An Algorithm for Extracting Phrases in a Space-Optimal Fashion

by

R.A. Wagner

Technical Report
No. 71-93
February 1971

Department of Computer Science
Cornell University
Ithaca, New York 14850

An Algorithm for Extracting Phrases in a Space-Optimal Fashion

Introduction:

The algorithm PARSE computes and prints a minimum-space form of a textual message, MS. The minimization is performed over all possible "parses" of MS into sequences of <u>phrase references</u> and <u>character strings</u>. Each <u>phrase reference</u> represents one of a finite collection, P, of phrases. The collection, P, must be selected before PARSE is applied.

Assumptions and Requirements:

PARSE assumes that the unit of storage is the "byte", defined such that one byte can hold either a single character of text, or an integer i in the range $0 \le i < W$. (For IBM 360 equipment, W = 256 = 2**8.)

PARSE also assumes that the number of different phrases in the collection P is no larger than W**PHC, and that each message to be parsed contains fewer than W**CHC characters of text.

The parameter values CHC=PHC=1 appear appropriate on IBM 360 equipment, when PARSE is applied to short messages, such as compiler error messages.

PARSE requires 2 arguments. The first is the message to be parsed: the second is the table of common phrases which may be used in the parse.

PARSE assumes that an external procedure HASH is present, HASH(MS,I,K) is defined as follows:

Let $H_1, H_2, \ldots, H_m$ be a sequence of indices such that among them they exhaust all entries $P(H_i)$ such that

SUBSTR(MS,I.3) = SUBSTR($P(H_i)$.1.3).

(That is. the H_i's include indices for every phrase $P(H_i)$ which agrees with characters I. I + 1, and I + 2 of the given message. Other indices may occur among the H_i's, as well).

Then HASH(MS.I,0) = H_1. HASH(MS,I.H_j) = H_{j+1}, and HASH(MS,I.H_m) = 0.

A "hash table" procedure can easily be modified to yield this performance; an equally usable, although slower version returns MØD(K+1, M+1) on every call.

Methods:

The method used to determine which phrases to extract from the given message is described in [1]. The resulting parsed message requires least space, assuming that messages are storable only as described in [1] - that is. as sequences of

```
     C  <number> <character string>
   | P  <number>
```

representing a literal string of characters. and a reference to a common phrase, respectively.

Internally, PARSE uses a single array. Z, parallelling the function arrays G and H. to retain the information needed for "re-constructing the parsed form of the message.

$$Z(I) \begin{cases} = K, \text{ if } G(I) > H(I). \text{ where K is the number} \\ \text{of the "best" common phrase matching MS at I. or} \\ = J. \text{ if } G(I) = H(I). \quad (G(I) < H(I) \text{ is impossible.}) \end{cases}$$

J gives the index of the end (plus one) of the character string starting at I. (In this case. the best parse at I begins with this character string.) J satisfies: $G(J) > H(J)$ and for all k, $I \leq k < J$, $G(k) = H(k)$.

Results:

To make the printed form of the parsed message more intelligible, PARSE prints: 'C <number>' as '#ddd'

'P <number>' as '%ddd'.

where "ddd" is the 3-digit decimal representation of <number>
In practice, a number representing a character count or phrase index can be stored as as an integer, in place of CHC or PHC characters respectively.

The program PARSE returns the number of bytes needed to store MS, given the particular set of extractable phrases in P.

References

1. Wagner, R.A.,

   Common phrases and minimum-space text storage, CACM

2. Bell, Jr.R.

   The quadratic quotient method: a hash code eliminating secondary clustering. CACM 13, 1, pp. 107-109 (Feb 1970)

Figure 2. The PARSE Algorithm

```
PARSE:    PROC(MS,P) RETURNS(FIXED BINARY);
          DCL  (MS,P(*)) CHAR(*) VARYING;
          DCL  N;
          DCL  HASH RETURNS(FIXED BINARY);
          DCL (CHC,   /* BYTES PER CHARACTER-COUNT   */
               PHC)   /* BYTES PER PHRASE-INDEX  */
               STATIC EXTERNAL FIXED BINARY;

          N=LENGTH(MS);
          BEGIN;
               DCL(G,H,Z)(N+1) FIXED BINARY;
               DCL (I,J,K,L,T) FIXED BINARY;

               G(N+1)=3; H(N+1)=1; J,Z(N+1)=N+1;
          MSGP:     DO I=N BY -1 TO 1;
                    K=HASH(MS,I,0B);
                    H(I), G(I) = MIN( G(I+1)+1, H(I+1)+CHC+2 );
                    Z(I)=J;
               /* J HOLDS INDEX OF END+1 OF NEXT CHAR-STRING  */
               M1:  DO WHILE (K>0);
                         L=LENGTH(P(K));
                         IF L ¬> N-I+1 THEN
                         IF L < N THEN
                         IF SUBSTR(MS,I,L)=P(K) THEN DO;
                              T=H(I+L)+PHC+1;
                              IF H(I)>T THEN DO;
                                   H(I)=T;  Z(I)= K;  J=I;
                                   END;
                              END;
                         K=HASH(MS,I,K);
                         END M1;
                    END MSGP;

               PUT SKIP EDIT(H(1),N+3,': ')(2 F(4),A);
               I=1; GOTO B1;
          BUILD:
               IF H(I)<G(I) THEN DO;
                    PUT EDIT('%', Z(I))(A,P'999');
                    I=I+LENGTH(P( Z(I)));
                    END;
               ELSE DO;
                    J=Z(I)-I;
                    PUT EDIT('#',J,SUBSTR(MS,I,J))(A,P'999',A);
                    I=Z(I);
                    END;
          B1:  IF I¬>N THEN GOTO BUILD;
               PUT EDIT('.')(A);
               RETURN(H(1));
          END PARSE;
```

Figure 2: An acceptable HASH procedure

```
H:      PROC(MS,I,K) RETURNS(FIXED BINARY);
 DCL    MS CHAR(*), J FIXED BINARY(31,0),
        (HT(0:200) INIT((201)0),
              KJ, HP INIT(197),
              HX,HY,HZ) FIXED BINARY STATIC;
        DCL (CHC,    /* BYTES PER CHARACTER-COUNT    */
              PHC)    /* BYTES PER PHRASE-INDEX   */
              STATIC EXTERNAL FIXED BINARY;

        CALL HCMN(K);
        RETURN(HT(HZ));

 HCMN:        PROC(K);
        IF K  = 0 THEN
              IF LENGTH(MS)-I < PHC+1 THEN HZ=-1;
              ELSE DO;
                    UNSPEC(J)=UNSPEC(SUBSTR(MS,I,PHC+2));
                    HZ=MOD(J,HP);
                    HY=J/HP;
                    HX=0;
                    END;
        ELSE DO;
              HX=MOD(HX+HY,HP);
              HZ=MOD(HX+HZ,HP);
              END;
        HZ=HZ+1;
        RETURN;
        END HCMN;

ER:     ENTRY(MS,I,K);
        IF LENGTH(MS) < PHC+2 THEN RETURN;
        KJ=0;
  E1:   CALL HCMN(KJ);
        KJ=HT(HZ);
        IF KJ >0 THEN GOTO E1;
        HT(HZ)=K;
        RETURN;
        END HASH;
```

Figure 3: A driver for the PARSE procedure

```
DRIVER:  PROC OPTIONS(MAIN);
         DCL  MS CHAR(256) VARYING:
         DCL NP,M:
         DCL (HASH RETURNS(FIXED BINARY), ENTER)
              ENTRY(CHAR(256) VARYING,FIXED BINARY,FIXED BINARY);
         DCL PARSE RETURNS(FIXED BINARY);
         DCL (CHC,    /* BYTES PER CHARACTER-COUNT    */
              PHC)    /* BYTES PER PHRASE-INDEX   */
              STATIC EXTERNAL FIXED BINARY;

         CHC,PHC=1;       /*  COUNT/INDEX SIZE = 1 BYTE   */
         GET SKIP LIST(NP,M);
         BEGIN;
              DCL P(NP) CHAR(M) VARYING;
              DCL NB,NA,I,J;

              NB,NA=0;
              DO I=1 TO NP:
              GET SKIP LIST(P(I));
              CALL ENTER(P(I),1,I);
              END:

         PUT PAGE LIST('PHRASES, AND THEIR PARSED FORMS');
         DO I=1 TO NP;
              PUT SKIP(2) EDIT(I, '   ''' || P(I) || '''')
                   (F(4),A);
              NA=NA+PARSE(P(I),P);
              END:

         PUT PAGE LIST('MESSAGES:');
     L1: GET SKIP LIST(MS);
         PUT SKIP(2) LIST( '''' || MS || '''' );
         IF MS='' THEN GOTO L2:
         NB=NB+LENGTH(MS)+CHC+2:
                  /* ALLOW FOR STRING-OVERHEAD + END MARK   */
              NA=NA+PARSE(MS,P):
              GOTO L1:

     L2: PUT SKIP EDIT('FINAL STATISTICS:',
         'WITHOUT PHRASE EXTRACTION:',NB,
         'AFTER PHRASE EXTRACTION:',NA,
         'SAVING:',NB-NA,
         '   (',(NB-NA)*100/NB,'%)')
              (A,3(SKIP,A,F(5)),A,F(5,1),A);
         RETURN;
         END DRIVER;
```

Figure 4: Sample input files

(a) (2 phrases, 4 messages). Illustrates heavily overlapping phrases.

```
3 LISTING OF INPUT STREAM .

      2,10
     'AAAAA'
     'AAAAAAA'
     'AAAAAAAAAA'
     'AAAAAAAAAAAAA'
     'AAAAAAAAAAAAAAA'
     'AAAAAAAAAAAAAAAA'
     ''
```

(b) (4 phrases, 23 messages). These messages are the first 23 numbered error messages from the syntactic analysis section of the PL/C compiler.

```
3 LISTING OF INPUT STREAM

          5,20
     'EXTRA '
     'MISSING '
          'IMPROPER '
     'SEMI-COLON'
     'EXPRESSION'
     'EXTRA ('
     'MISSING ('
     'EXTRA )'
     'MISSING )'
     'EXTRA COMMA'
     'MISSING COMMA'
     'EXTRA SEMI-COLON'
     'MISSING SEMI-COLON'
     'MISSING :'
     'MISSING ='
     'IMPROPER *'
     'MISSING *'
     'EXTRA END'
          'MISSING END'
          'MISSING KEYWORD'
          'INCOMPLETE EXPRESSION'
          'MISSING EXPRESSION'
          'MISSING VARIABLE'
          'MISSING ARGUMENT, 1 SUPPLIED'
          'EMPTY LIST'
          'IMPROPER NOT'
          'IMPROPER ELEMENT'
          'UNTRANSLATABLE STATEMENT'
     ''
```

Figure 5: Result of applying DRIVER to the cards listed in figure 4a. Note that phrase 2 is itself reduced in size by PARSE, while each of the messages are reduced to strings of phrase references alone.

```
PHRASES, AND THEIR PARSED FORMS

    1  'AAAAA'
    8   8:  #005AAAAA.

    2  'AAAAAAA'
    7  10:  #002AA%001.


MESSAGES:

'AAAAAAAAAA'
    5  13:  %001%001.

'AAAAAAAAAAAA'
    5  15:  %001%002.

'AAAAAAAAAAAAAA'
    5  17:  %002%002.

'AAAAAAAAAAAAAAA'
    7  18:  %001%001%001.

''
FINAL STATISTICS:
WITHOUT PHRASE EXTRACTION:   63
AFTER PHRASE EXTRACTION:   37
SAVING:   26  ( 41.3%)
```

Figure 6: Result of applying DRIVER to the cards listed in figure 4b.

```
PHRASES, AND THEIR PARSED FORMS

    1   'EXTRA '
    9    9:  #006EXTRA .

    2   'MISSING '
   11   11:  #008MISSING .

    3   'IMPROPER '
   12   12:  #009IMPROPER .

    4   'SEMI-COLON'
   13   13:  #010SEMI-COLON.

    5   'EXPRESSION'
   13   13:  #010EXPRESSION.



MESSAGES:

'EXTRA ('
    6  10:  %001#001(.

'MISSING ('
    6  12:  %002#001(.

'EXTRA )'
    6  10:  %001#001).

'MISSING )'
    6  12:  %002#001).

'EXTRA COMMA'
   10  14:  %001#005COMMA.

'MISSING COMMA'
   10  16:  %002#005COMMA.

'EXTRA SEMI-COLON'
    5  19:  %001%004.

'MISSING SEMI-COLON'
    5  21:  %002%004.

'MISSING :'
    6  12:  %002#001:.
```

Figure 6: continued

```
'MISSING ='
   6  12:  %002#001=.

'IMPROPER *'
   6  13:  %003#001*.

'MISSING *'
   6  12:  %002#001*.

'EXTRA END'
   8  12:  %001#003END.

'MISSING END'
   8  14:  %002#003END.

'MISSING KEYWORD'
  12  18:  %002#007KEYWORD.

'INCOMPLETE EXPRESSION'
  16  24:  #011INCOMPLETE %005.

'MISSING EXPRESSION'
   5  21:  %002%005.

'MISSING VARIABLE'
  13  19:  %002#008VARIABLE.

'MISSING ARGUMENT, 1 SUPPLIED'
  25  31:  %002#020ARGUMENT, 1 SUPPLIED.

'EMPTY LIST'
  13  13:  #010EMPTY LIST.

'IMPROPER NOT'
   8  15:  %003#003NOT.

'IMPROPER ELEMENT'
  12  19:  %003#007ELEMENT.

'UNTRANSLATABLE STATEMENT'
  27  27:  #024UNTRANSLATABLE STATEMENT.

''
FINAL STATISTICS:
WITHOUT PHRASE EXTRACTION:  376
AFTER PHRASE EXTRACTION:  283
SAVING:    93  ( 24.6%)
```