PSYCHOMETRIKA—VOL. 56, NO. 1, 147–152
MARCH 1991
COMPUTATIONAL PSYCHOMETRICS

AN EFFICIENT ALGORITHM FOR PARAFAC OF THREE-WAY DATA WITH LARGE NUMBERS OF OBSERVATION UNITS

HENK A. L. KIERS AND WIM P. KRIJNEN

UNIVERSITY OF GRONINGEN

The CANDECOMP algorithm for the PARAFAC analysis of $n \times m \times p$ three-way arrays is adapted to handle arrays in which $n > mp$ more efficiently. For such arrays, the adapted algorithm needs less memory space to store the data during the iterations, and uses less computation time than the original CANDECOMP algorithm. The size of the arrays that can be handled by the new algorithm is in no way limited by the number of observation units (n) in the data.

Key words: CANDECOMP, PARAFAC.

Harshman (1970) developed a method for factor analysis of three-way arrays, called PARAFAC (PARAllel profiles FACtor analysis), and proposed an algorithm for his method identical to the CANDECOMP algorithm proposed by Carroll and Chang (1970) for fitting their INDSCAL model. The CANDECOMP/PARAFAC problem is that of minimizing

$$\mathrm{f}(A, D_1, \ldots, D_p, B) = \sum_{k=1}^{p} \|X_k - AD_kB'\|^2, \tag{1}$$

where X_k denotes the $n \times m$ matrix which forms the k-th frontal section of the three-way array, $k = 1, \ldots, p$; A, B, and $D_1, \ldots, D_p$ are matrices containing model parameters to be estimated, with A of order $n \times r$, B of order $m \times r$, and D_k diagonal, of order $r \times r$, where r is the number of factors to be extracted. The CANDECOMP/PARAFAC algorithm (Carroll & Chang, 1970) can be described in terms of A, B, $D_1, \ldots, D_p$, and $X_1, \ldots, X_p$ as follows (see Kroonenberg, 1983, pp. 114–116):

```
Step 0. Initialize the matrices $B$ and $D_1, \ldots, D_p$.
Step 1. Compute the start for $A$ according to $A = (\Sigma_k X_k B D_k)(\Sigma_k D_k B'BD_k)^{-1}$.
Step 2. Iteratively update $B, D_1, \ldots, D_p, A$, and evaluate f according to the steps:
        2a. $B := (\Sigma_k X_k' A D_k)(\Sigma_k D_k A'AD_k)^{-1}$;
        2b. $D_k 1 := (A'A * B'B)^{-1}(\mathrm{Diag}\, A'X_k B)1$, for $k = 1, \ldots, p$ (with 1 denoting the $r$-vector with unit elements, and * the Hadamard product);
        2c. $A := (\Sigma_k X_k B D_k)(\Sigma_k D_k B'BD_k)^{-1}$;
        2d. Evaluate $\mathrm{f}(A, D_1, \ldots, D_p, B)$.
            If $\mathrm{f}^{\mathrm{old}} - \mathrm{f}^{\mathrm{new}} < \varepsilon$, where $\varepsilon$ is some arbitrary small value, then go to Step 3, otherwise repeat Step 2.
```

The authors are obliged to Jos ten Berge for his comments on an earlier version of this paper. The research of Henk A. L. Kiers has been made possible by a fellowship of the Royal Netherlands Academy of Arts and Sciences.

Requests for reprints should be sent to Henk A. L. Kiers, Department of Psychology, University of Groningen, Grote Kruisstraat 2/1, 9712 TS Groningen, THE NETHERLANDS.

Step 3. Postprocess A, B, and $D_1, \ldots, D_p$ (for instance, normalize A and B to unit column sums of squares).

The bulk of the computations is done in Step 2, which is typically repeated many times. Although different program versions may differ with respect to the order of Steps 2a through 2d, or with respect to the choice of convergence criterion, essentially all CANDECOMP/PARAFAC algorithms are based on these steps.

The above described algorithm has been proven to converge monotonically (Carroll & Chang, 1970) and is known to work well in practice. However, the size of data sets that can be analyzed using this algorithm is rather limited, because during the computations, the complete three-way array and the three parameter sets have to be stored in the working space of the computer (RAM). In many practical situations, the sample size (number of observation units) poses irresolvable problems for the program on personal computers, and even on mainframe hardware, data size problems may occur. One may resort to fitting the covariances instead of the original data and use the PARAFAC2 algorithm (Harshman, 1972, also, see Harshman & Lundy, 1984b) for that purpose, but then a different model is fitted, and consequently, a different problem is solved. The present paper offers an algorithm for fitting the original PARAFAC model that can handle data sets of any sample size; the only limitations are set by the numbers of variables and (frontal) sections. It is based on aggregating the data over the individual observation units. For data with n large compared to mp, this not only saves memory space, but is also faster, as will be demonstrated below.

Updating the A Matrix Implicitly

When n is large, the original CANDECOMP/PARAFAC algorithm causes storage problems for $X_1, \ldots, X_p$, and A, which are of orders $n \times m$ and $n \times r$, respectively. The modification of the CANDECOMP/PARAFAC algorithm proposed here avoids the use of these large matrices, using the much smaller matrices $X_j'X_k$, $X_k'A$, and $A'A$, $j, k = 1, \ldots, p$, instead. This modification is based on the fact that during the Steps 2a, 2b, and 2d, matrix A occurs only in the combinations $X_k'A$ and $A'A$. To check this statement for Step 2d, we expand $f(A, D_1, \ldots, D_p, B)$ as:

$$f(A, D_1, \ldots, D_p, B) = \sum_{k=1}^{p} \operatorname{tr} X_k'X_k - 2 \operatorname{tr} \sum_{k=1}^{p} X_k'AD_kB' + \operatorname{tr} \left(A'A \sum_{k=1}^{p} D_kB'BD_k \right).$$

It is useful to note that after updating A, $A(\Sigma_k D_k B'BD_k) = (\Sigma_k X_k BD_k)$, and hence, f can be simplified as

$$f(A, D_1, \ldots, D_p, B) = \sum_{k=1}^{p} \operatorname{tr} X_k'X_k - \operatorname{tr} \sum_{k=1}^{p} X_k'AD_kB'. \tag{2}$$

When n is large, it takes less space to store the $m \times r$ matrices $S_k \equiv X_k'A$, $k = 1, \ldots, p$, and the $r \times r$ matrix $C \equiv A'A$ than the $n \times r$ matrix A and the $n \times m$ matrices $X_1, \ldots, X_p$. However, when A and $X_1, \ldots, X_p$ are not stored anymore, we have to deal with S_k, $k = 1, \ldots, p$, and C throughout the algorithm. In particular, instead of updating A (Step 2c), we have to update S_k, $k = 1, \ldots, p$, and C, as follows:

$$S_k = X_k'A = (\Sigma_j X_k'X_j BD_j)(\Sigma_j D_j B'BD_j)^{-1}, \tag{3}$$

$k = 1, \ldots, p$, and

$$C = A'A = (\Sigma_k D_k B'BD_k)^{-1}(\Sigma_j \Sigma_k D_j B'X_j'X_k BD_k)(\Sigma_k D_k B'BD_k)^{-1}. \tag{4}$$

Apart from B and $D_1, \ldots, D_p$, both expressions involve the $n \times m$ matrices X_k, $k = 1, \ldots, p$. However, these only occur in cross-products $R_{jk} \equiv X_j'X_k$, $j, k = 1, \ldots, p$, which are all of order $m \times m$. Hence the updates for S_k, $k = 1, \ldots, p$, and C can be computed without using X_k explicitly. As a consequence, replacing Steps 1 and 2c (for updating A) by two steps in which S_k, and C are updated, and basing the computations in Steps 2a, 2b, and 2d on S_k, and C instead of A, modifies the CANDECOMP/PARAFAC algorithm such that it no longer needs to store the possibly very large matrices A and $X_1, \ldots, X_p$. The modified algorithm needs to store the matrices $S_k(m \times r)$, $C(r \times r)$, and $R_{jk}(m \times m)$, $j, k = 1, \ldots, p$ (i.e., $pmr + r^2 + p^2m^2$ real numbers), where the original algorithm needs to store $X_k(n \times m)$ and $A(n \times r)$, $k = 1, \ldots, p$ (i.e., $pnm + nr$ real numbers), apart from the matrices B, $D_1, \ldots, D_p$, and possible (small) working arrays. As a result, the modified algorithm requires less RAM than the original algorithm whenever $n > mp + r^2(mp + r)^{-1}$, and in particular, when $n \gg mp$.

The above modifications of the CANDECOMP/PARAFAC algorithm yield the following "implicit" algorithm:

```
Step 1. Read cross-product matrices $R_{jk}$, $j, k = 1, \ldots, p$, or read (raw) data
        matrices such that cross-product matrices are built while reading the original data.
Step 2. Initialize $B$ and $D_1, \ldots, D_p$.
Step 3. Compute starting values for $S_k$, $k = 1, \ldots, p$, and $C$, based on (3) and (4).
Step 4. Iteratively perform the following steps, until convergence:
        4a. $B := (\Sigma_k S_k D_k)(\Sigma_k D_k C D_k)^{-1}$;
        4b. $D_k 1 := (C{*}B'B)^{-1}(\text{Diag } S_k'B)1$, $k = 1, \ldots, p$;
        4c. Update $S_k$ according to (3), $k = 1, \ldots, p$;
        4d. Update $C$ according to (4);
        4e. Evaluate $f = \Sigma_k \text{ tr } R_{kk} - \Sigma_k \text{ tr } S_k D_k B'$, based on (2).
            If $f^{old} - f^{new} < \varepsilon$ for some small value $\varepsilon$, go to Step 5, otherwise
            repeat Step 4.
Step 5. Postprocess $B$, and $D_1, \ldots, D_p$ (for instance, normalize (the implicit) $A$
        and $B$ to unit column sums of squares).
Step 6. If the original data are still available, compute $A$ according to
        $A = \Sigma_k X_k B D_k (\Sigma_k D_k B'BD_k)^{-1}$ (optional).
```

Computation Times

Apart from the gain in RAM needed when $n \gg mp$, the implicit algorithm is also faster than the original algorithm, since the former uses smaller matrices, and multiplications with these smaller matrices require fewer operations. To evaluate the differences in computation time between the two algorithms, we conducted a small comparative study with generated data sets (sampled randomly from a uniform distribution on $[-1, 1]$) of different orders, crossing three levels of n (12, 24, and 36) with two levels of m (4 and 8) and two levels of p (2 and 3). In sum, 60 three-way arrays were generated (five replications in each cell of the design) and analyzed using both the new algorithm and our version of the original algorithm, programmed according to the description in the introduction. In all analyses, $r = 2$, and the same (rational) start for B and $D_1, \ldots, D_p$ was used. As a consequence, both algorithms yield the same intermediate updates for B and $D_1, \ldots, D_p$, and find the same function values after each iteration, within rounding error. The convergence criterion was $\varepsilon = 0.001$, which for our purpose led to

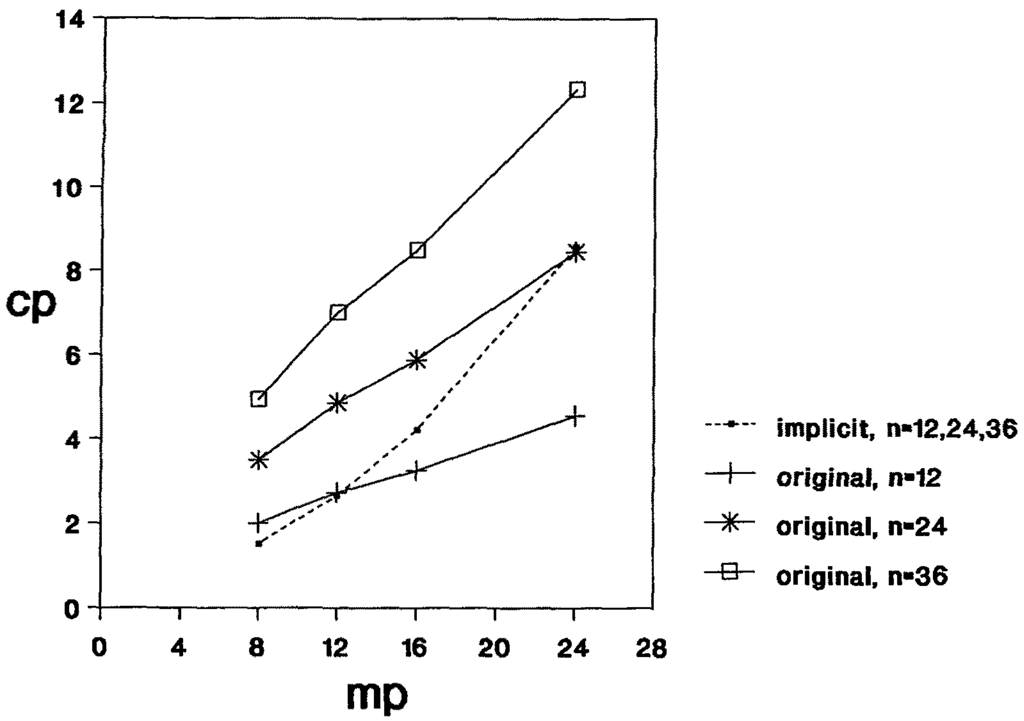

FIGURE 1
Mean computation times per iteration (cp) as a function of the product of m and p, for both the implicit and the original algorithm.

sufficient precision, because our interest is in computation times rather than determining the solution sufficiently accurately. The study was conducted on an Olivetti M24 (8086).

Because analyses of different data sets usually took different numbers of iterations (ranging from 7 through 112, with a mean of 34.5 iterations), we have computed the computation times per iteration (cp). For each (n, m, p)-combination, the implicit algorithm used virtually the same cp over all five replications, as did the original algorithm. In Figure 1 we report the mean cp over the five replications. The horizontal axis shows the value of mp (which in our study was 8 ($m = 4, p = 2$), 12 ($m = 4$, $p = 3$), 16 ($m = 8, p = 2$), and 24 ($m = 8, p = 3$), respectively). For both algorithms, and for all three different values of n, the corresponding mean cp has been plotted. Because cp does not depend on n, for the implicit algorithm the three lines coincide. Figure 1 displays clearly that with increasing n, computation time increases considerably for the original algorithm. The line depicting cp for the implicit algorithm can be seen as a demarcation line: points above the line refer to cases where the original algorithm is slower than the implicit algorithm ($n > mp$), whereas points below this line ($n < mp$) refer to cases where the original algorithm is faster. It is readily verified that $n \approx mp$ at the points where the lines for the different algorithms cross (implying that they are equally fast).

Imposing Orthogonality Constraints on Matrix A

In certain situations the unconstrained CANDECOMP/PARAFAC method leads to uninterpretable "degenerate" solutions, with highly correlated columns of A. Harshman and Lundy (1984a) propose to remedy this by constraining A to be columnwise orthonormal. That is, instead of minimizing (1) over arbitrary A, they propose to minimize (1) over columnwise orthonormal A. The solution to this minimization problem can be derived as follows (see Kroonenberg, 1983, p. 112, for a similar case). It is readily verified that minimizing (1) over A, subject to $A'A = I_r$, is equivalent to maximizing

$$g(A) = \text{tr} \sum_{k=1}^{p} X_k B D_k A'. \tag{5}$$

According to Cliff (1966), the columnwise orthonormal A maximizing (5) is given by

$$A = (\Sigma_j X_j B D_j)(\Sigma_j \Sigma_k D_j B' X_j' X_k B D_k)^{-1/2}. \tag{6}$$

To avoid using large matrices in the algorithm, we propose to update $S_k = X_k' A$ (instead of A), as follows:

$$\begin{aligned} S_k &= X_k'(\Sigma_j X_j B D_j)(\Sigma_j \Sigma_k D_j B' X_j' X_k B D_k)^{-1/2} \\ &= (\Sigma_j R_{kj} B D_j)(\Sigma_j \Sigma_k D_j B' R_{jk} B D_k)^{-1/2}, \end{aligned} \tag{7}$$

for $k = 1, \ldots, p$. This updating procedure only uses B, $D_1, \ldots, D_p$, and the R_{jk} matrices. The other steps of the algorithm, that is, those of updating B and $D_1, \ldots, D_p$, and that of evaluating f can be done as in the previous section. It should be noted that $C = A'A$, which was updated in each cycle of the algorithm of the previous section, is constant here, that is $C = I_r$. Using (7) and the procedures for updating B, and $D_1, \ldots, D_p$ described earlier, we have an algorithm for PARAFAC with orthogonality constraints which, once again, does not use any matrices of row- or column-order n. A PC-program using implicit updatings for A, both unconstrained and orthogonally constrained, is available from the second author.

Discussion

The above described modification of the CANDECOMP/PARAFAC algorithm has been presented as an efficient method for handling three-way arrays with large n. However, after a proper permutation of the array, the same procedure can be used when m is large compared to np, or when p is large compared to nm.

The PARAFAC program allows for a large number of preprocessing options. Above, preprocessing has been assumed to have been incorporated in the matrices X_k already. If this is not the case, some types of preprocessing (e.g., standardizing the variables in each frontal section), can simply be incorporated in the process of converting the original data to the R_{jk} matrices. Other preprocessing procedures, however, need to be carried out separately. The same holds for handling missing data. In the original CANDECOMP/PARAFAC algorithm it is possible to iteratively reestimate missing data from the model, but in our implicit algorithm one is forced to choose values for missing data in the preprocessing stage.

The above modification of the CANDECOMP/PARAFAC algorithm shows that the computations of the solution for B, and $D_1, \ldots, D_p$, do not involve the X_k matrices, but only the cross-product matrices $R_{jk}, j, k = 1, \ldots, p$. As a consequence, different data sets with the same cross-product matrices yield the same solutions for B, and $D_1, \ldots, D_p$. This parallels the situation of principal components analysis (PCA), where the loadings depend on the correlation matrix only. As PCA, CANDECOMP/PARAFAC analyzes first- and second-order moments only.

References

Carroll, J. D., & Chang, J. J. (1970). Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition. *Psychometrika*, *35*, 283–319.

Cliff, N. (1966). Orthogonal rotation to congruence. *Psychometrika*, *31*, 33–42.

Harshman, R. A. (1970). Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-mode factor analysis. *UCLA Working Papers in Phonetics*, *16*, 1–84.

Harshman, R. A. (1972). PARAFAC2: Mathematical and technical notes. *UCLA Working Papers in Phonetics*, *22*, 31–44.

Harshman, R. A., & Lundy, M. E. (1984a). Data preprocessing and the extended PARAFAC model. In H. G. Law, C. W. Snyder, J. A. Hattie, & R. P. McDonald (Eds.), *Research methods for multimode data analysis* (pp. 216–284). New York: Praeger.

Harshman, R. A., & Lundy, M. E. (1984b). The PARAFAC model for three-way factor analysis and multidimensional scaling. In H. G. Law, C. W. Snyder, J. A. Hattie, & R. P. McDonald (Eds.), *Research methods for multimode data analysis* (pp. 122–215). New York: Praeger.

Kroonenberg, P. M. (1983). *Three-mode principal component analysis*. Leiden: DSWO Press.

Manuscript received 1/22/90
Final version received 7/27/90