Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computer Physics Communications 166 (2005) 58–65

Computer Physics Communications

www.elsevier.com/locate/cpc

Four-index integral transformation exploiting symmetry [☆]

Shigeyoshi Yamamoto [a,*], Umpei Nagashima [b]

[a] *Faculty of Liberal Arts, Chukyo University, 101-2 Yagoto-Honmachi, Showa-ku, Nagoya 466-8666, Japan*
[b] *National Institute of Advanced Industrial Science and Technology, Grid Technology Research Center, 6-9-3 Higashi-Ueno, Taito-ku, Tokyo 110-0015, Japan*

Received 20 June 2003; received in revised form 2 January 2004; accepted 6 January 2004

Available online 13 December 2004

Abstract

We present a Fortran implementation of four-index integral transformation in the LCAO-MO (linear combination of atomic orbitals–molecular orbitals) framework that exploits symmetry. Electron correlation calculations, such as configuration interaction (CI) calculations, usually require electron repulsion integrals to be transformed to a molecular orbital basis from a basis using atomic orbitals. In large molecular systems it is vital to exploit the sparsity of integrals in making this transformation. By exploiting symmetry, the sparsity of integrals is fully utilized, the size of intermediate file is minimized, and the computational cost is reduced. The present algorithm is simple and can readily be added to existing quantum chemistry program packages.

Program summary

Title of program: SYM4TR (symmetry adapted 4-index integral transformation)
Catalogue identifier: ADUW
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/ADUW
Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland
Computers: IBM/AIX, HP Alpha server/Tru64, PC's/Linux
Program language used: Fortran 95
Number of lines in distributed program, including test data, etc.: 4519
No. of bytes in distributed program, including test data, etc.: 32 095
Distributed format: tar gzip file
Nature of physical problem: Molecular orbital calculations including electron correlation effects usually require electron repulsion integrals to be transformed from an atomic orbital (AO) basis to a molecular orbital (MO) basis. By exploiting the sparsity of molecular integrals, the computational cost and memory needed for the transformation are minimized.

[☆] This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

[*] Corresponding author.
E-mail address: syamamot@lets.chukyo-u.ac.jp (S. Yamamoto).

doi:10.1016/j.cpc.2004.01.008

Method of solution: The sparsity of molecular integrals is exploited. The program treats only nonzero integrals. The length of running indices in DO loops is reduced using the block-diagonal form of the MO coefficient matrix. In the present program, the point group is limited to D_{2h} and its subgroups.

PACS: 31.15.Ar

Keywords: Four-index integral transformation; Symmetry adaptation; Sparsity; Point group; Electron repulsion integral; Two-electron integral

1. Introduction

In post-Hartree–Fock molecular orbital calculations, molecular integrals usually have to be transformed from an atomic orbital (AO) basis to a molecular orbital (MO) basis. A typical example is a configuration interaction (CI) calculation that accounts for electron correlation effects. The integral transformation is a simple arithmetic operation corresponding to Eq. (1).

$$(ij|kl) = \sum_p^N \sum_q^N \sum_r^N \sum_s^N (pq|rs) C_p^i C_q^j C_r^k C_s^l. \tag{1}$$

Here, $(pq|rs)$ is the AO basis 2-electron repulsion integral:

$$(pq|rs) = \iint \chi_p^*(1)\chi_q(1)\frac{1}{r_{12}}\chi_r^*(2)\chi_s(2)\,\mathrm{d}v_1\,\mathrm{d}v_2. \tag{2}$$

The C_p^i are MO coefficients, and N is the total number of basis functions (AOs). We assume that N is approximately 500. The MO φ^i is expanded in basis functions χ_p with coefficients C_p^i:

$$\varphi^i = \sum_p^N \chi_p C_p^i. \tag{3}$$

A Gaussian-type function is usually used as the basis function. We use the letters p, q, r, s for the AO index, and i, j, k, l for the MO index.

Of the various integrals involved, transformation of four-index two-electron repulsion integrals has a high computational cost and demands large memory. This is sufficiently demanding to be an impediment in post-HF calculations for large molecules.

Many algorithms [1–4] have been proposed to improve the efficiency of transformation of integrals. Among them the Bender–Shavitt method [1,2], the Elbert method [3], and the Saunders–van Lenthe method [4] are well known. There is a comprehensive review by Wilson [5] for these methods. The most recent one is the Saunders–van Lenthe method and our present method is similar to it.

There are some algorithms designed for specific applications in which the MO indices are restricted. For example, Olsen and Yeager [6] proposed a partial transformation algorithm for second-order MC-SCF calculations. By contrast the present algorithm is designed for general transformation.

Integral-driven transformation algorithms have been proposed to avoid the saving of integrals in external storage. One example is the integral-driven direct configuration interaction (CI) method due to Mochizuki et al. [7], in which AO integrals are generated and transformed to the MO basis in-core for each iteration of Davidson's diagonalization procedure [8]. Recently Schütz et al. [9] presented integral-based implementations of electron correlation methods, including the internally contracted multireference configuration interaction (MRCI). However, these integral-based approaches require much CPU time in compensation for reducing the external storage requirement.

The present article supposes that half-transformed integrals $(ij|rs)$ can be held in a temporary external memory (usually a file on magnetic disk). The basic structure of our algorithm is quite similar to the Saunders–van Lenthe method, but in our method, the sparsity of AO integrals is explicitly considered, that is, only nonzero integrals are treated.

We then present a program which minimizes the computational cost and file size by taking into account the molecular symmetry and the sparsity of the two-electron integrals. We shall see that values as high as $N \approx 500$ can be handled by this means. It is assumed that basis functions are symmetrized as symmetry orbitals (SO), and from now on we write SO rather than AO.

The present program is designed for nonrelativistic quantum chemistry calculations. For the relativistic case, refer to Ref. [10].

2. Algorithm

2.1. Loop structure

It is well known that the transformation of Eq. (1) is efficiently implemented by decomposing the process into the following four steps. This stepwise method is called the N^5 algorithm [11].

$$(iq|rs) = \sum_{p}^{N} (pq|rs) C_p^i, \tag{4}$$

$$(ij|rs) = \sum_{q}^{N} (iq|rs) C_q^j, \tag{5}$$

$$(ij|ks) = \sum_{r}^{N} (ij|rs) C_r^k, \tag{6}$$

$$(ij|kl) = \sum_{s}^{N} (ij|ks) C_s^l. \tag{7}$$

When N is large, the whole integral data exceeds the main memory capacity. To reduce the memory requirement, half-transformed integrals $(ij|rs)$ are saved in a temporary intermediate file.

Here, the process is decomposed into two parts, as proposed by Diercksen [12]. The first stage is $(pq|rs) \to (ij|rs)$, and the second is $(ij|rs) \to (ij|kl)$. The same decomposition has been made by Saunders and van Lenthe [4].

To process each step efficiently, and in particular to vectorize the program, the integrals should be blocked with respect to their index in advance. In the first step, the integrals $(pq|rs)$ are to be ordered according to the canonical index of r and s, i.e. $r(r-1)/2 + s$ $(r \geqslant s)$. Also, in the third step, the integrals $(ij|rs)$ are processed sequentially according to the order of the rs index.

The loop structure of the calculation is shown in Figs. 1 and 2. The basic structure is quite similar to that of the Saunders–van Lenthe method, but the sparsity of the integrals is taken into account explicitly in our method. In other words, only AO integrals having absolute value larger than a threshold are treated. Fig. 1 shows the first and second steps (first part), and Fig. 2 shows the third and fourth steps (latter part). The program assumes only that symmetrically unique SO integrals $(pq|rs)$ are prepared at the start. These integrals are saved in a file, but their order can be random. This randomness is needed to make the program adaptable to existing quantum chemical program packages. Therefore, before the transformation step, our program first gathers SO integrals with respect to the rs index. Here, it should be noted that in the present method SO integrals of which pq and rs indices are exchanged are also gathered into the given rs block. In the rs block, the pq indices can appear in random order. The exchange of the pq and rs enlarges the size of the SO integral file approximately by factor two.

This ordering is processed by applying the backward-chaining technique of Yoshimine [13]. The pq indices in an ordered rs block do not need to be sorted. Our program uses 2-pass binsort algorithm [14] for this purpose, however it should be noted that this process is not sorting but gathering of integrals.

2.2. I/O of half-transformed integrals

To vectorize the third step, the half-transformed integrals $(ij|rs)$ should be processed in the order of the rs index. However, at the end of the second step, the half-transformed integrals $(ij|rs)$ are written to an intermediate file in the order of the ij index. The I/O sequence changes between the second and third steps. This change in the processing order is implemented by Yoshimine's backward-chaining technique [13]. The Yoshimine method has been frequently exploited by several authors [5] and its application is not new at all. However, it would be preferable to give a detailed explanation of the algorithm to clarify the program structure.

The I/O for the intermediate file is illustrated in Fig. 3. Integrals with rs index block are processed in the order of the ij index and are written to an intermediate direct access file in that order. In the example shown in Fig. 3, the half-transformed integrals are written in the order $A \to B \to C$ for the first rs block. The size of each block is determined such that integrals in the blocks of A, B, and C can be held in the main memory. For the second rs block, the output order is $D \to E \to F$, and for the last rs block it is $G \to H \to I$.

```
    DOUBLE PRECISION C(#SO,#MO), IQRS(#MO,#SO)
    DOUBLE PRECISION IJRS(#RS_IN_BLOCK,#MOMO)
    INTEGER SO2MO_RANGE(2,#SO), MO2SO_RANGE(2,#MO)
    INTEGER RS(2,#MAXRS), BACK_REC(#BLOCK_IJ)

!-----------------------------------------------------

    RS = 0; BACK_REC = 0 ; irec = 0
!
    DO block.rs = 1, #BLOCK_RS
     IJRS = 0.0
!
     DO rs_in_block = 1, #RS_IN_BLOCK
      IQRS = 0.0
!
!   --- 1st step (p,q,r,s) -> (i,q,r,s) ----
!    read nonzero SO integrals for given rs-block
!    #word, r, s, PQ(i:#word), PQRS(i:#word)
!
     DO word = 1, #WORD
      IF( p == q ) THEN
       DO i = SO2MO_RANGE(1,p), SO2MO_RANGE(2,p)
        IQRS(i,q) = IQRS(i,q) + PQRS(word) * C(p,i)
       END DO
      ELSE
       DO i = SO2MO_RANGE(1,p), SO2MO_RANGE(2,p)
        IQRS(i,q) = IQRS(i,q) + PQRS(word) * C(p,i)
       END DO
       DO i = SO2MO_RANGE(1,q), SO2MO_RANGE(2,q)
        IQRS(i,p) = IQRS(i,p) + PQRS(word) * C(q,i)
       END DO
      END IF
     END DO
!
!   ---- 2nd step (i,q,r,s) -> (i,j,r,s) ----
!
     ij = 0
     DO i = 1, #MO
      DO j = 1, i
       ij = ij + 1
       xijrs = 0.0
       DO q = MO2SO_RANGE(1,j), MO2SO_RANGE(2,j)
        xijrs = xijrs + IQRS(i,q) * C(q,j)
       END DO
       IJRS(rs_in_block,ij) = xijrs
      END DO
     END DO
    END DO
!
    DO block_ij = 1, #BLOCK_IJ
     CALL WRITE_DA(irec, BACK_REC, block_ij,,,)
    END DO
   END DO
```

Fig. 1. First half of transformation $(pq|rs) \rightarrow (ij|rs)$.

```
    DOUBLE PRECISION C(#SO,#MO), IJRS(#MAXRS,#BLOCK_IJ)
    DOUBLE PRECISION IJKS(#SO,#MO)
    INTEGER SO2MO_RANGE(2,#SO), MO2SO_RANGE(2,#MO)
    INTEGER RS(2,#MAXRS), BACK_REC(#BLOCK_IJ)

!------------------------------------------------------

    DO block_ij = 1, #BLOCK_IJ
!
!   ---- 3rd step (i,j,r,s) -> (i,j,k,s) ----
!
    IJRS = 0.0
    DO block_rs = 1, #BLOCK_RS
     irec = BACK_REC(block_ij)
     CALL READ_DA(irec, BACK_REC, block_ij,,,)
    END DO
    DO ij_in_block = 1, #IJ_IN_BLOCK
     IJKS = 0.0
!
     DO rs = 1, #MAXRS
      IF( |IJRS(rs,ij_in_block)| < THIJRS ) CYCLE
      IF( r == s ) THEN
       DO k = SO2MO_RANGE(1,r), SO2MO_RANGE(2,r)
        IJKS(s,k) = IJKS(s,k) + IJRS(rs,ij_in_block) * C(r,k)
       END DO
      ELSE
       DO k = SO2MO_RANGE(1,r), SO2MO_RANGE(2,r)
        IJKS(s,k) = IJKS(s,k) + IJRS(rs,ij_in_block) * C(r,k)
       END DO
       DO k = SO2MO_RANGE(1,s), SO2MO_RANGE(2,s)
        IJKS(r,k) = IJKS(r,k) + IJRS(rs,ij_in_block) * C(s,k)
       END DO
      END IF
     END DO
!
!   ---- 4th step (i,j,k,s) -> (i,j,k,l) ----
!
     kl = 0
     DO k = 1, #MO
      DO l = 1, k
       kl = kl + 1
       IF( ij < kl ) CYCLE
       xijkl = 0.0
       DO s = MO2SO_RANGE(1,l), MO2SO_RANGE(2,l)
        xijkl = xijkl + IJKSC(s,k) * C(s,l)
       END DO
       IF( DABS(xijkl) < THRESHOLD_IJKL ) CYCLE
!      accumulate i, j, k, l, xijkl into buffer
!      if buffer full, write buffer on file
      END DO
     END DO
    END DO
   END DO
```

Fig. 2. Second half of transformation $(ij|rs) \rightarrow (ij|kl)$.

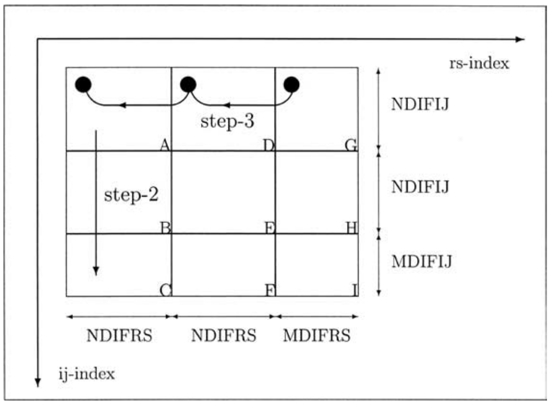

Fig. 3. Exchange of I/O sequence in intermediate file.

For each I/O block, the record number of the previous block is written as the first record, to enable backward-chaining. This first record in the I/O block is followed by subsequent records containing the ij indices, the rs indices and the $(ij|rs)$ integrals. By using the previous record number saved in the first record in a block (designated by a solid circle in Fig. 3), integrals of given ij-index block are read in direct access mode.

In the third step, the I/O blocks are read in the order $G \to D \to A$ for the first ij block and these integrals are processed together. For the second ij block, the reading order is $H \to E \to B$, and for the last ij block it is $I \to F \to C$. Though the writing is done consecutively, the reading is performed by direct access.

The constant NDIFIJ in Fig. 3 is the total number of different ij indices in a rs block. MDIFIJ is also the number of different ij indices, but this is for the last rs block. Similarly, NDIFRS is the total number of different rs indices in an ij block.

The size of the intermediate file is minimized by saving only nonzero integrals. Half-transformed integrals having absolute value smaller than a threshold ($\approx$ 1.0D–10) are not saved. During this ordering process, some of the associated ij indices may be dropped. From this reason, ij indices should be also saved in the intermediate file together with the $(ij|rs)$ integrals and the rs indices.

2.3. Reduction of loop length by symmetry

When the molecule has spatial symmetry, wasteful arithmetic operations for vanishing elements can be avoided by classifying the AO integrals into $(\alpha\alpha\alpha\alpha)$, $(\alpha\beta\alpha\beta)$, $(\alpha\alpha\beta\beta)$, and $(\alpha\beta\gamma\delta)$ types by their symmetry as described in Refs. [5,15].

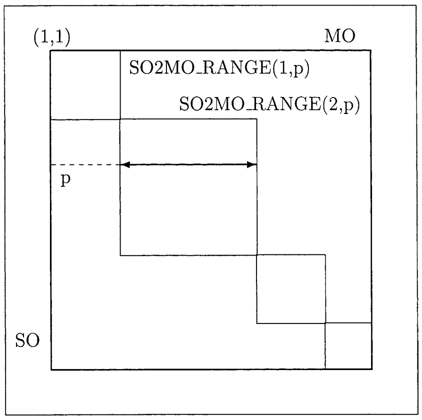

Fig. 4. Running indices in block-diagonal MO coefficient matrix.

In our method instead, the spatial symmetry of molecule is taken into account by exploiting symmetry-adapted basis function. The zero-valued integrals by the symmetric restriction are not contained in the files and therefore only symmetrically nonzero integrals appear. The simplicity of the program is maintained by taking this approach.

In addition, in our method, wasteful arithmetic operations are removed by restricting the running indices of the innermost loops. When the molecule has spatial symmetry, it is possible to transform the matrix of MO coefficients to block-diagonal form (see [16]). The loop length in each transformation step can then be reduced. For example, in the case of D_{2h} symmetry, an eightfold reduction is attainable.

This reduction is easily realized by preparing two integer arrays, SO2MO_RANGE and MO2SO_RANGE. SO2MO_RANGE$(1, p)$ and SO2MO_RANGE$(2, p)$ refer respectively to the starting and ending MO indices of the MO coefficient matrix for given SO index p. Since these two values are common for all SO indices belonging to the same irreducible representation, they can be stored in more compact form. In the present program, however, they are kept in the array SO2MO_RANGE, to keep the program transparent and gain fast memory access. The second array, MO2SO_RANGE, contains the SO index range which corresponds to the reverse of SO2MO_RANGE. It follows that MO2SO_RANGE$(1, i)$ and MO2SO_

RANGE$(2, i)$ refer respectively to the starting and ending SO indices for a given MO index i. Fig. 4 shows this situation schematically for the case of four irreducible representations.

3. Program description

The present program does not contain any SO integral generation routines, since these are large and complicated. RHF (restricted Hartree–Fock) SCF routines are also not included. There are many widely distributed quantum chemical program packages, and integrals and MOs can be generated using these. Examples of free quantum chemical program packages include GAMESS [17] and DALTON [18]. Users are expected to modify the present program to import integrals and MOs from an existing quantum chemical program.

However, it is highly desirable to provide the present software as a complete stand-alone program, so that users can verify that the program is correctly installed. Minimum routines and data are therefore provided for this purpose.

The present software system contains two programs. One is the SYM4TR program, and the other is the PREPAR program which simply converts text format files of the MO coefficient matrix and SO basis integrals to binary files. The SYM4TR program consists of arterial routines (transformation) and supplementary routines (ordering integrals).

The calling sequence of the arterial routines is as follows.

```
WTHDRV-+-WTHSTB
        +-WTHOPT
        +-WTHCL1---WTHWDA
        +-WTHCL2---WTHRDA
```

3.1. *WTHDRV*

Subroutine WTHDRV is a main driver for the subordinates subroutines. This routine allocates arrays dynamically.

3.2. *WTHSTB*

Subroutine WTHSTB sets arrays SO2MO_RANGE and MO2SO_RANGE.

3.3. *WTHOPT*

Subroutine WTHOPT optimizes the size of an I/O block, i.e. the number of ij and rs indices in a block. These data are saved in variables NTIMIJ, NDIFIJ, NTIMRS, NDIFRS, etc.

3.4. *WTHCL1*

Subroutine WTHCL1 is the main part of the first half of the transformation (first and second steps). The loop structure is shown in Fig. 1.

3.5. *WTHWDA*

Subroutine WTHWDA writes half-transformed integrals and their associated data in the intermediate direct access file.

3.6. *WTHCL2*

Subroutine WTHCL2 is a main part of the second half of the transformation (third and fourth steps). The loop structure is shown in Fig. 2.

3.7. *WTHRDA*

Subroutine WTHRDA reads half-transformed integrals and their associated data from the intermediate direct access file.

4. Test run and sample input and output

For a test run we used the water molecule, which has symmetry C_{2v}. The molecular geometry is taken from the benchmark calculations of Refs. [19,20].

However, the basis set used here is the STO-3G minimal set [21], instead of the double-zeta set used in the reference papers. SO integrals are given with their indices in a text file (soint.txt). The RHF MOs of the ground state are also given as a text file (mocoef.txt) with their MO and SO indices. Other miscellaneous data, such as molecular symmetry information, is given in the inf.txt file.

The transformed integrals comprise the output data. Their values and indices are listed in the text file (moint.txt).

5. Performance data

We measured the performance of this program by applying it to free-base porphin (FBP, $C_{20}N_4H_{14}$), which has D_{2h} symmetry. This molecular system is the same as in Ref. [22]. The number of basis functions are 91 a_g, 79 b_{1g}, 30 b_{2g}, 30 b_{3g}, 27 a_u, 33 b_{1u}, 84 b_{2u}, 86 b_{3u}, a total of 460. Linear dependence in the basis set reduces the number of active MOs to 416. These are 82 a_g, 70 b_{1g}, 28 b_{2g}, 28 b_{3g}, 25 a_u, 31 b_{1u}, 75 b_{2u}, 77 b_{3u}. The SO integrals and RHF MOs were generated by the JAMOL4 program [23].

The computer used was the IBM eServer pSeries 650 model 6M2 at Nagoya City University, having SPECfp2000 value of 1295 (cf. [24]). This machine is equipped with an 8-way Power4+ microprocessor of clock rate 1.45 GHz, 128 MB L3cache, and 32 GB main memory. External storage comprises Ultra160 SCSI disk drive of 1 TB.

Computation was carried out with 8-byte double precision. The record length of the intermediate direct access file is 19 KB. We set a threshold of 1.0D–10 to cease working with any integral. The program used 1.6 GB memory. Even with less memory, there was no severe degradation of the throughput. Performance was measured without parallelization.

Total number of different rs indices: 102,412
Total number of nonzero SO integrals: 662,175,999
Nonzero ratio of SO integrals: 89.479%
Total number of nonzero transformed integrals: 491,921,496
SO integral file size: 15.906 GB
Intermediate file size: 18.603 GB
Transformed integral file size: 11.816 GB
NTIMIJ (number of ij blocks): 45
NTIMRS (number of rs blocks): 45
NDIFIJ (number of different ij indices in a block): 1944
NDIFRS (number of different rs indices in a block): 2296
CPU time for 1st and 2nd steps: 45 m 55 s
CPU time for 3rd and 4th steps: 41 m 03 s
Elapsed time for 1st and 2nd steps: 51 m 59 s
Elapsed time for 3rd and 4th steps: 49 m 01 s

These figures clearly show that complete integral transformation is realistic for $N \approx 500$ with current computer resources.

Acknowledgements

The authors thank Professor Hiroshi Tatewaki of Nagoya City University for encouragement and valuable comments.

References

[1] C.F. Bender, J. Chem. Phys. 9 (1972) 547–554.
[2] I. Shavitt, in: H.F. Schaefer III (Ed.), Methods of Electronic Structure Theory, Plenum Press, New York, 1977, pp. 189–275.
[3] S.T. Elbert, in: C. Moler, I. Shavitt (Eds.), Numerical Algorithms in Chemistry: Algebraic Methods, LBL 8158, Lawrence Berkeley Laboratory, University of California, Berkeley, 1978, p. 129.
[4] V.R. Saunders, J.H. van Lenthe, Mol. Phys. 48 (1983) 923–954.
[5] S. Wilson, in: S. Wilson (Ed.), Methods in Computational Chemistry, vol. 1, Plenum Press, New York, 1987, pp. 251–309.
[6] J. Olsen, D.L. Yeager, Int. J. Quantum Chem. 23 (1983) 789–809.
[7] Y. Mochizuki, N. Nishi, Y. Hirahara, T. Takada, Theor. Chim. Acta 93 (1996) 211–233.
[8] E.R. Davidson, J. Comput. Phys. 17 (1975) 87–94.
[9] M. Schütz, R. Lindh, H.-J. Werner, Mol. Phys. 96 (1999) 719–733.
[10] H.M. Quiney, in: S. Wilson (Ed.), Methods in Computational Chemistry, vol. 2, Plenum Press, New York, 1988, pp. 227–278.
[11] K.C. Tang, C. Edminston, J. Chem. Phys. 52 (1972) 547–548.
[12] G.H.F. Diercksen, Theor. Chim. Acta 33 (1974) 1–6.
[13] M. Yoshimine, J. Comput. Phys. 11 (1973) 449–454.
[14] A.V. Aho, J.E. Hopcroft, J.D. Ullman, Data Structures and Algorithms, Addison-Wesley Publishing Company, USA, 1983.
[15] P.R. Taylor, C.W. Bauschlicher Jr., D.W. Schwenke, in: S. Wilson (Ed.), Methods in Computational Chemistry, vol. 3, Plenum Press, New York, 1989, pp. 63–146.
[16] P.R. Taylor, in: B.O. Roos (Ed.), Lecture Notes in Quantum Chemistry, Springer-Verlag, Berlin, 1992, pp. 89–176.
[17] M.W. Schmidt, et al., J. Comput. Chem. 14 (1993) 1347–1363.
[18] T. Helgaker, et al., DALTON, A molecular electronic structure program, Release 1.2, 2001.
[19] P. Saxe, H.F. Schaefer III, N.C. Handy, Chem. Phys. Lett. 79 (1981) 202–204.
[20] W. Duch, J. Karwowski, G.H.F. Diercksen, Chem. Phys. Lett. 144 (1988) 421–422.
[21] W.J. Hehre, R.F. Stewart, J.A. Pople, J. Chem. Phys. 51 (1969) 2657–2664.
[22] S. Yamamoto, H. Tatewaki, O. Kitao, G.H.F. Diercksen, Theor. Chem. Acc. 106 (2001) 287–296.
[23] H. Kashiwagi, T. Takada, E. Miyoshi, S. Obara, F. Sasaki, JAMOL4, A library program of the Computer Center of the Institute for Molecular Science, Okazaki, Japan, 1987.
[24] SPEC (Standard Performance Evaluation Corporation), http://www.spec.org.