A partially structure-preserving algorithm for the permanents of adjacency matrices of fullerenes✩

Heng Liang, Fengshan Bai *

Department of Mathematical Sciences, Tsinghua University, Beijing, 100084, P.R. China

Received 7 April 2004; accepted 2 August 2004

Abstract

A partially structure-preserving method for sparse symmetric matrices is proposed. Computational results on the permanents of adjacency matrices arising from molecular chemistry are presented. The largest adjacency matrix of fullerenes computed before is that of $\text{C}_{60}$ with a cost of several hours on supercomputers, while only about 6 min on an Intel Pentium PC (1.8 GHz) with our method. Further numerical computations are given for larger fullerenes and other adjacency matrices with $n = 60, 80$. This shows that our method is promising for problems from molecular chemistry.

© 2004 Elsevier B.V. All rights reserved.

PACS: 02.10.Eb; 02.10.Sp; 81.05.Tp

Keywords: Permanent of matrix; Adjacency matrix; Structure-preserving algorithm; Fullerenes

1. Introduction

The permanent of an $n \times n$ matrix $A = [a_{ij}]$ is defined as

$$\text{Per}(A) = \sum_{\sigma \in \Lambda_n} \prod_{i=1}^{n} a_{i\sigma(i)}$$

(1)

where $\Lambda_n$ denotes the set of all possible permutations of $\{1, 2, \ldots, n\}$. The numerical computation of the permanent recently attracts more and more attentions from mathematicians and computer scientists [1–8], statistical physicists [9–11] and chemists [12–17]. The definition of $\text{Per}(A)$ looks similar to the determinant of matrix, however, it is much harder to be computed. Valiant [2] proves that computing the permanent of a $(0, 1)$ matrix is a #P-complete problem in counting. Even for matrices with only three nonzero entries in each row and column, evaluating their permanents is still a #P-complete problem [3].

The best-known algorithm on precise evaluation of the permanent, due to Ryser [18] and later improved by Nijenhuis and Wilf [19], runs $O(n2^{n-1})$ in complexity. For $(0, 1)$ sparse matrices, there are sev-
eral other precise algorithms, such as direct expansion method [12] and Kallman’s method [20]. Delic and Cash [15] intensively review and compare those algorithms. They are tested on different platforms, such as Cray T3D, SGI as well as Intel PC’s. Kallman’s algorithm performs the best over all others, and shows the lowest growth rate with respect to \( n \) in computational costs. Thus we always make comparison with Kallman’s algorithm in this paper.

It is only possible to make the precise calculation method faster if the special structural properties of matrices can be used intensively. In this paper, we will focus mostly on adjacency matrices arising from molecular chemistry [21] so that the structure properties of matrices are explored accordingly. They are all \((0,1)\) sparse matrices and have the following structure properties in common:

1. **symmetric**;
2. degree (i.e. row or column sum) of either 3 or 2;
3. all 0 entries along the main diagonal.

Among the adjacency matrices, the most interesting ones are those of fullerenes [12, 13, 22] which satisfy a stronger structure property instead of (2):

\[(2)' \text{degree always 3.}\]

Hence the computation for fullerenes is much harder.

A partially structure-preserving method, which improves a hybrid algorithm for sparse symmetric matrices, is proposed. Adjacency matrices used in [15] are calculated with our methods. The computational results show that the hybrid algorithm is remarkably faster than Kallman’s method. The partially structure-preserving method can speed up further by making use of the structure properties of adjacency matrices. The computation for \( C_{60} \) takes more than 6 hours on SGI Onyx2 (1 processor) with Fortran 77 using Kallman’s algorithm, while only about 6 min on an Intel Pentium PC (1.8 GHz) with Matlab using the partially structure-preserving method. Further numerical computations are presented for other fullerenes and matrices with even larger \( n \). The results show that our new method is promising for problems from molecular chemistry.

## 2. Algorithms

### 2.1. Ryser–Nijenhuis and Wilf method (R-NW)

Let \( S \) be any set, and \(|S|\) denote the number of elements of the set \( S \). Ryser [18] shows that the permanent of matrix \( A \) can be given by

\[
\text{Per}(A) = (-1)^n \sum_{S} (-1)^{|S|} \prod_{i=1}^{n} \left( \sum_{j \in S} a_{ij} \right).
\]

(2)

Here \( A \) is the \( n \times n \) matrix and \( S \) runs over all the subsets of \( \{1, 2, \ldots, n\} \). This method is based on the inclusion and exclusion principle.

The computational cost of Ryser formula is \( O(n^2 \cdot 2^n) \). Nijenhuis and Wilf [19] improve Ryser method through the following considerations.

1. It is enough to process only the subsets of \( \{1, 2, \ldots, n-1\} \). This reduces the computational cost of Ryser method by a factor of 2.
2. A Hamiltonian walk on an \((n-1)\)-cube (Chapter 1 of [19]) is used in the implementation. A slight change in the calculation for a set \( S \) yields the result for the successor of \( S \). This accomplishes a reduction of a factor \( n/2 \) in the computational cost.

So the total computational cost becomes \( O(n \cdot 2^{n-1}) \). We call the improved version R-NW method which is very efficient for small and dense matrices.

### 2.2. A hybrid method

Consider an expansion

\[
\text{Per} \begin{pmatrix} a & b & x^T \\ y_1 & y_2 & Z \end{pmatrix} = \text{Per} \begin{pmatrix} 0 & 0 & x^T \\ y_1 & y_2 & Z \end{pmatrix} + \text{Per}(a y_2 + b y_1 \ Z)
\]

\[
= \text{Per}(A_1) + \text{Per}(A_2),
\]

(3)

where \( a \) and \( b \) are scalars, \( x^T \) is an \((n-2)\)-dimensional row vector, \( y_1 \) and \( y_2 \) are both \((n-1)\)-dimensional column vectors, and \( Z \) is an \((n-1) \times (n-2)\) matrix. This expansion is used to establishing an approximate algorithm for permanent by [11]. Combining the expansion (3) with R-NW algorithm, a hybrid algorithm is constructed. Essentially, the method eliminates two
Algorithm H (Hybrid).

\[ P = H(A) \]
\[ n \leftarrow \text{the order of } A \]
\[ s \leftarrow \text{the minimal number of nonzero entries in one row or column of } A \]
\[ \text{if } n > 2 \text{ and } s < 5, \text{ then divide } A \text{ into } A_1, A_2 \text{ as (3)}, \]
\[ \text{and} \]
\[ P = H(A_1) + H(A_2) \]
\[ \text{else return } \text{by R-NW}(A). \]

The algorithm above is recursive. Here R-NW(A) means that the permanent of A is evaluated with R-NW method.

2.3. A partially structure-preserving method

The following expansion can partially preserve the structure properties (1)–(3) so as to improve the efficiency in computation further.

\[
\begin{align*}
\text{Per} & \left( \begin{array}{ccc}
0 & a & b \\
0 & 0 & 0 \\
x & y_1 & y_2 \\
\end{array} \right) \\
& = \text{Per} \left( \begin{array}{ccc}
0 & a y_2 + b y_1 & 0 \\
0 & 0 & a y_2 + b y_1 \\
x & y_1 & y_2 \\
\end{array} \right) + 2abc \cdot \text{Per}(Z) \\
& + \text{Per} \left( \begin{array}{ccc}
0 & 0 & x^T \\
0 & 0 & x^T \\
y_1 & y_2 & Z \\
\end{array} \right) \\
& = \text{Per}(A_1) + a \text{Per}(A_2) + \text{Per}(A_3) + 2 \text{Per}(B),
\end{align*}
\]

where \(a, b\) and \(c\) are scalar, \(a = 2abc\) is a constant, \(x^T, y_1^T\) and \(y_2^T\) are all \((n - 3)\)-dimensional row vector, \(Z\) is an \((n - 3) \times (n - 3)\) symmetric matrix with all diagonal entries 0.

Notice that matrices \(A_1, A_2\) and \(A_3\) are symmetric and still have all 0 along diagonal. Hence the permanent of the matrix \(A\) can be computed by the sum of permanents of matrices \(A_1, A_2\) and \(A_3\), together with the permanent of a nonsymmetric matrix \(B\). Algorithm S, a partially structure-preserving recursive method, is given as follows. The nonsymmetric matrix \(B\) is computed with Algorithm H. As long as any of the matrix becomes small and/or dense, R-NW will be used.

Algorithm S (Partially structure-preserving).

\[ P = S(A) \]
\[ n \leftarrow \text{the order of } A \]
\[ s \leftarrow \text{the minimal number of nonzero entries in one row or column of } A \]
\[ \text{if } n > 4 \text{ and } s < 5, \text{ then divide } A \text{ into symmetric matrices } A_1, A_2, A_3 \text{ and a nonsymmetric matrix } B \text{ as (2)}, \]
\[ \text{and} \]
\[ P = S(A_1) + a S(A_2) + S(A_3) + 2 H(B) \]
\[ \text{else return } \text{by R-NW}(A). \]

Table 1 shows our computational results on an Intel Pentium PC (1.8 GHz). The matrices are constructed randomly such that satisfy the structure properties (1), (2)’ and (3) in the last section. One can see that the speedup ratio of Algorithm S versus Algorithm H increases as \(n\) grows.

### Table 1

<table>
<thead>
<tr>
<th>(n)</th>
<th>(\text{Per}(A))</th>
<th>(H(s))</th>
<th>(S(s))</th>
<th>Speedup ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>33317</td>
<td>0.421</td>
<td>0.263</td>
<td>1.60</td>
</tr>
<tr>
<td>40</td>
<td>1260673</td>
<td>4.08</td>
<td>2.41</td>
<td>1.69</td>
</tr>
<tr>
<td>50</td>
<td>12237781</td>
<td>57.9</td>
<td>33.5</td>
<td>1.73</td>
</tr>
<tr>
<td>60</td>
<td>294614944</td>
<td>689</td>
<td>383</td>
<td>1.80</td>
</tr>
</tbody>
</table>

3. Numerical results

Computational results on some fullerenes with the number of carbon atoms from 30 to 82 are reported first. Results by Delic and Cash [15] are compared with our computations. Then permanents of some general matrices with structure properties (1)–(3) are calculated.

3.1. The permanents of the adjacency matrices of fullerenes

In this subsection, five test matrices which are all adjacency matrices of fullerenes are computed. The
Table 2

<table>
<thead>
<tr>
<th>Fullerene</th>
<th>( n )</th>
<th>( \text{Per}(A) )</th>
<th>( S (s) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_{30}(C_2) )</td>
<td>30</td>
<td>29621</td>
<td>0.328</td>
</tr>
<tr>
<td>( C_{44}(T) )</td>
<td>44</td>
<td>2478744</td>
<td>8.22</td>
</tr>
<tr>
<td>( C_{60}(I_h) )</td>
<td>60</td>
<td>395974320</td>
<td>355.2</td>
</tr>
<tr>
<td>( C_{70}(D_{5h}) )</td>
<td>70</td>
<td>9193937544</td>
<td>3606</td>
</tr>
<tr>
<td>( C_{82}(C_2) )</td>
<td>82</td>
<td>418912566060</td>
<td>60730</td>
</tr>
</tbody>
</table>

Data are taken from fullerenes gallery by Yoshida [23]. The computational results are shown in Table 2.

Note that \( C_{60} \) is the largest fullerene computed before on supercomputers. We can compute \( C_{70} \) and \( C_{82} \) with Algorithm S simply using PC.

3.2. Comparison with results before

Delic and Cash [15] show that type of CPU, platform, compiler and programming language are all factors to the efficiency of algorithm implementation. In their tests, Kallman’s method on SGI Onyx2 (1 processor) with Fortran 77 runs fastest, and it takes about 6.12 hrs for the case of \( C_{60} \). Algorithm S takes only about 6 min for the same problem on PC (1.8 GHz), as shown in Table 2. We use Matlab 5.3/6.1 here as the programming language for simplicity. Matlab is an interpreted programming language. It runs slower than Fortran and C in general. So our algorithms should be able to run even faster by porting them to Fortran or C.

The comparison between Kallman’s method and Algorithm S is shown in Table 3. Results of Kallman’s method are all taken from [15]. That the Intel 133 MHz runs faster than the Intel 166 MHz is due to the difference in compiler. It indicates that WATCOM compiler is about ten times faster than Fortran compiler at that time. Note that test matrices with \( n = 30, 44, 60 \) are all degree 3, whereas those with \( n = 48, 52, 56 \) have row sums either 3 or 2 (and hence sparser). So they are divided into two subgroups. In order to rule out the factor of hardware, our Matlab 5.3 program is run on an Intel Pentium (133 MHz). That is also a kind of computer used by [15].

Regardless factors of programming language and compiler, Algorithm S is still about 50 times faster than Kallman’s method for the case \( n = 60 \).

Computational costs increase about 25 times for Algorithm S and at least 145 times for Kallman’s algorithm, as \( n \) grows from \( n = 30 \) to \( n = 44 \). And they increase about 40 times for Algorithm S and at least 250 times for Kallman’s algorithm, as \( n \) grows from \( n = 44 \) to \( n = 60 \). This indicates that Algorithm S may still be promising as \( n \) goes larger.

We further find that the logarithm of running times are almost linearly increase with respect to \( n \) for both Algorithm S and Kallman’s method. The logarithm of running time of Algorithm S (we take the case of Intel PC 1.8 GHz) has the increase scope about 0.336 with \( n \) and that of Kallman’s method (we take the case of SGI Onyx2, which is the best in [15]) has the increase scope 0.506 with \( n \). The relations between \( \log_2(T) \) and \( n \) of Algorithm S and Kallman’s method are shown in Fig. 1, where the \( T \) denotes the running time, \( K \) denotes Kallman’s method and \( S \) denotes Algorithm S. For fullerenes, the time complexity of Algorithm S method is roughly \( O(2^n/3) \) and that of Kallman’s method is roughly \( O(2^n/2) \). Note that the time complexity of R-NW is \( O(n^2) \). As \( n \) is little larger, such differences are crucial.

3.3. Further numerical computation

In order to show the efficiency of our methods further, we construct two sets of matrices with \( n = 60 \) and \( n = 80 \), respectively. They all have the structure properties (1)–(3). Let \( m \) denote the number of rows (columns) with degree 2, and the rest of rows (columns) all have degree 3. Numerical results are given in Tables 4 and 5.

From the computational results one can conclude that Algorithms H and S are high speed. They are promising for large scale very sparse problems.
4. Discussions and conclusions

In this paper, a structure-preserving method is proposed for finding the permanents of adjacency matrices arising from molecular chemistry. Fullerenes up to $C_{82}$ are computed with our new method, while $C_{60}$ is the largest fullerenes ever computed before. This shows that the structure-preserving method is promising for larger fullerenes. Approximate algorithm may essentially be the unique way for even larger fullerenes, since it is a #P-complete problem in computation [3].

Compare with the precise methods discussed in this paper, approximate algorithms can produce a reasonable estimation for $\text{Per}(A)$ within a relatively short CPU time. Results obtained are no longer exact. Approximate algorithms on permanent can roughly be divided into the following classes: random Laplacian expansion method [6], reduction to determinants [5], Markov chain Monte Carlo [4,8] and importance sampling method [10]. Abundant theoretical analysis has been developed for those methods. Works still need to be done for solid applications.

Acknowledgments

The authors thank referees for their careful reading and helpful suggestions and comments.

References