MATCH
Communications in Mathematical and in Computer Chemistry

MATCH Commun. Math. Comput. Chem. **69** (2013) 831-842

ISSN 0340 - 6253

Improved Algorithms for Permanent and Permanental Polynomial of Sparse Graph

Baojun Yue, Heng Liang*, Fengshan Bai†

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

(Received May 31, 2012)

Abstract

A novel algorithm for permanent of sparse graph is proposed, which combines a new permanent expansion formula and graph bisection. Hence the improved algorithm for permental polynomial of sparse graph is followed. Computational results on the permanents and permanental polynomials of fullerene graphs are presented. The new algorithms increase the computable scale for permanents and permanental polynomials dramatically on PC within acceptable time.

1 Introduction

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

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

where Λ_n denotes the set of all possible permutations of $\{1, 2, ..., n\}$.

The permanent function arises in a number of fields, including mathematics [1], physical sciences [2,3] and molecular chemistry [4–6]. However, computing the permanent of a matrix is proved to be a $\#P$-complete problem in counting [7]. Even for 3-regular matrices, which are with 3 nonzero entries in each row and column, evaluating their permanents is still a $\#P$-complete problem [8].

*E-mail address: hliang72@gmail.com

†Supported by National Science Foundation of China 10871115.

The best-known algorithm for precise evaluation of the permanent of general matrix is due to Ryser, Nijenhuis and Wilf [9]. It is $O(n2^{n-1})$ in time complexity. This method only works for small matrices. It is only possible to make the precise calculation faster, if the special structure properties of matrices can be used intensively. In this paper, we will focus on sparse matrices. Several precise algorithms have been proposed by exploring the structure properties of sparse matrices intensively, such as Kallman's method [10], expansion algorithm [11] and partially structure-preserving algorithm [12].

The permanental polynomial of a graph G is defined as

$$P(G, x) = per(xI - A), \tag{2}$$

where A is the adjacency matrix of the graph G with n vertices, and I is the identity matrix of order n.

For a graph with n vertices, its permanental polynomial can be computed by $n/2+1$ permanents of $n \times n$ matrices and an $(n+1)$-element FFT [13]. Hence the efficiency of the algorithm for permanental polynomial is determined by that of the algorithm for permanents.

In this paper, a new permanent expansion formula is given. Then a novel algorithm for permanent of sparse graph is proposed, which is based on graph bisection. Hence the improved algorithm for permental polynomial of sparse graph is followed. The fullerene graphs used in [10, 12, 13] are calculated by new methods. The computational results show that the new permanent algorithm is faster than the algorithm proposed in [12] when $n \geq 70$ and the new permanental polynomial algorithm is faster than the algorithm proposed in [13] algorithm when $n \geq 30$. The algorithms proposed here increase the computable scale from C_{100} to C_{150} for permanents and from C_{60} to C_{90} for permanental polynomials on PC under the same time and precision tolerance.

In the next section, the new permanent expansion formula is presented. In section 3, graph bisection problem and algorithms are introduced briefly. Then our sparse permanent and permanental polynomial algorithms are proposed, which combine the new permanent expansion formula and graph bisection. The numerical results are given in section 4. Some discussions are made in section 5.

2 A Permanent Expansion in Block

Consider a 0-1 matrix $M = [m_{ij}]_{p\times q}$. If elements $m_{i_1,j_1}, ..., m_{i_k,j_k}$ have no row or column in common, the set of k elements is called a k-**permutation** of matrix M; and if $\prod_{s=1}^{k} m_{i_s j_s} = 1$, then the k-permutation is called nonzero.

Let $[A|\{i_1, ..., i_k\}, \{j_1, ..., j_s\}]$ be the $(n-k)\times(n-s)$ matrix obtained by deleting the i_1th,$\cdots$,i_kth rows and j_1th,$\cdots$,j_sth columns from the matrix A. The main result of this section is as follows. The proof of this theorem is presented in the Appendix.

Theorem 2.1. *Assume*

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},$$

where A_{11}, A_{22} are $n_1 \times n_1$ and $n_2 \times n_2$ respectively. Let $n_s = min\{n_1, n_2\}$. The set consisting of all the nonzero k permutations in A_{12} is denoted as Λ_{1k}, the set consisting of all the nonzero k permutations in A_{21} is denoted as Λ_{2k}, $(1 \le k \le n_s)$. Then

$$perm \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = perm(A_{11}) \cdot perm(A_{22}) +$$

$$\sum_{k=1}^{n_s} \sum_{\substack{a_{i_1 j_1},...,a_{i_k j_k}\in\Lambda_{1k} \\ a_{s_1 l_1},...,a_{s_k l_k}\in\Lambda_{2k}}} perm[A_{11}|\{i_1, ..., i_k\}, \{l_1, ..., l_k\}] \cdot perm[A_{22}|\{s_1, ..., s_k\}, \{j_1, ..., j_k\}].$$

It is clear that the sparser the off-diagonal blocks in matrix A, the fewer expansion terms in the expansion. The graph bisection algorithm is applied to achieve this goal.

3 The Algorithms

3.1 Graph bisection

Consider a graph $G(V, E)$, where V denotes the set of vertices and E denotes the set of edges. The graph bisection problem is to partition V into two parts (subsets) V_1 and V_2, such that the parts are disjoint and the number of edges between them is minimized.

Graph bisection is an NP-complete problem. The problem has many applications, such as in VLSI placement and routing problems [14]. Many fast heuristics have be developed [15, 16].

Fullerene type graphs are exactly small average degree graphs. Thus we adopt compaction combining with Kernighan-Lin algorithm [15] in this paper.

Essentially, the bisection algorithm gives a reordering of n vertices of the graph G. The adjacency matrix with new vertex order is $B = PAP^T$, where P is a permutation matrix.

3.2 The Permanent Algorithm Based on Graph Partition

The permanent expansion formula in Theorem 2.1 could be efficiently used in constructing a numerical algorithm for permanent only when the matrices A_{12} and A_{21} are sparse enough. For n vertices of the graph G, we use the graph bisection algorithm to reorder the rows and columns of A, and evaluate the permanent via expanding the matrix that has been reordered.

The permutation matrix P is obtained by graph bisection algorithm. Then let

$$B = PAP^T = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix},$$

in which the nonzero elements are gathered in diagonal blocks B_{11} and B_{22} and $perm(A) = perm(B)$. Here B_{11} is $n_1 \times n_1$ matrix and B_{22} is $n_2 \times n_2$ matrix.

For any k-permutations $\sigma_1 = \{b_{i_1,j_1}, \cdots, b_{i_k,j_k}\}$ in B_{12}, $\sigma_2 = \{b_{s_1,l_1}, \cdots, b_{s_k,l_k}\}$ in B_{21}, let $[B_{11}|\{i_1, ..., i_k\}, \{l_1, ..., l_k\}]$ and $[B_{22}|\{s_1, ..., s_k\}, \{j_1, ..., j_k\}]$ be denoted as $[B_{11}|\sigma_1, \sigma_2]$ and $[B_{22}|\sigma_1, \sigma_2]$.

The function H (hybrid algorithm), which is proposed to compute the permanent of sparse matrix in [11], is called directly in the following algorithm. Then we give an algorithm for sparse permanents based on graph bisection as follows.

Algorithm SP (sparse permanent algorithm based on graph bisection)

Input: A sparse graph G and its adjacency matrix A

Step 1: Using graph bisection algorithm to obtain the permutation matrix P.

Let $B = PAP^T = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$.

Step 2: Finding all the nonzero permutations of B_{12}, denoted by Λ_{1k} $(k = 1, \cdots, m_1)$, and all the nonzero permutations of B_{21}, denoted by Λ_{2k} $(k = 1, \cdots, m_2)$.

Step 3: Let $m = min(m_1, m_2)$ and $p = H(B_{11}) \cdot H(B_{22})$,
for k=1 to m

$$p = p + \sum_{\sigma_1 \in \Lambda_{1k}, \sigma_2 \in \Lambda_{2k}} H([B_{11}|\sigma_1, \sigma_2]) \cdot H([B_{22}|\sigma_1, \sigma_2]).$$

Output: $perm(A) = p$.

3.3 The Improved Permanental Polynomial Algorithm

let $\omega_N = e^{(-2\pi i)/N}$ be the Nth root of unity. Take x_j be the ω_{n+1}^j $(j = 0, 1, \cdots, n)$. The values of $p_n(x_i) = per(x_iI - A)$ with $i = 0, 1, 2, \cdots, n$ are computed by the algorithm SP, then all the coefficients of $per(xI - A)$ can be solved by the following linear system of equations.

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^n \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \\ a_n \end{pmatrix} = \begin{pmatrix} per(x_0I - A) \\ per(x_1I - A) \\ \vdots \\ per(x_{n-1}I - A) \\ per(x_nI - A) \end{pmatrix}. \tag{3}$$

The system (3) is the discrete Fourier transformation and can be solved rapidly, robustly and accurately by Fast Fourier Transformation (FFT) in $O(n \log n)$ time.

4 Numerical Results

In order to illustrate the efficiency of our algorithms, we have applied them to fullerene graphs. A fullerene C_n is a polyhedral carbon cage with n atoms. The adjacency matrices of fullerenes are symmetric 3-regular (0,1) matrices. The permanent and permanental polynomial of a chemical graph G is of interest in chemical graph theory [4, 6, 17–21]. For the permanents of this kind of matrix, several precise algorithms have been proposed by exploring the structure properties of matrices intensively.

So far as we know, the partially structure-preserving algorithm proposed in [12] is the most efficient algorithm for permanents of fullerene structure. And the the best one for permannetal polynomial is the algorithm proposed in [13], which adapts the permanent algorithm to FFT. Our new algorithms are compared with them.

The numerical experiments are carried on a Linux system using Intel Core i3 530 CPU(2930 MHz) and 4 GB RAM. Fortran 90 and MATLAB are used as programming language. All computations are performed under double-precision and quadruple-precision arithmetic.

4.1 Results on Permanents of Fullerene Graphs

Some fullerenes with the number of carbon atoms from 30 to 120 are computed. The data are taken from [22]. The computational results are shown in Table 1, which give the

comparison between the new algorithm and the partially structure-preserving algorithm. For simplicity, we denote the partially structure-preserving algorithm as algorithm P, and the graph bisection based algorithm proposed in this paper as algorithm SP.

Note that C_{82} was the largest computed fullerenes reported before. We compute C_{120} with algorithm SP simply using PC. Because the permanent of C_{120} is over 16 significant bits, quadruple-precision is necessary for computing the permanents of $C_{\geq 120}$.

When $n = 30$, the computational time of algorithm P is far shorter than that of algorithm SP. The computational times of the two algorithms are almost the same when $n = 70$. The computational time of algorithm SP is only no more than half of that of algorithm P when n increases to 80. When $n > 100$, the algorithm P can hardly afford the computational costs.

Table 1. Computational results of algorithm B for fullerenes

			Computational time (second)		
Fullerene	n	Per(A)	Algorithm P	Algorithm SPB	Speedup
$C_{30}(C_{2\nu})$	30	29621	0.004	0.063	0.066
$C_{44}(T)$	44	2478744	0.12	0.53	0.226
$C_{60}(I_h)$	60	395974320	5.34	13.80	0.387
$C_{70}(D_{5h})$	70	9193937544	55.75	42.39	1.315
$C_{80}(C_2)$	80	189275868081	360.44	147.24	2.452
$C_{90}(C_{2v})$	90	5206290577049	4703.23	484.56	9.705
$C_{100}(C_2)$	100	116149608133433	65712.58	836.22	78.581
$C_{120}(Td)$	120	81917875997012096	*	19315.03	*

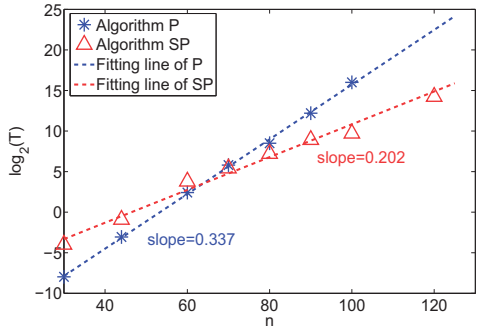

Figure 1. The comparison between the partially structure-preserving algorithm (P) and graph bisection based algorithm (SP)

As n grows from $n = 30$ to $n = 100$, the computational times increase about 12,000 times for algorithm SP and more than 10^7 times for algorithm P. This indicates that the algorithm SP may still be promising as n goes larger.

We further observe that the logarithm of running times are almost linearly increase with respect to n for both algorithmd SP and P. The increase scope of logarithm of running time is about 0.202 with n for algorithm SP, and about 0.337 with n for algorithm P. The relations between $\log_2(T)$ and n on algorithm SP and P are shown in Fig.1, where the T denotes the running time. For fullerenes, the time complexity of algorithm SP is roughly $O(2^{n/5})$ and that of algorithm P is roughly $O(2^{n/3})$. As n becomes larger, such difference is crucial.

4.2 Results on Permanental Polynomials of Fullerene Graphs

We compare the improved permanental polynomial algorithm, which is denoted by algorithm I for simplicity, with the algorithm proposed in [13], which is denoted by algorithm F. The Table 2 gives the comparison between these two algorithms for C_{20} to C_{52}. The computations are made under double-precision.

Table 2. The comparision of computational times between algorithm F and I for fullerenes

Fullerene	n	Algorithm F	Algorithm I	Speedup
C_{20}	20	0.01	0.006	1.67
$C_{30}(C_{2\nu})$	30	0.73	0.35	2.09
$C_{40}(D_{2h})$	40	36.58	7.52	4.86
$C_{44}(T)$	44	191.77	38.20	5.02
$C_{50}(D_{5h})$	50	2128.14	365.60	5.82
$C_{52}(D_2)$	52	5406.21	892.32	6.06

Table 3. Computing time (seconds) for the coefficients of the permanental polynomials

Fullerene	$C_{30}(C_{2\nu})$	$C_{40}(D_{2h})$	$C_{44}(T)$	$C_{50}(D_{5h})$
CPU time	15.25	130.40	620.60	1448.97
Fullerene	$C_{52}(D_2)$	$C_{56}(T_d)$	$C_{60}(I_h)$	$C_{70}(D_{5h})$
CPU time	5015.64	8756.15	15180	171970

When $n > 54$, the quadruple-precision is necessary due to the effect of rounding error.

Table 3 presents the running times of computing the permanental polynomials of C_{30} to C_{70} with quadruple-precision by our new method. The logarithm of running times still grow almost linearly with respect to n, as shown in Fig 2.

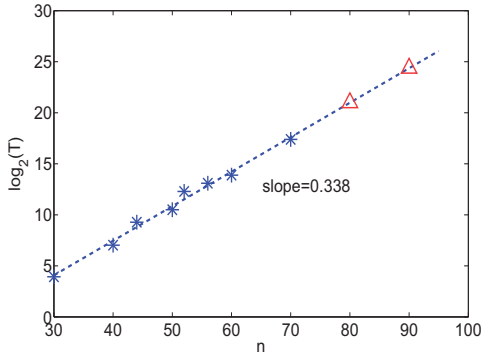

Figure 2. The relations between $log_2(T)$ and n

The linear regression equation is given by

$$Y = 0.33798n - 6.05771, \quad R^2 = 0.989142 \tag{4}$$

The slope of the line is about 0.338, hence the time complexity of Algorithm I is roughly $O(2^{0.24n})$ for fullerene graphs. Note that the time compulexity of Algorithm F is roughly $O(2^{0.59n})$ [13]. The labeled two triangles denote the predictions for the computing times of C_{80} and C_{90}. They are about 675 and 6982 hours respectively. The algorithm can be parallelized easily. Hence the computation for $C_{\geq 80}$ is promising.

Then we consider the accuracy of the computation for permanental polynomials of fullerene graphs. The coefficients of a permanental polynomial are all integer, while Fast Fourier transformation gives output in real numbers. The part of FFT shows the computational precision. Let cof_C_n be the coefficients vector of C_n derived from FFT directly. Take

$$error = \max_{1 \leq k \leq n+1} |round(cof\_C_n(k)) - cof\_C_n(k)|$$

be a measure of the computational precision. Table 4 gives the trend of the computational precision with n from 30 to 70. The round error is less than 10^{-12} when n is up to 70. Fig 3 shows the error trend. The linear regression equation is given by

$$Y = 0.31262n - 35.19053, \quad R^2 = 0.982561 \tag{5}$$

According to this trend, the permanental polynomial of larger fullerenes will be accurately computed, because the computational precision is adequate. The two triangles denote the predictions of the errors, and they are no more than 10^{-5} and thus acceptable.

Table 4. The trend of the computational precision with n

Fullerene	$C_{30}(C_{2\nu})$	$C_{40}(D_{2h})$	$C_{44}(T)$	$C_{50}(D_{5h})$
error	4.23×10^{-26}	3.03×10^{-23}	4.34×10^{-22}	1.40×10^{-20}
Fullerene	$C_{52}(D_2)$	$C_{56}(T_d)$	$C_{60}(I_h)$	$C_{70}(D_{5h})$
error	4.20×10^{-20}	8.32×10^{-19}	1.13×10^{-17}	4.57×10^{-13}

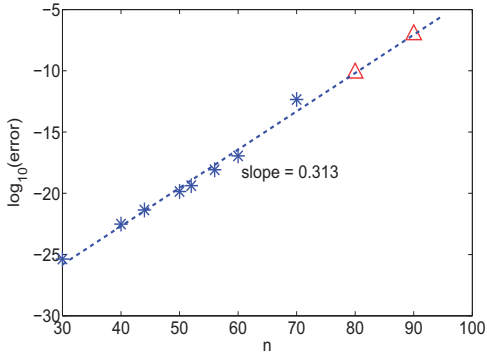

Figure 3. The error trend

5 Discussions and Conclusions

In this paper, a novel algorithm based on graph bisection is proposed for computing the permanents of sparse matrices. For fullerene-type graph, the efficiency of computation and computable scale of permanents and permanental polynomials achieve the significant increase. The computational times of the new algorithms increase much slower than those of the other existing algorithms when n grows. This shows that the new methods are promising for larger fullerenes. And this makes it possible to compute the permanents of large fullerenes in bulk and explore the way in which they depend on the molecular

structure more profoundly [21]. Though the examples computed in this paper are all fullerenes, the algorithms are equally applicable to other types of chemical structures as well.

The efficiency of the new algorithms relies on the performance of the graph bisection. The sparser the matrix B_{12} and B_{21} in Algorithm SP, the more efficient in the permanent evaluation. The graph bisection used in the paper can be regarded as a preconditioner for the algorithm on sparse permanents induced by Theorem 2.1. It is one of the most popular algorithm, but not necessarily the best here. Finding more efficient bisection algorithm for this kind of structure is a meaningful way to improve the algorithm further.

Appendix

Consider the permanent expansion in block. The following result is well known [1].

Lemma 5.1. *Assume*

$$A = \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix},$$

where A is an $n \times n$ matrix, both A_{11} and A_{22} are not square matrices. Then

$$perm(A) = 0.$$

The proof of Theorem 2.1

Let

$$A_x = \begin{bmatrix} A_{11} & x \cdot A_{12} \\ x \cdot A_{21} & A_{22} \end{bmatrix},$$

where x is regarded as a parameter. It is clear that

$$perm(A_x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n.$$

Therefore $perm(A) = a_0 + a_1 + \cdots + a_n$.

According to all the nonzero permutations of A_{12} and A_{21} to do Laplace expansion for $perm(A_x)$, the coefficient a_t can be represented as

$$\sum_{p+q=t} \sum_{\substack{a_{i_1 j_1},\ldots,a_{i_p j_p} \in \Lambda_{1p} \\ a_{s_1 l_1},\ldots,a_{s_q l_q} \in \Lambda_{2q}}} perm \begin{bmatrix} [A_{11}|\{i_1,\cdots,i_p\},\{l_1,\cdots,l_q\}] & 0 \\ 0 & [A_{22}|\{s_1,\cdots,s_q\},\{j_1,\cdots,j_p\}] \end{bmatrix}.$$

By the result of Lemma 1, when $p \neq q$,

$$perm \begin{bmatrix} [A_{11}|\{i_1,\cdots,i_p\},\{l_1,\cdots,l_q\}] & 0 \\ 0 & [A_{22}|\{s_1,\cdots,s_q\},\{j_1,\cdots,j_p\}] \end{bmatrix} = 0.$$

Hence $a_t = 0$ when t is odd.

Only when $p = q = \frac{t}{2}$, a_t may be nonzero. Hence the coefficient of the highest order term of $perm(A_x)$ is $2n_s$. For any $1 \le k \le n_s$,

$$a_{2k} = \sum_{\substack{a_{i_1 j_1},\cdots,a_{i_k j_k} \in \Lambda_{1k} \\ a_{s_1 l_1},\cdots,a_{s_k l_k} \in \Lambda_{2k}}} perm \begin{bmatrix} [A_{11}|\{i_1,\cdots,i_k\},\{l_1,\cdots,l_k\}] & 0 \\ 0 & [A_{22}|\{s_1,\cdots,s_k\},\{j_1,\cdots,j_k\}] \end{bmatrix}$$

$$= \sum_{\substack{a_{i_1 j_1},\cdots,a_{i_k j_k} \in \Lambda_{1k} \\ a_{s_1 l_1},\cdots,a_{s_k l_k} \in \Lambda_{2k}}} perm([A_{11}|\{i_1,\cdots,i_k\},\{l_1,\cdots,l_k\}]) \cdot perm([A_{22}|\{s_1,\cdots,s_k\},\{j_1,\cdots,j_k\}]).$$

It is easy to know that $a_0 = perm(A_{11}) \cdot perm(A_{22})$. Hence we have

$$perm \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = perm(A_{11}) \cdot perm(A_{22}) +$$

$$\sum_{k=1}^{n_s} \sum_{\substack{a_{i_1 j_1},\cdots,a_{i_k j_k} \in \Lambda_{1k} \\ a_{s_1 l_1},\cdots,a_{s_k l_k} \in \Lambda_{2k}}} perm[A_{11}|\{i_1,...,i_k\},\{l_1,...,l_k\}] perm[A_{22}|\{s_1,...,s_k\},\{j_1,...,j_k\}].$$

References

[1] H. Minc, *Permanents*, Addison-Wesley, Reading, 1978.

[2] I. Beichl, F. Sullivan, Approximating the permanent via importance sampling with application to the dimer covering problem, *J. Comput. Phys.* **149** (1999) 128–147.

[3] Y. Huo, H. Liang, S. Liu, F. Bai, Computing monomer–dimer systems through matrix permanent, *Phys. Rev. E* **77** (2008) 016706.

[4] D. Kasum, N. Trinajstić, I. Gutman, Chemical graph theory. 3. On the permanental polynomial, *Croat. Chem. Acta* **54** (1981) 321–328.

[5] N. Trinajstić, *Chemical Graph Theory*, CRC Press, Boca Raton, 1992.

[6] I. Gutman, Permanents of adjacency matrices and their dependence on molecular structure, *Polycyclic Arom. Comp.* **12** (1998) 281–287.

[7] L. Valliant, The complexity of computing the permanent, *Theor. Comput. Sci.* **8** (1979) 189–201.

[8] P. Dagum, M. Luby, Approximating the permanent of graphs with large factors, *Theor. Comput. Sci.* **102** (1992) 283–305.

[9] A. Nijenhuis, H. S. Wilf, *Combinatorial Algorithms for Computers and Calculators*, Academic Press, New York, 1978.

[10] G. Delic, G. G. Cash, The permanent of 0,1 matrices and Kallman's algorithm, *Comput. Phys. Commun.* **124** (2000) 315–329.

[11] H. Liang, S. Huang, F. Bai, Hybrid algorithms for evaluating permanents of sparse matrices, *Appl. Math. Comput.* **172** (2006) 708–716.

[12] H. Liang, F. Bai, A partially structure-preserving algorithm for the permanents of adjacency matrices of fullerenes, *Comput. Phys. Commun.* **163** (2004) 79–84.

[13] Y. Huo, H. Liang, F. Bai, An efficient algorithm for computing permanental polynomials of graphs, *Comput. Phys. Commun.* **175** (2006) 196–203.

[14] S. Bhatt, T. Leithton, A framework for solving VLSI graph problems, *J. Comput. Sys. Sci.* **28** (1984) 300–343.

[15] B. W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, *Bell Sys. Tech. J.* **49** (1970) 291–307.

[16] T. Bui, S. Chaudhuri, T. Leighton, M. Sipser, Graph bisection algorithm with good average case behavior, *Combinatorica* **7** (1987) 171–191.

[17] G. G. Cash, The permanental polynomial, *J. Chem. Inf. Comput. Sci.* **40** (2000) 1203–1206.

[18] G. G. Cash, Permanental polynomials of the smaller fullerenes, *J. Chem. Inf. Comput. Sci.* **40** (2000) 1207–1209.

[19] G. G. Cash, A differential operator approach to the permanental polynomial, *J. Chem. Inf. Comput. Sci.* **42** (2002) 1132–1135.

[20] I. Gutman, G. G. Cash, Relations between the permanental and characteristic polynomials of fullerenes and benzenoid hydrocarbons, *MATCH Commun. Math. Comput. Chem.* **45** (2002) 55–70.

[21] Q. Chou, H. Liang, F. Bai, Remarks on the relations between the permanental and characteristic polynomials of fullerenes, *MATCH Commun. Math. Comput. Chem.* **66** (2011) 743–750.

[22] P. W. Fowler, D. E. Manolopoulos, *An Atlas of Fullerenes*, Oxford Univ. Press, Oxford, 1995.