N90-25055

Minimizing Distortion and Internal Forces in Truss Structures by Simulated Annealing

by
Professor Rex K. Kincaid
Mathematics Department
The College of William and Mary
Williamsburg, VA 23185

Inacuracies in the length of members and the diameters of joints of large truss reflector backup structures may produce unacceptable levels of surface distortion and member forces. However, if the member lengths and joint diameters can be measured accurately it is possible to configure the members and joints so that root-mean-square (rms) surface error and/or rms member forces is minimized.

Following Greene and Haftka (1989) we assume that the force vector **f** is linearly proportional to the member length errors $\mathbf{e}_M$ of dimension NMEMB (the number of members) and joint errors $\mathbf{e}_J$ of dimension NJOINT (the number of joints), and that the best-fit displacement vector **d** is a linear function of **f**. Let NNODES denote the number of positions on the surface of the truss where error influences are measured. Let $\mathbf{U}_M$ (NNODES x NMEMB) and $\mathbf{U}_J$ (NNODES x NJOINT) denote the matrices of influence coefficients. Then $\mathbf{d} = \mathbf{U}_M\mathbf{e}_M + \mathbf{U}_J\mathbf{e}_J$. Concatenating $\mathbf{e}_M$ with $\mathbf{e}_J$ and $\mathbf{U}_M$ with $\mathbf{U}_J$ yields $\mathbf{d} = \mathbf{Ue}$.

Let **D** be a positive semidefinite weighting matrix (in our computational experiments we let **D** be an identity matrix) denoting the relative importance of the surface nodes where distortion is measured. The mean-squared displacement error can then be written as

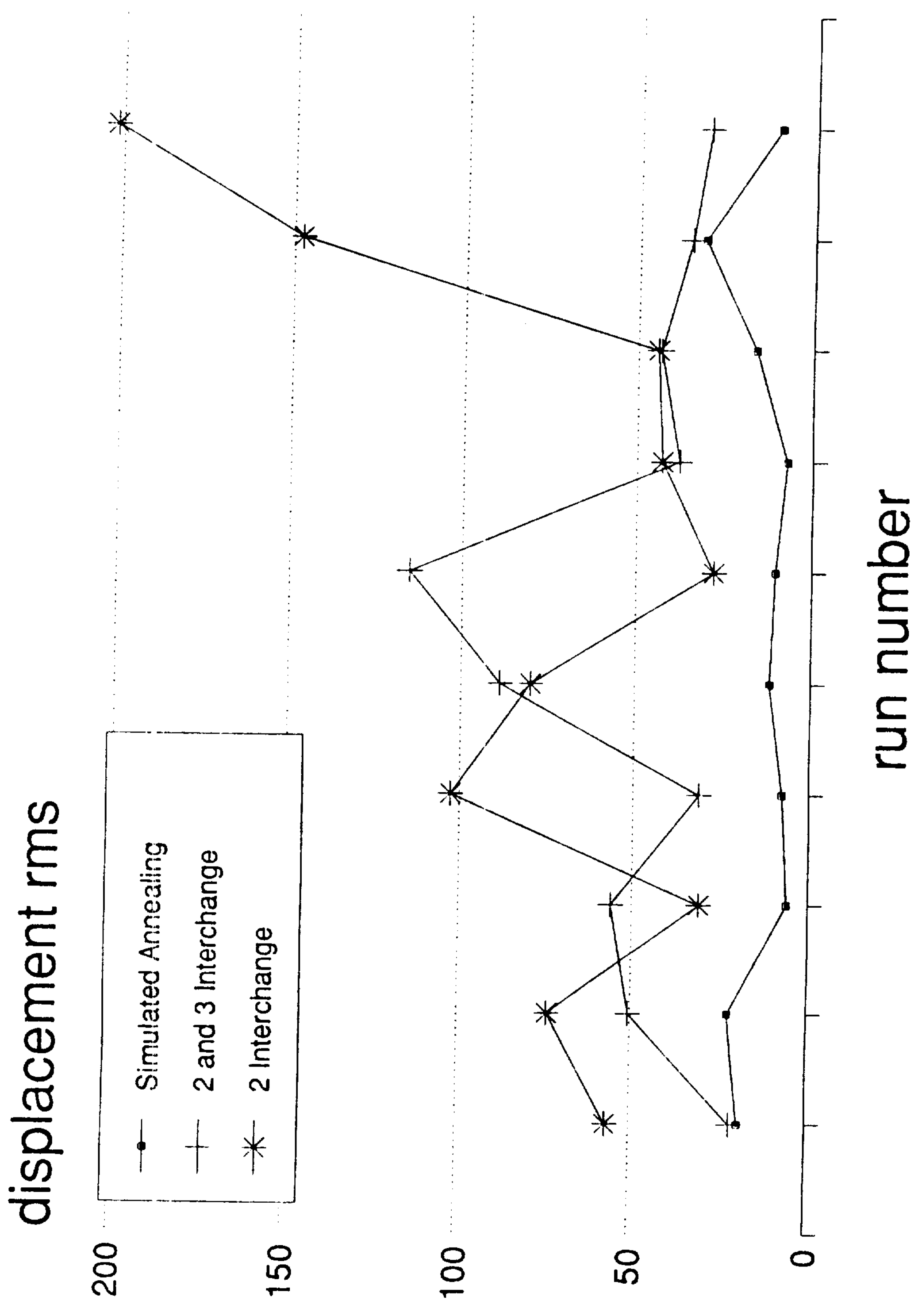
$$\mathbf{d}^2_{rms} = \mathbf{e}^T\mathbf{U}^T\mathbf{DUe} = \mathbf{e}^T\mathbf{He}.$$

A similar construction can be derived for mean-squared member force error, $\mathbf{s}^2_{rms}$ (see Greene and Haftka (1989)). Minimizing $\mathbf{d}^2_{rms}$ (or $\mathbf{s}^2_{rms}$) can be formulated as a combinatorial optimization problem. That is, finding the permutation of the components of $\mathbf{e}_M$ and $\mathbf{e}_J$ that minimizes $\mathbf{d}^2_{rms}$ (or $\mathbf{s}^2_{rms}$) is equivalent to minimizing $\mathbf{d}^2_{rms}$ (or $\mathbf{s}^2_{rms}$) directly. Unfortunately there (NMEMB!)*(NJOINT!) possiblities to consider. Hence, an enumeration scheme is out of the question. However there are many combinatorial optimization problems with exponentially large solution spaces that can be solved by algorithms whose time complexity is bounded by a polynomial function of the problem parameters.

To classify this problem we compare it to a similar combinatorial optimization problem. In particular, when only the member length errors are considered, minimizing $\mathbf{d}^2_{rms}$ is equivalent to the quadratic assignment problem. The quadratic assignment problem is a well known NP-complete problem in the operations research literature. Hence, minimizing $\mathbf{d}_{rms}^2$ is also an NP-complete problem. Moreover, if a problem is NP-complete it is highly unlikely that an algorithm exists which can determine an optimal solution in polynomial time and, therefore, (polynomial time) heuristic solution techniques should be employed. Greene and Haftka (1989) tested two heuristics of the same type. They use pairwise interchange and triple interchange of the members and joints to reduce $\mathbf{d}_{rms}^2$. The focus of our research has been the development of a simulated annealing algorithm to reduce $\mathbf{d}_{rms}^2$. The plausibilty of this technique has been its recent success on a variety of NP-complete combinatorial optimization problems including the quadratic assignment problem.

Simulated annealing was first proposed and used in statistical mechanics in the early 1950's (see Metropolis et al. (1953)). However, not until Cerny (1982) was simulated annealing used to solve a NP-complete combinatorial optimization problem--the traveling salesman problem. A physical analogy for simulated annealing is the way liquids freeze and crystallize. As the liquid is cooled slowly the atoms line themselves up and form a pure crystal that is completed ordered. The pure crystal is the minimum energy for this system. The basic procedure consists of a loop over a random displacement generator that produces changes in the objective function value. If this change is negative the displacement is accepted and the objective function is reduced. If this change is non-negative the displacement is accepted probabalistically. That is, uphill climbs are accepted with some positive probability which decreases as the temperature decreases. Simulated annealing must be used with some care. In addition to determining how to generate random displacements, one must also pick a starting temperature T, a cooling rate TFACTR, and a stopping temperature T_f. If these parameters are not chosen appropriately simulated annealing may produce poor results and/or run for an exponential amount of time.

Figure 1 is a graph of the objective function value ($\mathbf{d}_{rms}^2$) for ten random starting arrangements of the components of **e** for three different heuristics. All computational experiments were done on a MicroVAX. The two interchange heuristic is very fast (an average cpu time of 1.1 minutes per run) but produces widely varying results. The two and three interchange heuristic provides less variability in the final objective function values but runs much more slowly (an average cpu time of 68 minutes per run). Simulated annealing produced the best objective function values for every starting configuration and was faster than the two and three interchange heuristic (an average cpu time of 42 minutes per run).


displacement rms
200
150
100
50
0
Simulated Annealing
2 and 3 Interchange
2 Interchange
run number

```
C
C*********** MAIN PROGRAM *************
C
        INTEGER NMEMB,NJOINT,NROW
        PARAMETER  (NMEMB=102,NJOINT=31,NROW=NMEMB+NJOINT,NW=19)
        INTEGER IORDER(NROW),NDIM
        DOUBLE PRECISION H,M,ZQAP,T,TFACTR,ZCHK,PSUM,TSUM
        DIMENSION H(NROW,NROW),M(NROW),PSUM(NROW)
     $,DFDE(NW,NROW),DWDE(NW,NROW)
        CHARACTER*35 MSG
        REAL TIM(20)
        OPEN(UNIT=5,FILE='QAP2.IN',STATUS='OLD')
        OPEN (UNIT=6,FILE='QAPJNT.OUT',STATUS='UNKNOWN')
        OPEN (UNIT=7,FILE='MTEN.DAT',STATUS='OLD')
C
C*********************************************************************
C
C       Read in input data. Influence matrix H=UDU, member
C       length errors M, joint diameter errors M, displacement
C       derivatives DWDE, force derivatives DFDE, and initial
C       objective function value ZQAP.  The input file QAP.IN
C       is created by GENQAP.FOR.
C
C*********************************************************************
C
        DO 21 I=1,NROW
               READ(5,901)  (H(I,J),J=1,NROW)
21      CONTINUE
        DO 20 I=1,NMEMB
               READ(5,901)  (DFDE(I,J),J=1,NROW)
20      CONTINUE
        DO 22 I=1,NW
               READ(5,901)  (DWDE(I,J),J=1,NROW)
22      CONTINUE
        DO 2400 J=1,3
               DO 17 I=1,NROW
                      IORDER(I)=I
17             CONTINUE
               READ(7,901)  (M(I),I=1,NROW)
C
               READ(7,902)  ZQAP
C
               READ(7,900)  MSG
C
C*********************************************************************
C
C       Use the largest eigenvalue of H to provide a bound on
C       the difference between the largest and smallest objective
C       function values.  For this H, 9.779335 is the appropriate
C       eigenvalue.
C
C*********************************************************************
C
               T=0.0
               DO 79 I=1,NROW
                      T=T+M(I)*M(I)
79             CONTINUE
               T=T*10*9.779335
               TFACTR=0.96
900            FORMAT(1X,A)
901            FORMAT(1X,5E16.12)
902            FORMAT(1X,E16.12)
C
C
C********* End initialization and echo results
C
C
               WRITE(6,*)  'ITERATION ',J
               WRITE(6,*)  'Start Temperature= ',T,TFACTR
               WRITE(6,*)  'Starting ZQAP=',ZQAP
               CALL SECOND(TIM(J))
C
               CALL ANNEAL(M,H,IORDER,NMEMB,NROW,TFACTR,ZQAP,T)
C
               CALL SECOND(TIM(10+J))
               WRITE(6,*)  'Execution time ',TIM(10+J)-TIM(J)
               WRITE(6,*)  'Final annealing objective value ',ZQAP
C
               CALL OBJCHK(M,IORDER,H,PSUM,NROW,ZCHK)
C
```

```
                WRITE(6,*) 'Obj. value check ',ZCHK
                WRITE(6,*) (IORDER(I),I=1,NROW)
                WRITE(6,*) 'Final temperature ',T
2400    CONTINUE
        STOP
        END
C
C
C
        SUBROUTINE  ANNEAL(M,H,IORDER,NMEMB,NROW,TFACTR,ZQAP,T)
C
C
C***********************************************************************
C
C       This algorithm finds the permutation of the components
C       of the vector M that minimizes the product MHM for any
C       real symmetric positive definite matrix H.  There are
C       NROW (NMEMB+NJOINT) components of M and H is NROW by NROW.
C       The array IORDER(I) specifies the permutation of M.  On
C       input, the elements of IORDER may be set to any permutation
C       of the numbers 1 to NROW.  This routine will return the best
C       alternative permutation it can find.
C
C       T is the current temparture.
C       NOVER is the max number of swaps tried at any temperature T.
C       NLIMIT is the max number of successful swaps before continuing.
C       TFACTR is the annealing schedule, Tnew=Told*TFACTR.
C       ZQAP denotes the objective function value at any time T.
C       DE denotes the change in ZQAP when two components are swapped.
C
C***********************************************************************
C
C
        INTEGER NMEMB,IORDER(NROW),N(2),NOVER,NLIMIT,IDUM
        DOUBLE PRECISION M,H,TFACTR,ZQAP,DE,T,TSUM
        DIMENSION M(NROW),H(NROW,NROW)
        LOGICAL ANS
        NOVER=10*NROW
        NLIMIT=1*NROW
        IDUM=-1
        NSUCC=1
        NCNT=0
        NJOINT=NROW-NMEMB
C
C******* Loop until temperature is too small or NSUCC=0.
C
        DO WHILE (NCNT.LT.600.AND.NSUCC.GT.0)
                NCNT=NCNT+1
                NSUCC=0
C
C******* Local search of neighbors of current assignment
C
                DO 12 K=1,NOVER
C
C******* N(1) and N(2) are the two components of M to be swapped.
C
                IF (RAN3(IDUM).GT.0.76692) THEN
                        N(2)=1+INT(NJOINT*RAN3(IDUM))
                        N(1)=1+INT((NJOINT-1)*RAN3(IDUM))
                        IF (N(2).EQ.N(1).AND.N(2).EQ.NJOINT) THEN
                                N(1)=N(1)-1
                        ELSE IF (N(2).EQ.N(1)) THEN
                                N(2)=N(2)+1
                        ENDIF
                ELSE
                        N(2)=1+INT(NMEMB*RAN3(IDUM))
                        N(1)=1+INT((NMEMB-1)*RAN3(IDUM))
                        IF (N(2).EQ.N(1).AND.N(2).EQ.NMEMB) THEN
                                N(1)=N(1)-1
                        ELSE IF (N(2).EQ.N(1)) THEN
                                N(2)=N(2)+1
                        ENDIF
                ENDIF
                        CALL SWPCST(M,H,IORDER,NMEMB,NROW,N,DE)
                        CALL METROP(DE,T,ANS)
                                IF (ANS) THEN
                                        NSUCC=NSUCC+1
                                        ZQAP=ZQAP+DE
                                        CALL SWAP(IORDER,NROW,N)
                                ENDIF
                        IF (NSUCC.GE.NLIMIT) GOTO 2
12                 CONTINUE
2                  T=T*TFACTR
           END DO
           WRITE(6,*) 'NCNT',NCNT
           RETURN
           END
C
           SUBROUTINE SWPCST(M,H,IORDER,NMEMB,NROW,N,DE)
C
C
C*********************************************************************
C
C          This subroutine returns the value of the change in the
C          objective function for a proposed swap of two positions
C          in the current permutation assignment IORDER.  On output
C          DE is the value of the change (+ or -).
C
C*********************************************************************
C
C
           INTEGER NMEMB,IORDER(NROW),N(2),I1,J1,K,K1,ITMP
           DOUBLE PRECISION M,H,LTSUM,RTSUM,DIFF,DE,SQDIFF
           DIMENSION M(NROW),H(NROW,NROW)
C
C********* initialization
C
           DE=0.0
           RTSUM=0.0
           LTSUM=0.0
           I1=IORDER(N(1))
           J1=IORDER(N(2))
C
C********* put indices of M in ascending order, I1 < J1
C
           IF (I1.GT.J1) THEN
                   ITMP=I1
                   NTMP=N(1)
                   I1=J1
                   N(1)=N(2)
                   J1=ITMP
                   N(2)=NTMP
           ENDIF
C
C*********************************************************************
C
C          This section of the code computes the change in the objective
C          function value, DE, in linear time.  To do this, a pointer array
C          IORDER is used to keep track of the switches in the array M.
C          Since only two components of M are switched at any one time only
C          two rows and two columns of the matrix H need be considered to
C          compute DE.
C
C*********************************************************************
C
           DO 12 K=1,NROW
                   K1=IORDER(K)
                   IF (K1.EQ.I1.OR.K1.EQ.J1) GOTO 12
                           LTSUM=LTSUM+H(K,N(2))*M(K1)
                           RTSUM=RTSUM+H(K,N(1))*M(K1)
12         CONTINUE
           DIFF=M(J1)-M(I1)
           SQDIFF=(M(J1)**2)-(M(I1)**2)
           DE=(SQDIFF*H(N(1),N(1)))+(2*DIFF*RTSUM)
      $      -(SQDIFF*H(N(2),N(2)))-(2*DIFF*LTSUM)
           RETURN
           END
C
C
           SUBROUTINE SWAP(IORDER,NROW,N)
C
C
C*********************************************************************
C
C          This routine performs the actual swap in IORDER between
C          positions N(1) and N(2).  On output IORDER is modified to
C          reflect this exchange.
C
C*********************************************************************
C
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
C
        INTEGER NROW, IORDER(NROW),N(2),ITMP
C
C
        ITMP=IORDER(N(1))
        IORDER(N(1))=IORDER(N(2))
        IORDER(N(2))=ITMP
        RETURN
        END
C
C
        SUBROUTINE METROP(DE,T,ANS)
C
C
C******************************************************************
C
C       Metropolis algorithm.  ANS is a logical variable which
C       issues a verdict on whether to accept a reconfiguration
C       which leads to a change DE in the objective function E.
C       If DE<0, ANS = .TRUE., while if DE > 0, ANS is only
C       .TRUE. with probability exp(-DE/T), where T is a
C       temperature determined by the annealing schedule.
C
C******************************************************************
C
C
        DOUBLE PRECISION DE,T
        PARAMETER(JDUM=1)
        LOGICAL ANS
        ANS=(DE.LT.0.0).OR.(RAN3(JDUM).LT.EXP(-DE/T))
        RETURN
        END
CC
C
        FUNCTION RAN3(IDUM)
C
C
C*******************************************************************
C
C       Returns a uniform random deviate between 0.0 and 1.0.
C       Set IDUM to any negative value to initialize or
C       reintialize the sequence. (see Numerical Recipes p. 199)
C
C*******************************************************************
C
C
        PARAMETER (MBIG=1000000000,MSEED=161803398,MZ=0,FAC=1./MBIG)
        DIMENSION MA(55)
        DATA IFF /0/
C
C************Initialization
C
        IF(IDUM.LT.0.OR.IFF.EQ.0)THEN
                IFF=1
                MJ=MSEED-IABS(IDUM)
                MJ=MOD(MJ,MBIG)
                MA(55)=MJ
                MK=1
                DO 11 I=1,54
                        II=MOD(21*I,55)
                        MA(II)=MK
                        MK=MJ-MK
                        IF(MK.LT.MZ)MK=MK+MBIG
                        MJ=MA(II)
11              CONTINUE
                DO 13 K=1,4
                        DO 12 I=1,55
                                MA(I)=MA(I)-MA(1+MOD(I+30,55))
                                IF(MA(I).LT.MZ)MA(I)=MA(I)+MBIG
12                      CONTINUE
13              CONTINUE
                INEXT=0
                INEXTP=31
                IDUM=1
        ENDIF
C
C*********End initialization
C
        INEXT=INEXT+1
        IF(INEXT.EQ.56)INEXT=1
```

```
      INEXTP=INEXTP+1
      IF(INEXTP.EQ.56)INEXTP=1
      MJ=MA(INEXT)-MA(INEXTP)
      IF(MJ.LT.MZ)MJ=MJ+MBIG
      MA(INEXT)=MJ
      RAN3=MJ*FAC
      RETURN
      END
C
C
      SUBROUTINE SECOND(TIM)
      TIME0=0.0E+00
      TIM=SECNDS(TIME0)
      RETURN
      END
C
C
C
      SUBROUTINE OBJCHK(M,IORDER,H,PSUM,NROW,ZCHK)
C
      INTEGER NROW,IORDER,I1,J1
      DOUBLE PRECISION ZCHK,M,H,PSUM
      DIMENSION M(NROW),H(NROW,NROW),IORDER(NROW),PSUM(NROW)
C
      ZCHK=0.0
      DO 5 I=1,NROW
              I1=IORDER(I)
              PSUM(I)=0.0
              DO 4 J=1,NROW
                      J1=IORDER(J)
                      PSUM(I)=PSUM(I)+H(I,J)*M(J1)
4             CONTINUE
              ZCHK = ZCHK + PSUM(I)*M(I1)
5     CONTINUE
      RETURN
      END
C
C
```