Research Journal of Applied Sciences, Engineering and Technology 4(23): 5206-5211, 2012
ISSN: 2040-7467

Submitted: April 25, 2012 Accepted: May 13, 2012 Published: December 01, 2012

A Generalized Permanent Estimator and its Application in Computing Multi-Homogeneous Bézout Number

[1]Hassan M.S. Bawazir, [2]Ali Abd Rahman and [2]Nor'aini binti Aris
[1]Department of Mathematics, Faculty of Education, Hadhramout University of Science and Technology, Seiyun, Hadhramout, Yemen
[2]Department of Mathematics, Faculty of Science, Universiti Teknologi Malaysia, Malaysia

Abstract: The permanent of a matrix has many applications in many fields. Its computation is #P-complete. The computation of exact permanent large-scale matrices is very costly in terms of memory and time. There is a real need for an efficient method to deal well with such situations. This study designs a general algorithm for estimating the permanents of the complex square or non-square matrices. We prove that the Multi-Homogeneous Bézout Number (MHBN) can be estimated efficiently using the new algorithm. Further, a proposition that provides some analytic results is presented and proved. The analytic results show the effectiveness and the efficiency of our algorithm over some recent methods. Furthermore, with the new algorithm we can control the accuracy as we need. A large amount of numerical results are presented in this study. By applying the algorithm that estimates MHBN we extend the applicability of the algorithm.

Keywords: Multi-homogeneous bézout number, permanent, polynomial system, random path

INTRODUCTION

The permanent of a matrix is very important and has many applications. Its computation is #P-complete (Liang *et al*., 2007). The computation of exact permanent of large or even moderate matrices is very costly in terms of memory and time. There is a real need for an efficient method to deal well with such situations.

Liang *et al.* (2007) presented an estimator of permanents of square 0-1matrices. In this study we present a general estimator, which can estimate the permanent of any matrix, square or non-square, integer matrix or even general complex matrix, the focus will be on non-negative real matrices. With the new method we can deal very efficiently with large-scale matrices and we can control the accuracy as we need.

An application of the new method is the estimation of MHBN. In this case we recognize the problem from two sides.

Firstly, MHBN of a multivariate polynomial system is an estimation of the number of isolated solutions of the system and it is also the number of solution paths that can be traced to reach all isolated solutions. A polynomial system can be homogenized in many ways, each giving its own MHBN. The space of all ways of homogenizing the system (variable partitions) increases exponentially as the system size grows, this problem is an NP-hard problem (Malajovich and Meer, 2007); we dealt with this problem in details in Bawazir and Abd Rahman (2010).

Secondly, the computation of an MHBN at a fixed partition, especially when such a partition has a number of elements close to the system size, is equivalent to computing the permanent of a matrix (Verschelde, 1996). By Liang *et al.* (2007) such a problem is #P-complete. In this study we will focus on computing the MHBN at a fixed partition as an application of the general estimator of the permanents.

The Multi-Homogeneous Bézout Number (MHBN): Consider the multivariate polynomial system:

$$F(x) = f_1(x), f_2(x), \ldots, f_n(x)) \tag{1}$$

where, $x(x_1, x_2, \ldots, x_n) \in C^n$

Let $z = \{z_1, z_2, \ldots, z_m\}$ be an m-partition of the unknowns $X = \{x_1, x_2, \ldots, x_n\}$ where,

$$z_j = \{z_{j1}, z_{j2}, \ldots, z_{jkj}\}, j = 1, 2, \ldots, m. \; j = 1, 2, \ldots, m$$

Define the degree matrix of the system $F(x) = 0$ as the following:

$$D = \begin{pmatrix} d_{11} & d_{12} & \ldots & d_{1m} \\ d_{21} & d_{22} & \ldots & d_{2m} \\ . & . & . & . \\ . & . & . & . \\ d_{n1} & d_{n2} & \ldots & d_{nm} \end{pmatrix} \tag{2}$$

Corresponding Author: Hassan M.S. Bawazir, Department of Mathematics, Faculty of Education, Hadhramout University of Science and Technology, Seiyun, Hadhramout, Yemen

where, d_{ij} is the degree of polynomial f_i w.r.t. the variable z_j. The degree polynomial of F w.r.t. the partition z is defined as:

$$f_D(y) = \prod_{i=1}^{n} \sum_{j}^{m} d_{ij} y_j \quad (3)$$

The m-Multi-Homogeneous Bézout Number (or in short m-MHBN) of F w.r.t. the partition z equals the coefficient of the monomial $y^k = y^{k1}_1 y^{k2}_2 \ldots y^{km}_m$ in the degree polynomial $f_D(y)$, (3) and denoted by B_m, where k = (k_1, k_2, …, k_m) with k_j = #(z_j), j = 1, 2, …, m and $\sum_{j=1}^{m} k_j = n$.

RELATIVE METHODOLOGY

Expanding the degree polynomial, (3), is called Basic Algorithm (BA) (Wampler, 1992). It is easy to find that BA has:

$$N = \prod_{i=1}^{m} \binom{n_i}{k_i}$$

terms (Also the number of addition operations), where n_1 = n and $n_i = n_{i-1} - k_{i-1}$ for i = 2, 3, …, n. This number N ranges from N = 1 for a 1-MHBN up to N = n for an n-MHBN, i.e., m = n.

In the following we describe two algorithms provided by Wampler (1992). The first is the Row Expansion algorithm (RE) and the second is the Row Expansion with Memory (REM). For RE, consider the degree matrix (2), D, first in row 1 of D, suppose we choose element d_{1j}, to complete the degree product we must choose only one element from each of the remaining rows and only k_j elements from the j th column, so this procedure can be done by applying the same steps on a minor matrix derived from the matrix D by eliminating the row 1 and the column j, denote this procedure by B(D, k-e_j, 1), where ej is the j th row of the identity matrix of degree m. Then, the row expansion algorithm computes the Bézout number as the sum along the first row expansion procedure and can be expressed by the following recursive relation:

$$B(D,k,i) = \begin{cases} 1, & if\ i = n+1 \\ \sum_{j=1, k_j \neq 0}^{m} d_{ij} \times B(D, k - e_j, i+1), & otherwise \end{cases} \quad (4)$$

where, m-MHBN is B(D, k, 1).

This formula computes directly the appropriate coefficient, i.e., m-MHBN, so it saves the operations in comparison to expanding the degree polynomial (3).

By Wampler (1992), for the most expensive case, m = n, we have k_i = 1, I = 1, 2, …, n and the number of multiplications comes to be (1/1!+1/2!+…+ 1/(n−1)!)n!. As n increases, this rapidly approaches (e-1)n!.

Table 1: PC time for computing MHBN using RE

n	Time: t(n)(sec)	Time (min)	t(n)/t(n-1)
10	268.32	4.47	10.01
11	2964.93	49.42	11.05
12	36235.33	603.92	12.22

Table 2: PC time for computing MHBN using REM

n	Time: t(n)(sec)	Time (min)	t(n)/t(n-1)
18	352.98	5.88	2.05
19	713.58	11.89	2.02
20	1431.66	23.86	2.01

For the row expansion with memory REM, Wampler (1992) uses the same recursive relation (4) with giving consideration to the repeated subtotals as shown from the following example.

Example 1: Let n = 4, m = 2 and k = (2, 2). Using BA Bézout number B is:

$$B = d_{11}d_{21}d_{32}d_{42} + d_{11}d_{22}d_{31}d_{42} + d_{11}d_{22}d_{32}d_{41} + d_{12}d_{21}d_{31}d_{42} + d_{12}d_{21}d_{32}d_{41} + d_{12}d_{22}d_{31}d_{41}$$

which uses 18 multiplications. Using RE, the following is obtained:

$$B = d_{11}(d_{21}d_{32}d_{42} + d_{22}(d_{31}d_{42} + d_{32}d_{41}) + d_{12}(d_{21}(d_{31}d_{42} + d_{32}d_{41}) + d_{22}d_{31}d_{41})$$

which uses only 12 ultiplies. We note that the expression B(D, (1, 1), 3) = $d_{31}d_{42}+d_{32}d_{41}$ appears twice by both B(D, (1, 2), 2) and B(D, (2, 1), 2), so REM,avoids such cases.

Thus, REM requires a memory array of dimension $(k_1+1)\times(k_2+1)\times\ldots\times(k_m+1)$. The number of multiplications in the worst case m = n is $n2^{n-1}$ which is much smaller than that of RE. This improvement comes at the expense of a memory array that in the worst case has 2^n (Wampler, 1992).

We have done some numerical experiments to show the cost of Wampler's methods. Table 1 and 2 show the PC time for computing n−MHBN of the known cyclic n−roots problem by using RE and REM respectively. The algorithms are implemented in MATLAB and executed on a personal computer with Intel (R) Pentium (R) Dual CPU E2160 1.80 GHz CPU and 2.00 GB of RAM.

From the fourth column of the Table 1, we conclude that the time devoted using RE at step n is about n times the time devoted in the step (n−1), this is consistent with the analysis of the algorithm RE. From this observation we can estimate the time at n = 20 by 5.08×10^{10} h or 7.05×107 months. By the same way, from Table 2, the time devoted at step n using REM is about double the time at step n−1 by which we conclude that we need about 34 days to compute (n = 30) n-MHBN of such a problem. Further, the best recent algorithm REM cannot be applicable for the case n>27 in such a PC, because it cannot provide a memory array larger than 2^{27} elements.

The recent methods for minimizing Bézout number such as Genetic Algorithm (Yan *et al*., 2008) used RE and

REM algorithms for finding the minimum MHBN for moderate systems up to n = 11. From such cases, we conclude that there is a need to use efficient methods to deal well with large-scale systems.

Liang *et al.* (2007) presented an estimator of the permanents of the square 0-1matrices; it is called random path and shortly RP. This estimator is built up on the estimator of Rasmussen (1994) by adding the column pivoting property.

In this study we construct a general permanent estimator over RP. The new one is constructed for estimating the permanent of square or non-square matrices. Furthermore, it can be applicable for computing MHBN.

In the following we state in short about RP algorithm.

Random path method: The permanent of an n×n matrix $A = (a_{ij})$ is defined by:

$$Per(A) = \sum_{\gamma} \prod_{i=1}^{n} a_{i\gamma(i)} \tag{5}$$

where, γ ranges over all permutations of {1, 2, …, n}. The definition of the permanent looks similar to the definition of the determinant of the matrix, but it is harder to compute. Using the Gauss elimination method, we convert the original matrix into an upper triangular matrix, as a result the determinant of the original matrix is equal to the product of the diagonal elements of the converted one and this property makes the computation of the determinant easier. There is no such property in the case of the permanent computation.

Consider an n×n matrix $A = (a_{ij})$ with entries 0, 1 Define:

$$M_j = \{i: a_{ij} = 1, 1 \le i \le n\}, j = 1, 2, \ldots, n.$$

Namely, M_j is the set of the row indices of nonzero elements in the j th column of the matrix A.

Definition 1: Liang *et al*. (2007) A permutation $\gamma = (i_1, i_2, \ldots, i_n)$ is called random path of the matrix A, if there exists $1 \le j \le n$, such that $a_{i_t t} = 1$ for t = 1, 2, …, j and $M_{j+1} \backslash \cup_{t=1}^{j} \{i_t\} = \varnothing$. Random path with j = n is called a feasible path; while that with j < n is called a broken path.

Definition 2: Liang *et al*. (2007) The path value of a feasible path $\gamma = (i_1, i_2, \ldots, i_n)$ is defined as $M_\gamma = |M_1||M_2\backslash\{i_1\}|\ldots|M_n|\backslash\{i_1, i_2, \ldots, i_{n-1}\}$ where $|M_i|$ is the number of the elements in the set M_i.

Here is the RP algorithm.

Algorithm 1: Liang *et al*. (2007) Random Path algorithm (RP).

Input: A- an n×n 0-1 matrix, M = 1, I = ∅.

Output: X_A- the estimate for Per(A).

Step 1: For j = 1 to n

- Select c_j such that $|M_{cj} \backslash \cup_{t=1}^{j-1} \{a_t\}| = \min_{s \in \{1, \ldots, n\} \backslash I} \{|M_s \backslash \cup_{t=1}^{j-1} \{at\}|\}$; $I = I \cup \{c_j\}$
- Choose a_j from $Mcj \backslash \cup^{j-1}_{t=1} \{a_t\}$ uniformly at random; $M = M|M_{cj} \backslash \cup^{j-1}_{t=1} \{a_t\}|$.
- End

Step 2: Generate $X_A = M$.

General Random Path Algorithm (GRPA): To establish our algorithm we first present some necessary concepts.

Definition 3: Let $A = (a_{ij})$ any complex matrix of degree n×m, where $1 \le m \le n$, let $k = (k_1, k_2, \ldots, k_m) \in N^m$ where $\sum_{i=1}^{m} k_i = n$. We define the permanent of the matrix A with respect to the vector k as the following:

$$Per(A) = \sum_{\gamma} \prod_{i=1}^{m} \prod_{j=1}^{k_i} a_{\gamma_j^{(i)} i} \tag{6}$$

where, $\gamma = (\gamma^{(1)}, \gamma^{(2)}, \ldots, \gamma^{(m)})$ with $\gamma^{(i)} = (\gamma^{(i)}_1, \gamma_2^{(i)}, \ldots, \gamma^{(i)}_{ki})$ ranges over all permutations of {1, 2, …, n} such that $\gamma^{(i)} \cap \gamma^{(j)} = \varnothing$ for all $i \ne j, 1 \le i, j \le m$. If m > nthen for suitable vector k we define Per(A) as the permanent of its transpose.

We find that the permanent of a square matrix is a special case from Definition 3 with m = n and k = (1, 1, … ,1). Furthermore, this definition is consistent with the definition of m-MHBN, B_m, so B_m is considered as a special permanent as following: Let A = D the degree matrix with $k = (k_1, k_2, \ldots, k_m)$ the vector that holds the cardinalities of the partition sets of the variables, so B_m = Per(D, k).

Consider an n×m complex matrix $A = (a_{ij})$. Define:

$$M_j = \{i: a_{ij} \ne 0, 1 \le i \le n\}, j = 1, 2, \ldots, m.$$

Namely, M_j is the set of the row indices of nonzero elements in the j th column of the matrix A. Let $k = (k_1, k_2, \ldots, k_m) \in N^m$ with $\sum_{i=1}^{m} k_i = n$. For s > m, $M_s = \varnothing$.

Definition 4: A permutation $\gamma = (\gamma^{(1)}, \gamma^{(2)}, \ldots, \gamma^{(m)})$ of {1, 2, … , n} with $\gamma^{(i)} = (\gamma^{(i)}_1, \gamma^{(i)}_2, \ldots, \gamma^{(i)}_{ki})$ is called a general random path of the matrix A, if there exists $1 \le j \le m$, such that $a_{\gamma_t^{(i)} i} \ne 0$ for t = 1, 2, … , k_i, i = 1, … , j and $M_{j+1} \backslash \{\gamma^{(i)}_t: t = 1, \ldots, k_i, i = 1, \ldots, j\} = \varnothing$. Random path with j = m is called a feasible path; while that with j < m is called a broken path.

Definition 5: The path value of a feasible path $\gamma = (\gamma^{(1)}, \gamma^{(2)}, \ldots, \gamma^{(m)})$ with $\gamma^{(i)} = \left(\gamma_1^{(i)}, \gamma_2^{(i)}, \ldots, \gamma_{k_i}^{(i)}\right)$ is defined as $s = \prod_{i=1}^{m}\left(\binom{s_i}{k_i}\prod_{j=1}^{k_i} a_{\gamma_j^{(i)} i}\right)$ where $s_i = |M_i \backslash \{\gamma^{(r)}_t : t = 1, \ldots, k_r, r = 1, \ldots, i-1\}|$ and $\binom{s_i}{k_i} = \frac{(s_i)!}{(k_i)!(s_i - k_i)!}$.

For the illustration of the previous concepts we consider the following example.

Example 2: Consider the following matrix:

$$A = \begin{pmatrix} 2 & 0 & 1 \\ -1 & 5 & -4 \\ 0 & 1 & 0 \\ 3 & 0 & 2 \\ 1 & -2 & 1 \end{pmatrix}$$

Suppose we need to find Per(A, k) with k = (2, 2, 1) then by the definitions 4, 5 the following are two general random paths and their values:

- $(\gamma^{(i)}_1, \gamma^{(1)}_2, \gamma^{(2)}_1, \gamma^{(2)}_2, \gamma^{(3)}_1) = (1, 5, 2, 3, 4)$ and its value is:

$$\left(\binom{s_1}{k1} a_{\gamma_1^{(1)}1} a_{\gamma_1^{(1)}1}\right)\left(\binom{s_2}{k_2} a_{\gamma_1^{(2)}1} a_{\gamma_1^{(2)}1}\right)\left(\binom{s_3}{k_3} a_{\gamma_1^{(3)}1}\right)$$
$$= \left(\binom{4}{2}(2)(1)\right)\left(\binom{2}{2}(5)(1)\right)\left(\binom{1}{1}(2)\right)$$
$$= [(6)(2)(1)][(1)(5)(1)][(1)(2)]$$
$$= 120$$

- $(\gamma^{(1)}_1, \gamma^{(1)}_2, \gamma^{(2)}_1, \gamma^{(2)}_2, \gamma^{(3)}_1) = (1, 2, 3, 5, 4,)$ and its value is:

$$\left[\binom{s_1}{k_1} a\gamma_1^{(1)} a_{\gamma_2^{(1)}1}\right]\left[\binom{s_2}{k_2} a_{\gamma_1^{(2)}1} a_{\gamma_2^{(2)}1}\right]\left[\binom{s_3}{k_3} a_{\gamma_1^{(3)}1}\right]$$
$$= \left[\binom{4}{2}(2)(-1)\right]\left[\binom{2}{2}(1)(-2)\right]\left[\binom{1}{1}(2)\right]$$
$$= [(6)(2)(-1)][(1)(1)(-2)][(1)(2)]$$
$$= 48$$

In the following we establish our algorithm.

Algorithm 2: General Random Path Algorithm (GRPA).
Input: -A -an n × m complex matrix:

$k = (k_1, k_2 \ldots, k_m) \in \mathbb{N}^m$ with $\sum_{i=1}^{m} k_i = n$

$N \in \mathbb{N}$ and M = 0

Output: $E_{(A, k)}$ - the estimate for Per(A, k).
Step 1: Rearrange the columns of A such that:

$$|M_1| \le |M_2| \le \ldots \le |M_m|$$

Step 2: For i = 1 to N
For j = 1 to m

- Choose $\{\gamma^{(j)}_1, \ldots, \gamma^{(j)}_{k_j}\}$ from $M_j \backslash \{\gamma^{(r)}_w : w = 1, \ldots, k_r, r = 1, \ldots, j-1\}$ uniformly at random.
- Set $a_j = 0$, then $a_j = \prod_{t=1}^{k_j} a_{\gamma_t^{(j)} j}$
- Set $b_j = 0$, then $b_j = \binom{s_j}{k_j}$ where s_j as in Definition 5.

End

$$M = M + \prod_{j=1}^{m} a_j b_j$$

End

M = M/N

Step 3: $E_{(A, k)} = M$

Proposition 1: Algorithm 2 requires n×m + n + 5 m + 2 memory variables, which can be reduced to n×m + n + 2 m + 5.

Proof: A is an array of n×m dimension, the vector $\gamma = (\gamma^{(1)}, \gamma^{(2)}, \ldots, \gamma^{(m)})$, with $\gamma^{(i)} = (\gamma^{(i)}_1, \gamma^{(i)}_2, \ldots, \gamma^{(i)}_{k_i})$ for i = 1, 2, … , m, is of dimension $k_1 + k2 + \ldots + k_m = n$, $(k_1, k_2, \ldots, k_m)$ is of m dimension. Further, $(s_1, s_2, \ldots, s_m)$, $(a_1, a_2, \ldots, a_m)$ and $(b_1, b_2, \ldots, b_m)$ have a total of 3m memory variables, in this case we can use just 3 variables s, a and b instead of 3m variables by accumulating the product in the second 'For' loop of the algorithm. Furthermore we have m+2 variables $(M_1, M_2, \ldots, M_m)$, M and N. Therefore we have in total n×m + n + m + 3 m + m + 2 n = m×n + 5 m + 2 memory variables which can be reduced to n × m + n + 2 m + 5 memory variables.

We note that in the worst case m = n, Algorithm 2 requires $n^2 + 3n + 5$ memory variables.

RESULTS AND DISCUSSION

Numerical results are given for twelve different polynomial systems ranges from size 10 up to 20. For each system we pick up 100 variable partitions, taken at random. The algorithm 2 (GRPA) is used to estimate their MHBNs. The algorithm is implemented for two levels of accuracy N = 100 and N = 1000. The results are compared with exact solutions, which are computed firstly using REM. The algorithms are implemented in MATLAB and executed on a personal computer with Intel (R) Pentium (R) Dual CPU E2160 1.80 GHz CPU and 2.00 GB of RAM.

We considered four types of polynomial systems as following: cyclic n-roots problem (Björck and Fröberg, 1991), economic n problem (Morgan, 1987), noon n problem (Noonberg, 1989) and reduced cyclic n problem (Verschelde, 1996).

The Table 3, 4, 5, 6, 7, 8, 9 and 10 show the percentages that relative errors of the computational results fall into in the specific ranges. Each table shows the results for three systems of sizes 10, 15 and 20. The last column of each table, which is denoted by '%' represents the percentage of the PC time using GRPA to the PC time using REM.

The Table 3 and 4 are for the cyclic n polynomial system; they are for two accuracy levels N=100, 1000 respectively, while the Table 5 and 6 are for the economic n polynomial system, which are for two accuracy levels N=100, 1000, respectively. The results of the noon n polynomial system are in the Table 7 and 8 for two levels of accuracy, N = 100,1000. Finally, the Table 9 and 10 are for the reduced cyclic n polynomial system for two accuracy levels N = 100,1000 , respectively.

We have three notes here. Firstly, in the case of the size of the system being small, say 10, using GRPA with accuracy N = 1000, we note that the consuming time by GRPA is bigger than that of REM, refer to the first row in the Table 4, 6, 8 and 10, but sometimes in small systems it is enough to take the level of accuracy as N = 100, refer to the first row in the Table 3, 5, 7 and 9.

Secondly, for those systems with size larger than 10 say 15, 20 as in our examples, the consuming time by GRPA is much smaller than that of REM even with using high accuracy N = 1000, refer to the second and third rows in the Table 3, 4, 5, 6, 7, 8, 9 and 10.

Thirdly, for large systems, we note that the time consumed by GRPA becomes smaller and smaller compared with that of REM as system size increases, refer to the last column in the Table 3, 4, 5, 6, 7, 8, 9 and 10. This column reflects the percentage of GRPA Time to REM Time.

Table 3: GRPA results for cyclic n problem with N=100

|Relative error |<							
n	0.05	0.10	0.15	0.20	GRPA time	REM time	%
10	55	86	97	100	20s	43 s	46.51
15	43	85	98	100	31s	15 m 12 s	3.40
20	45	76	92	99	43s	6 h 13 m 34 s	0.19

Table 4: GRPA results for cyclic n problem with N=1000

|Relative error |							
n	0.05	0.10	0.15	0.20	GRPA time	REM time	%
10	100	100	100	100	3 m 26 s	43 s	479.07
15	93	100	100	100	5 m 9 s	15 m 12 s	33.88
20	94	100	100	100	7 m 6 s	6 h 13 m 34 s	1.90

Table 5: GRPA results for economic n problem with N=100.

|Relative error |							
n	0.05	0.10	0.15	0.20	GRPA time	REM time	%
10	55	80	95	100	20 s	40 s	50.00
15	43	83	93	100	31 s	17 m 51 s	2.89
20	45	76	92	99	41 s	5 h 20 m 30 s	0.21

Table 6: GRPA results for economic n problem with N = 1000.

|Relative error |							
n	0.05	0.10	0.15	0.20	GRPA time	REM time	%
10	92	94	100	100	3 m 11 s	40 s	477.50
15	90	96	100	100	5 m	17 m 51 s	28.01
20	89	95	100	100	6 m 36 s	5 h 20 m 30 s	2.06

Table 7: GRPA results for noon n problem with N = 100.

|Relative error |							
n	0.05	0.10	0.15	0.20	GRPA time	REM time	%
10	66	94	99	100	24 s	43 s	55.81
15	53	92	99	100	34 s	16 m 29 s	3.44
20	51	85	99	100	44 s	5 h 37 m 48 s	0.22

Table 8: GRPA results for noon n problem with N=1000.

|Relative error |							
n	0.05	0.10	0.15	0.20	GRPA time	REM time	%
10	99	100	100	100	3 m 44 s	43 s	520.93
15	98	100	100	100	5 m 36 s	16 m 29 s	33.97
20	98	100	100	100	7 m 10 s	5 h 37 m 48 s	2.12

Table 9: GRPA results for reduced cyclic n problem with N = 100.

|Relative error |							
n	0.05	0.10	0.15	0.20	GRPA time	REM time	%
10	56	82	98	100	21 s	40 s	52.50
15	43	81	94	98	32 s	15 m 6 s	3.53
20	41	74	94	98	44 s	5 h 56 m 19 s	0.21

Table 10: GRPA results for reduced cyclic n problem with N = 1000

|Relative error |<							
n	0.05	0.10	0.15	0.20	GRPA time	REM time	%
10	100	100	100	100	3 m 28 s	40 s	520.00
15	95	100	100	100	5 m 12 s	15 m 6 s	34.43
20	90	100	100	100	7 m 13 s	5 h 56 m 19 s	2.03

The numerical result shows that with GRPA we can save much time compared with REM. The efficiency increases exponentially as the system size grows; for example, for a system of size n = 10, GRPA estimates MHBN in about 53% of the time needed by REM which is the best of the recent methods; while for a system of size n = 20 we need just about 2% of that of REM. Consequently, the speed up of GRPA implementation becomes higher and higher as the system size grows.

Further, in the worst case when m = n, GRPA requires only $n^2 + 3n + 5$memory variables, this is by Proposition 1, while REM requires 2^n memory variables, in other words, the memory storage needed by REM increases exponentially while that of GRPA increases in polynomial time. Therefore GRPA is an efficient tool for handling the computation of MHBNs for large systems and when REM becomes intractable or impossible.

CONCLUSION

A General Random Path Algorithm (GRPA) for approximating the permanent of a general complex matrix is proposed in this study. The GRPA is constructed as a general algorithm (method) over RP algorithm. RP algorithm is used for approximating the permanent of square 0-1 matrices while GRPA estimates the permanent of square or non-square complex matrices.

The new algorithm has been applied successfully in a new application field, that is the computation of MHBN, which is considered as a special case of the general permanent. The large amount of numerical results in the previous section shows considerable accuracy of estimating MHBN. The method is flexible in controlling the accuracy. Further, the presented proposition proves analytically the effectiveness and the efficiency of our method over some recent methods.

ACKNOWLEDGMENT

All praise is due only to ALLAH, the lord of the worlds. Ultimately, only ALLAH has given us the strength and courage to proceed with our entire life. We would like to thank the Ministry of Higher Education, Malaysia for granting us FRGS Vot. 78521 and Vot. 78215. We also would like to thank Hudramout University of Science and Technology for their generous support.

REFERENCES

Bawazir, H.M.S. and A. Abd Rahman, 2010. A tabu search method for finding minimal multi-homogeneous bézout number. J. Math. Stat., 6(2): 105-109.

Björck, G. and R. Fröberg, 1991. A faster way to ount the solutions of inhomogeneous systems of algebraic equations with applications to cyclic n-roots. J. Symbolic Comp., 12: 329-336.

Liang, H., S. Linsong, B. Fengshan and L. Xiaoyan, 2007. Random path method with pivoting for computing permanents of matrices. Appl. Math. Comput., 185: 59-71.

Malajovich, G. and K. Meer, 2007. Computing multi-homogeneous bézout numbers is hard. Theory Comput. Syst., 40: 553-570.

Morgan, A.P., 1987. Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems. 1st Edn., Prentice-Hall, Englewood Cliffs, New Jersey, pp: 148.

Noonberg, V.W., 1989. A neural network modeled by an adaptive Lotka-Volterra. SIAM J. Appl. Math., 49: 1779-1792.

Rasmussen, L.E., 1994. Approximating the permanent: A simple approach. Random Struct. Algorithms., 5: 349-361.

Verschelde, J., 1996. Homotopy continuation methods for solving polynomial systems using continuation for solving polynomial systems. Ph.D. Thesis. Katholieke Universiteit Leuven, pp: 178, Retrieved from: http://proquest.umi.com/pqdweb?did=739104711&sid = 2&Fmt = 2&clientId = 21690&RQT = 309&Vname = PQD,

Wampler, C.W., 1992. Bézout number calculations for multi-homogeneous polynomial systems. Appl. Math. Comput., 51: 143-157.

Yan, D., J. Zhang, B. Yu, C. Luo and S. Zhang, 2008. A genetic algorithm for finding minimal multi-homogeneous Bézout number. pp: 301-305.Proceeding of the 7th IEEE/ACIS International Conference on Computer and Information Science, IEEE Computer Society.