THE TECHNOLOGICAL INSTITUTE

NORTHWESTERN UNIVERSITY

EVANSTON, ILLINOIS

SYSTEMS RESEARCH MEMORANDUM No. 129

The Technological Institute The College of Arts and Sciences
Northwestern University

EXTREME POINT SOLUTIONS IN MATHEMATICAL PROGRAMMING: AN OPPOSITE SIGN ALGORITHM

by

A. Charnes, K. Kortanek* and W. Raike

*University of Chicago and Northwestern University

June 1965

This paper is an extension and revision of research memorandum ONR #84 and SRM #104 for presentation at the TIMS meeting, Dallas, Texas, February 1966.

Part of the research underlying this report was undertaken for the Office of Naval Research, Contract Nonr-1228(10), Project NR 047-021, and for the U. S. Army Research Office - Durham, Contract No. DA-31-124-ARO-D-322 at Northwestern University. Reproduction of this paper in whole or in part is permitted for any purpose of the United States Government.

SYSTEMS RESEARCH GROUP

A. Charnes, Director

Several important and efficient methods of solution of specific types of linear programming problems, such as the "flooding" or "flow" methods of Boldyreff and Ford-Fulkerson for network flows and the "decomposition" principle or "mixing routines" of Dantzig-Wolfe or Charnes-Cooper, have the distressing feature for managerial applications of sometimes providing optimal solutions which are <u>not</u> extreme point (or basic) solutions. Thereby, any immediate and automatic availability of the optimal dual evaluators of the original problem is not forthcoming. Indeed, several such experiences have resulted in abandonment of these techniques for managerial studies in certain industrial firms.

Aside from sensitivity analysis, it is also often desirable to start from an operable ("feasible") solution, not necessarily optimal, which is either suggested by company personnel or from various pertinent qualitative considerations, and to be able to proceed automatically to an extreme point (= <u>basic</u>) solution which is at least as good as the suggested one. 1/ (Thereafter any one of the various methods could be employed to achieve optimality.)

The purpose of this paper is to exhibit how part of the technique of proof of the opposite sign theorem 2/ can be employed in a simple algorithmic manner to achieve this end. We present an ALGOL code for executing this algorithm in a manner compatible (as a procedure) with standard programs.

Consider for the moment the linear programming problem written in the form:

1/ There have been constructive methods which reduce any feasible solution to a basic solution, but which have not considered this important feature, or other features such as ability to employ a known but not necessarily feasible basis.

2/ See reference [3] on semi-infinite programming for a proof of it in general form.

$$\max f(\lambda) = \sum_{j=1}^{n} c_j \lambda_j$$

$$\text{subject to } \sum_{j=1}^{n} P_j \lambda_j = P_o$$

$$\lambda \geq 0$$

where P_j, P_o are all m-vectors. Let $\Lambda = \{ \lambda: \sum_{j=1}^{n} P_j \lambda_j = P_o , \lambda_j \geq 0 \}$.

We may assume that the problem is regularized. 1/ This means that the solution set is non-empty and bounded, so the Opposite Sign Theorem applies:

Λ is generated by its extreme points $\iff$: $\sum_{j=1}^{n} a_j P_j = 0$, not all

$a_j = 0 \implies$ some a_r , a_s are of opposite sign.

Now suppose some collection $B = \{v_1, v_2, \ldots, v_q , \; q \leq m\}$ is a basis for the subspace of R^m spanned by the P_j. (Some or all of the v_i and P_j may be identical; it is not required that B be composed only of certain P_j's.) For notational convenience, rename any v_i which is not present in the collection $\{P_j , j=1,\ldots,n\}$ as P_{n+i}, and let $I = \{ i : P_i \in B\}$.

Let $B^{\#}$ denote a left inverse of the $m \times q$ basis matrix B; 2/ then $a^T = B^{\#} P_k$ is the expression of P_k relative to B, or

$$\sum_{i \in I} a_i P_i = P_k ,$$

$$\text{so that} \quad \sum_{i \in I} a_i P_i + a_k P_k = 0, \quad \text{where } a_k = -1.$$

Now P_k can replace any P_i in the basis with $a_i \neq 0$, and by the opposite sign theorem some a_r and a_s are of opposite sign.

Let λ be <u>any</u> solution to $P\lambda = P_o$, $\lambda \geq 0$. Assume that some vector $P_k \notin B$ occurs with positive λ-component in the expression of P_o; if no such vector exists then we already have an extreme point solution.

We now must consider the two possibilities which arise relative to the values of the corresponding a_j and λ_j components:

1/ See Charnes-Cooper, reference [2], p. 424.

2/ See [4]; note that the matrix B need not be square.

<u>Case 1</u>. There exists some $\alpha_i \neq 0$ with corresponding $\lambda_i = 0$.

Choose one of these, say α_r, to designate a "remove" vector. No change will be effected in the functional to be considered until we have only P_j's in the basis; until this occurs we need not be concerned with the opposite sign property but are only choosing a basis from among the P_j vectors.

<u>Case 2</u>. For every $\alpha_i \neq 0$, $\lambda_i > 0$.

Form the quantities $\rho_1 = \min\limits_{\substack{\lambda_j > 0 \\ \alpha_j > 0}} \dfrac{\lambda_j}{\alpha_j}$ and $\rho_2 = \min\limits_{\substack{\lambda_j > 0 \\ \alpha_j < 0}} \dfrac{\lambda_j}{|\alpha_j|}$,

whose existence is guaranteed by the opposite sign theorem. Then $\lambda^1 = \lambda - \rho_1\alpha$, $\lambda^2 = \lambda + \rho_2\alpha$ will have every component ≥ 0 <u>and</u> each will have at least one more zero component than λ as long as the number of $\lambda_j > 0$ is greater than q. Note also that because λ is a solution it is true that $\lambda + \rho\alpha$ is also a solution for any real ρ, since $\sum_j \alpha_j P_j = 0$, so that in particular λ^1 and λ^2 are both feasible solutions.

Now let $f(\lambda)$ be convex, i.e., for any λ^1, λ^2

$$f((1-\gamma)\lambda^1 + \gamma\lambda^2) \leq (1-\gamma)f(\lambda^1) + \gamma f(\lambda^2), \quad 0 \leq \gamma \leq 1 \text{ holds.}$$

Since $\lambda = \dfrac{\rho_2}{\rho_1+\rho_2}\lambda^1 + \dfrac{\rho_1}{\rho_1+\rho_2}\lambda^2$, it follows that

$$f(\lambda) \leq \frac{\rho_2}{\rho_1+\rho_2} f(\lambda^1) + \frac{\rho_1}{\rho_1+\rho_2} f(\lambda^2)$$, so that <u>not both</u>

$f(\lambda^1) < f(\lambda)$ and $f(\lambda^2) < f(\lambda)$; i.e., one of λ^1, λ^2 yields at least as great a functional value as λ. Thus, the j which yields the minimum in the definition of ρ_1, when $f(\lambda^1) \geq f(\lambda^2)$ (or of ρ_2 when $f(\lambda^2) \geq f(\lambda^1)$) serves to designate a vector P_r which is to be replaced by P_k in the basis. At this point we begin again with the new basis and new feasible λ as before.

Thus at each stage we remove one or more vectors from the original set, while maintaining at least as great a value as in the previous stage, until the original set is reduced to a set of linearly independent vectors, i.e., the corresponding λ is an extreme point with at least as great a functional value as the original given solution. (A corresponding result holds when a concave functional is be be minimized.) We can summarize the above discussion in the following algorithm:

Step 1. Start with a feasible solution λ , $\sum_{j=1}^{n} P_j\lambda_j = P_o$, $\lambda \geq 0$, and a basis $B = \{P_i \ , i\varepsilon I\}$.

Step 2. Find any $\lambda_k>0$ with $P_k \not\varepsilon B$ to designate a vector to enter the basis; if none is available, λ is an extreme point and the process terminates.

Step 3. Obtain the α_i in $\sum_{i\varepsilon I} \alpha_i P_i + \alpha_k P_k = 0$, where $\alpha_k = -1$, by computing $\alpha^T = B^{\#} P_k$.

Step 4. (i) If there exist $\alpha_i \neq 0$ with corresponding $\lambda_i = 0$, choose one, say α_r , to designate a remove vector P_r . Replace P_r by P_k in B, compute the new $B^{\#}$, and go to step 2.

(ii) Otherwise set $\rho_1 = \min\limits_{\substack{\alpha_i>0 \\ \lambda_i>0}} \dfrac{\lambda_i}{\alpha_i}$ and $\rho_2 = \min\limits_{\substack{\alpha_i<0 \\ \lambda_i>0}} \dfrac{\lambda_i}{|\alpha_i|}$.

Step 5. Let $\lambda^{(1)} = \lambda-\rho_1\alpha$ and $\lambda^{(2)} = \lambda+\rho_2\alpha$. Let r_1 , r_2 be a pair of indices for which ρ_1 , ρ_2 achieve their respective minima. Choose P_{r_1} or P_{r_2} to be removed (= P_r) according as $f(\lambda^{(1)})$ or $f(\lambda^{(2)})$ is the larger.

Step 6. Substitute P_k for P_r in the basis, compute the new $B^{\#}$, and return to step 1 with $\lambda^{(i)}$ as a new λ .

Observe that the algorithm terminates when a λ is reached such that each P_k with $\lambda_k>0$ is in the current basis.

Notice again that it is possible to choose an initial basis consisting entirely of artificial or slack vectors (even though they may not be part of the original set of vectors), and that the procedure will automatically find a basic solution in terms of the original vectors.

It should be expressly noted that the process of choosing a vector to enter the basis (any vector with $\lambda_j>0$, and not already in the basis, may be chosen), computing a column of α_j ("substitution ratios"), selecting a vector to be removed from the basis (the one at which either the ρ_1 or the ρ_2 minimum ratio was achieved), and transforming the matrix of the basis inverse to reflect this change of basis corresponds closely in structure with that of the modified simplex method of A. Charnes and C. E. Lemke, 1/ although the actual criteria governing these operations are different. In particular, just as in the modified simplex method, the possibility of an unbounded solution is indicated by the absence of a positive α_i (pivot element), although, since there are normally two candidates for removal from the basis, this condition is critical only if the other candidate would cause a worsening of the solution; e.g., if we are maximizing, the availability of an infinite minimum will not cause an error stop; we would simply remove the vector associated with the minimum ρ_2 ratio from the basis if no decrease in the functional is caused thereby.

It is conceivable that the initial "solution" presented to the procedure could be really not a solution at all. If λ is the "solution" presented and $P\lambda = \bar{P} \neq P_o$, the resulting "basic solution" will be a basic solution to $P\lambda = \bar{P}$, since the algorithm never refers to the actual stipulations vector P_o ! For production use, therefore, it would be worth while to check before using the procedure that the solution proposed really is feasible.

The ALGOL procedure presented below assumes that $f(\lambda)$ is linear, i.e., $f(\lambda) = c^T\lambda$, which is both concave and convex. The roles of the formal parameters used in the procedure declaration are described in the following table.

1/ See [5] .

Formal Parameter	Description
m	The number of constraints. (The number of rows in P.)
n	The number of columns in P, exclusive of P_o.
size	The number of vectors in the basis, which is the dimension of the space spanned by the P_j. In most cases this will be equal to m.
basis	The list of indices of the P_i in the starting basis; if no starting basis is supplied then the elements of this array are irrelevant when the procedure is called.
matrix	The m x n constraint matrix.
inverse	A size x m (left) inverse of the initial basis; again, the elements are irrelevant if no basis is specified. One additional row should be adjoined which will contain the dual evaluators.
lambda	The initial feasible solution, whose components must be in the same order as the columns of the coefficient matrix.
c	The components of the objective, in the same order as those of lambda.
objective	An integer which should be zero or one if minimization or maximization, respectively, is desired.
option	An indicator which should be set to one if no starting basis is specified in which case the m x m identity matrix will be used, so that size must be equal to m; if a basis and inverse are provided, this indicator should be set to 2.
unbounded	The name of a sentinel which will normally be set to zero but which assumes a non-zero value in case an infinite optimum is attainable.

In terms of the ALGOL identifiers used as formal parameters of the procedure, upon completion of the process the integer array, basis, contains the numbers of the vectors in the basic solution; the array, lambda, contains the actual values of the λ_j, and unbounded is equal to 0; if not zero, it was set to the number of the vector attempting to enter the basis when unboundedness was noticed and the process terminated. The basis (left) inverse is contained in inverse, which also includes a size + 1st row containing the dual evaluators for the current basic solution.

ALGOL Procedure

```
procedure purify (m, n, size, basis, matrix, inverse, lambda, c, objective,
                  option, unbounded) ;

value m, n, size, objective, option ;

integer m, n, size, objective, option, unbounded ;

real array matrix, inverse, lambda, c ;

integer array basis ;

begin integer i, j, k, r, r1, r2 ;

      array alpha [1 : size + 1] ;

      real rho1, rho2, t, d1, d2 ;

      unbounded : = 0 ;

      comment if necessary, generate augmented identity matrix for the
              left inverse and dual evaluators ;

      if option = 1 then for j : = 1 step 1 until m do

      begin for i : = 1 step 1 until m + 1 do

            inverse [i,j] : = if i = j then 1 else 0 ;

            basis [j] : = n+j

      end

      look for some lambda greater than zero to determine a vector
      to enter the basis ;
```

```
again : for j: =1 step 1 until n do

        if lambda [j] <10-7 then lambda [j] : = 0

        else begin for i : =1 step 1 until size do

                                if basis [i] =j then goto skip it ;

                                comment if we get to here, vector j was
                                        not already in the basis ;

                                k: =j ;

                                goto come in ;

skip it:        end

        if we get to here, no vector k can be found with lambda [k] > 0 and
        which is not already in the basis, hence we are done. ;

        goto finished ;

come in: for i:=1 step 1 until size + 1 do

        begin alpha [i] : = 0 ;

                for j: = 1 step 1 until m do

                alpha [i] : = alpha [i] + inverse [i,j]* matrix [j,k]

        end ;

        alpha [size + 1] : = alpha [size + 1] - c[k] ;

        rho1   : = rho2   : = 10 20 ;

        comment now find two candidates for removal ;

        for i  : = 1 step 1 until size do

        if alpha [i] >  10 -7 then

                begin if basis [i] > n then

                        begin r : = i ; goto transform end

                        else if lambda [basis [i] ] < 10 -7 then

                        begin r : = i ; goto transform end
```

```
                else begin t := lambda [basis [i] ] / alpha [i] ;
                           if t < rho1  then
                           begin rho1  := t ; r1 := i end
                     end
          end
     else if alpha [i] < -10 -7 then
          begin if basis [i] > n then
                     begin r := i ; goto transform end
                else if lambda [basis [i] ] < 10 -7 then
                     begin r := i ; goto transform end
                else begin t := - lambda [basis [i] ] / alpha [i] ;
                           if t < rho2  then
                           begin rho2  := t ; r2 := i end
                     end
          end ;
     if lambda [k] < rho2  then
                begin rho2  := lambda [k] ; r2 := -1 end ;
     d1 := c[k] * rho1  ;
     d2 := -c[k] * rho2  ;
     for i := 1 step 1 until size do
     begin t := c [basis [i] ] * alpha [i] ;
          d1 := d1 - rho1  * t ;
          d2 := d2 + rho2  * t
     end ;
     comment now choose the best vector to be removed, either r1 or r2 ;
```

```
        if objective = 0 ≡ d1 ≤ d2
        then goto takeout 1 else goto takeout 2 ;
takeout 1:  if rho1  = 10 20 then begin unbounded : = k ; goto finished end ;
        for i : = 1 step 1 until size do
               lambda [basis [i] ] : = lambda [basis [i] ] - rho1  * alpha [i];
        lambda [k] : = lambda [k] + rho1   ;
        r : = r 1 ;
        goto transform ;
takeout 2:  for i : = 1 step 1 until size do
               lambda [basis [i] ] : = lambda [basis [i] ] + rho2  * alpha [i];
        lambda [k] : = lambda [k] - rho2 ;
        if r2 = -1 then goto again else r : = r2 ;
transform:  for j : = 1 step 1 until m do
        begin inverse [r,j] : = inverse [r,j] / alpha [r] ;
               for i : = 1 step 1 until size + 1 do
               if i ≠ r then
                    inverse [i,j] : = inverse [i,j] - inverse [r,j] * alpha [i]
        end ;
        basis [r] : = k ;
        goto again ;
finished:  end ;
```

Example

We present a simple example with a summary of the iterations taken to reach a basic solution, starting from an initial basis not consisting of vectors taken from the original coefficient matrix.

$$\begin{array}{ll} \max & 2x_1 + x_2 + 4x_3 + 3x_4 + x_5 \\ \text{subject to} & 5x_1 + x_2 + 6x_3 \qquad + 3x_5 = 26 \\ & 7x_1 + x_2 + 2x_3 - x_4 - 2x_5 = 5 \\ & \text{and all } x_i \geq 0 \end{array}$$

Stage	Come-in Vector	Remove Vector	Current Basis	λ	$f(\lambda)$	Case
Initial	--	--	$\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$	(2,1,1,6,3,0,0)	30	--
1	P_1	P_6	$\begin{pmatrix}5 & 0\\ 7 & 1\end{pmatrix}$	(2,1,1,6,3,0,0)	30	Case 1
2	P_2	P_7	$\begin{pmatrix}5 & 1\\ 7 & 1\end{pmatrix}$	(2,1,1,6,3,0,0)	30	Case 1
3	P_3	P_1	$\begin{pmatrix}1 & 6\\ 1 & 2\end{pmatrix}$	(0,17,0,6,3,0,0)	38	Case 2
4	P_4	P_3	$\begin{pmatrix}1 & 0\\ 1 & -1\end{pmatrix}$	(0,17,0,6,3,0,0)	38	Case 1
5	P_5	P_5	$\begin{pmatrix}1 & 0\\ 1 & -1\end{pmatrix}$	(0,26,0,21,0,0,0)	89	Case 2

The <u>optimum</u> value for this problem is 104 - 3/5 .

REFERENCES

[1.] Charnes, A., and W. W. Cooper, "The Strong Minkowski-Farkas-Weyl Theorem for Vector Spaces over Ordered Fields," Proc. Nat. Acad. Sci. U.S.A., 44, No. 9.

[2.] Charnes, A., and W. W. Cooper, Management Models and Industrial Applications of Linear Programming, Vols. I and II, New York, John Wiley and Sons, 1961.

[3.] Charnes, A., W. W. Cooper, and K. Kortanek, "Duality in Semi-infinite Programs and Some Works of Haar and Caratheodory," Management Science, Vol. 9, No. 2, Jan. 1963.

[4.] Charnes, A., and M. Kirby, "A Linear Programming Application of a Left Inverse of a Basis Matrix," ONR Research Memo No. 91.

[5.] Charnes, A., and C. E. Lemke, "A Modified Simplex Method for Control of Roundoff Error in Linear Programming," Proc. ACM, Pittsburgh, May 1952.

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Northwestern University	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

EXTREME POINT SOLUTIONS IN MATHEMATICAL PROGRAMMING: AN OPPOSITE SIGN ALGORITHM

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Research paper

5. AUTHOR(S) (Last name, first name, initial)

Charnes, Abraham; Kortanek, Kenneth O.; and Raike, William M.

6. REPORT DATE June 1965	7a. TOTAL NO. OF PAGES 13	7b. NO. OF REFS 5
8a. CONTRACT OR GRANT NO. Nonr-1228(10) b. PROJECT NO. NR 047-021 c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) Systems Research Memorandum No. 129 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) None	

10. AVAILABILITY/LIMITATION NOTICES

Releasable without limitation on dissemination.

11. SUPPLEMENTARY NOTES Extension and revision of ONR #84 and SRM #104	12. SPONSORING MILITARY ACTIVITY Logistics and Mathematical Statistics Branch - Office of Naval Research Washington, D. C. 20360

13. ABSTRACT

Several important and efficient methods of solution of specific types of linear programming problems have the feature of sometimes providing optimal solutions which are not extreme-point (or basic) solutions, so that important and useful analyses provided by knowledge of the optimal dual evaluators are not available. It is also often desirable to be able to begin with a solution suggested by knowledgeable persons with experience in the field (or other considerations) and to proceed immediately to a basic solution at least as good as the suggested one.

In this paper it is shown how part of the technique of proof of the opposite sign theorem can be employed in a simple algorithm to achieve this end. This method is equally valid when maximizing a nonlinear but convex objective function. A tested ALGOL code is provided for executing the algorithm in a manner compatible (as a procedure) with other programs.

DD FORM 1 JAN 64 1473

Unclassified

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
linear programming convex programming extreme point basic solution opposite sign theorem experience solutions sensitivity analysis dual evaluators						

INSTRUCTIONS

1. **ORIGINATING ACTIVITY:** Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2a. **REPORT SECURITY CLASSIFICATION:** Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. **GROUP:** Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. **REPORT TITLE:** Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. **DESCRIPTIVE NOTES:** If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. **AUTHOR(S):** Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. **REPORT DATE:** Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7a. **TOTAL NUMBER OF PAGES:** The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. **NUMBER OF REFERENCES:** Enter the total number of references cited in the report.

8a. **CONTRACT OR GRANT NUMBER:** If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. **PROJECT NUMBER:** Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. **ORIGINATOR'S REPORT NUMBER(S):** Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. **OTHER REPORT NUMBER(S):** If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. **AVAILABILITY/LIMITATION NOTICES:** Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through
____________________."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through
____________________."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through
____________________."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. **SUPPLEMENTARY NOTES:** Use for additional explanatory notes.

12. **SPONSORING MILITARY ACTIVITY:** Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. **ABSTRACT:** Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)*, or *(U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. **KEY WORDS:** Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, roles, and weights is optional.