run_` scripts. The `saint2` program may also be called directly, but doing so may require manual editing of the configuration file; the `run_` scripts create configuration files automatically.

To fold in vitro (from a fully extended state):

`run_invitro` *runs id fasta library moves temperature*

eg. `run_invitro 20000 1a1xA 1a1xA.fasta lib_1a1xA 10000 2.5`

To fold cotranslationally (square brackets indicate optional parameters):

`run_cotrans` *runs id* `r|n` *fasta library growth_moves additional_moves temperature* [`fixed|linear|`(*codon_file tunnel_length*)] [`rev`]

eg. `run_cotrans 20000 1a1xA r 1a1xA.fasta lib_1a1xA 9000 1000 2.5 linear rev`

eg. `run_cotrans 20000 1a1xA r 1a1xA_codons.fasta lib_1a1xA 9000 1000 2.5 1a1xA_TAIprofile.csv 50`

The arguments are as follows:

runs is the total number of runs to perform (one structure is generated per run).

id is unique identifier for the target protein, eg. 1a1x or 1a1xA.

`r|n` (`r` or `n`) specifies whether or not to model the ribosome wall.

fasta is a file containing the amino acid sequence (when no *codon_file* is specified) or nucleotide sequence (when a *codon_file* is specified) in FASTA format. This can be extracted from a PDB file using the program `get_fasta`, or extracted from the codon (.csv) file using `csv_to_fasta`,. Lines starting with “>” are comments. There is one character per amino acid or three per codon (using the characters GATC). Newlines are ignored.

library is a library fragment file containing a set of fragments for each position in the target. See ***Fragment Libraries*** for a complete description.

moves is the total number of moves to perform during in vitro folding. Use the program `calc_invitro_moves` to find the appropriate number of moves to use for an in vitro run, given the configuration file for a cotranslational run (see ***Appendix B*** for an explanation).

growth_moves is the number of moves to perform during the growth phase of cotranslation.

additional_moves is the number of moves to perform after the peptide is full grown.

temperature is the Simulated Annealing temperature to use for accepting or rejecting moves.

`fixed` specifies that there are a fixed (constant) number of moves after each extrusion, ie. the value `growth_moves` is evenly divided between the extrusions. This is the default.

`linear` specifies that the number of moves after each extrusion is proportional to the number of extrusions so far (ie. it increases linearly). The total number of moves is equal to the value of `growth_moves`.

codon_file is a file containing the codon for each residue, along with the TAI concentration. The number of moves after each extrusion is proportional to the inverse of the TAI concentration, and sums to the value of `growth_moves`.

`rev` indicates the residues should be extruded in reverse order (C-terminus instead of N-terminus first).

Generated Configuration Files

When `run_cotrans` or `run_invitro` is called, a configuration file is automatically generated using another script called `create_config`. The name of the generated configuration file contains all of the parameter information (except for the number of runs); for example, the command:

```
run_invitro 20000 1a1xA 1a1xA.fasta lib_1a1xA 10000 2.5
```

creates a configuration file called

```
config_1a1xA_i_lib_1a1xA_10000_t2.5
```

The `_i_` stands for “in vitro”; cotranslational folding uses `_c_` instead.

If the configuration file exists already, it is not changed.

Outputs

The scripts `run_cotrans` and `run_invitro` both produce the same kind of output:

- The configuration file.
- A set of structures (one per run).
- The score for each structure (`res_scores`).
- The top 1000 scoring structures (directory `top`).
- The pairwise RMSDs between the these structures (`res_rmsds`).
- The clusters for these structures (`res_clusters`).
- The central structure in the central cluster (starting with `best_`), ie. the one with the smallest sum of squared RMSD values to all other structures in the cluster.

The configuration file is created in the current directory. The rest of the files are placed in a directory with a similar name to the configuration file (without the `config_` prefix); for example, `1a1xA_i_lib_1a1xA_10000_t2.5`.

A separate subdirectory is created for every group of 1000 runs. The directories are named "0", "1", etc. If such directories already exist, the next highest number is used. Note that this number is also used as the random number seed (to guarantee that the runs are all different). A separate process is submitted to perform the runs in each directory.

If more runs for the same target are required later, call `run_cotrans` or `run_invitro` in the same directory as before with the same arguments (but with any number of runs). The original arguments are included at the start of the configuration file.

Each structure has a similar name to the configuration file (without the `config_` prefix), followed by `_s`*(seed)*`_`*(num)*, where *seed* is the random number seed (the name of the subdirectory) and *num* is a number from 000 to 999; for example, `1a1xA_i_lib_1a1xA_10000_t2.5_s0_002`.

A subdirectory called `top` is also created, containing links to the top 1000 scoring structures.

An example file hierarchy is shown below.

```
-- runs/
    |
    +-- config_1a1xA_i_lib_1a1xA_10000_t2.5
    |
    +-- 1a1xA_i_lib_1a1xA_10000_t2.5/
         |
         +-- 0/
         |   |
         |   +-- 1a1xA_i_lib_1a1xA_10000_t2.5_s0_000
         |   |
         |   +-- 1a1xA_i_lib_1a1xA_10000_t2.5_s0_001
         |   ...
         |
         +-- 1/
         |   |
         |   +-- 1a1xA_i_lib_1a1xA_10000_t2.5_s1_000
         |   ...
         ...
         |
         +-- best_1a1xA_i_lib_1a1xA_10000_t2.5_s3_247      ← the single best structure
         |
         +-- res_clusters
         |
         +-- res_rmsds
         |
         +-- res_scores
         |
         +-- top/
              |
              +-- 1a1xA_i_lib_1a1xA_10000_t2.5_s1_125
              |
              ...
```

If it doesn't work ...

If no "best_*" file is created, there was an error somewhere. Any messages generated by saint2 can be found in 0/err_0, 1/err_1, and so on. The simplest way to view these is by saying (for the previous example):

```
cat 1a1xA_i_lib_1a1xA_10000_t2.5/*/err_*
```

What to do with the output

The best_ file contains the best structure found. If the native structure is known, the first thing to do is to run TMscore to compare the best structure to the native. Note that TMscore requires exactly the same residue numbering in the two files to work properly; to renumber a PDB file, use the program write_pdb.

Another thing to check is the size of the clusters in the res_clusters file (see ***Output File Formats***). If the size of the first few clusters is too small (eg. size 1), redo the clustering with a higher threshold than the default (printed at the top of the res_clusters file), eg.:

```
cluster res_rmsds 10 > clusters_10
```

This should be fast to run (about one second).

The output file gives the central structure in the central cluster, as well as the centre of each cluster. The central structure in "cluster 0" (which is the largest cluster) is an alternative "best" structure.

Side chains can be added to the final structure using SCWRL. There is an installation package for SCWRL 4 in the SAINT 2 bin directory.

Scripts and Parameters

After saint2 generates all of the output structures, run_cotrans or (run_invitro) calls a script called cluster_results. This script uses the following parameters:

- TOP is the number of structures to cluster (1000).
- CUTOFF is the threshold used for clustering (6.0).

It calls the following programs:

- score_pdb to score each structure (creating the file res_scores).
    - This calls saint2 to do the scoring, using the configuration file $SAINT2/data/config_for_scoring.
- get_pairwise_rmsds to find the RMSDs (creating res_rmsds).
    - This calls rmsd, which calls TMscore to find the RMSD values.
- cluster to do the actual clustering (creating res_clusters).

Output File Formats

`res_scores`

The output file `res_scores` contains one line per structure, containing the `saint2` score, a space, then the filename. The scores are sorted from lowest (best) to highest (worst).

`res_rmsds`

The output file `res_rmsds` starts with a single line containing the number of structures (eg. 1000), a followed by a blank line, then one line per structure of the form *number=filename*. This assigns a unique number to each structure (starting from 0).

This is followed by a blank line, and then one line for each pair of structures, of the form *num1 num2 rmsd*, where num1 and num2 are the structure numbers.

For example:

```
1000

0=top/1a1x_c_n_lib_1a1xA_10_2_t2.5_fixed_s0_023
1=top/1a1x_c_n_lib_1a1xA_10_2_t2.5_fixed_s0_068
2=top/1a1x_c_n_lib_1a1xA_10_2_t2.5_fixed_s0_081
...

0 1 22.462
0 2 19.909
0 3 17.909
...
998 999 20.891
```

`res_clusters`

The output file `res_clusters` starts with a line like:

```
235 clusters, largest = 120 (cut off = 6)
```

This indicates the number of clusters (235), the size of the largest cluster (120) and the cut off value used (6.0).

A blank line follows, then a line like:

```
central cluster 3 top/1a1xA_i_lib_1a1xA_10000_t2.5_s1_024
```

The “central cluster” is the one whose central structure (see below) has the smallest sum of squared RMSD values to the central structures of all of the other clusters.

This is followed by a blank line, then one line for each cluster (in descending order of size). The lines look like:

```
cluster 0 size 120 centre top/1a1xA_i_lib_1a1xA_10000_t2.5_s3_280
cluster 1 size 103 ...
```

The format is: cluster *id* size *size* centre *filename*. The filename indicates the most central structure in the cluster (the one with the smallest sum of squared RMSD values to all other structures in the cluster).

This is followed by a blank line, then one line for each structure, giving its cluster id. The lines look like:

```
c0 top/1a1x_c_n_lib_1a1xA_10_2_t2.5_fixed_s6_371
c0 ...
...
c1 ...
```

Calling `saint2` Directly

The `run_` scripts described in the section ***Running SAINT 2*** use a program called `saint2` to do the actual work. This program may be used directly to perform structure prediction, or to get information about a single protein (such as its score).

For a complete list of command line options, run `saint2` without any arguments.

Structure Prediction

To run `saint2` normally, say:

`saint2` *configuration_file* `[ -n` *number_of_runs* `] [ -o` *output_filename* `]`

Configuration files are described below. Arguments in square brackets are optional; by default, the number of runs is one, and the output is sent to `stdout` (along with debug

output). If there is more than one run, the suffix _000, _001, etc. is appended to the output filename.

To repeat a previous run exactly, use the option `-s` *number* to specify the random number seed to use. The value of the seed is printed near the beginning of `saint2`'s output.

If the `-s` option is not present on the command line, the random number seed is based on the current time. This means that SAINT 2 should not be called several times simultaneously with exactly the same arguments, or the results will be identical.

Scoring a Single Chain

The `saint2` program may also be used to score a single protein chain (which is what the script `score_pdb` does):

`saint2` *configuration_file* `--` *pdb_file* `[` *chain* `] [ -t ] [ -a ]`

If a chain is not specified, the first chain in the PDB file is used.

The configuration file is required because it contains the scoring parameters (the weights to use and the location of the score data files). Since these are the only parameters used, any existing configuration file is suitable.

Without the `-t` option, this command prints the score for the chain (one component at a time, followed by the total), along with some other information about the chain such as the amino acid sequence and the radius of gyration.

With the `-t` option, the torsion and bond angles for every residue are printed instead.

The `-a` option means that all atoms will be used for scoring instead of just backbone atoms (which affects the RAPDF score).

Configuration Files

The configuration file contains all of the parameters SAINT 2 needs to run. To generate an "empty" configuration file, say:

```
saint2 -c
```

The result will look something like the following. The # character indicates a comment; the rest of the line is ignored.

```
[Sequence]

type = amino            # "amino" or "nucleotide"
file = ...              # filename (FASTA format)
#seq = ...              # sequence
#index = 0              # index of sequence in FASTA file (starting from 0)
#id = ...               # id of sequence in FASTA file
```

The `type` parameter indicates whether the sequence contains amino acids (one character per residue) or nucleotides (three per residue). The value of the `type` parameter is either "`amino`" or "`nucleotide`".

The sequence itself may either be in a separate file (the `file` parameter) or in this file (the `seq` parameter); note that a long line may continued by starting the next line with "+". If the FASTA file contains multiple sequences, the `index` or `id` parameter may be used to specify the sequence.

```
[General]

sequential = true       # whether is sequential or non-sequential
moves = 10000           # move limit (excluding growth phase)
#none selected = 0      # stop after none selected this many times in a row
reverse = false         # whether to extrude in reverse order (C first)
#start_structure = ...  # Starting structure (PDB file)
#native_structure = ... # Native structure (PDB file)
```

`Sequential` indicates whether to fold cotranslationally or in vitro. The value may be "`true`", "`false`", "`t`", "`f`", "`yes`", "`no`", "`y`" or "`n`" (upper or lower case).
`Moves` is either the total number of moves to perform (in vitro) or the number of moves to do after the peptide is fill grown (for cotranslation).
If `none_selected` is not zero, it specifies the maximum number of moves in a row with no new structure being selected. If this limit is reached, the run ends (as if the move limit specified by `moves` had been reached).
`Reverse` indicates whether extrusion should be in the reverse direction to normal (C-terminus instead of N-terminus first).
If a `start_structure` (PDB file) is specified, it is used as the initial structure (using

ideal bond lengths). This only makes sense for in vitro folding. (Note that a particular chain can be extracted from a PDB file with the `write_pdb` *program).*
If the `native_structure` *(PDB file) is specified, it is used in debugging.*

```
[Extension]

initial = 9                     # initial number of residues extruded
extrude = 1                     # number of residues to extrude at a time
growth_moves = 10000            # total number of moves during growth

type = fixed
move_distribution = fixed       # distribution of max moves between
                                # extrusions ("fixed" or "linear")

#type = codon
#codonfile = ...                # TAI profile csv file
#tunnel_length = 0              # number of residues in tunnel (delay)
```

This section only applies to cotranslational folding.

`Initial` *is the (minimum) initial number of residues to extrude.*
`Extrude` *is the number of residues to add when an extrusion happens.*
`Growth_moves` *is the total number of moves to perform during the "growth" phase.*

`Type` *is either "*`fixed`*" or "*`codon`*". Each type has its own associated set of parameters.*

For type "fixed":
`Move_distribution` *determines how the* `growth_moves` *are divided in between the extrusions. The value "*`fixed`*" means the moves are divided evenly; "*`linear`*" means that the number of moves between extrusions is proportional to the number of extrusions so far.*

For type "codon":
`Codonfile` *specifies a file containing tRNA adaptation index for each codon.*
`Tunnel_length` *specifies the number of residues between the most recently extruded residue and the corresponding codon (see* ***Cotranslation and Extrusion****).*

```
[Scoring]

type = combined

long_solvation_file = ...
long_orientation_file = ...
long_rapdf_file = ...
long_torsion_file = ...

short_solvation_file = ...
short_orientation_file = ...
short_rapdf_file = ...
```

```
short_torsion_file = ...

long_weight_solvation = 1.0
long_weight_orientation = 1.0
long_weight_lj = 1.0
long_weight_rapdf = 1.0
long_weight_hbond = 1.0
long_weight_torsion = 1.0
weight_ribosome = 1.0

short_weight_solvation = 1.0
short_weight_orientation = 1.0
short_weight_lj = 1.0
short_weight_rapdf = 1.0
short_weight_hbond = 1.0
short_weight_torsion = 1.0
```

The `type` is always "`combined`".
The prefixes "`short`" and "`long`" refer to chains that are less than or equal to 150 residues, or longer than 150 residues respectively.
The `file`s contain pre-calculated data for each scoring term.
The `weight` values are the relative weights to give each scoring term when they are combined (see ***Weight Determination*** *for the actual weights used).*

```
[Strategy]

#type = strict
#number = 1

type = monte
temperature = 1.0
number = 1

#type = boltz
#temperature = 1.0
#number = 10
```

The strategy type determines how many candidate structures are generated (the "`number`" parameter) and the method used to accept or reject each structure.
Type "`strict`" means that only better scoring structures are accepted (downhill search).
Type "`monte`" means Monte Carlo selection using the Metropolos criterion.
Type "`boltz`" means that the probabilities are based on the Boltzmann distribution.

```
[Movement]

type = fragment
lib = ...                       # fragment library location
```

`Type` specifies how a "move" is performed ("`fragment`" means fragment replacement).
`Lib` is the name of the fragment library.

Fragment Libraries

There are several different fragment library formats available: *Explicit* format, *Rosetta* format and *SAINT 2* format. SAINT 2 can only use *SAINT 2* format fragment libraries.

- *Explicit* format contains `ATOM` records extracted from the original PDB files.
- *Rosetta* format is used by the structure prediction program `rosetta`. It contains references to the original PDB files and the torsion angles, but no bond angles.
- *SAINT 2* format contains the torsion and bond angles for each residue.

The `convert_fragments` program converts an *explicit* format fragment library to *SAINT 2* format. For example:

```
convert_fragments 1jo8A_complete_pdb_library > lib_1jo8A
```

Similarly, the `convert_rosetta_fragments` program converts from *Rosetta* to *SAINT 2* format. The fragments are taken from the database in the HPCC directory:

```
/home/jellis/data/pdb/data/structures/divided/pdb
```

... and so if there are references in the *Rosetta* library file to PDB ids that are not present in this database, the corresponding fragments will be excluded from the *SAINT 2* library.

Codon Files

Codon files contain one line per residue in the chain, with three comma separated values: residue number (starting from 1), codon (three letter code), and tRNA adaptation index (0 < TAI <= 1; the number of moves is proportional to the inverse of this value).

Score Data Files

The data files for the RAPDF, Solvation, Orientation and Torsion scores were generated using a program called `mkd` ("make data"). The data is based on a set of proteins obtained from the PISCES server (see ***The Scoring Function***).

Supporting Programs

The scripts and programs provided in the `saint2/bin` directory are as follows (executables are followed by a "*", and scripts are not):

`average`	Print the average of all values in the first field in the input.
`calc`	Evaluate a mathematical expression (improved `expr`).
`calc_invitro_moves*`	Given a cotranslational configuration file, calculate the equivalent number of in vitro moves (see ***Appendix B***). `[src/timing/calc_invitro_moves.cpp]`
`cluster*`	Hierarchical clustering program used by the `cluster_results` script. `[src/clustering/cluster.cpp]`
`cluster_results`	Perform final structure clustering (see ***Outputs*** => ***Scripts and Parameters***)
`convert_fragments*`	Convert a fragment library from *Explicit* to *SAINT 2* format (see ***Fragment Libraries***). `[src/mkdata/convert_fragments.cpp]`
`correlation`	Find the correlation coefficient for the first two fields in the input.
`count`	Count the number of occurrences of each line in the input.
`create_config`	Used by `run_cotrans` and `run_invitro` to create a SAINT 2 configuration file.
`csv_to_fasta`	Extract the codon sequence from a codon file (comma separated, with codons in the second field) and output it in FASTA format.
`do_cmd_outfile`	Execute a command, and send the output to a particular file (like running "`cmd > filename`").
`field`	Print the nth field in the input.
`filter_len_11*`	Given a list of fragment source PDB files and positions, filter out the ones that are not length 11 (ie. length 9 plus the residue before and after the fragment) unless they come at the start or end of the sequence (in the target as well as the source PDB). `[src/mkdata/filter_len_11.cpp]`
`fullpath`	Print the full pathname of a file or directory.
`get_fasta*`	Extract the amino acid sequence from a PDB file, and output it in FASTA format. `[src/mkdata/get_fasta.cpp]`

`gdt_ts`	Print the GDT_TS value between two PDB files (uses `TMscore`).
`get_pairwise_rmsds`	Get the RMSD between all pairs in a set of PDB files.
`get_res_scores_gdt_rmsd`	Given a `res_scores` file (created by `run_cotrans` or `run_invitro`) and the native structure, find the GDT_TS and RMSD to the native for each structure in the file.
`install_Scwrl4_Linux*`	Installs the `scwrl4` program (for adding side chains to a structure).
`oneline`	Put the input all on a single line.
`rmsd`	Print the RMSD value between two PDB files (uses `TMscore`).
`run_cotrans`	Top level SAINT 2 script (see ***Running SAINT 2***).
`run_invitro`	Top level SAINT 2 script (see ***Running SAINT 2***).
`run_then_finish`	Run a command, then create a file called “finished” when the command has complete.
`saint2*`	The main SAINT 2 program. `[src/main/main.cpp]`
`score_pdb`	Print the SAINT 2 score for a structure.
`TMscore*`	Compare two structures and print things like GDT_TS and RMSD. Note that the structures must have exactly the same numbering (use `write_pdb` to do this).
`val`	Print a list of integers.
`wait_for_file`	Wait until a file exists, then run a command.
`write_pdb*`	Write out a single chain from a PDB file. `[src/write/main.cpp]`

3. Design

Overall Program Flow

The high level operation of SAINT 2 is as follows.

1. Parse the command line arguments.
   This determines the number of runs to perform.

2. Read the configuration file.
   This determines the type of folding (cotranslational or in vitro) and the other parameters.

3. Read the fragment library and score data files specified in the configuration file.

4. Perform the number of runs specified on the command line.
   For each run:

   4.1. Initialise the structure:

      4.1.1. For cotranslational folding, a starting fragment is selected at random. If the fragment is shorter than the initial length specified in the configuration file, "extrusion" is performed repeatedly until is the starting length is reached.

      4.1.2. For in vitro folding, the chain is initialised to a fully extended state (all torsion angles are set to 180 degrees, and all bond lengths and angles are set to ideal values).

   4.2. Perform the number of moves specified in the configuration file.
      For each move:

      4.2.1. For cotranslational folding, if the protein chain is not yet full grown, determine whether it is time for the next extrusion (based on the number of moves performed so far). If so, extrude the number of residues specified in the configuration file (i. e. increase the current length of the chain, and insert a random fragment ending at the new end position).

      4.2.2. Select a residue position P at random. All valid positions have an equal probability of being selected. A valid position is one where there is at least one fragment in the fragment library which starts at this position and does not end beyond the current end of the chain (for in vitro folding, the "current end" is the C terminal residue; for cotranslational folding, the "current end" is the number of residues extruded so far).

4.2.3. Select a random valid fragment from the fragment library which starts at position P. A valid fragment is one which does not go beyond the "current end" as described above. Probabilities are weighted by the score specified in the fragment library (a fragment with score X is half as likely to be selected as a fragment with a score of 2X).

4.2.4. Generate a candidate structure by performing the fragment replacement. If the configuration file specifies that multiple candidates should be generated, repeat the above two steps to create each candidate.

4.2.5. Determine the score for each candidate structure. The score is a weighted combination of the RAPDF, Solvation, Lennard-Jones, Hydrogen Bonding, Orientation and Torsion scores; weights are specified in the configuration file. A score for the ribosome wall may also be specified (only scored during the cotranslational "growth" phase).

4.2.6. Determine whether to accept one of the candidate structures, using the method specified in the configuration file. For example, the "Boltzmann" method selects one (or possibly none) of the structures based on the relative score change, with a higher probability of accepting more negative changes (since low scores are more desirable).

4.2.7. If one of the candidate structures was accepted, replace the current structure with that structure.

4.3. After all moves have been performed, output the full grown structure with the best (lowest) score found during the run.

5. If more than a thousand structures have been generated, find the best scoring one thousand structures. Cluster the resulting structures and find the most central one in each cluster. Out of these, choose one of the best scoring structures (requires subjective judgement).

6. Run `scwrl4` on the final structure to add side chain atoms.

Design Details

The most important aspects of the design are described in the sections below. The SAINT 2 scoring function has a major section of its own in another part of this document.

Fragment Libraries

Each target requires its own individual fragment library. Fragments may vary in length, but it is common for every fragment in a library to be nine residues long.

Each fragment contains six angles per residue: the three torsion angles (phi, psi and omega), and the bond angles between the main chain atoms (N, CA and C).

This is sufficient to recreate the structure of a protein with high accuracy (a GDT_TS score of 1.0), if ideal bond lengths are also used. However, if the fragment bond angles are replaced with ideal bond angles, then structures cannot be accurately recreated, and steric clashes often occur. (See ***Appendix A*** for ideal bond length and angle values).

Each fragment also has a score, which determines its probability of being selected (see ***Fragment Selection***). Fragment scores in the libraries generated by S. Srivastava are based on the similarity of the fragment's secondary structure to the predicted secondary structure.

Internal Representation

A protein chain is represented by SAINT 2 using its backbone atom positions only (C, N, CA, CB and O) using 3D Cartesian coordinates. It would be theoretically possible to call an external program such as `scwrl4` to calculate the side chain atom positions after each move, but the time cost would be prohibitively expensive, and writing a local library to perform side chain optimisation was outside the scope of the SAINT 2 project.

Since the fragment library contains only torsion and bond angles for the main chain atoms (C, N and CA), the CB and O positions are calculated using ideal bond angles. (The four bonds around a CA atom form a tetrahedral shape, making it straightforward to calculate the position of the CB atom; and the C, CA, N and O backbone atoms lie on approximately the same plane, enabling the O position to be calculated to sufficient accuracy as well).

The torsion angles (phi, psi and omega) are also stored for each residue. Phi is undefined for the N terminal residue, and psi and omega are undefined for the C terminal residue.

Fragment Replacement

When a fragment is replaced, the torsion and bond angles in the current structure are replaced with those in the fragment. Ideal bond lengths are always used (see ***Appendix A***).

Torsion angles involve four atoms. SAINT 2 starts from C-terminal-most end of the fragment and works toward the N-end, determining each atom position from the fragment torsion and bond angles and the positions of the last three atoms.

After a fragment is replaced, the structure beyond the N-end of the fragment is rotated in such a way that all existing bond and torsion angles retain their current values.

Fragment Selection

A "move" in SAINT 2 consists of selecting a fragment at random and trying it out (performing the fragment replacement and accepting or rejecting the new structure based on the change in score).

All positions along the chain where a fragment replacement could occur have an equal probability of selection. For each position, the fragment library contains a set of possible fragments. Each fragment has a (positive) score; for the current version of SAINT 2, the score is based on the secondary structure similarity between the fragment and the predicted secondary structure.

The probability of selecting a particular fragment is proportional to its score (for example, a fragment with a score of 60 is twice as likely to be selected as a fragment with a score of 30).

Cotranslation and Extrusion

For cotranslational folding, the user may specify the following values:

- E1, the initial number of residues extruded (default 9).
- E2, the number of residues to extrude at a time (default 1).
- GM, the total number of moves to perform during the growth phase (ie. before the final extrusion happens).
- MD, the move distribution between extrusions:
    - Constant (GM divided by the number of extrusions).
    - Linear (a constantly increasing amount starting from one, adding up to GM).
    - Codon based (see below).
- FM, the number of moves to perform after the peptide is full grown.

If the move distribution (MD) is codon based, then SAINT 2 requires the tRNA Adaptation Index for each codon (an organism specific value between zero and one). Since low tAI values are associated with "slow" codons, the number of moves corresponding to a particular codon is proportional to the inverse of its tAI value.

Because of the existence of the ribosome tunnel, it is difficult to decide which codon should be used for which extrusion. If there was no tunnel at all, the codon for the most recently extruded residue could be used to decide on the number of moves to perform. However, around 50-70 residues may be present in the tunnel, which is a very restrictive area for folding; alpha helices can form inside the tunnel, but not multi stranded beta sheets, and certainly not the native structure (unless it is just one long helix).

To allow for the tunnel, the user may specify a "delay" between the codon and the most recently extruded ribosome. For example, with a delay of 70, the number of moves to perform after residue 30 is extruded would be determined by the codon for residue 100 (since 70 of the residues are still in the tunnel, leaving only the last 30 free to move). Note that even this approach has issues, since the actual number of residues in the tunnel is unknown (and variable, even within a single run).

Residues within the delay length of the C-terminus are given a fixed number of moves equal to the average of the other moves (since they have no corresponding codon).

Clustering the Final Structures

After a large number of candidate structures have been generated, they may be clustered into groups of similar structures, and then a final structure chosen from one of these groups.

The clustering script uses hierarchical clustering with average linkage, using RMSD as the distance measure (calculated using the `TMscore` program). Since all pairs of structures must be compared, the time cost is proportional to the square of the number of structures, so the total number of structures should be first reduced to around 1000 (the best scoring ones).

The output of the script is the most "central" structure for each cluster (having the lowest total squared RMSD to all of the other structures in the cluster).

Subjective judgement must be used to decide which of these structures to use.

The `scwrl4` program may be used to add side chain atoms to the final structure.

Reverse SAINT 2

In order to investigate the importance of directionality in cotranslational protein folding, SAINT 2 may be run in "reverse" mode. In this mode, residues are extruded from the C-terminus instead of the N-terminus; in all other aspects, the behaviour is unchanged.

4. Scoring

Why Develop a New Scoring Function?

The SAINT 2 scoring function was created specifically for this project, even though there are several other freely available scoring functions for protein structures. Some limitations of these functions are:

- Force Fields such as CHARMM and AMBER are designed for use with molecular dynamics, and require all atoms to be present; SAINT 2 only stores backbone atoms.
- Rosettta's scoring function is opaque (not described in sufficient detail in the publications, and not very accessible in the current source code).
- Other knowledge based scoring functions such as Victor/FRST (described in the following section) tend to be designed to evaluate near-native structures rather than the kinds of structures SAINT 2 would encounter during the process of fragment replacement, which are often highly uncompact. SAINT 2 required a scoring function that would be able to handle any kind of structure as input.

The SAINT 2 scoring function also includes a component specifically designed for cotranslational folding (the Ribosome Wall score).

The Scoring Function

SAINT 2 uses a knowledge based scoring function inspired by Victor/FRST:

The victor/FRST function for model quality estimation, Silvio C.E. Tosatto, 2005

... which uses a weighted combination of RAPDF, Solvation, Torsion and Hydrogen Bonding potentials.

In addition to these, SAINT 2 uses a Lennard-Jones and an Orientation term, as well as an optional Ribosome Wall score.

Low values are "good" for all of these score types, since the overall score is intended to mimic an energy function, and the native conformation is assumed to be the one with the lowest energy.

For the RAPDF, Solvation, Torsion and Orientation scores, the score data was derived from the PISCES data set:

```
cullpdb_pc30_res2.0_R0.25_d100213_chains5488
```

obtained from the PISCES web site (`dunbrack.fccc.edu`) on May 4th, 2010.

Some chains were found to contain missing or invalid data, and so the total number used was 5425 out of the original 5488.

RAPDF Score

"RAPDF" stands for Residue-specific All-atom conditional Probability Discriminatory Function. It is described in the paper:

An all-atom distance-dependent conditional probability discriminatory function for protein structure prediction, Samudrala and Moult, 1998

RAPDF was originally designed for protein structures whose non-hydrogen atom positions are all known, including side chain atoms. SAINT 2 only stores the positions of the five backbone atoms (N, CA, C, O and CB), but using the RAPDF function with just these atoms still seems to perform well.

The RAPDF approach assigns every atom in each type of amino acid a separate "atom type", resulting in 167 distinct types (for example, "Lysine alpha carbon" or "Asparagine main chain oxygen"). SAINT 2 uses 99 of these atom types (5 for each amino acid except for Glycine, which has no beta carbon).

The RAPDF function is based on the theory that each pair of atom types has a particular propensity to be a certain distance apart (on average over all proteins). To calculate this propensity, a non-redundant, high accuracy subset of the PDB was used (the PISCES data set mentioned above), and the frequency of each pair of atoms at various distances was counted. As in the *Victor/FRST* paper, 18 different distance bins were used (0-3, 3-4, 4-5 ... 19-20).

To translate a frequency into a score, the following formula was used:

n1 = total number of atom pairs (a, b) at distance d
t1 = total number of atom pairs of any type at distance d
n2 = total number of atom pairs (a, b)
t2 = total number of atom pairs of any type
ratio1 = n1 / t1
ratio2 = n2 / t2
score(a, b, d) = -log(ratio1 / ratio2)

where a and b are the atom types, and d is the distance bin.

This formula assigns a negative value to situations with a high propensity of occurrence relative to the average amino acid.

The RAPDF data file stores the pre-calculated score values for all pairs of atom types. Missing or sparse values (frequency less than 7) were set to zero; this only occurs for very low distances. When calculating the score, SAINT 2 linearly interpolates between the bins.

The RAPDF score is the total of the score for every pair of atoms. Atom pairs within the same or an adjacent residue are not included.

Solvation Score

Solvation is a measure of exposure to the surrounding solvent (water). Different amino acid types are expected to have different propensities for burial within the protein.

The Solvation score is based on the algorithm in the *Victor/FRST* paper. The number of other CB atoms within 10 Angstroms of a residues's own CB atom is counted, and the score is based on the propensity for that amino acid type to have exactly that many surrounding CB atoms (for Glycine, which has no CB atom, the CA atom is used instead). The bins used are (0-3, 4, 5 ... 37+).

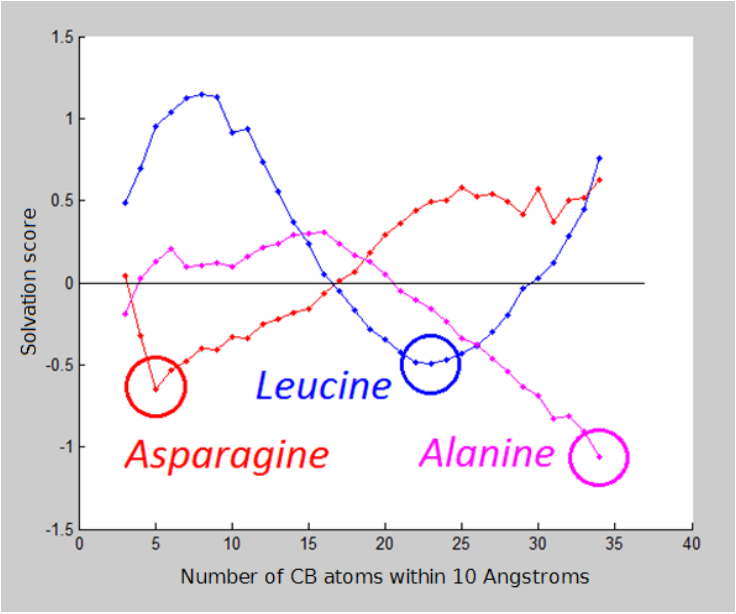
As with RAPDF, score values were derived from the PISCES data set mentioned above. The formula for the score is:

n1 = total number of residues of type a with exactly n surrounding CB atoms
t1 = total number of residues of any type with exactly n surrounding CB atoms
n2 = number of residues of type a
t2 = total number of residues of any type
ratio1 = n1 / t1
ratio2 = n2 / t2
score(a, n) = -log(ratio1 / ratio2)

where a is the amino acid type and n is the number of surrounding CB atoms.

The Solvation score is the total of the scores for each residue. Just as for RAPDF, the pre-calculated score values are stored in a data file. Missing or sparse values (frequency less than 7) were filled in by interpolation from the nearest bins.

The Solvation scores for three types of amino acid are show below. Optimal (lowest scoring) values are indicated with a circle.



Hydrogen Bonding Score

When the weights for each scoring term were calculated, the weight for Hydrogen Bonding was assigned a value of zero, so it is **not actually used** in SAINT 2.

The Hydrogen Bonding score is minus the total number of hydrogen bonds. The formula for detecting where hydrogen bonds occur is described in the *Victor/FRST* paper.

Torsion Angle Score

The Torsion Angle score is **not actually used** in the current version of SAINT 2, since it was causing non-native conformations to sometimes have a significantly better score than the native structure.

This score is based on each amino acid's propensity to have a particular pair of phi/psi angles. The formula is similar to the one described in the *Victor/FRST* paper; various bin sizes were tried, from 10x10 degrees (as in the paper) to 3x3 degrees.

The main issue with Torsion Angle scoring is that there are many undefined values (empty areas of the Ramachandran plot), and assigning a value of zero to these areas can result in them becoming more favourable than low frequency but valid areas (this problem does not tend to occur for RAPDF scoring, since the undefined values correspond to rarer situations). But assigning a high penalty to these areas (making them effectively forbidden) seems to have too strong an effect, since rare phi/psi angles still sometimes occur in real protein structures.

Lennard-Jones Score

The Lennard-Jones score was introduced to prevent steric clashes (overlapping atom positions), and to act as "glue" between residues that are close together. If this scoring term is omitted, the structures produced are not compact enough (neighbouring residues do not "touch").

The Lennard-Jones potential between atom types a and b at distance d is:

$LJ_{ab} = (C12_{ab} / d^{12}) - (C6_{ab} / d^{6})$

where

$C12_{ab} = epsilon_{ab} * sigma_{ab}^{12}$
$C6_{ab} = 2 * epsilon_{ab} * sigma_{ab}^{6}$

The epsilon and sigma values were taken from the OPLS-AA force field (see ***Appendix A*** for details).

A cut off distance of 2.5 * S is used (where S is the distance at which the potential equals exactly zero), beyond which the Lennard-Jones score is assigned a value of zero. This causes

a small discontinuity of about 1.5% of the well depth (the minimum value of the function), which is not significant for the purposes of SAINT 2.

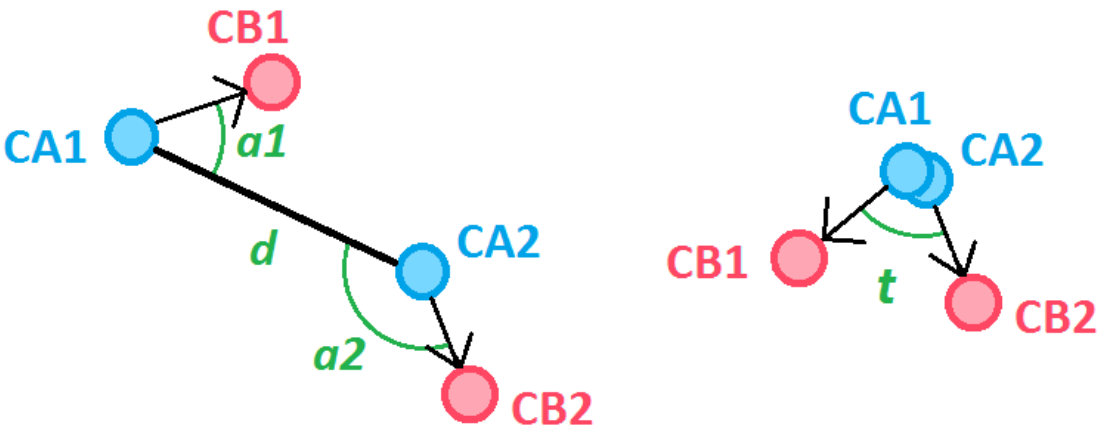
Because SAINT 2 only stores backbone and not side chain atoms, the Lennard-Jones score could potentially result in structures that are too compact, but this does not seem to happen very much in practice.

When the Lennard-Jones score is calculated for a structure, pairs of atoms in the same or an adjacent residue are not included.

The Lennard-Jones potential is sometimes used just to detect steric clashes. A steric clash is defined to occur if any pair of atoms has a Lennard-Jones potential of 20 or more.

Orientation Score

The Orientation score is similar to the RAPDF score. It is involves the propensity for each amino acid pair to be a certain distance apart (using CA atoms only), but includes the relative orientation of their side chains as well.



In the above diagram:

CA1 and CA2 are the alpha carbons of the two residues;
CB1 and CB2 are the beta carbons of the two residues (for Glycine, the estimated position where a beta carbon would be is used instead);
d is the distance between CA1 and CA2;
a1 is the angle CA2-CA1-CB1;
a2 is the angle CA1-CA2-CB2;
t is the torsion angle formed by CB1 and CB2 around CA1-CA2.

The values of a1, a2 and t are used to assign residue pairs to ten different “bins”:

<table>
<tr><td></td><td>a1 < 45</td><td colspan="2">45 <= a1 < 90</td><td>a1 >= 90</td></tr>
<tr><td>a2 < 45</td><td>bin 1</td><td colspan="2">bin 2</td><td>bin 3</td></tr>
<tr><td rowspan="2">45 <= a2 < 90</td><td rowspan="2">bin 4</td><td>t < 90</td><td>t >= 90</td><td rowspan="2">bin 7</td></tr>
<tr><td>bin 5</td><td>bin 6</td></tr>
<tr><td>a2 >= 90</td><td>bin 8</td><td colspan="2">bin 9</td><td>bin 10</td></tr>
</table>

Bins 5 and 6 are the only ones that make use of the torsion angle t because if either angle is less than 45 degrees, the torsion angle can easily fluctuate; and if either angle is greater than 90 degrees, the side chains are pointing “away” from each other, and the torsion angle is not as relevant.

As for RAPDF, the score values are obtained from the frequencies in the PISCES data set. 18 different distance bins were used (0-3, 3-4, 4-5 ... 19-20), along with the 10 angle bins described above, resulting in 180 different bins for each pair of amino acid types.

To translate a frequency into a score, the following formula was used:

n1 = total number of residue pairs (a, b) in bin B
t1 = total number of residue pairs of any type in bin B
n2 = total number of residue pairs (a, b)
t2 = total number of residue pairs of any type
ratio1 = n1 / t1
ratio2 = n2 / t2
score(a, b, B) = -log(ratio1 / ratio2)

where a and b are the amino acid types of the two residues, and B is the bin they fall into (based on the distance and angle values described above).

Just as for RAPDF, the pre-calculated score values are stored in a data file. Missing or sparse values (frequency less than 7) are set to zero.

The Orientation score is the total of the score for every pair of residues, not including residues adjacent in sequence.

Ribosome Wall Score

The presence of the ribosome restricts the possible conformations during the growth phase of cotranslational folding. A simple model of the ribosome is a two dimensional plane, beyond which no residue is permitted to move.

However, such a strict model of the ribosome wall can conflict with the fragment replacement approach used by SAINT 2. When a residue is extruded, it is possible that there is no fragment available which does not cross over the ribosome wall (because the position of the residue is not on the "outside" of the growing structure).

Instead of the "hard" ribosome wall described above, SAINT 2 uses an (optional) "soft" wall. Residues are permitted to cross the plane of the ribosome wall, but there is a scoring penalty for doing so, depending on the distance beyond the wall.

To simplify (and speed up) the ribosome wall scoring, the CA atom of the most recently extruded residue always has coordinates (0, 0, 0), and the structure is rotated so that the centre of gravity (average position) of all of the CA atoms lies on the negative x axis.

The ribosome wall is the plane $x = -5$ (which is a little more lenient than using $x = 0$), making the distance a particular atom is beyond the ribosome wall equal to: $-5 - atom\_x$.

If this value is positive, the scoring penalty is the distance squared. Only alpha carbons are included, so the total Ribosome Wall score equals:

$$\text{Sum for all CA atoms with } atom\_x < -5\text{: } (-5 - atom\_x)^2$$

Score Normalisation

The various scoring terms described above (other than the Ribosome Wall score) are normalised in two ways: by chain length, and to a similar scale. This is done so that a constant weight can be assigned to each term when they are combined together, and so that the weights are understandable.

In the original Victor/FRST paper, the weights were not on the same scale, and so it was not possible to see which scoring terms had a stronger weight than others. The authors did not normalise each scoring term by length either, which is a shortcoming.

The Wiggler Approach

In order to normalise the scores by length, sample scores for proteins of various lengths were required – not just for the native structure, but also for the kinds of intermediate structures SAINT 2 might encounter (which would be much less compact than the native).

To generate such intermediate structures, a program called `decoygen` was used to perturb the torsion angles of a structure to the point where the GDT_TS to the native structure reached a specified threshold. This program was inspired by the `wiggler` program created

by the Deane group, with some additional constraints: torsion angles were restricted to lie within the appropriate region of the Ramachandran plot for each amino acid type, and steric clashes were prevented (using the Lennard-Jones potential described under ***Scoring***).

The `decoygen` algorithm is:

- Read the native structure.
- Set the current structure to the native structure.
- Repeat until the GDT_TS of the current structure and the native exceeds the required threshold:
    - Perturb every torsion angle by a random amount up to plus or minus (50 / protein length) degrees, excluding angles that would enter an illegal region of the Ramachandran plot.
    - If no steric clashes are detected, accept the new structure.
- Output the structure.

The allowed Ramachandran regions were identified by dividing the plot into 10x10 degree bins, and finding the frequency for each bin for amino acid type using the PISCES data set referred to earlier in this document. A bin was determined to be "legal" if its total frequency was greater than five, or if any of the eight adjacent bins (horizontally, vertically and diagonally) had a frequency greater than five (to err on the side of leniency).

Normalisation

Eighty decoys (structures) were generated for 505 of the 5425 proteins in the PISCES data set, each with a range of GDT_TS values evenly spaced between 0.4 and 1.0. These decoys were used both to determine the score weights and to normalise the scores by length. The full PISCES set was also used for determining the RAPDF, Orientation, Solvation and Torsion scores, but if the 505 proteins are excluded from the full set and the scores recalculated, similar weights are obtained, so there does not seem to be an issue using the same PISCES set for decoy generation.

For each term except for the Ribosome Wall score, the raw score for each decoy was calculated. The initial approach taken was to plot all raw scores against the GDT_TS value, and attempt to find a function which would normalise the data, but this approach was not very successful: after the optimal weights were determined (as described in the next section), it was still easy to find structures with a lower score than the native.

A more successful approach was to find the minimum and maximum score for each decoy, and find a function f(s) such that f(max) - f(min) was independent of the protein length. The result was then scaled by a constant to make the average value approximately equal to 100 (so that the score weights determined later could be meaningfully compared).

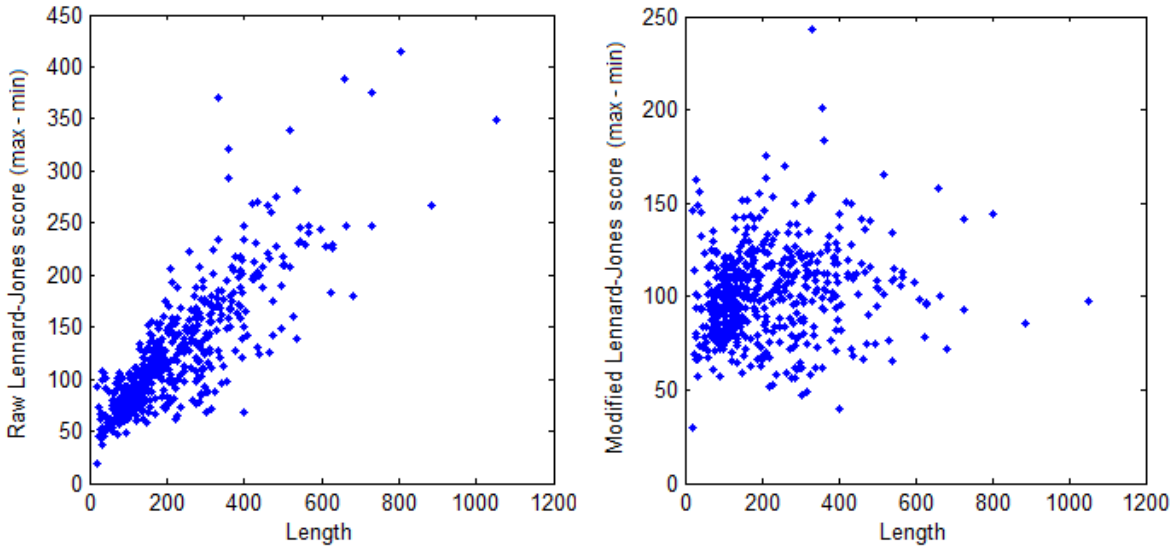
During the investigation, it was found that short proteins (150 residues or less) behaved differently to longer proteins for some of the scoring terms, and required different functions. Discontinuity is not an issue, because scores for different lengths are never compared.

The normalisation functions are:

Term	Short proteins (<= 150)	Long proteins (> 150)
RAPDF	score / length * 7	score / sqrt(length) * 0.75
Solvation	score / sqrt(length) * 60	score * 2.7
Torsion	score * 8	score / log(length) * 35
Hydrogen Bonding	score / sqrt(length) * 28 + 50	
Lennard-Jones	score / (length + 200) * 350	
Orientation	score / (length + 20) * 42	

The functions in the above table were arrived at subjectively, but they appear to perform well; structures with a score lower than the native are far less common than for the alternative approach described above.

The raw and modified Lennard-Jones scores for all 505 decoy proteins are shown below. The vertical axis for the raw plot (left) is the maximum minus the minimum score. For the modified plot (right), the vertical axis is f(max) - f(min), where f is the function for Lennard-Jones in the table above.



Weight Determination

A similar approach was used to determine the scoring term weights as in the *Victor/FRST* paper: a downhill search was performed to find the maximum possible correlation between the weighted score value and the GTS_TS value for all of the decoys mentioned in the ***Score Normalisation*** section above.

Unlike the *Victor/FRST* paper, however, the native structure's score was subtracted from the score value (so that the scores for different proteins could be meaningfully combined).

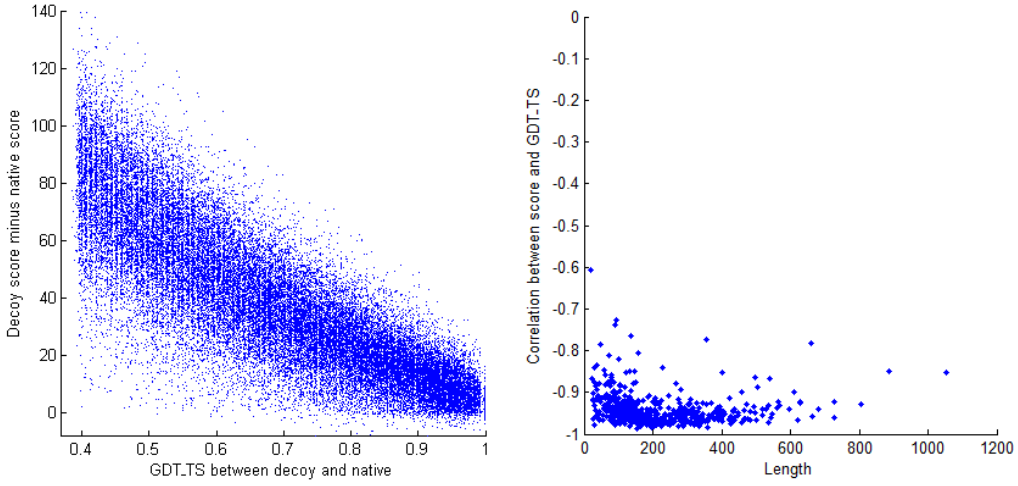
"Short" and "long" proteins were treated separately, and none of the weights was permitted to go below zero. The weights were normalised to add up to one. The final values are shown below (Torsion is excluded because it was causing problems; see section ***Torsion Score***).

Term	Short proteins (<= 150)	Long proteins (> 150)
RAPDF	0.156	0.303
Solvation	0.262	0.282
Hydrogen Bonding	0	0
Lennard-Jones	0.505	0.304
Orientation	0.077	0.111

Note that the Lennard-Jones potential is more important for short proteins (probably because the RAPDF and Solvation scores are more consistent for long proteins), and that the Hydrogen Bonding score has been excluded completely.

Using the above values, the correlation between the weighted score and the decoy GDT_TS values is -0.87 for long proteins and -0.79 for short proteins. The correlation is negative because low scores correspond to high GDT_TS values.

The plot on the left shows (decoy score minus native score) versus GDT_TS for all decoys; the overall correlation is -0.84. The plot on the right shows the correlation between score and GDT_TS for each individual protein. The average correlation is -0.94; a similar value is obtained if the 505 decoy proteins are omitted from the PISCES data set when determining the RAPDF, Orientation and Solvation scores, as described under ***Normalisation***.



5. Reflections and Further Work

Simulation versus Search

SAINT 2 attempts to combine two different approaches to protein structure prediction: simulation of "what really happens" during protein synthesis (extruding residues one at a time from the N-terminus, controlling the extrusion rate using codons and the presence of the ribosome wall), and direct search for the lowest energy state (fragment replacement).

These two approaches conflict with each other in some ways: fragment replacement is "non-physical" in that it allows large jumps in conformation space to occur. More problematically, using fragment replacement during cotranslational growth relies on the assumption that the intermediate state is a substructure of the native, or at least that that intermediate can be represented to a reasonable degree of accuracy from the available fragments.

Research into intermediate protein structures has been quite limited, but it seems likely that more flexibility is required than the current SAINT 2 model provides to be able to model intermediate states accurately.

One possibility is to combine the protein specific fragment library with a "general" fragment library to make a wide variety of conformations available at every position along the peptide. However, this could make the final native structure more difficult to locate.

A possible solution could be to vary the score for general fragments depending on where they are along the chain. The fragment score for the general fragments at the most recently extruded residue could be assigned a value equal to the best native fragment, then the value could be gradually decreased to zero over the next twenty positions or so. When the peptide is released from the ribosome, all general fragments could be assigned a score of zero, leaving only the native fragments selectable.

Appendix A: Constants

Ideal Bond Lengths and Angles

The following values were taken from *Proteins: Structures and Molecular Properties* by T. E. Creighton (2nd edition), 1993.

Backbone atom bond lengths:

C-C = 1.52 Å
N-C = 1.33 Å
N-CA = 1.45 Å
C-O = 1.23 Å

Backbone atom bond angles:

N-CA-C = 109.5°
CA-C-N = 115.6°
C-N-CA = 121.9°

Lennard-Jones Parameters

The "epsilon" and "sigma" values for the Lennard-Jones potential were taken from the OPLS-AA force field, as defined in:

`http://dasher.wustl.edu/tinker/distribution/params/oplsaa.prm`

The values for a pair of carbon, oxygen or nitrogen atoms are:

sigma_{CC}: 3.75	epsilon_{CC}: 0.105
sigma_{OO}: 2.96	epsilon_{OO}: 0.21
sigma_{NN}: 3.25	epsilon_{NN}: 0.17

To calculate epsilon and sigma values where the two atom types are different, the following formulas may be used:

$\text{epsilon}_{ab} = \text{sqrt}(\text{epsilon}_{aa} * \text{epsilon}_{bb})$	*(geometric mean)*
$\text{sigma}_{ab} = (\text{sigma}_{aa} + \text{sigma}_{bb}) / 2$	*(arithmetic mean)*

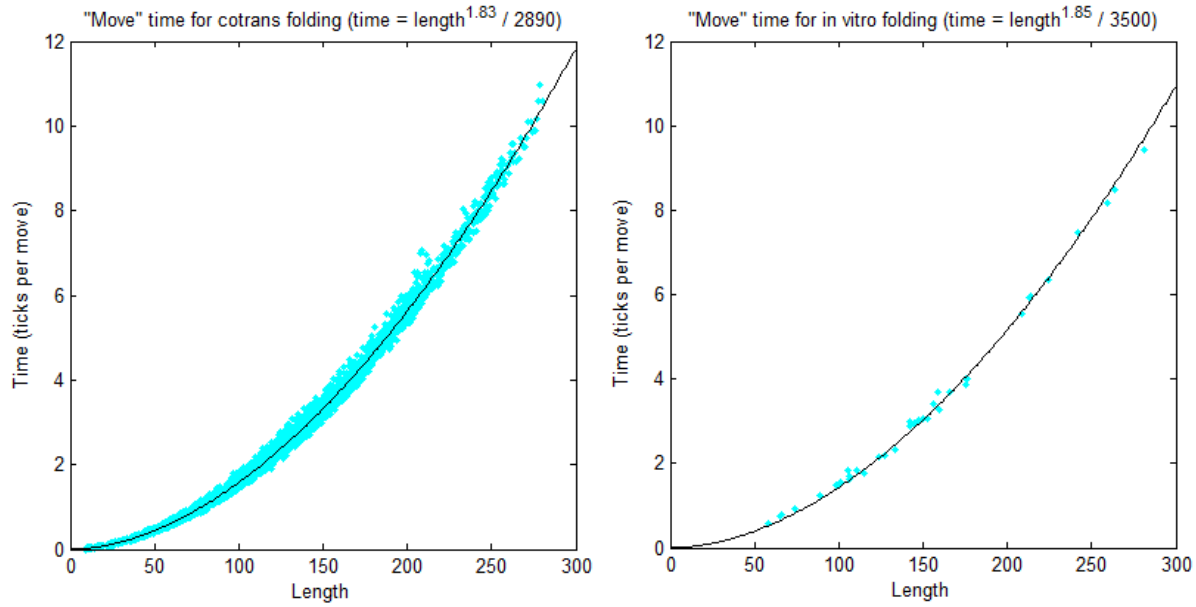
Appendix B: Move Equivalency

To be able to fairly compare in vitro and cotranslational folding, the same amount of processing time should be allocated to each. Simply using the same total number of moves is not enough, since scoring short peptides (eg. during the growth phase of cotranslation) takes much less time than scoring long ones.

As well as the length, the compactness of the structure affects the scoring time – for example, the RAPDF scoring function can quickly exclude residue pairs that are a long distance apart; more calculations are required for residues that are close to each other.

A set of 39 proteins was used to find the average time for a single in vitro move or cotranslational move. The proteins and their chains were: 1a17 A, 1a1x A, 1ag9 B, 1au1 A, 1bd8 A, 1bm8 A, 1byi A, 1dhj A, 1dj0 A, 1em8 A, 1ew4 A, 1f18 A, 1f86 A, 1g8e A, 1gp0 A, 1gpq A, 1huk A, 1i40 A, 1jke A, 1jkx A, 1jo8 A, 1k6k A, 1k96 A, 1kpf A, 1l9l A, 1lho A, 1lm8 V, 1lug A, 1m5w A, 1or7 C, 1ozw A, 1pod A, 1q6o A, 1qbj A, 1r6q A, 1ra9 A, 1rya A, 1tqg A and 3vub A.

The plots below show the average move time (in "ticks", approximately one hundredth of a second) for various peptide lengths. Times are averaged over fifty moves.



The cotranslational plot (left) contains one data point for every extrusion of every protein. The black line is the curve:

$$Cotrans\ move\ time = length^{1.83} / 2890$$

The in vitro plot (right) displays one data point per protein. The curve shown is:

$$In\ vitro\ move\ time = length^{1.85} / 3500$$

The two curves are very similar (at length 300, the cotranslational time is about 8% higher than the in vitro time).

Using these formulas, it is possible to calculate the equivalent number of in vitro moves for a cotranslational run, depending on the number of moves after each extrusion. The `calc_invitro_moves` script automates this process.

Example

For a protein of length 150, starting with 9 residues extruded and extruding one at a time with 50 moves after each extrusion, the total number of cotranslational moves during the growth phase is 7050 moves. The total time for these moves is 8717.4 ticks.

At length 150, the time for one in vitro move is 3.032 ticks, and so the equivalent number of in vitro moves is 8717.4 / 3.032 = 2875 moves (about two fifths the number of cotranslational moves).