and from among sentences can be used for locating classes of common meaning at various levels of abstraction. The expression of individual words, pairs of words, and short strings in the multi-dimensional space can then be examined for semantic relevance.

REFERENCE:

BENNETT, E. M.; MAYER, R. P.; ET AL. COLLAD-I, a command language laboratory demonstration (preliminary concepts). The MITRE Corp., Technical Memorandum TM-3001, Mar. 1960.

TARGETEER

Computer Sciences Department, The RAND Corporation, Santa Monica, California

Reported by: F. M. Tonge (September 1960)

Descriptors: **computer application, programming, aircraft routing, dynamic programming**

This routine determines the highest expected target value routing of aircraft through a target complex. Routing is done by an iterative dynamic programming technique modified to recognize aircraft range restrictions and to prevent multiple visits by any aircraft to a single target.

Location, range and number of weapons of each aircraft, target values, and intertarget distances and survival probabilities are required as input data. The routine prints out the targeting of each aircraft on each iteration of the routing scheme and a final summary of aircraft and target statuses. The relative maximum numbers of aircraft, targets, and weapons can be varied by recompiling. This routine was developed as a research tool for evaluating routing techniques.

Conference on Information Retrieval Oriented Languages

October 6, 1961
RCA, Princeton, N. J.

By the Subcommittee on Information Retrieval of the ACM Special Interest Committee on Computer Languages

So far, no computer language designed especially for storage and retrieval applications has received widespread publicity. However, a number of languages have been developed for searching files, and languages have been designed for other specific problems. It would be interesting to know how many of these languages have been machine tested, and to know how many are being used. Are these languages similar to each other? If not, how do they differ? What are their specifications? What are the limitations? Probably, some of the 'blue sky' languages include 'things to come'.

The subcommittee needs all the information it can recover, discover and uncover about IR languages. If you have first-hand experience in designing, programming, testing or using a machine language for storage and retrieval problems, please contact one of the subcommittee members:

HERB KOLLER, R & D Group, U. S. Patent Office, Washington 25, D. C.
JACK MINKER, Astro-Electronic Products Division, RCA, Princeton, New Jersey
MANDY GREMS, IBM Corp., White Plains, New York

Algorithms

ALGORITHM 58
MATRIX INVERSION

DONALD COHEN
Burroughs Corporation, Pasadena, Calif.

```
procedure invert (n) array: (a);
comment  matrix inversion by Gauss-Jordan elimination;
    value n;
    array a;  integer n;
begin
    array b, c [1:n];  integer i, j, k, ℓ, p;
    integer array z [1:n];
        for j := 1 step 1 until n do z[j] := j;
        for i := 1 step 1 until n do begin
        k := i;  y := a[i, i];  ℓ := i − 1;  p := i + 1;
        for j := p step 1 until n do begin
        w := a[i, j];  if abs(w) > abs(y) then begin
        k := j;  y := w end end;
        for j := 1 step 1 until n do begin
        c[j] := a[j, k];  a[j, k] := a[j, i];
        a[j, i] := −c[j]/y;  b[j] := a[i, j] := a[i, j]/y end  ;
        a[i, i] := 1/y;  j := z[i];  z[i] := z[k];  z[k] := j  ;
        for k := 1 step 1 until ℓ, p step 1 until n do
        for j := 1 step 1 until ℓ, p step 1 until n do
        a[k, j] := a[k, i] − b[j] × c[k] end;  ℓ := 0  ;
back:   ℓ := ℓ + 1;  k := z[ℓ];  if ℓ ≦ n then begin
        for j := ℓ while k ≠ j do begin
        for i := 1 step 1 until n do begin
        w := a[j, i];  a[j, i] := a[k, i];  a[k, i] := w end  ;
        go to back end
        end invert.
```

ALGORITHM 59
ZEROS OF A REAL POLYNOMIAL BY RESULTANT PROCEDURE

E. H. BAREISS and M. A. FISHERKELLER
Argonne National Laboratory, Argonne, Ill.

```
procedure  RES (n, c, alpha, mu, re, im, rt, gc)  ;  value n,
           c, alpha  ;  integer n, alpha  ;  integer array
           mu  ;  array c, re, im, rt, gc  ;
comment    RES finds simultaneously all zeros of a polynomial of
    degree n with real coefficients, c_j (j = 0, ...  n), where c_n
    is the constant term. The real part, re_i , and imaginary part,
    im_i , of each zero, with corresponding multiplicity, mu_i , and
    remainder term, rt_i , (i = 1, ... , n), are found and a poly-
    nomial with coefficients gc_j (j = 0, ... , n), is generated from
    these zeros. Alpha provides an option for local or nonlocal
    selection of M, the number of root-squaring iterations, and
    delta and epsilon, acceptance criteria. If alpha = 1, these
    parameters are assigned locally. If alpha = 2, M, delta and
    epsilon are set equal to the global parameters Mp, deltap,
    and epsilonp, respectively. In cases where zeros may be found
```

more than once, the superfluous ones are eliminated by factorization. The method has been described by E. H. Bareiss (J. ACM 7, Oct. 1960, pp. 346–386).   ;

```
        begin integer M   ; real delta, epsilon   ; switch U :=
        U1, U2   ;
        go to U [alpha];
U1:             M := 10   ; delta := 0.2   ; epsilon := 10⁻⁸   ;
                go to START   ;
U2:             M := Mp   ; delta := deltap   ; epsilon :=
                epsilonp   ;
START:          begin integer CT, nu, nuc, beta, m, j, jc, k,
                i, p   ; Boolean ROOT   ;
                real X, Y, GX, rp   ; array a, ac [0:n, 0:M],
                R, Rc, t [0:n],
                   s [-1:n], ag [-2:n], rh, q, G, F [1:2×n]   ;
                switch S := S1, S2   ; switch T := T1, T2   ;
                switch V := V1, V2   ;
                real procedure min (u,v)   ; real u,v   ;
                   min := if u ≦ v then u else v   ;
                real procedure SYND (W, Q, I, T)   ;
                integer I   ; real W,Q   ;
                   array T   ;
SYNTHETIC       begin s [-1] := 0   ; s [0] := T [0]   ; for
  DIV:          m := 1 step 1 until I do
                   s [m] := T [m] - W*s [m - 1] - Q×s
                [m - 2]   ;
                if Q = 0 then SYND := abs (s[I]) else
                   SYND := abs (W/2×s [I - 1] + s[I])
                end SYND   ;
                CT := beta := 1   ; for j := 0 step 1 until
                n do a [j,0] := c[j]   ;
SQUARING        begin integer e1   ; real h   ; for m :=
  OPERATION:    1 step 1 until M do
                begin for j := 1 step 1 until n do
                begin h := 0   ; for e1 := 1 step 1 until
                min (n - j, j) do
                   h := h + (-1) ↑ e1 × a [j - e1,m - 1] × a
                (j + e1, m - 1]   ;
                   a [j,m] := (-1) ↑ j × (a [j, m - 1] ↑
                   2 + 2×h) end end end   ;
                for j := 0 step 1 until n do R [j] := (-1) ↑
                j×a [j, M - 1] ↑ 2/a [j,M]   ;
                   j := 0   ; nu := 1   ;
RD:             if (1 - delta ≦ R [j]) ∧ (R [j] ≦ 1 + delta)
                then
                begin rp := (a [j,M]/a [j - nu, M]) ↑ (1/(2 ↑
                M×nu))   ;
                go to T [beta] end   ;
1:              nu := nu + 1   ;
2:              j := j + 1   ; if j = n then go to S [beta]
                else go to RD   ;
3:              nu := 1   ; go to 2   ;
T1:             rh [CT] := rp   ; X := rp + epsilon × rp   ;
                Y := X + epsilon × rp   ;
                for k := 0 step 1 until n do t [k] := abs (c[k]) ;
                   F [CT] := SYND (Y,0,n,t) - SYND
                   (X,0,n,t)   ;
                   G [CT] := SYND (rh [CT],0,n,c)   ; if
                   F [CT] > G [CT] then
                begin ROOT := true   ; q [CT] := 0   ;
                CT := CT + 1   ; F [CT] := F[CT - 1] end   ;
                rh [CT] := -rp   ; G [CT] := SYND (rh
                [CT],0,n,c)   ;
                if F [CT] > G [CT] then begin ROOT :=
                true   ; q [CT] := 0   ; CT := CT + 1   ;
                F [CT] := F [CT - 1] end   ; if nu = 1 then
                go to 2   ;
                q [CT] := rp ↑ 2   ; nuc := nu   ; jc := j   ;
                for j := 0 step 1 until n do
                begin Rc [j] := R [j]   ; ac [j,M] := a [j,M]
                end   ;
RESULTANT:      begin real h   ; array b [-1:n + 1,
                -1:n + 1], A [1:n],
                   r [0:n, 0:n], CB [-1:n + 1]   ;
                b [-1,0] := CB [-1] := CB [n + 1] := 0   ;
                for j := 0 step 1 until n do
                CB [j] := c[j]   ; b [0,0] := 1   ; for k :=
                1 step 1 until n do
                begin b [k,-1] := 0   ; for j := 0 step 1
                until k do
                   b [k + 1,j] := b [k,j - 1] - q [CT] × b
                   [k - 1,j]   ;
                   b [k + 1, k + 1] := h := 0   ; for j :=
                   n - k step -1 until 0 do
                   h := h + (CB [j]×CB [k + j] - CB [j - 1]
                   ×CB[k + j + 1]) × q [CT] ↑ (n - k - j)   ;
                   A [k] := (-1 ↑ k×h   ; for j := 0 step
                   1 until k - 1 do
                begin r [0,j] := 0   ; r [k,j] := r [k - 1,j] +
                A [k] × b [k,j] end   ;
                   r [k,k] := A[k] end   ; beta := 2   ; for
                   j := 0 step 1 until n do
                   a [j,0] := r [n,j] end   ; go to SQUAR-
                   ING OPERATION   ;
T2:             if (rp/2) ↑ 2 ≧ q [CT] then go to 3   ; rh
                [CT] := rp   ;
                G [CT] := SYND (rh [CT], q [CT], n,c)   ;
                if F [CT] > G [CT] then
                begin CT := CT + 1   ; F [CT] := F
                [CT - 1]   ; q [CT] := q [CT - 1] end   ;
                rh [CT] := -rp   ; G [CT] := SYND [rh [CT],
                q [CT], n,c)   ;
                if F [CT] > G [CT] then begin CT := CT
                + 1   ; F [CT] := F [CT - 1]   ;
                q [CT] := q [CT - 1] end   ; go to 3   ;
S2:             for j := 0 step 1 until n do begin a [j,M] :=
                ac [j,M]   ;
                R [j] := Rc [j] end   ; j := jc   ; beta := 1   ;
                if ROOT then go to 3 else
                   nu := nuc   ; go to 1   ;
S1:             ag [-2] := ag [-1] := 0   ; ag [0] := 1   ;
                for j := 1 step 1 until n do
                ag [j] := 0   ; k := 1   ; i := n   ; m := 1   ;
                for j := 0 step 1 until n do
                   t [j] := c [j]   ;
MULT:           mu [m] := 0   ; p := if q [k] = 0 then 1
                else 2   ;
IT:             GX := SYND (rh [k], q [k],i,t)   ; if F [k]
                > GX then
                begin for j := 1 step 1 until n do
                   ag [j] := ag [j] - rh [k] × ag [j - 1] + q
                   [k] × ag [j - 2]   ;
                   mu [m] := mu [m] + p   ; i := i - p   ;
                   for j := 0 step 1 until i do
                   t [j] := s [j]   ; go to IT end else if
                   mu [m] ≠ 0 then begin
                   rt [m] := G [k]   ; go to V [p] end else
                   go to D   ;
V1:             re [m] := rh [k]   ; im [m] := 0   ; go to E   ;
V2:             re [m] := rh [k]/2   ; im [m] := sqrt (q [k] -
                re [m] ↑ 2)   ;
E:              m := m + 1   ;
D:              k := k + 1   ; if k ≦ CT ∧ m ≦ n then go to
                MULT   ;
                for j := 0 step 1 until n do gc [j] := ag [j] end
                end RES
```

CERTIFICATION OF ALGORITHM 23
MATHSORT (Wallace Feurzeig, *Comm. ACM*, Nov., 1960)

RUSSELL W. RANSHAW
University of Pittsburgh, Pittsburgh, Pa.

The MATHSORT procedure as published was coded for the IBM 7070 in FORTRAN. Two deficiencies were discovered:

1. The TOTVEC array was not zeroed within the procedure. This led to some difficulties in repeated use of the procedure.

2. Input vectors already in sort on nonsort fields were unsorted. That is, given the sequence

31, 21, 32, 22, 33,

Mathsort would produce, for a sort on the 10's digit:

22, 21, 33, 32, 31,

which is definitely out of sequence.

The following modified form of the procedure corrects these difficulties. Note the transformation of symbols.

```
procedure  MATHSORT (I, O, T, n, k, S); value n, k;
           array I, O; integer array T; integer procedure S;
           integer n, k;
begin      for i := 0 step 1 until k - 1 do T[i] := 0;
           for i := 1 step 1 until n do T[S(I[i])] := T[S(I[i])] + 1;
           for i := k - 2 step -1 until 0 do T[i] := T[i] +
           T[i + 1];
           for i := 1 step 1 until n do
                begin  O[n + 1 - T[S(I[i])]] := I[i];
                       T[S(I[i])] := T[S(I[i])] - 1;
                end
end MATHSORT.
```

Using the MATHSORT procedure ten times and having the procedure S supply each digit in order, 1000 random numbers of 10 digits each were sorted into sequence in 31 seconds. The method of locating the lowest element, interchanging with the first element, and continuing until the entire list has been so examined yielded a complete sort on the same 1000 random numbers in 227 seconds. Using the Table-Lookup-Lowest command in the 7070 yielded 56 seconds for the same set of random numbers.

CERTIFICATION OF ALGORITHM 30
NUMERICAL SOLUTION OF THE POLYNOMIAL EQUATION (K. W. Ellenberger, *Comm. ACM*, Dec. 1960)

WILLIAM J. ALEXANDER
Argonne National Laboratory,* Argonne, Ill.

ROOTPOL was coded by hand for the LGP-30 using the ACT-III Compiler with 24 bits of significance. The following corrections were found necessary.

(a) $b_{-1} := b_{-2} := c_{-1} := c_{-2} := d_{-1} := d_{-2} := e_{-1} := e_{-2} := 0$ *should be*
$b_{-1} := b_{-2} := c_{-1} := c_{-2} := d_{-1} := e_{-1} := h_{-1} := 0$

(b) $m :=$ entier $((n+1)/2)$ *should be*
$m :=$ entier $((n-1)/2)$

(c) $j_{n-j} := s$ *should be* $h_{n-j} := s$

(d) $q := h/h_{n-2}$ *should be* h_n/h_{n-2}

(e) $cj := b_j - p \times c_j - 1 - q \times c_{j-2}$ *should be*
$c_j := b_j - p \times c_{j-1} - q \times c_{j-2}$

(f) **if** $n_{n-1} = 0$ **then go to** BNTEST *should be*
if $h_{n-1} = 0$ **then go to** BNTEST

(g) $s :=$ sqrt $(q - (p/2)^3)$ *should be*
$s :=$ sqrt $(q - (p/2)^2)$

(h) **for** $j := 0$ **step** 2 **until** n **do** $h_j := b_j$ *should be*
for $j := 0$ **step** 1 **until** n **do** $h_j := b_j$

(i) **go to** BAIRSTOW *should be* **go to** ITERATE

The following correction was found necessary in the given example (Refer to "On Programming the Numerical Solution of Polynomial Equations," by K. W. Ellenberger, *Comm. ACM 3*, Dec., 1960):

$f(x) = (.10098)\ 10^8\ x^4 - (.98913)\ 10^6\ x^2 + (.10000)\ 10^6\ x + (.10000)\ 10^1 = 0$ *should be*

$f(x) = (.10098)\ 10^8\ x^4 - (.98913)\ 10^6\ x^3 - (.10990)\ 10^6\ x^2 + (.10000)\ 10^6\ x + (.10000)\ 10^1 = 0$

With these corrections the results obtained agree with those given in the example.

For equations of higher order it was found necessary to avoid repeated scaling of the reduced equation in order to prevent floating point overflow. The range on the exponent in the ACT III system is $-32 \leqq e \leqq 31$.

Further floating point overflow difficulties were experienced when certain coefficients in the reduced equation became small but not zero. The following additions were made to avoid this fault:

(a) **for** $j := 0$ **step** 1 **until** n **do** $h_j := d_j$ *was replaced by*
for $j := 0$ **step** 1 **until** n **do begin if** abs $(h_j/d_j) < K$ **then** $h_j := d_j$ **else** $h_j := 0$ **end**

(b) **for** $j := 0$ **step** 1 **until** n **do** $h_j := b_j$ *was replaced by*
for $j := 0$ **step** 1 **until** n **do** begin **if** abs $(h_j/b_j) < K$ **then** $h_j := b_j$ **else** $h_j := 0$ **end**

With the above changes the following results were obtained:

$x^4 - 3x^3 + 20x^2 + 44x + 54 = 0$
$x = -.9706390 \pm 1.005808i$
$x = 2.470639 \pm 4.640533i$

$x^6 - 2x^5 + 2x^4 + x^3 + 6x^2 - 6x + 8 = 0$
$x = -.9999999 \pm .9999999i$
$x = 1.500000 \pm 1.322876i$
$x = .5000002 \pm .8660251i$

$x^5 + x^4 - 8x^3 - 16x^2 + 7x + 15 = 0$
$x = 3.000001$
$x = -2.000000 \pm 1.000001i$
$x = -.9999997$
$x = .9999998$

* Work supported by the U. S. Atomic Energy Commission