September 1997

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 N.A.R. 24:8, 1515-1524 (1996).

RAGA: RNA Alignment by Genetic Algorithm
Notredame, O'Brien and Higgins

This documentation covers the following programs:

SAGA V0.91	9/10/96
RAGA V0.91	3/09/97
PRAGA V0.91	3/09/97

WARNING

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:

```
<your directory>/SAGA /   SOURCE
                          PARAM
                          TESTCASE/ GLOBIN
                          DOC
                          README
```

1-Go into SAGA/SOURCE.
2-Type : *source unix_make.source*
This will create the executable saga_unix

3-Install <your directory>/SAGA/SOURCE/saga_unix into your .path

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 example provided in SAGA/TESTCASE/GLOBIN will allow you to run saga:

1-Go into that directory.
2-*saga (* return)
prompt for the main parameter file ? (return)
prompt for the output name *? GLOBIN* (return)
prompt for the sequence file ? *globin.pep* (return)
prompt for the reference alignment ? *msa_globin.aln* (return)
prompt for the rooted tree ? *rooted_globin.dnd* (return)
prompt for the weights ? *globin_msaw.ga (* return)

The program will then go on, until it finds the optimal solution. It takes approximately 100 generations for the program to reach that point (on average) and 50 more to stop. Every 5 generations, the program outputs some of its current results. You can access the best alignment produced so far by opening the file: best_<name of the output>.ga. In this case, the best alignment is stored in best_GLOBIN.ga

You don't necessarily have to enter all the names of the files at the prompt, they may all be entered in advance in a file like SAGA/TEST CASE/GLOBIN/globin_param.param.

```
$
********************************************
S     pep_file            globin.pep
S     ref_aln_file        msa_globin.aln
S     rooted_tree         rooted_globin.dnd
S     weight _file        globin_msaw.ga
S     experience_name     GLOBIN
********************************************
$
```

to use this type of files named a secondary parameter file, you must enter
saga s: globin_param.param

If you do so, saga will prompt you to reconfirm the values of the parameters. To avoid this, it is possible to turn on the flag "non_interactive"
saga s: globin_param.param non_interactive

which causes the program to run without interruption.

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

A-main parameters

All the parameters SAGA requires are in SAGA/PARAM. The program has been compiled in such a way that by default it reads the SAGA/PARAM/s0_ga_param.param file that contains the name of 6 files that contain the values of all the parameters required by SAGA.

BGA: SAGA/PARAM/s0_bga_param.param
DATA: SAGA/PARAM/s0_pb_data_param.param
SEED: SAGA/PARAM/s0_pb_seedQaram.param
DOS: SAGA/PARAM/s0_pb_dos_param.param
OPP: SAGA/PARAM/s0_pb_opp_param.param
OFP: SAGA/PARAM/sO_pb_of_param.param

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

saga m: <new main parameter file>

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 globin_param.param. When using such a file, the values of the parameters are first initialised in SAGA/PARAM and then reinitialised with the values contained in the secondary parameter file. The second parameter is indicated by the flag s:

saga m: <main parameter file> s: < secondary parameter file>

This file can contain new values for any of the parameters declared in
SAGA/PARAM/
s0_bga_param.param
s0_pb_data_param.param,
s0_pb_seed_param.param
s0_pb_dos_param.param
s0_pb_opp_param.param,
s0_pb_of_param.param.

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

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

C-Output

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

These files are:

-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

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

-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.

D-The default parameters in SAGA/PARAM

The parameters contained in SAGA/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.
We give below a brief explanation of the values of some of these parameters.

1-s0_bga_param.param

This file contains all the parameters needed by the GA but not specific of the problem. It also contains a few parameters of general interest.

OUTPUT_SCREEN (0)no screen output
(l)normal output

OUTPUT_FILE (0)no file written.
(n) result file written every n generation.

RANDOM_SEED (-1) seed=random number
(n) seed=n.

MAXPOP (n) population of the GA =n

MAX_GEN (n) keep going for a maximum of n generations

OVERLAPING_GEN_FACTOR: ratio of individuals to survive unchanged from one generation to the next.
(0) replaces all the individuals
(1) never replace any individual
(0.3) replace 30% of the individuals by new children.

ALLOW_DUPLICATE

(0) Do not allow identical individual in the same generation
(1)allow duplicates.

The RESCALING, PRESELECTION, SELECTION and REPLACE_MODE sections are referring to different schemes of GA descried by GOLDBERG. These values should not be modified, unless experimenting with the GA itself.

The DYNAMIC OPERATOR section describes the parameters used for estimating dynamically the usage of the different operators, as described in DAVIS.

FIT_TO_ADAPT
(0) causes the program to use constant probabilities of usage as initially set by the user.
(1) use the dynamic scheduling

MAX_STAB
(n) stops the program when it has not produced any improvement for n generations.

BIASE_POS:
(1) Make it more likely to apply the operators around areas containing gaps.
(0) turns the bias off.

2-s0_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.

rooted_tree_file
file containing the tree (rooted or not) set it to NO if no tree is available. It is better to use a rooted tree.

3-s0_pb_seed.param

This file contains variables that control the seeding.

SEEDING_MODE (0) random alignment used for seeding.
(l)Use the reference alignment. This second option causes the program to behave as an alignment improver with the reference alignment.

RANDOM_OFFSET
(n) controls the generation of random alignment with each sequence having an offset between 0 and n.

4- s0_pb_dos.param

This files defines the initial probabilities of usage for the operators.O indicates that the variable defines an operator. The first number (1 or 2) indicates the number of parents. The second value indicates the relative usage probability. These values have been tuned carefully and should provide a robust setting.

5- s0_pb_opp.param

This file contains some parameters specific of the operators. These values, as well have been tuned and should not be modified unless willing to experiment with the GA itself.

6- s0_pb_of.param

These parameters control the Objective Function used by SAGA. The gap penalties are of affine type:

gap score= *GAP_OP* + *GAP_EXT* * gap length

OF MODE (0): consider terminal gaps as internal gaps
(1): penalise terminal gaps with a gap opening penalty
(2): do not penalise terminal gaps at all

NATURAL_GAP
(0) use semi natural gap penalties.
(1) use natural gap penalties.

weight_file

This file contains the weights used for scoring the alignment. If instead of a name the value is "NO", then all the weights are set to 1. Otherwise, the indicated file is read. The weights must be in a specific format, for instance, for three sequences seql, seq2, seq3:

```
$
seql_name seq2_name <weight (1,2)>
seql_name seq3_name <weight (1,3)>
seq2_name seq3_name <weight (2,3)>;
$
```

matrice_name

the default matrices are in the header SAGA/SOURCE/matrices.h. Available matrices are:

blosum30mt
blosum40mt
blosum45mt
blosumS5mt
blosum62mt
blosum62mt2
blosum80mt
paml20mt
paml60mt
pam250mt
pam350mt
md_l20mt
md_120mt
md_120mt
id_mat

If the program does not recognize the *matrice_name* as a default matrix, it assumes it comes from a file. A file containing a matrix must have the following format:

order: ABCDEFGHIKLMNQPRSTVWXYZ
$
vl
v2 v3
v4 v5 v6
....
$

The matrix must be set for maximisation (identity with the highest score). To align DNA, the identity matrix must be used, where the Amino Acid GCTANX are considered as nucleotides.

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 the trees in the format used by ClustalW. Using Clustal is the easiest way to obtain a tree. But ClustalW tree are unrooted. As it is preferable to use a rooted tree, user may root theirs in retree, a program of the 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.