Volume 14 Number 1 1986 Nucleic Acids Research

A multiple sequence alignment program

Eric Sobel and Hugo M.Martinez*

Department of Biochemistry and Biophysics, University of California, San Francisco, CA 94143, USA

Received 16 July 1985

ABSTRACT

A program[1] is described for simultaneously aligning two or more molecular sequences which is based on first finding common segments above a specified length and then piecing these together to maximize an alignment scoring function. Optimal as well as near-optimal alignments are found, and there is also provided a means for randomizing the given sequences for testing the statistical significance of an alignment. Alignments may be made in the original alphabets of the sequences or in user-specified alternate ones to take advantage of chemical similarities (such as hydrophobic-hydrophilic).

INTRODUCTION

We previously reported on a method for efficiently finding subsequences which repeat in one or more sequences [1] and indicated that this method could be the basis for doing multiple sequence alignments. We here report on such an extension.

The report is divided into two main parts, the first dealing with an analytical description of the method for those interested in how it works, and the second devoted to a user-oriented description with examples.

ANALYTICAL DESCRIPTION OF THE METHOD

A subsequence of contiguous elements in a sequence (bases in nucleic acids or amino acids in proteins) is referred to as a segment, and a segment which occurs in two or more sequences is called a common segment, repeat or simply 'region'. For example, if the segment ACCTG occurs at position 12 of sequence #1 and at positions 45 and 100 of sequence #2, then there are two common segments which ACCTG gives rise to. One corresponds to the position pair (12,45) and the other to the position pair (12,100). If, in addition, we are concerned with three sequences and ACCTG also occurs at positions 20 and 50 of sequence #3, then ACCTG would give rise to the regions corresponding to the position triples (12,45,20),(12,100,20), (12,45,50) and

[1] Program available on 9-track tape in UNIX tar format at nominal distribution cost (mailing and tape).

Downloaded from http://nar.oxfordjournals.org/ at BIU Montpellier on March 12, 2014

(12,100,50).

As will be appreciated, the number of common segments which two or more sequences have in common is proportional to the product of the length of the sequences; hence the considerable computational complexity that would result unless some constraints are imposed. In practice, we therefore limit the length of a common segment to be above a specified size. This helps avoid the 'combinatorial explosion' which would otherwise occur in the number of common segments and is also aimed at getting above the 'noise' level.

Given a method for finding regions above a specified size in two or more sequences, alignment of the sequences amounts to piecing together some of these regions. To do this, it is convenient to introduce a partial ordering among the regions in the sense that region X 'precedes' region Y if the positions of X precede the corresponding positions of Y by at least the length of region X. Thus, suppose X is of length 5 and has the position triple (100,130,80), while Y has the position triple (104,140,90). There is an overlap relative to the position in the first sequence. But if the length of X were 4, then X would be said to precede Y. In general, if there is no overlap relative to corresponding positions when region lengths are considered, then one region will precede the other.

By an 'alignment' of two or more sequences relative to a set of regions is meant a sequence X1,X2,..,Xn of regions such that $X1 < X2 < \dots < Xn$. An alignment can be given a score, such as the sum of the lengths of the regions comprising it, and an 'optimal' alignment is one which has the largest possible score. Our program finds an optimal alignment and all the alignments which have a score within a specified range of the optimal, that is, 'near-optimal' alignments.

Procedure for Finding an Optimal Alignment

As an ordered sequence of regions, an alignment is like a 'path' made up of a sequence of steps, and to each path there can be assigned a score based on the length of each step and perhaps a penalty for some inter-step characteristic, such as a 'gap' that would arise when two successive regions require some insertions between them to maintain registration of the sequences. Among possible steps which can be pieced together, we are therefore seeking a path with a best score. This amounts to solving the 'longest path' problem for which many algorithms have been described in the computer science literature. We simply implement one of these as follows.

Let W(X) be the best score which can be achieved if the alignment were to start with region X. This is recursively defined as

$$W(X) = \max\{\mathrm{length}(X)+\mathrm{penalty}(X,Y)+W(Y)\}$$

in which the maximization is carried out for all the regions Y such that X precedes Y.

Downloaded from http://nar.oxfordjournals.org/ at BIU Montpellier on March 12, 2014

The term 'penalty(X,Y)' denotes the penalty incurred for a gap between the regions X and Y.

Procedure for Finding the Near-Optimal Alignments

Once the optimal alignment under the chosen scoring scheme is found, the program can also find all the alignments with scores within a user-specified range of the best alignment's score. Suppose B is the score of the optimal alignment, b% the user-specified range and $N = B-(bB/100)$. The method used to find these near-optimal alignments is that of Byers and Waterman [2]. The remarkable aspect of their approach is that it requires very little additional calculation and storage space once the best alignment is known. Indeed very quickly and without wasted effort one finds precisely those alignments with scores $>= N$ using the information necessary to find the optimal alignment. For each region r it was necessary to find and store the sum of the score of r plus the best score possible of an alignment of the sequences between r and the end of the sequences. Call this sum M(r).

To find the near-optimal alignments use the following procedure:

Set $i = 1$.

Do step i: Find the score, call it W_i, from the beginning of the sequences up to the (i-1)-st region in the alignment under construction. (Thus W_1 will always equal zero.) Now consider the set S_i containing all regions which could possibly occur next in this maximal alignment. If S_i is empty, then a maximal alignment has been found and the procedure outlined in the next paragraph is implemented. If S_i is nonempty, then we consider all regions r in S_i such that

$$W_i + M(r) + \text{(score of nonaligned portion between r and the (i-1)-st region in the alignment)} >= N. \qquad (1)$$

Call the set of such regions R_i. It will be shown presently that S_i nonempty implies R_i nonempty. The next stage is to pick out that member of R_i with largest M(r) + (score of nonaligned region between r and the (i-1)-st region in the alignment) and make this region the i-th region in the alignment. For each other member of R_i, say r', push the ordered pair (r',i) onto the stack. (A stack, also called a pushdown store, is simply an ordered block of memory which may later be accessed in a last-in-first-out manner.)

Do Step i+1.

Once S_i is empty for any i, the alignment that has been constructed is stored. (The sole exception being the first time S_i is empty, the optimal

Downloaded from http://nar.oxfordjournals.org/ at BIU Montpellier on March 12, 2014

alignment is reproduced.) Next, the top element is popped off the stack, i.e., the last element stored in the stack, say the ordered pair (r0,j), is retrieved. Now the j-th region in the alignment is replaced with region r0. Next, reenter the first half of this procedure at Step j+1, continue as before until S_i is empty for some i. The second half of the procedure is then repeated until both the stack and S_i are empty.

It remains to show that once the above procedure ends, exactly those near-optimal alignments with scores within the user-specified range have been found and stored. First, we know there is an alignment with score B, that is with score >= N. Thus at the initial Step 1 if S_1 is nonempty then so is R_1. Furthermore, at each stage regions are only put into position in the alignment if they satisfy equation (1) for that region and that position. Thus if repeat r0 has just been installed as the j-th region in the alignment, then there must be an alignment from r0 to the sequences' ends with score >= N - {W_j+(score of r0)+(score of non-aligned portion between r0 and the (j-1)-st region in the alignment)} =N - W_{j-1}. Since such an alignment of score >=N-W_{j+1} exists, in the subsequent Step j+1, S_{j+1} nonempty must imply R_{j+1} nonempty. This is precisely the same reasoning as at the initial Step 1. Thus whenever S_i is nonempty then R_i is nonempty. This shows that the above procedure cannot 'break-down'; that is, it proceeds as outlined above.

Equation (1) requires that each alignment found must be within the user-specified range. Hence, at each stage only those regions which will lead to an overall score >=N may be installed.

Finally, by using the computer science principle of 'depth first searching', we show that all the alignments with scores >= N have been found. Notice that at each Step i one either installs or saves for later installation all regions which could possibly be used, given the alignment up to position i, as the i-th region in the alignment, and still have the overall score >= N. Through the use of the stack, and its last-in-first-out accessibility, one installs a region, say region r at position i, and then finds all alignments with score >= N which vary only those regions in positions > i. Only then does one replace r with, say, r'. Again the possibilities for positions > i are searched but now with r' in position i, and so on. Eventually the last alternate region for position i will have been popped off the stack and yet again all the possibilities for positions > i searched. At this point the next element on the stack will be an alternate for the region in position i-1. Unless of course i = 1, in which case the stack will be empty. This simple searching technique, once the stack and S_i are both empty, will have recorded exactly those near-optimal alignments with score >= N. These alignments are then ordered, their scores reported to the user, and any or all are displayed as output.

Downloaded from http://nar.oxfordjournals.org/ at BIU Montpellier on March 12, 2014

USERS' DESCRIPTION OF THE METHOD

malign is a completely interactive program for finding alignments among two or more sequences. It works by using the **qrepeats** algorithm [1] to find common subsequences among your sequences, then aligning these common regions. The optimal alignment is written in the file malign.out. Each sequence must be in its own file with no other characters (except spaces). **malign** does not distinguish between upper and lower case letters. **malign** finds the alignment of the sequences that maximizes a similarity measure. By insertions of ., the sequences are made to match up as well as possible. For example:

```
acd.  [ABCDAB] .... [DCBAAC] d [ADCB] cddbabacddabc [CCCBAAB] ...
abdd  [ABCDAB] bbd. [DCBAAC] b [ADCB] ............. [CCCBAAB] ddc
bd..  [ABCDAB] cdba [DCBAAC] b [ADCB] cb........... [CCCBAAB] ...
```

malign can translate your sequences into each alphabet for which you can specify a translation file of exactly two lines. The first line is the original alphabet, then under each character is its translation into the new alphabet. For example, amino acid to hydrophilic-hydrophobic:

```
PGASTDENQKHR    FYMCMVLI
000000000000    11111111
```

You may then choose the alphabet in which the sequences will be aligned. A match in one alphabet may be a mismatch in another. Thus one must specify an alphabet to compute the optimal alignment for that alphabet. All alignments printed are printed in all alphabets supplied. **malign** computes the significant repeat length for the given number of sequences. The alignment procedure first finds all repeats, i.e. segments common to all sequences, of length >= the user-specified minimum repeat length. **malign** then forms the alignments using these repeats as building blocks. Thus **malign** will not recognize any similarity of length shorter than this minimum. This is an advantage due to the statistical insignificance of very short repeats and their large numbers (which would slow **malign** down). However some smaller repeats may combine to form an area of significantly high similarity. For instance a lengthy segment with more than 90% similarity might be considered significant, as in:

```
[AGTCAGTC] a [GTCAGTC]
[AGTCAGTC] g [GTCAGTC]
```

Hence the minimum length chosen should not only be small enough to include all repeats significant both alone and as above, but should also be large enough so that the number of repeats allows for a reasonable running time. In any case once a set of repeats has been found (the repeats previously found are remembered) one may reset the minimum length to obtain a more desired set. You may ask the program to write a list of the repeated regions which were used to form the alignment on a file called

Downloaded from http://nar.oxfordjournals.org/ at BIU Montpellier on March 12, 2014

malign.rep. If you know *a priori* that the sequences are homologous at their left or right ends you may want to score the gaps at the ends of the sequences. You may also decide whether these gaps should be penalized in the same way as an interior gap.

Malign finds the optimal alignment, i.e., the alignment which maximizes a similarity measure between the sequences. At present this similarity measure is found by adding together the lengths of the repeats in the alignment and then subtracting the score of each gap introduced by the alignment. A gap is any positive number of contiguous insertions-or-deletions needed to make the sequences align. The score of a gap is **constant1 + (n * constant2)** where **n** is the number of insertions-or-deletions in the gap, **constant1** is the user specified penalty for the existence of the gap, and **constant2** is the user-specified penalty per insertion-or-deletion in the gap. By making the gap's score a function of its size the alignments with more clustered repeats are favored. The more obvious advantage is that larger gaps are more heavily penalized. Recommended values are: **constant1**, 1.0 to 3.0 and **constant2**, 0.1 to 0.3. The larger the constants, the larger an off-axis repeat will have to be before it will be introduced. An example of the scoring of a non-aligned region, with constant1 = 2.0 and constant2 = 0.2:

```
] bdc... [
] c..... [
] cabdaa [
```

This gap has score= 2+(8 * 0.2) = 3.6

You may then test the significance of the optimal alignment by running the significance test which performs statistical analysis of the results by scrambling the sequences and then aligning them again. These randomized optimal alignments are then used to approximate a normal distribution, and thus find the probability that this score, or better, would occur by chance. You then choose how many of these randomized optimal alignments should be written to the file malign.rdm. **malign** normally finds the best (optimal) alignment. In addition, it can find the near-optimal alignments within a limit specified by significance or score. Each optimal or near-optimal alignment found is maximal in the sense that no repeat could be added to these alignments without lowering their score. As usual, after the near-optimal alignments have been found, one may start again with a different near-optimal limit; thus, initially the limit should be kept fairly close to the optimal alignment's score. Example: I have decided to run **malign** to compare 3 protein sequences; namely hemoglobin alpha chain of carp, HACA, goldfish, HAGY, and Port Jackson shark, HARKJ. I want the alignment to be computed in another alphabet, hydrophilic/hydrophobic. As the program is running it computes the significant repeat length to be >= 3. So I ask the

Downloaded from http://nar.oxfordjournals.org/ at BIU Montpellier on March 12, 2014

program to find the repeated regions of minimum length 2. I do not want to score the gaps at the ends of the sequences. The gap penalty per scored gap and per scored insertion-or-deletion is 1. To test the significance of my **malign** results, I choose to randomize the order of the elements in all 3 sequences 10 times, and then run the **malign** program on each randomized set. The program then prints the repeated regions and their respective statistical significance, the number of aligned elements, scored gaps, scored insertion-or-deletions and the total score. The program also prints the optimal alignment score for each randomization and computes the probability that such an alignment would occur by chance.

Example:

```
% malign

              MALIGN : Multi-sequence Alignment Program

How many sequences are there to be aligned ? 3

File name of sequence #1 ? HACA

File name of sequence #2 ? HAGY

File name of sequence #3 ? HARKJ

Do you wish to have the alignment computed or output
in alphabets other than that in which the sequences were entered ? yes

How many alphabets, INCLUDING the original,
do you wish to work with ? 2

The original alphabet of the sequences is alphabet #0.

Name of alphabet #1's translation file ? trans

WARNING: 8 characters were UNTRANSLATABLE by this translation file.
They were left unchanged.

What is the number of the alphabet in which the sequences should be
aligned ( `0' is the original alphabet ) ? 0

For these sequences it is expected that the longest common segment,
or repeat, which would occur by chance would have length about 2.02
with standard deviation 0.25 .  Thus for a repeat by itself to be
very significant it should have length >= 3

What is the minimum repeat length you wish used in the alignments ? 2
```

Downloaded from http://nar.oxfordjournals.org/ at BIU Montpellier on March 12, 2014

```
Repeats found so far:
      repeat              number of repeats statistical
      length               of this length         significance
        2                  292          0.5331
        3                   11          0.0000
        5                    2          0.0000

The total number of repeats found was.........: 305

Are you satisfied with these 305 repeats ? yes

Would you like a list of the repeats ? no

Should the left terminal gap be counted
in the overall score of the alignment ? no

Should the right terminal gap be counted
in the overall score of the alignment ? no

What do you wish the penalty per scored gap to be
(this penalty weight should be entered as a positive number) ? 1

What do you wish the penalty per scored insertion-or-deletion to be
(this penalty weight should be entered as a positive number) ? .1

The optimal alignment has:
      Number of aligned elements................:    35
      Number of scored gaps.....................:     4
      Number of scored insertions-or-deletions..:    13
      Score.....................................:    29.700

Do you wish to run the significance test ? yes

Do you wish to randomize the order of the elements in sequence #1
in addition to all the other sequences ? yes

How many randomizations should there be
to approximate the normal distribution ? 10

How many of the about-to-be-found `randomized' optimal
alignments should be written to `malign.rdm' ? 0

Randomization # 1 has optimal alignment score:    4.700
Randomization # 2 has optimal alignment score:    2.500
Randomization # 3 has optimal alignment score:    2.800
Randomization # 4 has optimal alignment score:    2.900
Randomization # 5 has optimal alignment score:    4.600
Randomization # 6 has optimal alignment score:    3.000
```

Downloaded from http://nar.oxfordjournals.org/ at BIU Montpellier on March 12, 2014

```
Randomization # 7 has optimal alignment score:    3.000
Randomization # 8 has optimal alignment score:    3.000
Randomization # 9 has optimal alignment score:    2.700
Randomization #10 has optimal alignment score:    3.000

The optimal alignments obtained amongst the 10 randomizations had:
      Mean number of aligned elements..:    4.200
      Mean number of scored gaps.......:    0.700
      Mean number of scored insertions.:    2.800
      Mean score.......................:    3.220
      Score standard deviation.........:    0.771

Also the no. of repeats found averaged...:  101.100
Longest repeat, when found, averaged.....:    2.600

Are you satisfied with this number of randomizations ? yes

We may define the STATISTICAL SIGNIFICANCE OF AN ALIGNMENT with
score X to be the probability that an alignment with score > X
would occur `by chance'.

Therefore we estimate the STATISTICAL SIGNIFICANCE
OF THE OPTIMAL ALIGNMENT
(to four decimal places) is.............:     0.0000

Do you wish to find near-optimal alignments of your original sequences? no

The optimal alignment is written in the file `malign.out'
```

The program has computed that the above alignment is very significant, and the probability of this alignment to happen by chance is effectively zero. Now I examine the output file malign.out. The file contains the alignment of my three proteins in two alphabets. Shown here:

```
% cat malign.out
The minimum repeat length allowed was 2.

The expected longest repeat occurring by `chance'
was of length 2.02 with standard deviation 0.25

The weighting scheme used here was:
      Neither end gap was counted toward the overall score.
      Weight per gap was............................:1.000
      Weight per insertion-or-deletion was..........:0.100

The number of repeats found amongst the 3 given sequences was......305
```

Downloaded from http://nar.oxfordjournals.org/ at BIU Montpellier on March 12, 2014

```
                    **** THE OPTIMAL ALIGNMENT ****

 The optimal alignment under these weights has:
      Number of aligned elements................:    35
      Number of scored gaps.....................:     4
      Number of scored insertions-or-deletions..:    13
      Score.....................................:    29.700

 SEQUENCE NAMES, THEIR LENGTHS and % ALIGNMENTS

 # 1:   HACA              142          24.65%
 # 2:   HAGY              145          24.14%
 # 3:   HARKJ             148          23.65%

 TABLE OF THE ALIGNED REPEATS - with statistical significance, length
                          and starting position in each sequence

 statistical     repeat    starting position in sequence:
 significance    length              # 1    # 2     # 3

   0.5331           2          3      3      7
   0.0000           5         25     25     32
   0.5331           2         31     31     38
   0.0000           3         34     34     41
   0.5331           2         39     39     46
   0.5331           2         42     42     49
   0.5331           2         58     61     66
   0.0000           3         75     78     82
   0.0000           5         95     98    102
   0.5331           2        120    123    126
   0.0000           3        127    130    133
   0.5331           2        137    140    143
   0.5331           2        141    144    147
                  ---
        Total :    35

 ALIGNMENT MAP #1 - showing distance between repeats and their lengths

   2-[ 2]- 20-[ 5]-  1-[ 2]-  1-[ 3]-  2-[ 2]-  1-[ 2]- 14-[ 2]- 15-[ 3]
   2-[ 2]- 20-[ 5]-  1-[ 2]-  1-[ 3]-  2-[ 2]-  1-[ 2]- 17-[ 2]- 15-[ 3]
   6-[ 2]- 23-[ 5]-  1-[ 2]-  1-[ 3]-  2-[ 2]-  1-[ 2]- 15-[ 2]- 14-[ 3]

 - 17-[ 5]- 20-[ 2]-  5-[ 3]-  7-[ 2]-  2-[ 2]-  0
 - 17-[ 5]- 20-[ 2]-  5-[ 3]-  7-[ 2]-  2-[ 2]-  0
 - 17-[ 5]- 19-[ 2]-  5-[ 3]-  7-[ 2]-  2-[ 2]-  0

 ALIGNMENT MAP #2 - showing sequences and aligned repeats [in brackets]
                  - in each given alphabet
```

Downloaded from http://nar.oxfordjournals.org/ at BIU Montpellier on March 12, 2014

```
In alphabet in which alignment was found:

  0  sl....  [SD] kdkaavkiawakispkaddi... [GAEAL] g [RM] l [TVY] pq [TK] t [
  0  sl....  [SD] kdkavvkalwakigsradei... [GAEAL] g [RM] l [TVY] pq [TK] t [
  0  ststst  [SD] ysaadraelaalskvlaqnaeaf [GAEAL] a [RM] f [TVY] aa [TK] s [

 41  YF] ahwadlspgsgpvk...  [HG] kkvimgavgdavski [DDL] vgglaslselhasklrv [DP
 41  YF] shwsdlspgsapvk(k,  [HG] k)timgavgdavski [DDL] vgalsalselhafklri [DP
 48  YF] kdykdftaaapsika..  [HG] akvvtalakacdhl. [DDL] kthlhklatfhgselkv [DP

 96  ANF] kilanhivvgimfylpgdfp [PE] vhmsv [DKF] fqnlala [LS] ek [YR]
 99  ANF] kilahnvivvigmlfpgdft [PE] bhmsv [DKF] fqnlala [LS] ek [YR]
103  ANF] qylsyclevalavhltefs. [PE] thcal [DKF] ltnvche [LS] sr [YR]

In alphabet #1:

  0  01....  [00] 000001010w0010000001... [00001] 0 [01] 1 [011] 00 [00} ,0 [
  0  01....  [00] 000011001w0010000001... [00001] 0 [01] 1 [011] 00 [00] 0 [
  0  000000  [00] 10000000100100110000001 [00001] 0 [01] 1 [011] 00 [00] 0 [

 41  11] 00w00100000010...  [00] 001110010001001 [001] 10010010010000101 [00
 41  11] 00w00100000010(0,  [00] 0)0110010001001 [001] 10010010010010101 [00
 48  11] 001001000000100..  [00] 00110010001001. [001] 00010010010000101 [00

 96  001] 01100011101111100010 [00] 10101 [001] 1001010 [10] 00 [10]
 99  001] 01100011111011100010 [00] b0101 [001] 1001010 [10] 00 [10]
103  001] 0110111010101010010. [00] 00101 [001] 1001100 [10] 00 [10]
```

The program has computed that there is about 24% homology between these 3 proteins; there is definitely something of significance here. The alignment between HACA, HAGY, and HARKJ, looks most promising. The alignment is created by aligning amino acid 1 of all three proteins. The distances between gaps is also very similar.

DISCUSSION

The program 'malign' has been in extensive use for nearly two years. It is written in the C language and readily portable to computers with sufficient memory. The compiled code on our VAX 11/750 under UNIX 4.2 requires a minimum of 80,000 bytes plus dynamically allocated storage depending on the number and size of the sequences handled.

Of the suggested directions for improvement, special attention is currently being given to relaxing the requirement that a common segment is indeed common to all the sequences. For instance, specifying that commonality correspond to any combination of four sequences when five are being aligned, allows for bad sections in one or more of the sequences which would otherwise render the alignment very poor. Such a relaxa-

Downloaded from http://nar.oxfordjournals.org/ at BIU Montpellier on March 12, 2014

tion in commonality has already been incorporated into a program called 'vurepeats' which is an improved version of 'qrepeats' originally reported on. It lists common segments found in all of N specified sequences as well as in all combination of n of these or more for $n < N$. This is an invaluable feature when trying to find out what several sequences have in common when what is 'common' may not be perfect.

Another direction for improvement is a weighting scheme in the scoring of an alignment which favors a common seqment in proportion to its statistical significance rather than just in proportion to its length, a feature which would be especially important when aligning proteins.

The method we have used for multiple sequence alignment employs what is now referred to as the 'regions' approach which relaxes the stringent resolution found in the dynamic programming approach to sequence alignment. A thorough comparison of these two approaches may be found in [3] as well as in [4]. The reader may also wish to consult [5] for the presentation of a multiple sequence alignment procedure based on a generalization of the Needleman and Wunsch dynamic programming approach to the alignment of two sequences.

ACKNOWLEDGEMENT

This work was in part supported by NSF grant #PCM-8314716.

*To whom correspondence and reprint requests should be addressed

REFERENCES

1. Martinez, H. M. (1983) Nucl. Acids Res. 11, 4629-4634.
2. Byers, T.H. and Waterman, M. S. (1984) Operat. Res. 46:473.
3. Waterman, M.S. (1984) Bull. Math. Biol. 46:473-500.
4. Davison, D. (1985) Bull. Math. Biol. 47:437-475.
5. Murata, M; J.S. Richardson; Joel L. Sussman (1985) Proc. Natl. Acad. Sci. 82:3073-3077.

Downloaded from http://nar.oxfordjournals.org/ at BIU Montpellier on March 12, 2014