TR-I-0237

Evaluation of metrics for text comparison with implication for recognition

Nathalie Delfosse Tsuyoshi Morimoto

1992.1

Abstract

This report describes work done on some texts from the ATR database, evaluating the relationship between text similarity and speech recognition accuracy. The work is in two parts: the first part is an application of the quantification theory IV, using a set of Japanese conversation texts with a corresponding set of English texts, and testing different definitions for the metric of the quantification. The correlation between distances is measured, as is the correlation between Japanese and English texts. Bigrams and trigrams (both a simplified form of syntax) have strong correlations in Japanese and in English, but vocabulary and simplified syntax appear only weakly correlated.

In the second part of this report, only Japanese texts are used. Two probabilistic grammars have been trained on different sets of conversation texts. For one target text, the recognition rate is higher if the distance to the training set is smaller. For one training set, the distance of the target texts to this set and the recognition rate seem not to be strongly correlated.

ATR 自動翻訳電話研究所
ATR Interpreting Telephony Research Laboratories

Contents

Abstract

1 Introduction 2

2 Similarity among metrics between texts in English and Japanese 3

2.1 Brief summary about Quantification Theory IV
2.1.a-Description of the method
2.1.b-Possible exploitation of Quantification Theory IV

2.2 Description of the linguistic corpus

2.3 Correlations between metrics and between texts
2.3.a-Definitions of the metrics
2.3.b-Matrix of distances
2.3.c-Correlation between matrixes
2.3.d-Correlation results table

2.4 Interpretation of the results

2.5 Discussion

3 Relationship between text similarity and recognition accuracy 14

3.1 General method

3.2 Probabilistic grammar

3.3 Training texts and target texts

3.4 Recognition results

3.5 Distances between the target texts and the training texts

3.6 Relationship between distances and recognition rates

4 Conclusion 22

References

Appendix

1 Introduction

In speech recognition using a language model, it is important to build the most suitable models as possible. Some language models are based on a stochastic grammar, which is supposed to be able to parse any sentence in its database. It is important to know how to choose relatively short and well adapted samples of linguistic texts for training such a grammar, because in practice it is impossible to find samples which really cover all the aspects of a language. In this report, we experiment a method based on metrics. The Quantification Theory IV proposes different kinds of metrics and ways of using test spaces for choosing text samples. In the first part of this report,we compare these metrics in order to find the characteristics of text similarity. In the second part, we try to find a relationship between the recognition rates of target texts and their distances to the training text sets for the stochastic grammar.

2 Similarity among metrics between texts in English and Japanese

2.1 Brief summary about Quantification Theory IV

2.1.a-Description of the method

The quantification theory IV (QIV) is one of the quantification methods proposed by Chikio Hayashi.

The goal of QIV is to create a metric between linguistic texts in order to analyse and distinguish them quantitatively. It is useful for selecting suitable linguistic text samples from which linguistic information can be extracted to build a linguistic model.

A similarity among data must be defined, that can be quantified. This similarity can be defined in different ways.

Let t_i and t_j be two linguistic texts. Two examples of the metric will be described.

Example 1 : topical metric.

Let P_i and P_j be the probability distributions of the nouns in t_i and t_j respectively. The distance d_w between the two texts can be defined as:

$$d_w\ (t_i, t_j) = \left(\sum_{t_i U t_j} \{ P_i(w) - P_j(w) \}^2 \right)^{1/2}$$

(where w represents a word belonging to the vocabulary used in the texts t_i and t_j). If t_i and t_j are identical, then $d_w\ (t_i, t_j)$ is 0.

Example 2 : grammatical metric.

The texts are supposed to be able to be parsed using a linguistic grammar. Such a grammar contains a set of linguistic rules, and each parsed sentence can be considered as a sequence of

rewriting grammar rules. In the same way as in example 1, it is possible to define the probabilities P_i and P_j of the grammar patterns in the texts t_i and t_j respectively. The grammatical metric d_g between the two texts can be defined as:

$$d_g\ (t_i,t_j) = (\Sigma_G \{ P_i(g) - P_j(g) \}^2)^{1/2}$$

where G represents the grammar used and g a grammar pattern belonging to G. We also have for this metric

$$0 <= d_g\ (t_i,t_j) <= 2^{1/2}$$

2.1.b-Possible exploitation of Quantification Theory IV

Quantification theory IV can be used for representing a set of linguistic texts in an n-dimensional text space. The axes can be determined with the n eigenvectors having the n largest eigenvalues of the symetrical matrix which contains the distances from every text of the set to the others. It seems that each axis tends to represent a special topic.

By then using a mesh-method, suitable text samples can be selected from the initial set, so that they are uniform with respect to unnecessary elements other than grammatical properties; this can be used for building a stochastic grammar.

2.2 Description of the linguistic corpus

Conversation texts from the ATR Dialogue Database (ADD) were used. The main topics of these texts are travel arrangement and conference registration. Some texts were input by keyboard, and in this case the data are rather artificial. The others are

taken from telephone conversations and are more natural. There is a difference in the languages of these two types of texts.

In order to obtain the comparisons, a large set of texts, with both Japanese and English versions was necessary. Morphological information was also necessary. Only one set of texts met all these conditions: the morphologically analysed conversation texts about travel arrangement, in Japanese and English, which were input by keyboard. These texts are contained in the unix files

/data3/MORPH/bunout/key/Kxxxx.DEC (for Japanese texts) and

/data3/MORPH-E/key/Kxxxx.SHP (for English texts)

For these texts, the number xxxx is from 3179 to 3425 -except 3310. Each word has been morphologically analysed.

Example (from the English text K3179):

```
K3179_000250_000350_003710_003710_leaving_leave_20_EB
  1      2      3      4      5      6      7    8  9
```

1: conversation ID
2: utterance ID
3: sentence ID
4: word ID1
5: word ID2
6: surface form
7: standard form
8: part of speech
9: conjugation

The analysis of Japanese words is about the same.

2.3 Correlations between metrics and between texts

2.3.a-Definitions of the metrics

QIV was applied to the set of texts, and different kinds of metric were tried. The same general formula can be used for describing them:

$$d_X\,(ti,tj) = \left(\sum X(ti) \cup X(tj) \; \{ Pi(x) - Pj(x) \}^2 \right)^{1/2}$$

In this formula, x represents an element of a text (like a word, a grammar pattern, a bigram...) and X(t) the set of those elements appearing in the text t.

Six metrics were applied. Among them, two types can be distinguished: the metrics concerning the *vocabulary* employed in the texts and the ones concerning *small syntaxes.*

Vocabulary metrics:

vi) "all words" metric is based on a non-restricted vocabulary. All the words of the texts are taken into account.

vii) "ind. words" metric is limited to independent words (in this work, words which seemed to be the most important were considered: nouns, verbs, adjectives and adverbs) and is supposed to be more a *topical* metric,.

Small syntax metrics: Distances based on bigrams and trigrams of surface forms were used. For these bi and trigrams, all the sequences of two or three consecutive words were considered, but because some surface expressions have a very low probability of appearing in some texts, distances based on bigrams and trigrams of grammatical categories were tested too. (There are about 30 categories corresponding to part of speech). Four metrics were tested

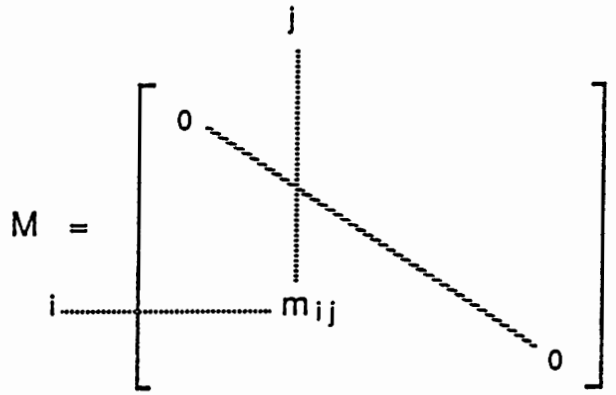
vi) "bigrams" vii) "trigrams"

viii) "cat. bigrams viiii) "cat. trigrams"

Correlations between these six metrics are shown in table 1.

2.3.b-Matrix of distances

Let { t_1, ..., t_n } be a set of linguistisc texts, and d a metric. The distances from each text to the others can be contained in a symmetrical matrix $M = [m_{ij}]_{1<=i<=n;\ 1<=j<=n}$ defined in this way:



$$m_{ij} = d(t_i, t_j) = m_{ji}$$

For each set of texts, the matrixes corresponding to each metric were generated, providing 12 matrixes in all. About 50 hours are needed for generating one matrix for 100 texts. Because of this calculation time, only 100 texts (from K3179 to K3278.) were used. The program to do this, written in C-language, is described in the appendix. The algorithm for producing the matrixes grows in n^2, and many reading and comparing operations are done with each text. That is the reason why this program is rather slow

As the matrixes are very large, they will not be included in this report.

2.3.c-Correlation between matrixes

As two corresponding sets of texts were used, and as the texts were always considered in the same order, the matrixes have special properties and they can be compared by statistical methods. Only the upper parts need to be compared.

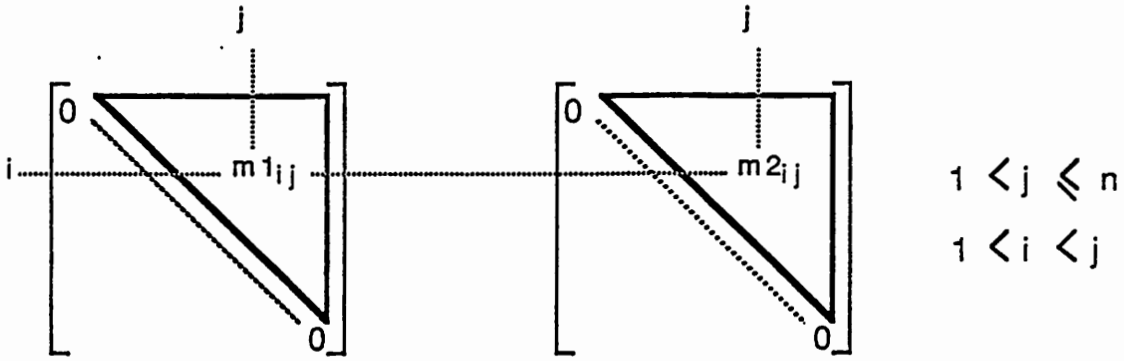
Comparison of Japanese and English

In this part, we compare pairs of matrixes, both obtained using the same metric, one for Japanese texts, one for English texts. Let $J = [\ j_{ij}\ ]$ and $E = [\ e_{ij}\ ]$ represent them. Then corresponding elements j_{ij} and e_{ij} represent the distances between two Japanese texts and between their English translations. It is then possible to calculate the correlation between the two sets for the chosen metric

$$\varrho = \mathrm{cov}(\ j_{ij}\ ,\ e_{ij}.) \,/\, (\ \sigma(\ j_{ij}\ ) \times \sigma(\ e_{ij}\ )\ ) \qquad 0 <= \varrho <= 1$$

$$(\ 1<j<=n;\ \text{for the elements } j_{ij} \text{ or } e_{ij},\ 1<i<j\ )$$

This number indicates the similarity that exists between Japanese and English for this metric



Comparison of two metrics

Let d1 and d2 be two metrics, and $M1 = [\ m1_{ij}\ ]$ and $M2 = [\ m2_{ij}\ ]$ be the matrixes for these metrics, generated with the Japanese or the English set of texts. This time, two corresponding elements represent the distances betwen two texts obtained with two different metrics. Then the correlation calculated in the same way indicates the similarity between the metrics

$$\varrho = \mathrm{cov}(\ m1_{ij}\ ,\ m2_{ij}\ ) \,/\, (\ \sigma(\ m1_{ij}\ ) \times \sigma(\ m2_{ij}\ )\ )$$

2.3.d-Correlation result table

Table1 summarizes the correlations obtained with the method described above. One hundred texts are used for each language, from K3179 to K3178. (Important results are in heavy type.)

		J	J	J	J	J
	CORRELATION (e)	All words	Ind. words	Bigrams	Trigrams	Cat. bigrams
J	All words	X	0.6283	0.5385	0.4504	-
J	Ind. words	0.6283	X	**0.4571**	0.3763	0.2895
J	Bigrams	0.5385	**0.4571**	X	**0.8943**	/
J	Trigrams	0.4504	0.3763	**0.8943**	X	/
J	Cat. bigrams	-	0.2895	/	/	X
J	Cat. trigrams	-	0.4496	/	/	0.7798
E	All words	0.5616	/	/	/	/
E	Ind. words	/	**0.6759**	/	/	/
E	Bigrams	/	/	**0.8213**	/	/
E	Trigrams	/	/	/	**0.9127**	/
E	Cat. bigrams	/	/	/	/	0.3835
E	Cat. trigrams	/	/	/	/	/

Table 1: Correlations for six metrics. 100 texts used

J	E	E	E	E	E	E
Cat trigrams	All words	Ind. words	Bigrams	Trigrams	Cat. bigrams	Cat. trigrams
-	0.5616	/	/	/	/	/
0.4496	/	**0.6759**	/	/	/	/
/	/	/	**0.8213**	/	/	/
/	/	/	/	**0.9126**	/	/
0.7798	/	/	/	/	0.3835	/
X	/	/	/	/	/	0.6424
/	X	0.6966	0.5367	0.3824	-	-
/	0.6966	X	**0.6180**	0.4738	0.5335	0.5493
/	0.5367	**0.6180**	X	**0.8440**	/	/
/	0.3824	0.4738	**0.8440**	X	/	/
/	-	0.5335	/	/	X	0.7880
0.6424	-	0.5493	/	/	0.7880	X

X: no possible correlation
/: no interesting correlation
-: no result

J: Japanese
E: English

2.4 Interpretation of the results

Some interesting correlations can be seen in table1:

J-bigrams/J-trigrams	0.8943
E-bigrams/E-trigrams	0.8440
bigrams J/E	0.8213
trigrams J/E	0.9127
J-ind words/J-bigrams	0.4571
E-ind words/E-bigrams	0.6180

It seems then that vocabulary, represented by the metric based on independent words, and syntax, represented by bigrams and trigrams (both kinds of simplified syntax) are two independent characteristics. The correlation between bigrams and trigrams is strong for both languages, and it can be noticed that the correlations between Japanese and English for these metrics are strong too. But the correlation between metrics based on words and bigrams is rather weak (the one between metrics based on words and trigrams is weak too, as it can be seen in table1).

Trying to associate these results to the use of a grammar, we can say that the simplified syntaxes correspond to non-terminal derivation rules (A --> B), whereas vocabulary characteristics correspond to pre-terminal symbol derivation rules. (W --> (symbol)). These two characteristics should be considered separately.

2.5 Discussion

It might be possible to improve the method, or the definitions of the metrics, in order to obtain better probability distributions. This was not done in the present study because the time needed for testing one possibility was too long, and only a limited number of tests could be done

3 Relationship between text similarity and recognition accuracy

3-1 General method

This part summarizes experiments done with some texts from the ADD database. The goal of these experiments was to try to find a possible relationship between text similarity and speech recognition accuracy.

At first, a probabilistic grammar had to be initialized with a set of texts. This grammar was then used in recognition experiments, and the target texts were compared to the training texts with the metrics described in part 2, so that any correlation between testual similarity and recognition results could be seen.

3-2 Probabilistic grammar

A grammar was iteratively trained on a set of texts to obtain the probabilities for the grammar rules. The training texts needed to be first parsed. The grammar, dp10org.gra. contains 1974 rules. The training program (written by Kenji Kita) generated both probabilistic rules and a grammar table. The probabilities were then copied into the grammar used for speech recognition (dp10sp.gra).

3-3 Training texts and target texts

The recognition program was only capable of processing a limited set of seven Japanese conference registration texts. They can be found in the unix files

/data3/MORPH/bunout/ifg/Ixxxx.DEC (where xxxx represents a number from 8013 to 8019). These texts have been morphologically analysed, but their contents are rather artificial.

Because these target texts were about conference registration, it was not possible to use the travel arrangement texts for training the grammar. That is the reason why the training texts are different from the texts used in part 2. There are two sets of morphologically analysed texts about conference registration: the ones input by keyboard ("set-K"), which can be found in the unix files /data3/MORPH/bunout/key/Kxxxx.DEC (where xxxx represents a number from 3001 to 3177, except 3047, or from 3426 to 3604). The size of this set is 355 texts. The other set ("set-T") contains 181 texts from the unix files /data3/MORPH/bunout/tel/Txxxx.DEC (where xxxx represents a number from 0001 to 0181). One probabilistic grammar was then generated for each training set.

3-4 Recognition results

Recognition experiments were done with each grammar. All the sentences of the target texts were parsed successfully when the training set-K was used, but one sentence could not be parsed with set-T, and two when the grammar was not probabilistic. Using a probabilistic grammar appears to improve the parsing of the target texts.

The recognition results are summarized in the following table.

Table 2: Recognition results for two grammars with seven target texts.

Recognition rate (%)	Set-K	Set-T
I 8013	87.5	90.0
I 8014	95.3	93.0
I 8015	95.3	93.0
I 8016	93.1	91.4
I 8017	89.8	89.8
I 8018	94.7	93.0
I 8019	95.2	92.1
Concatenated texts	93.2	91.8

3-5 Distances between the target texts and the training texts

The distances between each target text and each training text were calculated for different metrics - vocabulary and surface expressions The mean of these results was then taken as the distance from each target text to each training set.

Tables 3 and 4 sum up the distances to both training sets:

Set-K

Distances	All words	Ind words	Bigrams	Trigrams
I 8013	0.1331	0.2523	0.1692	0.2474
I 8014	0.1205	0.2297	0.1421	0.2149
I 8015	0.1334	0.2534	0.1530	0.2336
I 8016	0.1182	0.2399	0.1606	0.2347
I 8017	0.1233	0.2321	0.1617	0.2353
I 8018	0.1127	0.2190	0.1398	0.1849
I 8019	0.09944	0.1916	0.1434	0.2417
Concatenated texts	0.08846	0.1703	0.07921	0.1180

Table 3

Set-T

Distances	All words	Ind words	Bigrams	Trigrams
I 8013	0.1481	0.2554	0.1984	0.3284
I 8014	0.1319	0.2310	0.1682	0.2971
I 8015	0.1470	0.2555	0.1792	0.2908
I 8016	0.1270	0.2423	0.1791	0.2775
I 8017	0.1279	0.2285	0.1673	0.2575
I 8018	0.1255	0.2210	0.1507	0.2133
I 8019	0.1155	0.1919	0.1684	0.3316
Concatenated texts	0.1033	0.1717	0.1035	0.1775

Table 4

Table 3 & 4 : Distances from target texts to training sets for each metric

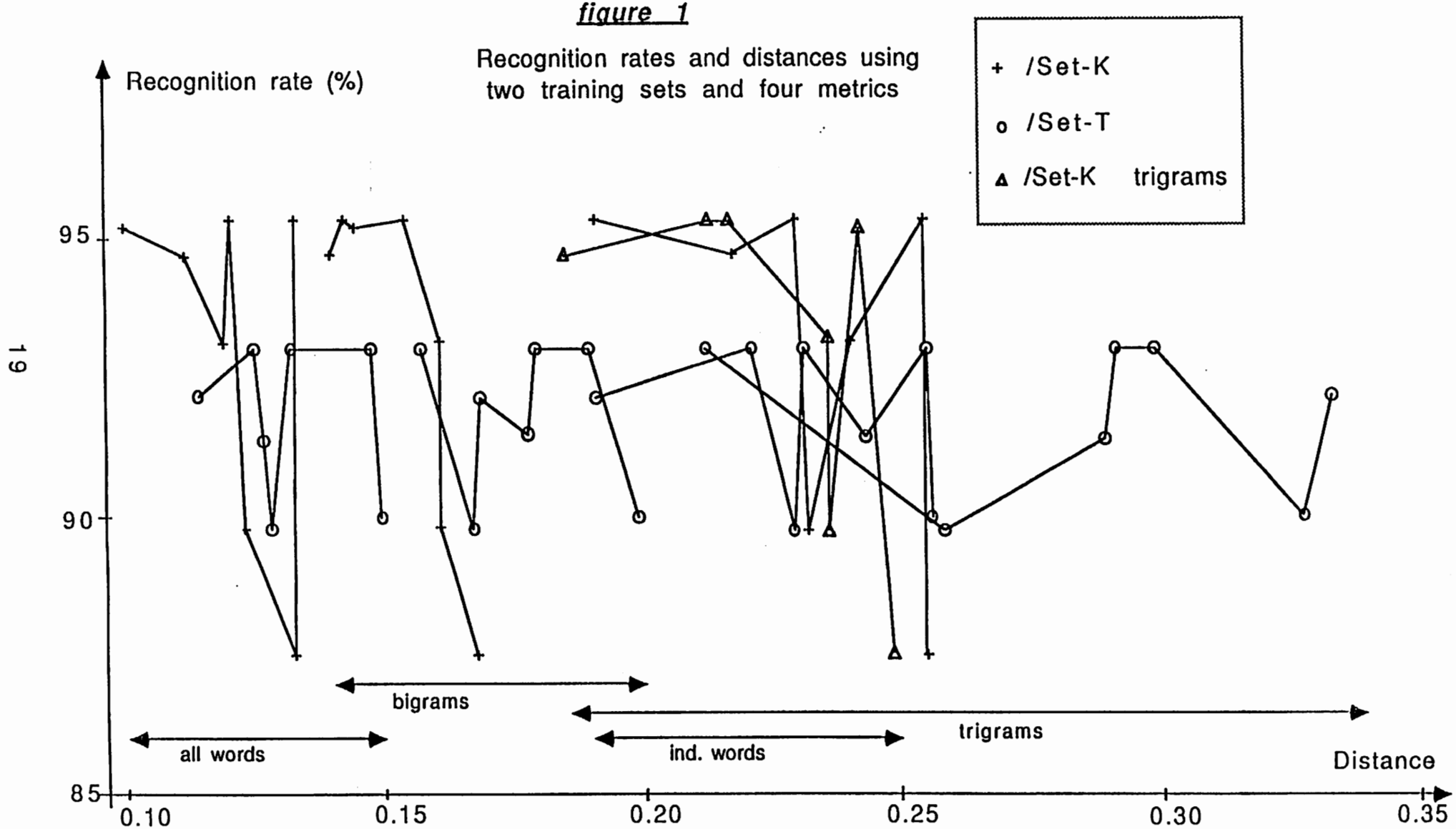
3.6 Relationship between distances and recognition rates

Except for one text, the recognition rate is higher when the training set-K is used; the parsing is better too. Correspondingly, for every metric, the distance from each target text to set-K is less than the distance to set-T.

See figure1. It can be clearly seen from this figure that there is no correlation between recognition rate and distances to the training set. Correlation coefficient were in the range of about -0.2 to -0.5. On the other hand, the almost constant correlation between set-K and set-T is well marked. This is particularely clear in figure 2, where trigrams are not considered and where five target texts are kept.

figure 1

Recognition rates and distances using two training sets and four metrics



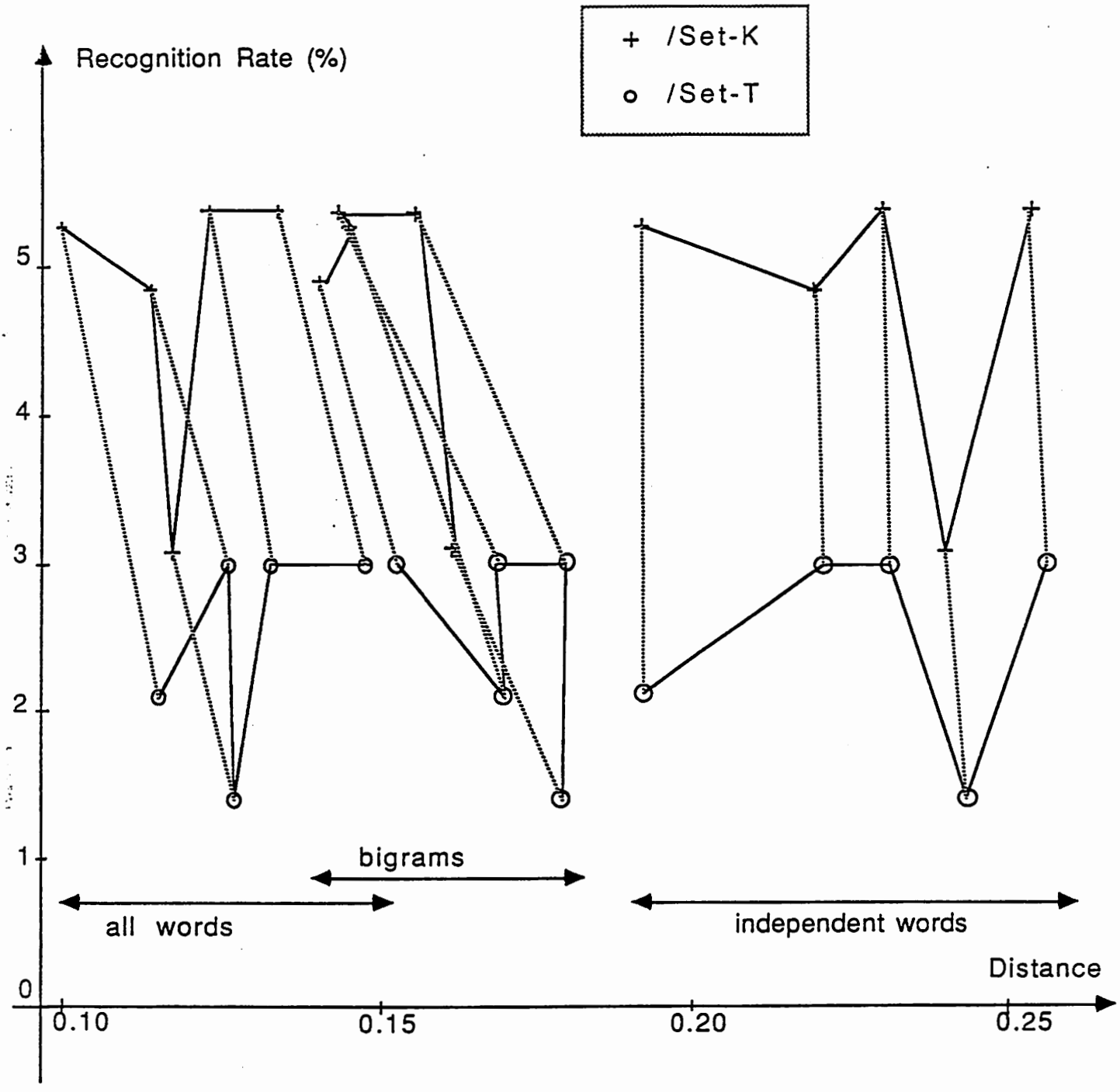


figure 2 Results for 3 metrics with 2 training sets for 5 target texts

See figure 2. In this figure, the points corresponding to the same target text compared with set-K or set-T have been connected with sotted lines. Although the points are not correlated, these vectors (sotted lines) are strongly correlated. The correlations are given in table 5

set-K/set-T	Correlation
All words	-0.79
Ind. words	-0.74
Bigrams	-0.71
Trigrams (not represented)	-0.88

Table 5: correlations between vectors representing the differences in the use of set-K or set-T

One possible conclusion that can be drawn from table 5 is that in almost every case, the recognition rate is higher and the distance lower when set-K is used. If the training set for which the distances to the target texts are higher is chosen, the recognition rate will probably be lower than by using the other training set, even if there is no correlation between recognition rate and distance to the training set for one training set. This result seems to be independent of the metric

For this study, only a few texts could be used for the recognition experiments. A larger set of target texts is required if significant conclusions are to be drawn.

4 Conclusion

In this report, we first looked at the similarities that can be observed between metrics for the comparison of linguistic texts in Japanese and English. For both languages, the conclusion is that two characteritics can be distinguished: we can define a similarity for vocabulary and another for syntax. These similarities correspond to pre-terminal and non-terminal derivation grammar rules respectively.

We then saw that the training set of texts used for refining the stochastic grammar can be chosen by evaluating the distance between this set and the target texts. A lower mean value of this distance improves the recognition rate.

References

T.Sakano, T.Morimoto *Design principle of language model for speech recognition* ICSLP 90 (1990).

K.Kita *A study on language modelling for speech recognition* (1991)

K.Shikano *Improvement of word recognition results by trigram models* ICASSP86 (1987)

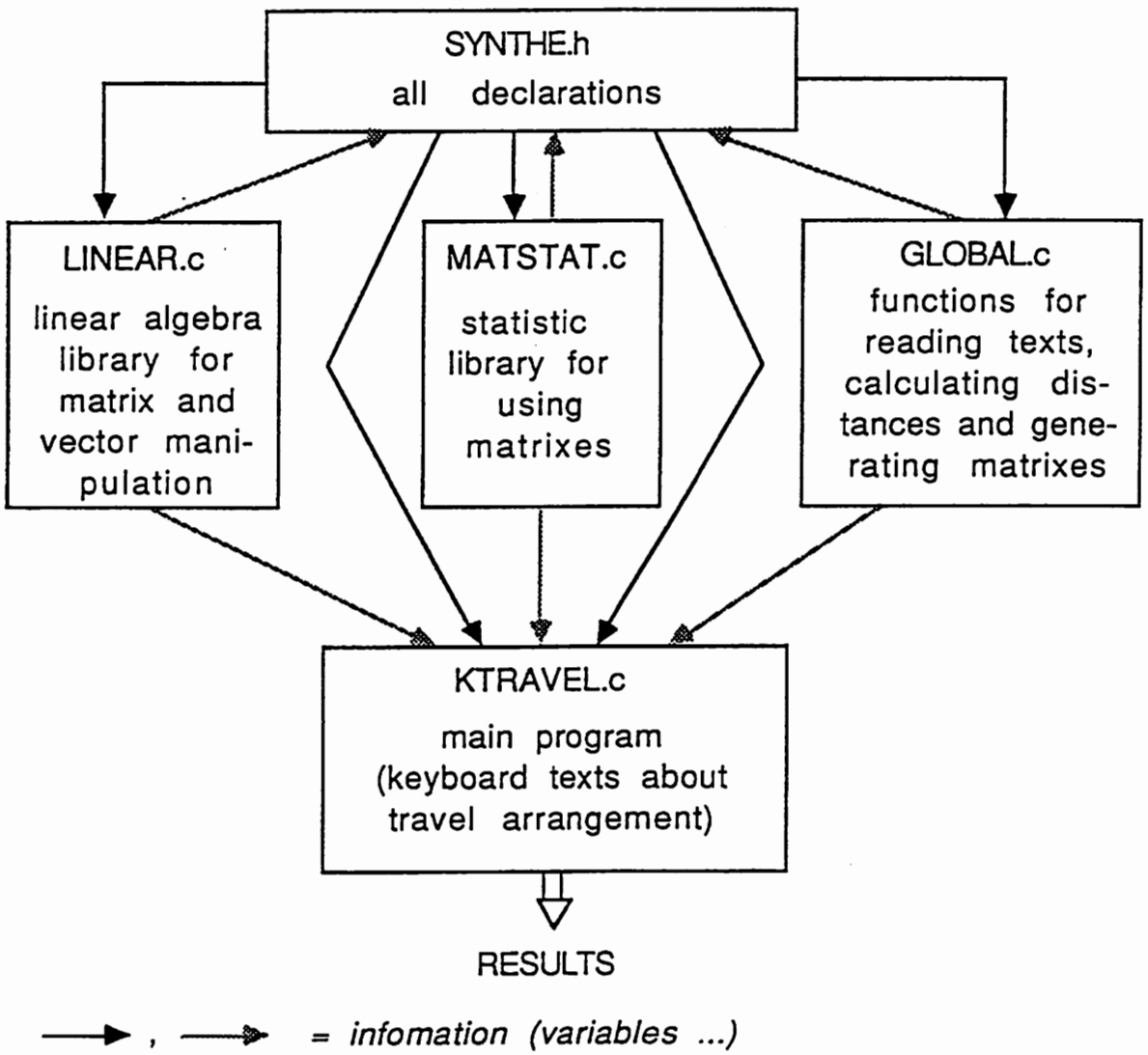
T.Ehara, N.Inoue, H. Kohyama, T.Hasegawa, F. Shohyawa, T.Morimoto *Contents of the ATR dialogue Database* (1990)

Appendix

This appendix contains the main part of the program producing the matrixes of distances described in part 2. Figure 3 explains how the program is built.

synthe.h (declarations), matstat.c (statistics) and global.c (main functions) are copied in this report.

figure 3: ***Description of the program used in part 2***



```
/****************************************
*                                       *
*              SYNTHE.H         *
*                                       *
****************************************/


/*********** linearl.c ************/
  /*    1- Constants      */
#define         VIP             1
#define         Vloc(i)         i + VIP -1
#define         Mloc(n. i. j)   (n+VIP)*(i+VIP-1)+VIP+j-1

#define         MATSIZE         50
#define         MAXSIZE         MATSIZE * MATSIZE
#define         MAXREPEAT       100


  /*    2- Functions      */
  extern            Vplus(). Vdiff().
                    Vnpro().
                    Vtrans().
                    Vprint().
                    Vsave().
                    Vread().
                    Vcopy():

  extern double     Vipro(). Vnorm():

  extern            Mplus(). Mdiff().
                    Mpro().
                    Mid(). Mtp().
                    Mprint().
                    Meigen().
                    Mcopy().
                    Msave().
                    Mread():

  extern            Msubmat().
                    Minv().
                    loc().
                    jacobi().
                    get_sin_cos():

  extern double     Mtr(). Mdet():


/************ matstatl.c **************/
  /*    Functions     */
  extern double     Mean().
                    Variance().
                    Sigma().
                    Covar().
                    Correlation():

  extern double     VMean().
                    VVariance().
                    VSigma().
                    VCovar().
                    VCorrelation():
```

```
/************ global1.c **************/
  /*    1- Constants      */
#define  ENUM             3000
#define  FNUM             500
#define  FNAME            100
#define  MSIZE            (FNUM+1)*(FNUM+1)
#define  VSIZE   FNUM+1


  /*    2- Functions      */
  extern     ReadFileName();

  extern     GenMatrix();

  extern     GenVector();

  extern     GenLength();
double  beta;

/*                  GenM_Me().
                    GenD_Me().
                    GenT_Me().
                    GenS1_Me().
                    GenS2_Me().
                    GenS3_Me();
*/

/************** Main1.c ***************/
  /*    - Main Variables -    */
  int        J_E;

  int        MonoG,
             DiG,
             TriG,
             MonoS,
             DiS,
             TriS;

  int        Fnum;
  FILE       *Fp[FNUM];
  char       Fname[FNUM][FNAME];

  double      M_Me[MSIZE],   D_Me[MSIZE],   T_Me[MSIZE],
             S1_Me[MSIZE], S2_Me[MSIZE], S3_Me[MSIZE],
              M_Ve[VSIZE],   D_Ve[VSIZE],   T_Ve[VSIZE],
             S1_Ve[VSIZE], S2_Ve[VSIZE], S3_Ve[VSIZE];
  double L_Ve[VSIZE];
```

```
#include <stdio.h>
#include <math.h>

#include "synthe.h"

                /*********************************
                *                                *
                *           GLOBAL.C             *
                *                                *
                *********************************/



/********** Variables *********/
struct MonoTag
  {
    int          key:
    charword[180]:
  };

struct MonoTag  M_table[ENUM]:

struct DiTag
  {
    int          key:
    charword[2][180]:
  };

struct DiTag    D_table[ENUM]:

struct TriTag
  {
    int          key:
    charword[3][180]:
  };

struct TriTag   T_table[ENUM]:

struct MonoSurf
  {

    int          key:
    int          morph:
  };

struct MonoSurf S1_table[ENUM]:

struct DiSurf
  {
    int          key:
    int          morph[2]:
  };

struct DiSurf   S2_table[ENUM]:

struct TriSurf
  {
    int          key:
    int          morph[3]:
  };

struct TriSurf  S3_table[ENUM]:

struct Suit
  {
    int          counter[2]:
```

```
};

struct Suit      Mcount[ENUM], Dcount[ENUM], Tcount[ENUM],
                 Sicount[ENUM], S2count[ENUM], S3count[ENUM];

int              M_EntryNum, D_EntryNum, T_EntryNum,
                 S1_EntryNum, S2_EntryNum, S3_EntryNum;


/********** Functions *********/
GetLine(fp, line)
  FILE  *fp;
  char  *line;
  {
    char  c;
    int   coma;
    int   position;
    int   ok;

    if( (c = fgetc(fp)) == EOF )
      return NULL;

    ungetc(c, fp);       /* push back */

    ok = 1;
    coma = 0;
    position = 0;

    if( J_E == 1 )
      while( ((c = fgetc(fp)) != '\n') && (c != EOF))
        {
          if( c == '.' )
            coma++;
          *line++ = c;
          if( ( ok != 2 ) && (coma == 2) && ( c == '.' ) )
            if( (c = fgetc(fp)) != '.' )
              ungetc(c, fp);
            else
              ok = 2;
          if( ( ok != 2 ) && (coma == 7) && ( c == '.' ) )
            if( (c = fgetc(fp)) != '.')
              ungetc(c, fp);
            else
              ok = 3;
        }
    else if( J_E == 2)
      while( ((c = fgetc(fp)) != '\n') && (c != EOF) )
        {
          ++position;
          if( (position == 16) && (c == '_') )
            ok = 2;
          *line++ = c;
        }
    *line = '\0';

    return ok;
  }

GetWord( line, word, part)
  char  *line;
  char  *word;
  int   *part;
  {
    int  i, j,
         coma,
         parttmp, conjugationtmp;
```

```
    coma = 0;
    i = 0;
    parttmp = 0;

    if( J_E == 1 )
      {
        while( coma < 7 )
          if( *line++ == ',' )
            coma++;
        while( *line != ',' )
          word[i++] = *line++;
        word[i] = '¥0';
        ++line;

        while( *line != ',' )
          parttmp = parttmp*10 + (*line++) - '0';
        ++line;
      }
    else if( J_E == 2 )
      {
        for( j = 0; j <= 34; j++ )
          ++line;
        while( *line != '_' )    /* printf("%c  ", *line++);*/
          ++line;
        ++line;

        while( *line != '_' )    /* printf("%c  ", *line); */
          word[i++] = *line++;
        word[i] = '¥0';
        ++line;

        while( *line != '_' )     /* printf("%c  ",*line); */
          parttmp = parttmp*10 + (*line++) - '0';
        ++line;
      }
    *part = parttmp;

    return;
  }

InitTable()
  {
    int i;

    for( i = 0; i < ENUM; i++)
      {
        M_EntryNum = 0;
        D_EntryNum = 0;
        T_EntryNum = 0;
        S1_EntryNum = 0;
        S2_EntryNum = 0;
        S3_EntryNum = 0;

        M_table[i].key = 0;
        D_table[i].key = 0;
        T_table[i].key = 0;
        S1_table[i].key = 0;
        S2_table[i].key = 0;
        S3_table[i].key = 0;

        (Mcount[i].counter)[0] = 0;
        (Mcount[i].counter)[1] = 0;
        (Dcount[i].counter)[0] = 0;
        (Dcount[i].counter)[1] = 0;
        (Tcount[i].counter)[0] = 0;
```

```
        (Tcount[i].counter)[1] = 0;
        (S1count[i].counter)[0] = 0;
        (S1count[i].counter)[1] = 0;
        (S2count[i].counter)[0] = 0;
        (S2count[i].counter)[1] = 0;
        (S3count[i].counter)[0] = 0;
        (S3count[i].counter)[1] = 0;
      }

    return;
  }

int  M_SearchEntry(word)
  char  word[180];
  {
    int  i;

    if( !M_EntryNum )
      return  -1;

    for(i=0; i<M_EntryNum; i++)
      if( !M_table[i].key )
        return -1;
      else if( !strcmp( word, M_table[i].word ) )
        return i;

    return -1;
  }

int  D_SearchEntry(word0, word1)
  char  word0[180],
        word1[180];
  {
    int  i;

    if( !D_EntryNum )
      return  -1;

    for(i=0; i<D_EntryNum; i++)
      if( !D_table[i].key )
         return -1;
      else if( ( !strcmp( word0, (D_table[i].word)[0]) )
            && ( !strcmp( word1, (D_table[i].word)[1]) ) )
         return i;

    return -1;
  }

int  T_SearchEntry(word0, word1, word2)
  char  word0[180],
        word1[180],
        word2[180];
  {
    int  i;

    if( !T_EntryNum )
      return  -1;

    for(i=0; i<T_EntryNum; i++)
      if( !T_table[i].key )
         return -1;
      else if( ( !strcmp( word0, (T_table[i].word)[0]) )
            && ( !strcmp( word1, (T_table[i].word)[1]) )
            && ( !strcmp( word2, (T_table[i].word)[2]) ) )
         return i;
```

```
    return -1;
  }

int  S1_SearchEntry(morph)
  int   morph;
  {
    int  i;

    if( !S1_EntryNum )
      return  -1;

    for(i=0; i<S1_EntryNum; i++)
      if( !S1_table[i].key )
         return -1;
      else if( morph == S1_table[i].morph )
         return i;

    return -1;
  }

int  S2_SearchEntry(morph0, morph1)
  int   morph0,
        morph1;
  {
    int  i;

    if( !S2_EntryNum )
      return  -1;

    for(i=0; i<S2_EntryNum; i++)
      if( !S2_table[i].key )
         return -1;
      else if( ( morph0 == (S2_table[i].morph)[0] )
            && ( morph1 == (S2_table[i].morph)[1] ) )
         return i;

    return -1;
  }

int  S3_SearchEntry(morph0, morph1, morph2)
  int   morph0,
        morph1,
        morph2;
  {
    int  i;

    if( !S3_EntryNum )
      return  -1;

    for(i=0; i<S3_EntryNum; i++)
      if( !S3_table[i].key )
         return -1;
      else if( ( morph0 == (S3_table[i].morph)[0] )
            && ( morph1 == (S3_table[i].morph)[1] )
            && ( morph2 == (S3_table[i].morph)[2] ) )
         return i;

    return -1;
  }


M_ClearEntry( k, Ment)
  int   k;
  int   Ment;
  {
    int  i;
```

```
    if( !M_EntryNum )
      return:
    if( Ment )
      for(i=0: i<Ment: i++)
        (Mcount[i].counter)[k] = 0:
    for(i=Ment: i<M_EntryNum: i++)
      {
        M_table[i].key = 0:
        (Mcount[i].counter)[0] = (Mcount[i].counter)[1] = 0:
      }
    M_EntryNum = Ment:

    return;
  }

D_ClearEntry( k. Dent)
  int   k:
  int   Dent:
  {
    int   i;

    if( !D_EntryNum )
      return;
    if( Dent )
      for( i=0; i<Dent: i++)
        (Dcount[k].counter)[k] = 0:
    for(i=Dent; i<D_EntryNum: i++)
      {
        D_table[i].key = 0:
        (Dcount[i].counter)[0] = (Dcount[i].counter)[1] = 0:
      }
    D_EntryNum = Dent:

    return;
  }

T_ClearEntry( k. Tent)
  int   k:
  int   Tent:
  {
    int   i:

    if( !T_EntryNum )
      return:
    if( Tent )
      for( i=0; i<Tent: i++)
        (Tcount[k].counter)[k] = 0:
    for(i=Tent: i<T_EntryNum: i++)
      {
        T_table[i].key = 0:
        (Tcount[i].counter)[0] = (Tcount[i].counter)[1] = 0:
      }
    T_EntryNum = Tent:

    return:
  }

S1_ClearEntry(k. Slent)
  int   k;
  int   Slent:
  {
    int   i:

    if( !S1_EntryNum )
      return:
```

```
    if( S1ent )
      for( i=0; i<S1ent; i++)
        (S1count[k].counter)[k] = 0;
    for(i=S1ent; i<S1_EntryNum; i++)
      {
        S1_table[i].key = 0;
        (S1count[i].counter)[0] = (S1count[i].counter)[1] = 0;
      }
    S1_EntryNum = S1ent;

    return;
  }

S2_ClearEntry( k, S2ent)
  int   k;
  int   S2ent;
  {
    int   i;

    if( !S2_EntryNum )
      return;
    if( S2ent )
      for( i=0; i<S2ent; i++)
        (S2count[k].counter)[k] = 0;
    for(i=S2ent; i<S2_EntryNum; i++)
      {
        S2_table[i].key = 0;
        (S2count[i].counter)[0] = (S2count[i].counter)[1] = 0;
      }
    S2_EntryNum = S2ent;

    return;
  }

S3_ClearEntry( k, S3ent)
  int   k;
  int   S3ent;
  {
    int   i;

    if( !S3_EntryNum )
      return;
    if( S3ent )
      for( i=0; i<S3ent; i++)
        (S3count[k].counter)[k] = 0;
    for(i=S3ent; i<S3_EntryNum; i++)
      {
        S3_table[i].key = 0;
        (S3count[i].counter)[0] = (S3count[i].counter)[1] = 0;

      }
    S3_EntryNum = S3ent;

    return;
  }

int  ReadText(i, fname)
  int   i;
  char  fname[100];
  {
    FILE  *fp;
    char  line[255];
    char  word[180];
    int   part;
    char  word0[180], word1[180], word2[180];
    int   morph0, morph1, morph2;
```

```
    int   conjugation:
    int   ok:

    word0[0] = word1[0] = word2[0] = '¥0':
    morph0 = morph1 = morph2 = -1:

    fp = fopen(fname. "r"):

    while( (ok = GetLine( fp. line)) )
      {
        if( ok == 2 )
          continue:
        if( ok == 3 )
          {
            word0[0] = word1[0] = word2[0] = '¥0':
            morph0 = morph1 = morph2 = 0:
            continue:
          }
        GetWord( line. word. &part ):

        if( (MonoG) || (DiG) || (TriG) )
          if( TestG(part) )
            {
              strcpy(word0. word1):
              strcpy(word1. word2):
              strcpy(word2. word ):

              if( MonoG )
                M_Insert(word2. i):
              if( DiG )
                D_Insert(word1. word2. i):
              if( TriG )
                T_Insert(word0. word1. word2.i):
            }
        if( (MonoS) || (DiS) || (TriS) )
          if( TestS(part) )
            {
              morph0 = morph1:
              morph1 = morph2:
              morph2 = part:

              if( MonoS )
                S1_Insert(morph2. i):
              if( DiS )
                S2_Insert(morph1. morph2.i):
              if( TriS )
                S3_Insert(morph0. morph1.morph2.i):
            }
      }
    fclose(fp):
    return:
  }

int  TestG(part)
  int part:
  {

    if( J_E == 1 )
      {
        if( (part != 1) && (part != 4) && (part !=5)
                                       && (part != 8) && (part != 32) )
          return 0:
      }
    else if ( J_E == 2 )
      {
        if( (part != 10) && (part != 20) && (part != 30) && (part != 40) )
```

```
            return 0;
        }

      return 1;
    }

int  TestS(part)
  int part;
  {
/*
      if( J_E == 1 )
        {
          if( (part != ) && (part != ) && (part != ) )
            return 0;
        }
      else if ( J_E == 2 )
        {
          if( (part != ) && (part != ) )
            return 0;
        }
*/
      return 1;
    }

int  M_Insert(word, i)
  char  word[180];
  int   i;
  {
    int   num;

    num = M_SearchEntry( word, M_table );
    if( num == -1 )              /* not entried */
      {
        M_table[M_EntryNum].key = 1;
        strcpy( M_table[M_EntryNum].word, word );
        (Mcount[M_EntryNum].counter)[i] = 1;
        M_EntryNum++;
      }
    else
        (Mcount[num].counter)[i] += 1;

    return;
  }


int  D_Insert(word0, word1, i)
  char  word0[180], word1[180];
  int   i;
  {
    int   num;

    if ( word0[0] == '¥0' )
      return;

    num = D_SearchEntry( word0, word1, D_table );
    if( num == -1 )              /* not entried */
      {
        D_table[D_EntryNum].key = 1;
        strcpy( (D_table[D_EntryNum].word)[0], word0 );
        strcpy( (D_table[D_EntryNum].word)[1], word1 );
        (Dcount[D_EntryNum].counter)[i] = 1;
        D_EntryNum++;
      }
    else
        (Dcount[num].counter)[i] += 1;
```

```
    return;
  }

int  T_Insert(word0, word1, word2, i)
  char  word0[180], word1[180], word2[180];
  int   i;
  {
    int   num;

    if ( word0[0] == '¥0' )
      return;

    num = T_SearchEntry( word0, word1, word2, T_table );
    if( num == -1 )              /* not entried */
      {
        T_table[T_EntryNum].key = 1;
        strcpy( (T_table[T_EntryNum].word)[0], word0 );
        strcpy( (T_table[T_EntryNum].word)[1], word1 );
        strcpy( (T_table[T_EntryNum].word)[2], word2 );
        (Tcount[T_EntryNum].counter)[i] = 1;
        T_EntryNum++;
      }
    else
        (Tcount[num].counter)[i] += 1;

    return;
  }

int  S1_Insert(morph, i)
  int   morph;
  int   i;
  {
    int   num;

    num = S1_SearchEntry( morph, S1_table );
    if( num == -1 )                      /* not entried */
      {
        S1_table[S1_EntryNum].key = 1;
        S1_table[S1_EntryNum].morph = morph;
        (S1count[S1_EntryNum].counter)[i] = 1;
        S1_EntryNum++;
      }
    else
        (S1count[num].counter)[i] += 1;

    return;
  }

int  S2_Insert(morph0, morph1, i)
  int   morph0, morph1;
  int   i;
  {
    int   num;

    if ( morph0 == -1 )
      return;

    num = S2_SearchEntry( morph0, morph1, S2_table );
    if( num == -1 )                      /* not entried */
      {
        S2_table[S2_EntryNum].key = 1;
        (S2_table[S2_EntryNum].morph)[0] = morph0;
        (S2_table[S2_EntryNum].morph)[1] = morph1;
        (S2count[S2_EntryNum].counter)[i] = 1;
        S2_EntryNum++;
      }
```

```
        else
            (S2count[num].counter)[i] += 1;

        return;
      }

    int  S3_Insert(morph0, morph1, morph2, i)
      int    morph0, morph1, morph2;
      int    i;
      {
        int    num;

        if ( morph0 == -1 )
          return ;

        num = S3_SearchEntry( morph0, morph1, morph2, S3_table );
        if( num == -1 )                        /* not entried */
          {
            S3_table[S3_EntryNum].key = 1;
            (S3_table[S3_EntryNum].morph)[0] = morph0;
            (S3_table[S3_EntryNum].morph)[1] = morph1;
            (S3_table[S3_EntryNum].morph)[2] = morph2;
            (S3count[S3_EntryNum].counter)[i] = 1;
            S3_EntryNum++;
          }
        else
            (S3count[num].counter)[i] += 1;

        return;
      }

    GenMatrix()
      {
        int          i, j;
        int           Ment,  Dent,  Tent,
                     Slent, S2ent, S3ent;

        InitTable();

        for(i=0; i<Fnum; i++)
          {
            ReadText(0, Fname[i]);
             Ment =  M_EntryNum;  Dent =  D_EntryNum;  Tent =  T_EntryNum;
            Slent = S1_EntryNum; S2ent = S2_EntryNum; S3ent = S3_EntryNum;
            printf("FICHIER :: %s¥n", Fname[i] );
            for(j=0; j<=i; j++)
              {
                if( i==j )
                  {
                    Diag(i);
                    continue;
                  }
                ReadText(1, Fname[j]);
                GenChose(i,j);
                ClearChose(1, Ment, Dent , Tent, Slent, S2ent, S3ent );
              }
            ClearChose(0, 0,0,0,0,0,0);
          }
        return;
      }

    Diag(i)
      int    i;
      {
        M_Me[Mloc(Fnum, i+1,i+1)] = 0.0;
        D_Me[Mloc(Fnum, i+1,i+1)] = 0.0;
```

```
    T_Me[Mloc(Fnum. i+1.i+1)] = 0.0:
    S1_Me[Mloc(Fnum.i+1.i+1)] = 0.0:
    S2_Me[Mloc(Fnum.i+1.i+1)] = 0.0:
    S3_Me[Mloc(Fnum.i+1.i+1)] = 0.0:

    return:
  }

ClearChose( k. Ment. Dent. Tent. S1ent. S2ent. S3ent)
  int   k:
  int   Ment. Dent. Tent. S1ent. S2ent. S3ent:
  {
    M_ClearEntry(k.Ment):
    D_ClearEntry(k.Dent):
    T_ClearEntry(k.Tent):
    S1_ClearEntry(k.S1ent):
    S2_ClearEntry(k.S2ent):
    S3_ClearEntry(k.S3ent):

    return;
  }

GenChose( i. j)
  int   i. j:
  {
    if( MonoG )
      GenElement( i.j. Mcount. M_EntryNum. M_Me):
    if( DiG )
      GenElement( i.j. Dcount. D_EntryNum. D_Me):
    if( TriG )
      GenElement( i.j. Tcount. T_EntryNum. T_Me):
    if( MonoS )
      GenElement( i.j. S1count.S1_EntryNum.S1_Me):
    if( DiS )
      GenElement( i.j. S2count.S2_EntryNum.S2_Me):
    if( TriS )
      GenElement( i.j. S3count.S3_EntryNum.S3_Me):

    return;
  }

GenElement( i. j. compt. EntryNum. Mat_Me)
  int           i. j:
  struct Suit   compt[ENUM]:
  int           EntryNum:
  doubleMat_Me[]:
  {
    int         k:
    int         WordNum[2]:
    double      metric:
    double      prob0. prob1:

    WordNum[0] = 0:
    WordNum[1] = 0:

    for(k=0: k<EntryNum: k++)
      {
        WordNum[0] += ( compt[k].counter)[0]:
        WordNum[1] += ( compt[k].counter)[1]:
      }

    metric = 0.0:
    for(k=0: k<EntryNum: k++)
      {
        prob0 = (double)( compt[k].counter)[0] / WordNum[0]:
        prob1 = (double)( compt[k].counter)[1] / WordNum[1]:
```

```
            metric += (prob0 - prob1) * (prob0 - prob1):
          }

        metric = sqrt(metric):

        Mat_Me[Mloc(Fnum.i+1.j+1)] = Mat_Me[Mloc(Fnum.j+1.i+1)] = metric:

        return:
      }

    /* ****************************
    ReadFileName()
      {
        int  i:
        char name[100]:

        i = 0;
        do
          {
            printf("[%d]Enter Text Name>>  ". i+1):
            rewind( stdin ):
            scanf("%s". name):
            if( name[0] == ':' )
              break:
            if( J_E == 1 )
              {
                strcpy( Fname[i]. "/data3/MORPH/bunout/key/K3" ):
                strcat( Fname[i]. name ):
                strcat( Fname[i]. ".DEC" ):
              }
            else if( J_E == 2 )
              {
                strcpy( Fname[i]. "/data3/MORPH-E/key/k3"):
                strcat( Fname[i]. name ):
                strcat( Fname[i]. ".shp" ):
              }
            if( !(Fp[i] = fopen(Fname[i]. "r")) )
              {
                fprintf(stderr. "Cannot open %s!!¥n¥n". Fname[i]):
                continue:
              }
            else
              fclose(Fp[i]):
            ++i:
          } while( i < FNUM ):
        if( i == 0 )
          {
            printf("This Process is aborted.¥n"):
            return -1:
          }

        Fnum = i:
        printf("The Number of Files is %d¥n¥n". Fnum):
        return 1:
      }
                                    ************************** */
```

```
#include <stdio.h>
#include <math.h>

#include "synthe1.h"


                /************************************
                *                                   *
                *           Statistics              *
                *               on                  *
                *       symmetrical matrixes        *
                *                                   *
                ************************************/



/********** Functions *********/

double  Mean(dim, Mat)
  int       dim;
  double    Mat[];
  {
    int         i,j;
    double      m;

    m = 0;

    if( dim >= 2 )
      {
        for( i = 1; i < dim; i++)
          for( j = i+1; j <= dim; j++)
            m += Mat[Mloc( dim, i, j)];
        m /= (dim * (dim - 1)) / 2 ;
      }

    return m;
  }

double  Variance(dim, Mat)
  int       dim;
  double    Mat[];
  {
    int         i,j;
    double      m;
    double      v;

    v = 0;

    if(dim >= 2 )
      {
        m = Mean(dim, Mat);
        for( i = 1; i < dim; i++)
          for( j = i+1; j <= dim; j++)
            v +=  Mat[Mloc( dim, i, j)] * Mat[Mloc( dim, i, j)] ;
        v /= (dim * (dim - 1)) / 2;
        v -= (m * m);
      }

    return v;
  }

double  Sigma(dim, Mat)
  int       dim;
  double    Mat[];
  {
```

```
    return (sqrt( Variance(dim. Mat)) ):
  }

double  Covar(dim. Mat1. Mat2)
  int       dim:
  double    Mat1[].
            Mat2[]:
  {
    int          i.j:
    double       m1.m2:
    double       cov:

    cov = 0:

    if(dim >= 2 )
      {
        m1 = Mean(dim. Mat1):
        m2 = Mean(dim. Mat2):
        for( i = 1: i < dim: i++)
          for( j = i+1: j <= dim: j++)
            cov +=  Mat1[Mloc( dim. i. j)] * Mat2[Mloc( dim. i. j)] :
        cov /= (dim * (dim - 1)) / 2:
        cov -= (m1 * m2):
      }

    return cov:
  }

double  Correlation(dim. Mat1.Mat2)
  int       dim:
  double    Mat1[].
            Mat2[]:
  {
    return ( Covar(dim. Mat1. Mat2) / Sigma(dim. Mat1) / Sigma(dim. Mat2) ):
  }
```