February1998

SEQUENCE ALIGNMENT by GENETIC ALGORITHM:

User Documentation for:

- -SAGA
- -RAGA
- -PRAGA

Cedric Notredame
Des Higgins

Useful References:

SAGA: Sequence Alignment by Genetic Algorithm
Notredame and Higgins **Nucleic Acids Research** 24:8, 1515-1524 (1996).

COFFEE: An Objective Function for Multiple Sequence Alignmnents Evaluation
Notredame, Holm and Higgins, **Bioinformatics** (in press)

RAGA: RNA Alignment by Genetic Algorithm
Notredame, O'Brien and Higgins **Nucleic Acids Research** 25:22, 4570-4580 (1996).

This documentation covers the following programs:

SAGA V0.93	20/02/98
RAGA V0.91	3/09/97
PRAGA V0.91	3/09/97

WARNING

This program comes with no warranty. The code should not be modified and/or redistributed without the permission of the authors:

cedric.notredame@ebi.ac.uk
des.higgins@ebi.ac.uk

ADDRESS:

The EMBL outstation
The EBI
Hinxton Hall
Hinxton
CB 10 lRQ CAMBRIDGE

DESCRIPTION

SAGA/RAGA/ are packages for sequence alignment.

SAGA is specially designed for multiple alignment of protein sequences.

RAGA is designed for the alignment of two RNA sequences, knowing the secondary structure of one of the sequences and using this information during the alignment.

PRAGA is a parallelisation module that can be used with SAGA or RAGA.

More detailed information can be found in the relevant papers indicated above.

SUMMARY

1 -INSTALLATION

A-SAGA

Once the program has been uncompressed and untared, you will find in the main directory:

```
/SAGA/      DOC
            INSTALL
            PARAM
            TESTCASES/ac_prot.....s_prot
            WORK
            SOURCE_0.93
```

1-Go into SAGA/INSTALL
2-type *unix_make.source <osf or sgi>*
(if you cannot run make.source, change the mode of the file 'chmod u+x make.source'
This will create the executable SAGA in SOURCE_0.93 as well as a list of aliases in INSTALL/saga_aliases. You should install this file so that it get sourced each time you login. Alternatively, you can source it.

3-Install the executable /SAGA/SOURCE/SAGA in your .path
4-**IMPORTANT**: To use SAGA optimally (and especially COFFEE) you should have ClustalW installed. If you don't you can obtain it from ftp.ebi.ac.uk. Once ClustalW is installed edit the file

PARAM/execuatbles_path.param

and type in the path for ClustalW:

```
S       CLUSTAL_PATH <your_path>/<executable_name>
```

B-RAGA

Once the program has been uncompressed and untared, you will find in the main directory:

```
/RAGA/      DOC
            INSTALL
            MATRICES
            PARAM
            TESTCASES/TC1...TC9
            WORK
            SOURCE
```

1-Go into RAGA/INSTALL
2-Type *source unix_make.source*
This will create the executable raga in SOURCE

3-Install the executable /RAGA/SOURCE/raga in your .path

C-PRAGA

Once the program has been uncompressed and untared, you will find in the main directory:

```
/RAGA/      INSTALL
            TEMP
            PARAM
            TESTCASES/TC1...TC9
            SOURCE
```

1-Go into /PRAGA/INSTALL

2-Type source unix_make.source
    This will create the executable praga in SOURCE
3-Install the executable /PRAGA/SOURCE/raga in your .path

NOTE: You should have a different copy of PRAGA with SAGA and RAGA if you run both programs. In this case, the PRAGA executable should be renamed praga_saga or praga_raga.

2- QUICK START

A-SAGA

The file INSTALL/saga_aliases contains aliases for three different versions of SAGA:

-saga_osf : the basic genetic algorithm set to optimize the weighted sums of pairs
-coffee_osf: SAGA set to use the COFFEE Objective Function
-T_coffee_osf: SAGA set to make a greedy non GA based alignment using the COFFEE OF (see Parameters for OF_MODE).

-1 go into INSTALL
-2 source saga_aliases
-3 go into WORK
-4 Type one of these three commands :

```
saga_osf s: fniii.sparam
coffee_osf s: fniii.sparam
T_coffee_osf s: fniii.sparam (or any other sparam file).
```

B-RAGA

1-Go into RAGA/WORK
2-Type *raga s: tc1.sparam*

the program will align the sequences of the test case or any other of the 9 test cases present in this directory. tc1.sparam is called a secondary parameter. It contains pointers to the sequences and structure present in RAGA/TESTCASE/TC1. At the end of each generation the results (the RAGA alignment) are stored in best_E_1.ga. Other files are produced. The nature of their content, as well as the content of the parameter files are explained in the next section.

C-PRAGA

To run PRAGA, SAGA or RAGA must have previously been installed. Use for running RAGA or SAGA the version of PRAGA that came along.

1-REQUIREMENTS

1-You need to be able to run simultaneously 13 RAGA or SAGA processes (15Mbytes each). Ideally, this means having access to thirteen machines.

2-All the machines you want to use must be mounted on a common disk containing PRAGA, RAGA. and run under a UNIX Operating System.

3 All the machines you want to use must support: the following UNIX command: rsh, up_time, ping, grep, at. If they do not, see your system manger.

2-SETTING UP AND RUNNING RAGA

1-You need an executable of RAGA compiled on each different type of machine you want to use. This executable will be identified by a suffix of your choice (e.g. raga_sgi for the sgi executable, raga_alpha for the alpha and so on). This can be done as follow:

a) log into the machine you are interested in
b) go into /RAGA/SOURCE
c) type *makefile raga*
d) type mv raga raga_(new suffix)

e) carry on until an executable has been produced for each type of platform you are interested in.

5-Go into PRAGA/PARAM
6-Edit the file machine.praga_param
7-enter the name of the machine, the extension corresponding to its architecture and the number of processors it contains. For instance:

```
$
capricorne   sgi    2
jupiter      sgi    1
columba      sgi    7
venus        alpha  8
$
```

8-go into /PRAGA/TESTCASES/TC1
9-type *praga s: tc1.praga_sparam*

3-HOW TO STOP PRAGA?

If you want to stop PRAGA you need to remove all the processes from all the machines. This can be done in several ways:

1-if PRAGA is still running:
echo 1 > finish_PR_1.ga

2- if PRAGA is still running
kill one of the PRAGA processes

3-if PRAGA has been stopped or has crashed:
source end_of_PR_1.pga. In this case, you must as well go into PRAGA/TEMP/IO and remove all the files.

4-if PRAGA has been stopped or has crashed
if everything failed, you will have to go on all the machines and kill the processes one by one. Have fun and don't forget to remove the garbage in PRAGA/TEMP/IO.

The results of PRAGA are in the file best_PR_1.ga. PR_1 is the name of the Experience. It is contained in the TC1.praga_sparam file. This file is a PRAGA secondary file. It can contain any parameter used in RAGA/SAGA/PRAGA. It must also contain a TESTCASE_FILE parameter that contains for RAGA: at least the sequence_file and the structure_file. This TESTCASE_FILE is in fact a RAGA secondary file (see previous section).
If the TESTCASE_FILE contains other RAGA parameters (such as gap penalties ....) These are replaced by any new value given in the PRAGA secondary file. On the other hand the parameters contained in the RAGA secondary file overwrite the values contained in the PRAGA main parameter file. At this point we do no recommend changing any of the default values contained by the PRAGA/PARAM files.

3-USING SAGA

INTRODUCTION-USER ADVISES

Saga is a versatile tool for multiple sequence alignments. It is mostly built around a Genetic Algorithm but can also perform more greedy optimizations. We have arranged the parameters for three specific configurations of SAGA:

1-saga_osf is the original SAGA and attempts to perform the optimization of the function used in the MSA program (weighted sums of pairs with affine gap penalties).

2-coffee_osf is an objective function that defines the best multiple sequence alignment as the one having the highest level of consistency with a set of pairwise alignments. You can generate the pairwise alignments using your favorite method or the defaults provided with the package

3-T coffee is a fast greedy optimization of the COFFEE function. Although it is less good in terms of mathematical optimization. A comparison made on structural alignments revealed that T coffee often outperforms COFFEE. In most cases, both methods outperform MSA when using pairwise alignments generated by ClustalW.

A-main parameters

The program reads by default the parameters in the PARAM directory. These should not be modified. Parameters value can be modify in two ways:

-secondary Parameters
-at the prompt

B-secondary parameters

In order to locally modify the values of the parameter kept in SAGA/PARAM, it is possible to use a second set of parameters kept locally and containing new values for some of the parameters, like fniii.sparam. When using such a file, the values of the parameters are first initialized in SAGA/PARAM and then reinitialised with the values contained in the secondary parameter file. The second parameter is indicated by the flag s, There can be as many secondary files as you wish. For instance

saga s: < secondary parameter file1> s: <scondary parameter file2>

will first read the values from file1, and then from file2. If two different values are read in files 1 and 2 for the same parameter, the effective value will be the last one (i.e. the one read in file2).

In a secondary file, the order of the parameters is irrelevant but the format must be kept. For instance, if all you want to indicate is that the population size should be 50 individuals, your secondary parameter file should look like that:

```
$
**************************
D MAXPOP 50
**************************
$
```

The format is as follow:

```
<Type:  D,S,F>     NAME        VALUE
<Type:  O>         NAME        VALUE1      VALUE2
```

$ signs indicate the start and the end of the file.
* are commented out
D: parameter with an int value

F: parameter with a Float value
S: String
O: Operator

Alternatively, it is possible to enter parameters at the prompt:

```
<saga> s: <file x> MAXPOP 20
```

In which case the prompt parameter are dominant on all the others. There is no need to declare the type of the parameter in this case, simply the name and the appropriate syntax.

A list of the parameters is given below and is also available from the program itself (section D) by calling:

```
saga_osf option: ALL
```

C-Output

By default SAGA outputs at least one file:
<EXP_NAME>.saga_aln

If a tree is computed, it will also output
<EXP_NAME>.saga_dnd

If SAGA is used with COFFEE (OF_MODE 8 or 9), it will also output two extra files:
<EXP_NAME>.saga_rel_aln that contains the local score of each residue in the alignment
<EXP_NAME>.saga_filt_aln that contains a filtered alignment (Capital for residues having a score higher than seven).

Finally SAGA outputs a file name <EXP_NAME>.saga_param that contains all the parameter values used in your run (including the value of the random seed). It can be use to EXACTLY reproduce a run.

D-The parameters of SAGA/PARAM

Online help is available by:
saga_osf help

or

saga_osf option: <parameter name> (it is possible to enter only a portion of the name)

MAXPOP

type: D
Description: Number of individuals in the genetic algorithm SAGA
value:
Typical=100

MAX_GEN

Type D
value:
Typical=1000

experience_name

type: S
value:
- name of your experience
- DEFAULT will use the prefix of the sequence file

pep_file

type: S
value:
file containing all the sequences that need to be aligned, any format accepted by ClustalW will do (PIR, Swissprot, GDE...)
NOTE: COMPULSORY

ref_aln_file

type: S
value:
contain in a ClustalW like format a reference alignment

rooted_tree

type: S
value:
- file name of a rooted/unrooted tree in Phylip Format.
- MAKE:MODE:NAME
  - MAKE: Indicates that the tree must be computed
  - MODE: Indicates the type of weights to use for the tree
    - ALN_LIB_WEIGHT indicates that inverted pairwise sequence weights will be used
    - SIMILARITY_1: Average similarity of each pair as taken from:
      - 1-the aln library
      - 2-if there is no aln library, a clustalW pwaise aln
      - 3-if clustalw is not install, a dynamic programming alignment made by saga
    - ......
    - SIMILARITY-7 (see see weight_file)
  - NAME Indicates The name (DEFAULT will give <experience_name>.saga_dnd

aln_lib

type: S
value:
- file_name:
  - gives the list of the alignments used for the COFFE table

    two possible formats:

    1-default:
```
$
al1
al2
al3
...
$
```
where al1, al2.... are (pairwise or not) alignments in SAGA format.

2-compressed:

```
#SEQUENCES
>name1 seq1
>name2 seq2
....
#DALIALN
ALN 1 name1 name2 weight
<first bloc start in seq1 first bloc end in seq 1><second bloc start second bloc end>
<first bloc start in seq1 first bloc end in seq 1><second bloc start second bloc end>
ALN 2 name1 name2 weight
<first bloc start in seq1 first bloc end in seq 1><second bloc start second bloc end>
<first bloc start in seq1 first bloc end in seq 1><second bloc start second bloc end>
.....
```

NOTE: -set the weights to 0 if you do not want them read here
-example of coded aln:

```
seq1 aaaaa----aaaa-a-a---a
seq2 ---bbbbbbbb--b-bbbb-
```

will appear as:

```
ALN 1 seq1 seq2 0
4 5 6 7 10 10 11 11
1 2 7 9 10 10 11 11
```

-MAKE:MODE:NAME
MAKE-indicates that the library should be made (pairwise)
with:MODE
MAKE:CLUSTALW Clustal if Clustal is intalled
MAKE:BASIC_DP Built in Dynamic Programing
and saved as:NAME
MAKE:CLUSTALW:DEFAULT
<experience_name>.compressed_pw_lib in a compressed format with weights set to 0

NOTE: -Sequences in the aln_lib do not need to be exactly the same as those in the pep_file.
-If aln_lib contains more sequences than pep_file, the COFFEE_TABLE will be computed using the sequences in aln_lib, but the alignment will only be made on the pep_file sequences.

GAP_OP
type: D
value: 8 when used with Pam 250.
It is the gap opening penalty when doing DP or using the sums of pairs objective functions.8

GAP_EXT
type: D
value: 12 when used with Pam 250

OF_MODE
type: D

value:

0:sop with gap terminal treated as internal(minimise)
1:sop with no gop for terminal gaps(minimise)
2:sop with no gop+gep for terminal gaps (minimise)
8: COFFEE for GA(maximise)
9: COFFEE for greedy alignmnent (see doc)

NATURAL_GAP

type: D
value:

with OF_MODE 0, 1, 2
-0 semi natural gap penalties
-1 natural gap penalty

matrice_name

type S
value

-blosum use the blosum series
-pam use the pam series
-other, use ad hoc matrix with format:

```
$
v1
v2 v3
v4 v5 v6
.........
$
```

v1, v2... are integers possibly negatives.
the order of the amino acids is: ABCDEFGHIKLMNQRSTVWXYZ

weight_file

type S
value

-name: file containing pairwise weights in a format:

```
$
name1 name2 weight
.....
$
```

that must include ALL the pairs of sequences
-SIMILARITY_1...SIMILARITY_7
-indicates that weights should be computed using sequence **similarity**

SIMILARITY_1->Number id/number of pairs
SIMILARITY_2->Number id/alignment length
SIMILARITY_3->Number id/sum of the seq length
SIMILARITY_4-> non supported at the moment
SIMILARITY_5-> non supported at the moment
SIMILARITY_6->number of id-number of gap op/minimum lenght
SIMILARITY_7->Avergae sim in the longest conserved segment defined as the longest segment where two identities are no more than 5 residues appart

s_weight_file

type S

value

-name: file containing the sequence weights

$

name weight

....

$

EVALUATE_ONLY

type D

value:

0 ignored

1 only evaluate the reference alignmnent using the selected objective function and exits

CLUSTAL_PATH

type: S

value

indicate the path for the ClustalW executable

SEEDING_MODE

type: D

value:

0 random alignments

1 use the referenc alignment

2 use the pairwise library

3 use the coffee table

4 do dp using the coffee table

E-FORMAT

1-sequences

All the sequences must be in the same file. SAGA is able to read EMBL, SwissProt, PIR, Pearson, GDE, and sequences from multiple alignments coming from CLUSTAL and PILEUP/MSF.

2-alignments

SAGA can read alignments from Clustal, ClustalW, SAGA and RAGA.

3-trees

SAGA can read and will generate unrooted trees in the format used by ClustalW (Phylip package). The trees produced that way can be read by SAGA.

F-BUGS REPORT

If you encounter a BUG, please:

1- if the program crashes in early stages (before the first generation)

a-check you are not running out of memory

b-check your sequences, trees and reference alignments

c-try to make the sequence names simpler.
d- check the syntax of the files you have been using

2- if the program crashes after a few generations
a-using the same value for RANDOM_SEED (!= -1) try to increase the value of MEM_FACTOR in SAGA/PARAM/s0_pb_data.param.
b-If the program keeps crashing or if the bug is of another type, please send an E mail
cedric.notredame@ebi.ac.uk
subj.: saga bugs report
indicate:
-name of the machine (type, OS ...)
-list and values of the parameters changed from the standard
sequences, tree, ref. alignment, weights ....
-IMPORTANT: value of the RANDOM_SEED
-a copy of the output or error message.

And I will try to fix the bug..... But if you fix the bug yourself :-) please let me know about it.

3-USING RAGA

A-main parameters

All the parameters RAGA requires are in RAGA/PARAM. The program has been compiled in such a way that by default it reads the RAGA/PARAM/ga_param.param file that contains the name of 6 files that with the values of all the parameters required by RAGA.

```
BGA: RAGA/PARAM/bga_param.param
DATA: RAGA/PARAM/pb_data_param.param
SEED: RAGA/PARAM/pb_seed_param.param
DOS: RAGA/PARAM/pb_dos_param.param
OPP: RAGA/PARAM/pb_opp_param.param
OFP: RAGA/PARAM/pb_of_param.param
```

This file is the file read by default, you can specify another file by using the flag m:

```
RAGA m: <new main parameter file>
```

B-secondary parameters

In order to locally modify the values of the parameter kept in RAGA/PARAM, it is possible to use a second set of parameters kept locally and containing new values for some of the parameters (see RAGA QUICK START). When using such a file, the values of the parameters are first initialised in RAGA/PARAM and then reinitialised with the second parameter. The second parameter is indicated by the flag s:

```
RAGA m: <main parameter file> s: < secondary parameter file>
```

This file can contain new values for any of the parameters declared in

RAGA/PARAM/
- bga_param.param
- pb_data_param.param,
- pb_seed_param.param
- pb_dos_param.param
- pb_opp_param.param,
- pb_of_param.param.

The order of the parameter is irrelevant but the format must be kept. For instance, if all you want to indicate is that the population size should be 50 individuals, your secondary parameter file should look like that:

```
$
*************************
D MAXPOP 50
*************************
$
```

The format is as follow:

```
<Type:  D,S,F>     NAME        VALUE
<Type:  O>         NAME        VALUE1     VALUE2
```

$ signs indicate the start and the end of the file.
* are commented out
D: parameter with an int value
F: parameter with a Float value

S: String
O: Operator

The names of the parameters as well as their type can be found in the PARAM/* file. The function of some of these is briefly explained below in section D.

C-Output

By default RAGA outputs a series of files. <output name> is the value of the parameter:
S experiment_name <output name>".

-best_<output name>.ga: contains the best scoring alignment produced so far
-best_<output name>.ga: contains the amount of CPU time used to produce the best alignment
-best_<output name>.ga: contains the number of generations required
-bilan<output name>.ga: contain a listing of the results/generation
-current_operator_<output_name>.ga: contains a listing of the usage probability for each operator, after each dynamic reevaluation.

The same files preceded by the letter g, like *gbest<output name>*.ga contain the same information, but relative to the last evaluated generation.

D-The default parameters in RAGA/PARAM

The parameters contained in RAGA/PARAM are the default parameters. They should not be modified directly. If a different set of parameters is needed.

1-create a new PARAM directory
2-modify the default through a secondary parameter file.
The usage of some of these parameters is explained in some of the sections below.

1-bga_param.param

These parameters are common for SAGA and RAGA, see the equivalent SAGA section.

2-pb_data.param

This file contains the name of the data files used for creating the multiple alignment. For the formats, see the FORMATS section.

ref_aln_file: name of the reference alignment (set to NO) if none is available.
seq_file: name of the two sequences used for the alignment
struc_file: name of the file containing the structure of the first sequence in the sequence file. The format must be as follow:

C
Number of pairs
<stem number> <residue1 indices> <residue2 indices> <residue1> <residue2> <0 if the stem is a pseudoknot, 1 if it is not> <CR>

3-pb_seed.param

This file contains variables that control the seeding. See the file itself for usage information.

4- pb_dos.param

Contains the initial probabilities for the operator usage. It has been tuned for optimality and should be kept unchanged

5- pb_opp.param

This file contains some parameters specific of the operators initial usage probabilities. It has been tuned for optimality and should be kept unchanged.

6- pb_of.param

These parameters control the Objective Function used by RAGA (see paper).

LAMBDA	Weight of the secondary structure in the evaluation
R_GAP_S_OP	Cost for opening a gap in a secondary structure.
R_GAP_OP	Cost for opening a gap in a loop.
R_GAP_EXT	Cost for extending a gap.
r_struc_matrice_file	matrix for secondary structure match
r_seq_matrice_file	matrix for primary structure match. These two matrices are in the RAGA/MATRICES directory. They can be replaced using the following format:

```
order: AGCUTXN
#
v1
v2      v3
v4      v5      v6
.....
```

E-FORMAT

1-sequences

All the sequences must be in the same file. RAGA is able to read EMBL, SwissProt, PIR, Pearson, GDE, and sequences from multiple alignments coming from CLUSTAL and PILEUP/MSF.

2-alignments

RAGA can read alignments from Clustal, ClustalW, RAGA and RAGA.

3-RNA Secondary Structure

The format must be as follow:

C<CR>
Number of pairs<CR>
<stem number> <residue1 indices> <residue2 indices> <residue1> <residue2> <0 if the stem is a pseudoknot, 1 if it is not> <CR>

F-TEST CASES

Test cases are in the RAGA/TESTCASES/TC1-TC9 directories. Each directory includes:

-lg* which is the alignment made by dynamic programming with local gap penalties.
-*.rna that contains the sequences
-*.full_struc that contains the structure of human ribosomal RNA
-*.sparam is a secondary parameter file for aligning these sequences with RAGA. The *.sparam files are also in the RAGA/WORK directory, where the runs should preferably be made.

G-BUGS REPORT

If you encounter a BUG, please:

1- if the program crashes in early stages (before the first generation)

a-check you are not running out of memory
b-check your sequences, trees and reference alignments
c-try to make the sequence names simpler.
d- check the syntax of the files you have been using.

2-if you are running out of memory during the seeding. Change the variable SEEDING_MODE from 4 to 0. This avoids making the seeding by random dynamic programming.

3- if the program keeps crashing please send an E mail to
cedric.notredame @ebi.ac.uk
subj: RAGA bugs report
indicate:
-name of the machine (type, OS ...)
-list and values of the parameters changed from the standard sequences, tree, ref alignment, weights ....
-IMPORTANT: value of the RANDOM_SEED
-a copy of the output or error message.

And I will try to fix the bug..... But if you fix the bug yourself :-) please let me know about it.

4-USING PRAGA

An extended information is not yet available for PRAGA. Nevertheless for most of the uses, the quick start should be enough. If you need to design a new test case with PRAGA, inspire yourself of the available test cases in PRAGA/TESTCASES. If you have some interest in PRAGA and wish to see the documentation made available, please let me know.