Using Metaheuristic Computations to Find the Minimum-Norm-Residual Solution to Linear Systems of Equations

Rodrigo S. Jamisola, Jr.[†], Elmer P. Dadios[†], and Marcelo H. Ang, Jr.[‡]

***Abstract*—This work will present metaheuristic computations, namely, probabilistic artificial neural network, simulated annealing, and modified genetic algorithm in finding the minimum-norm-residual solution to linear systems of equations. By demonstrating a set of input parameters, the objective function, and the expected results solutions are computed for determined, overdetermined, and underdetermined linear systems. In addition, this work will present a version of genetic algorithm modified in terms of reproduction and mutation. In this modification, every reproduction cycle is performed by matching each individual with the rest of the individuals in the population. Further, the offspring chromosomes result from crossover of parent chromosomes without mutation. The selection process only selects the best fit individuals in the population. Mutation is only performed when the desired level of fitness cannot be achieved, and all the possible chromosome combinations were already exhausted. Experimental results for randomly generated matrices with increasing matrix sizes will be presented and analyzed. It will be the basis in modeling and identifying the dynamics parameters of a humanoid robot through response optimization at excitatory motions.**

***Index Terms*—metaheuristic computation, probabilistic artificial neural network, simulated annealing, modified genetic algorithm, linear systems of equations**

I. INTRODUCTION

Metaheuristic computations are gaining wide acceptance through many fields of specialization because of their ability to find near-optimum solutions to seemingly unsolved problems using traditional mathematical techniques. Because of this, many computational applications found workable solutions which would have been almost impossible to find or impractical to implement using non-heuristic approaches. This work will present three metaheuristic methods, namely, probabilistic artificial neural network, simulated annealing, and modified genetic algorithm. All three methods will be used to solve the same set of randomly generated inputs to linear systems of equations and their computational performance are compared. Probabilistic artificial neural network is chosen because if its faster computation especially at more complicated systems compared to the other two methods, simulated annealing because of its ease of implementation, and genetic algorithm because of its high rate of convergence.

[†]Dept. of Electronics and Communications Engineering and Dept. of Manufacturing Engineering and Management, De La Salle University - Manila, 2401 Taft Ave, 1004 Manila, Philippines {jamisolar,dadiose}@dlsu.edu.ph.

[‡]Dept. of Mechanical Engineering, National University of Singapore, 9 Engineering Dr 1, Singapore 117576 mpeangh@nus.edu.sg.

This work is supported by the University Research Council of De La Salle University - Manila.

Artificial neural network (ANN) has addressed a wide range of real-world problems including autoregressive moving-average model model parameter estimation [1], identification of linear discrete time systems [2], augmentation of controller for underwater vehicles [3], 2D pattern recognition problems [4], and linear systems of equations [5] including time-varying systems [6] and linear and quadratic programming [7]. The ANN results presented here varies from [5] in that a probabilistic approach is used when ANN does not converge to the desired minimum residual value when a preset number of iterations is reached. Probabilistic neural networks applications include pattern [8] and power quality classification [9], volume segmentation in brain MR images [10], face recognition/decision [11], allocation of power loads [12], and freeway incident detection [13].

Computational applications of genetic algorithm (GA) include traffic engineering optimization [14], learning and structuring of artificial neural network [15], linear array synthesis problem [16], optimizing reactive power planning [17], optimizing piecewise linear function [18], solving linear bilevel programming [19], and automated linear modeling[20]. In this work, when the level of fitness has not reached the desired value but all possible chromosome combinations are exhausted, a probabilistic mutation process is performed. Some applications in probabilistic approach to genetic algorithm include alarm processing [21], optimal scheduling [22], tagging model [23], and in embedded systems [24].

Simulated annealing (SA) was first used to optimize NP-complete (nondeterministic polynomial time complete) problems including the physical design of computers such as partitioning, component placement, and wiring of electronic systems [25]. Then simulated annealing found many wide-ranging area of applications that expanded the usefulness of this optimization technique. Among many of these applications include transmission system planning [26], combination with genetic algorithm to solve some NP-hard problems [27], atmospheric correction of hyperspectral data over dark surfaces [28], EEG source localization [29], flow model application [30], and bioclustering of gene expression data [31]. Swarm behavior that is hybrid with simulated annealing is shown in [32].

The contribution of this work lies in the analysis of the convergence and computational complexity of each of the methods presented, together with the comparison between these methods based on the analysis presented. A set of inputs are specified together with the desired outputs. Then the

objective function is defined as the norm of the residual such that the iteration process will proceed to minimize it. Random vectors and matrices are generated and the corresponding solutions are computed. The sizes of the matrices are varied from square to non-square matrices, with increasing row or column, in order to deal with determined, overdetermined, and underdetermined linear systems of equations. The results are compared against QR factorization.

Another contribution of this work is the modification of genetic algorithm. This modification affects the selection, reproduction, and mutation process of the usual method of computing genetic algorithm. Normally, reproduction is performed from selected parents until the desired new population size is achieved. The derived offspring chromosomes are the results of crossover as well as mutation. Then the best fit individual are more likely to get selected in the selection process.

In the modified genetic algorithm presented in this work, a single reproduction cycle is performed by matching each individual to the rest of the population. The chromosomes of the resulting offsprings are derived via crossover from the chromosomes of both parents, without mutation. The resulting population after one reproductive cycle, that is, parents plus offsprings, are ranked according to their fitness level. The best fit individuals are selected and the rest of the individuals are discarded. Thus there is a possibility of chromosome homogeneity, such that no further advancement of the population will occur after a number of reproduction processes. When chromosome homogeneity approaches, mutation is introduced to create the necessary diversification of the chromosome combination in order to attain the desired population advancement. No comparison is performed between the unmodified genetic algorithm and the modified one, because this is not within the scope of this work. Although, it may be more logical to do this comparison, this work only compares the modified genetic algorithm against artificial neural network and simulated annealing.

Section II presents an overview of the three methods discussed. This is followed by the detailed discussion for each of the methods. Probabilistic artificial neural network is presented in Section III, modified genetic algorithm in Section IV, and simulated annealing in Section V. Results are shown in Section VI and are analyzed. Finally, Section VII gives the summary and conclusion of this work.

II. OVERVIEW

An overview of each of the methods presented in this work is shown in this section to give an introductory discussion before showing the full details of implementation. In solving the $\mathbf{Ax} = \mathbf{b}$ problem, the metaheuristic approach is to iteratively compute for the unknown value of $\mathbf{x}$ such that resulting computed value for $\mathbf{b}$ is close to its desired value by a minimum-residual norm.

A. Overview of Probabilistic Artificial Neural Network

Artificial neural network, being a function mapping computational approach, does not give too much emphasis on how the network is implemented but rather on getting the desired output from the given set of inputs. In general, the neural network implementation can be treated as a black box.

This work will try to move away from the traditional computational approach of the ANN by attempting to find the appropriate values of the weights, which is the unknown vector $\mathbf{x}$, in order to arrive at a desired solution of $\mathbf{Ax} = \mathbf{b}$. Matrix $\mathbf{A}$ and the desired output vector $\mathbf{b}$ will remain constant all throughout the computation. This method will use back-propagation technique in ANN to adjust the weights values $\mathbf{x}$ at every computational cycle. The idea therefore is to find the appropriate stepping strategy such that when the values of the weights are adjusted at every iteration, the resulting computation move closer to the desired output. The correct values of the weights $\mathbf{x}$ are declared to have found when the computed value of $\mathbf{Ax}$ is close enough to the desired input value $\mathbf{b}$ within the required tolerance ε.

A maximum number of iterations is set such that if the desired ε is not achieved up to this value, a random walk from the final output is performed to start a new iteration. The random walk is the method of escaping from a local minimum in the ANN computation presented in this work.

B. Overview of Modified Genetic Algorithm

The same $\mathbf{Ax} = \mathbf{b}$ problem is considered using genetic algorithm. An added input information on the number of individuals p in the initial population is required. The number of individuals in the population will be responsible for the reproductive process in order to find the individual whose chromosome combination results in the minimum (or maximum) fitness function at the required tolerance ε. The technique will compute the resulting floating point numbers directly by searching the solution space through possible floating point chromosome combinations for every reproduction, mutation, and selection process.

The GA presented in this work is modified from the original GA in that it deals with floating point numbers instead of binary numbers in composing its chromosome combinations. In addition, parents are matched in such a way that each individual in the population will have a chance at pairing with the rest of the population. This is performed at every mating cycle. Thus at the end of a mating cycle, all the possible chromosome combinations for the current population were already exhaustively searched, and the most fit individuals were identified. Further mating within the population will not yield any better fit individual. This mating strategy is an added novelty to the modified GA presented in this work.

If no solution was found, there is therefore a need to alter the chromosome combinations of the current population. This is achieved by introducing random mutation to the most fit p individuals in the population. The rest of the individuals will be discarded. A new mating cycle is then performed with chromosome combinations mutated within the vicinity of the best fit individual's chromosome combination. The described process will be repeated until the best fit individual or the solution is found.

The individual that is the solution has a chromosome combination with a corresponding fitness function that lies

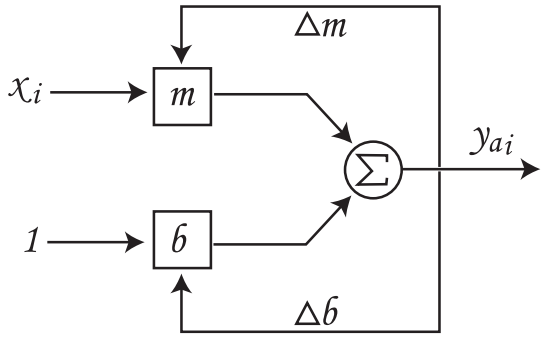

Fig. 1. The artificial neural network model that is used to solve for the constants of the line equation $\mathbf{y} = m\mathbf{x} + b$. The output y_{ai} is computed given the input x_i and 1, and the initial values of m and b. At every iteration k, the weights are updated such that $m^{k+1} = m^k + \Delta m$, $b^{k+1} = b^k + \Delta b$ where $\Delta m = \eta x_i(y_{di} - y_{ai})$ and $\Delta b = \eta(y_{di} - y_{ai})$.

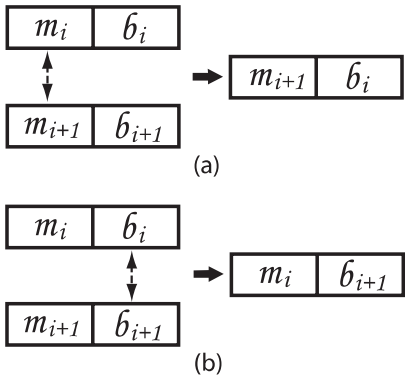

Fig. 2. Two parents of distinct chromosome combination $\{m_i, b_i\}$ and $\{m_{i+1}, b_{i+1}\}$ will result into two offsprings of chromosomes $\{m_{i+1}, b_i\}$ and $\{m_i, b_{i+1}\}$. This is performed by swapping the m's and the b's from both parents one at a time as shown in subfigures (a) and (b).

within the required value ε. In the same way as probabilistic ANN, a critical part in the computation of modified GA is the selection of the appropriate stepping strategy that will result in the advancement of the population.

C. Overview of Simulated Annealing

Simulated Annealing is based on the metallurgical concept of heating followed by slow cooling of metal allowing its lattice structure to rearrange and relieve stress, resulting into a more ductile material. The physical model of slowly decreasing temperature provides a way of randomly searching through the solution space, allowing a sacrifice of moving away from a seemingly better solution in order to move closer to the global optimum. This physical model provides a framework for optimization of more complex systems.

This optimization method will attempt to find the unknown vector $\mathbf{x}$, that will result into the closest value possible to the desired solution of $\mathbf{Ax} = \mathbf{b}$. The computation will step randomly in the solution space and compare the value of the objective function, which is the norm of the residual value. If the new objective function moves closer to the final solution, then the new solution will replace the current solution. However, if the new solution results into a higher norm of the residual value, this is not always discarded. In fact, this is checked against the the Metropolis criterion [33] such that the higher the temperature value, the higher is the probability of acceptance of the new solution with the higher residual value, as the computation searches through the solution space for the global optimum.

The Metropolis criterion which is compared against a random number creates a decision mechanism for the acceptance or non-acceptance of the new solution with a higher residual value than the current solution. This allows the local sacrifice of moving away from a local minimum for a future gain of moving towards the global solution.

III. PROBABILISTIC ARTIFICIAL NEURAL NETWORK

To present the method of solution discussed in this work, we first present the simple problem of solving for the unknowns for the equation of a line $\mathbf{y} = m\mathbf{x} + b$. This linear equation has a set of paired inputs $\{x_1, y_1\}, \{x_2, y_2\}, \{x_3, y_3\}, \ldots \{x_n, y_n\}$ where n is the sample size. The desired values $\mathbf{y}_d$ is the input vector $\mathbf{y}$. The weights to be adjusted are the corresponding m and b, such that the i-th component of the actual computed value $\mathbf{y}_a$ is $y_{ai} = mx_i + b$. The values of the m and b are adjusted for every iteration k of paired input $\{x_i, y_{di}\}$

$$\begin{bmatrix} m \\ b \end{bmatrix}^{k+1} = \begin{bmatrix} m \\ b \end{bmatrix}^{k} + \eta(y_{di} - y_{a_i}^k) \begin{bmatrix} x_i \\ 1 \end{bmatrix} \tag{1}$$

where η is the learning rate. The subscript i denotes the i-th component of the corresponding vector. A solution is found when the norm $||\mathbf{y}_d - \mathbf{y}_a|| < \varepsilon$. A diagram of the probabilistic ANN model is shown in Fig. 1.

The same approach is extended to the $\mathbf{Ax} = \mathbf{b}$ problem with inputs $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. The vector $\mathbf{x} \in \mathbb{R}^n$ is unknown and is computed. The desired vector $\mathbf{b}_d$ is the same as $\mathbf{b}$, while the vector $\mathbf{b}_a$ is computed. The i-th component of $\mathbf{b}_a$ is determined by

$$b_{ai} = \sum_{j=1}^{n} a_{ij} x_j \tag{2}$$

where $i = 1, \ldots, m$. At each k-th iteration the value of x_j is updated by

$$x_j^{k+1} = x_j^k + \eta a_{ij}(b_{di} - b_{ai}^k). \tag{3}$$

A solution is declared found when the norm $||\mathbf{b}_d - \mathbf{b}_a|| < \varepsilon$. The total number of floating-point operations (flops) for a given m sample size is $n_i[(2n-1)m + 3n]$, where n_i is the number of iterations performed. The first term inside the square brackets is due to the multiplication of $\mathbf{Ax}$, the second term is due to the weights update.

In this work, flop count as defined in [34] is used. One flop is counted for each addition, substraction, multiplication, division, or square root. Counting of flops is a preferred approach because it is more accurate and platform independent, as compared to recording of computer processing time.

IV. MODIFIED GENETIC ALGORITHM

As an initial approach, the genetic algorithm is used to solve for the simple equation of a line $\mathbf{y} = m\mathbf{x} + b$ in order to test its effectiveness in solving a linear equation. A set of input data points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ is specified by the user where n is the size of the input data. An additional information on the number of individuals p in a population is user-defined and is even.

A. Method of Reproduction

From the initial population, iterative reproduction, mutation, and selection are performed to search for the individual with the best set of chromosome combination in order to find a well-suited solution. Let the input $\mathbf{y}$ be the desired output $\mathbf{y}_d$, and the computed value $\mathbf{y}_a$ be the actual output defined as $\mathbf{y}_a = m\mathbf{x} + b$. Given a user-defined paired inputs $\{x_1, y_1\}, \{x_2, y_2\}, \ldots \{x_n, y_n\}$ and population size p, the optimization problem is to search for the best individual $i = 1, \ldots, p$, after iterative reproduction, mutation, and selection where

$$\mathbf{y}_{ai} = m_i\mathbf{x} + b_i \tag{4}$$

such that the corresponding error $||\mathbf{y}_d - \mathbf{y}_{ai}||$ is minimized.

An individual's chromosome will be composed of m and b. During reproduction, two parents of distinct chromosomes will produce two offsprings whose chromosomes result from the combination of chromosomes from both parents. Thus given two parents with chromosomes $\{m_i, b_i\}$ and $\{m_{i+1}, b_{i+1}\}$, the resulting two offsprings will have chromosomes $\{m_{i+1}, b_i\}$ and $\{m_i, b_{i+1}\}$ as shown in Fig. 2. Consequently for exactly the same set of parents, the resulting set of two offsprings will have identical chromosomes as another set of two offsprings from the same parents.

The idea in this method of genetic algorithm is to find two parents of distinct chromosome combination who will produce two offsprings that will result into an advancement of the population, that is towards an individual whose chromosome combination has the optimum solution. Consequently, parents with identical chromosomes will not result in an advancement. And parents with very close relations will result into an incremental advancement.

B. Matching of Parents

The method of pairing for the reproductive process is performed by matching two parents in an "*n choose k*" combination for all p individuals, that is,

$$C_2^p = \binom{p}{2} = \frac{p!}{2!(p-2)!} \tag{5}$$

such that each has a chance of pairing with the rest of the group as shown in Fig. 3. This type of reproductive process allows identification of all the possible combinations of the chromosomes within the population in just one iteration. This is one of the novel contributions of this work.

The resulting number of individuals in the population after each reproductive process is $n_k = 2C_2^p + p$, because each mating of parents will result into two offsprings, added by the number of p parents.

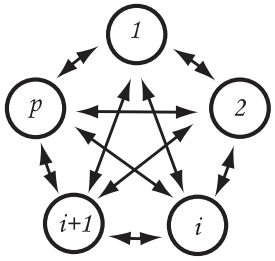

Fig. 3. The "*n choose k*" method of pairing two parents for the reproduction process such that each individual will have a chance of matching with the rest of the group. The index $i = 1, 2, \ldots, p$.

Two other methods of parent-matching were considered. One of the methods considered was random choice of partners. Random individual was picked and was paired with another randomly picked individual. Only one matching per individual was allowed for each reproduction process. Convergence is very slow because the probabilities of matching each individual to the rest of the group is much more limited compared to the "*n choose k*" method. In most cases, the maximum allowable iterations was already reached and no solution was found.

The other method tested was picking out the most fit individual to breed the rest of the population. This method also has very slow convergence because the resulting chromosome combination is limited to the best fit individual at the time of reproduction. In most cases, the best fit individual may become the bottleneck for the advancement of the population, instead of the catalyst for advancement.

C. Selection and Mutation

The fitness function for each i-th individual was calculated using $||\mathbf{y}_{ai} - \mathbf{y}_d||$ where $\mathbf{y}_{ai}$ was computed using Eq. 4. The resulting n_k individuals are sorted from the lowest to highest fitness function value, lowest being the best fit individual of the population. The top p individuals, parents included, are then chosen as new individuals in the population for the next reproduction process. The rest of the individuals are discarded. The issue faced by this method, especially when the best fit individual comes closer to the solution is the problem of homogeneity of the chromosome combination. This means that the values of the m's and the b's for different individuals are very close to each other. The reproductive process would be similar to the reproduction between very close relatives in the population, or in some cases was already a self-reproduction. In these cases, the advancement of the population is very small, or remains constant.

To avoid homogeneity in the population, mutation is performed once the top p individuals are chosen. These individuals are mutated by using the best fit individual's chromosome as the base and modifying it by taking random walks from this chromosome. Eventually, the resulting p individuals have randomly distributed chromosomes from the best fit individual's chromosomes. This is another novel contribution of this

work, in terms of the modification of genetic algorithm. The i-th individual is mutated at each k-th iteration cycle by

$$x_i^{k+1} = x_i^k + \eta \text{ rand() } ||\mathbf{y}_d - \mathbf{y}_{ai}^k|| \tag{6}$$

where rand() is random number generator within the range $[-1,1]$, and η is the step size.

Extending the same procedure to $\mathbf{Ax} = \mathbf{b}$, the user inputs now are $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$, and $\mathbf{x} \in \mathbb{R}^n$ is unknown. That is, the input $\mathbf{b}$ is the desired $\mathbf{b}_d$ and the the actual output $\mathbf{b}_a$ is computed for each i-th individual in the population by

$$\mathbf{b}_{ai} = \mathbf{A}\mathbf{x}_i \tag{7}$$

such that the fitness function $||\mathbf{b}_d - \mathbf{b}_{ai}||$ is minimized. The rest of the processes are identical to the equation of the line described above, except for the reproduction process where the combination of the chromosomes increased.

During the $\mathbf{Ax} = \mathbf{b}$ reproductive process, there are n chromosomes that will be swapped. The total number of individuals after one reproductive cycle is $n_k = \sum_{k=2}^{n} kC_2^p + p$. Thus the total number of flops after one iteration is approximately $n_k[(2n-1)m + n(2m+5)]$. The first term inside the square brackets is due to the multiplication of $\mathbf{Ax}$, and the second term is due to the computation of the fitness function and the update of $\mathbf{x}$. The swapping of chromosomes is not counted as a flop. Compared to probabilistic ANN, the flops for modified GA has almost the same terms, except that the terms in the square brackets are multiplied by n_k, instead of n_i.

V. SIMULATED ANNEALING

The same $\mathbf{Ax} = \mathbf{b}$ problem with randomly generated matrix $\mathbf{A}$ and vector $\mathbf{b}$ is considered using SA. To solve for the next value of $\mathbf{x}$, a random step is taken from its current or actual value $\mathbf{x}_a$, to get a new value $\mathbf{x}_n$,

$$\mathbf{x}_n = \mathbf{x}_a + \eta \text{ rand()} \tag{8}$$

where rand() is a pseudo random number generator and η is the step size. The function of the energy of the system is defined as $\Delta E = ||\mathbf{b}_d - \mathbf{b}_n|| - ||\mathbf{b}_d - \mathbf{b}_a||$ such that the new $\mathbf{x}_n$ replaces $\mathbf{x}_a$ when ΔE is negative. When ΔE is positive this means that the random step resulted in an output value $\mathbf{b}_n = \mathbf{A}\mathbf{x}_n$ that moves farther away from the solution. This new output value is not thrown away immediately, instead a probability function using the Metropolis criterion,

$$P(\Delta E) = \exp(-\Delta E / k_B T) \tag{9}$$

where T is the "temperature" of the material and k_B is the temperature factor, decides whether $\mathbf{x}_n$ can replace $\mathbf{x}_a$. This is what is referred to as a local sacrifice for a future global gain. This is SA's mechanism of escaping local minima. A pseudo random number generator is used such that when $P(\Delta E) >$ rand(), $\mathbf{x}_n$ replaces $\mathbf{x}_a$, otherwise $\mathbf{x}_a$ remains and a new random step is performed to find a new $\mathbf{x}_n$.

The total number of flops used in SA is $n_i[12n(m+1)]$ where n_i is the number of iterations. This is of the same order as that of probabilistic ANN which is $O(mn)$. Due to the size of the matrices considered that is not too large, we can roughly approximate that the number of flops for the SA is six times the number of flops for the probabilistic ANN for the same number of iterations.

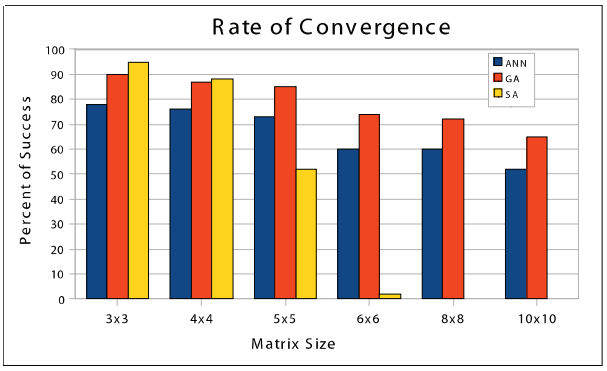

Fig. 4. The rate of convergence of $\mathbf{Ax} = \mathbf{b}$ for the corresponding matrix sizes shown, computed through probabilistic ANN, modified GA, and SA. For each matrix size, 100 matrices were randomly generated and computed for all methods. QR factorization always converged and is not shown.

VI. RESULTS AND ANALYSIS

The linear equation $\mathbf{Ax} = \mathbf{b}$ is solved for randomly generated inputs of matrix $\mathbf{A}$ and vector $\mathbf{b}$, and the unknown vector $\mathbf{x}$ is computed. The same random inputs are used for the three different metaheuristic methods, namely, probabilistic ANN, modified GA, and SA. These results are benchmarked against QR factorization. For random square matrices the matrix sizes are 3×3, 4×4, 5×5, 6×6, 8×8, and 10×10. To show the case for non-square matrices, the column or row size is fixed at 5 while the row or column size is varied from 3 to 10, using the same increment as the square matrices case. This experimental setup will consider determined, underdetermined and overdetermined linear systems with varying matrix sizes. For each matrix size, 100 randomly generated matrices and vectors are used. The random walks for probabilistic ANN is set at 10 such that the maximum number of iterations for non-convergence is 10,000. This is the same maximum number of iterations for SA. For the modified GA, the maximum number of iterations is set at 1,000 which is multiplied by the number of individuals in the population p. Initially p is set at 10.

A. Rate of Convergence

From the 100 randomly generated matrix $\mathbf{A}$ and vector $\mathbf{b}$, the rate of convergence for different square matrix sizes computed through probabilistic ANN, modified GA, and SA are compared against QR factorization and is shown in Fig. 4. A method of computation is said to have successfully found a solution when the 2-norm of the error is less than the desired $\varepsilon = 0.01$.

For the metaheuristic computations their accuracy in finding a solution to determined linear system of equations is always less than 100%. But QR factorization always converged and is not shown in the figure. SA has the highest rate of convergence at around 96% for a 3×3 matrix size but has zero rate

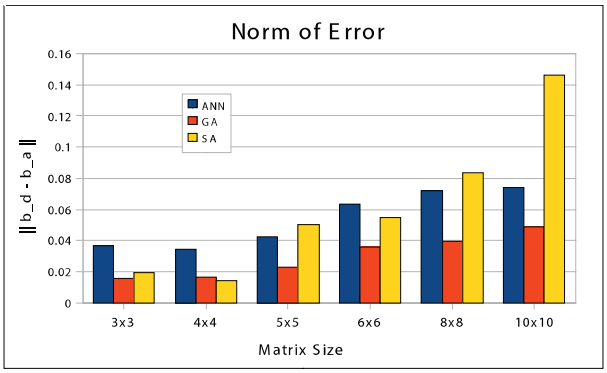

Fig. 5. The average norm of error for the 100 randomly generated matrices for each size shown, computed through probabilistic ANN, modified GA, and SA. Convergence is declared when the norm of error is less than $\varepsilon = 0.01$. The norm of error for QR factorization is always very small and is not shown.

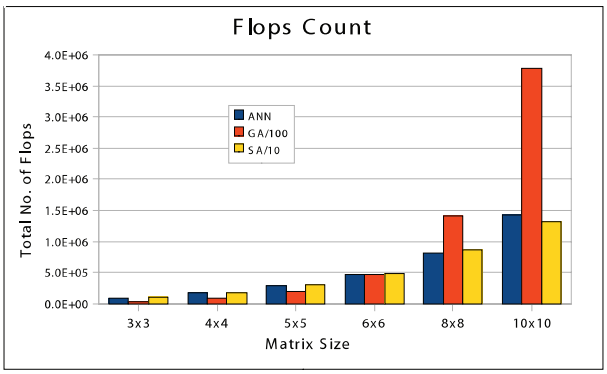

Fig. 6. The average number of floating-point operations (flops) for probabilistic ANN, modified GA, and SA corresponding to each of the matrix sizes shown.

of convergence for matrix sizes 8×8 and 10×10. The maximum rate of convergence for modified GA is at 90% for a 3×3 matrix size and at 65% for a 10×10 matrix size. The probabilistic ANN converges at a lower rate at 78% for a 3×3 matrix size and at 52% for a 10×10 matrix size. Both probabilistic ANN and modified GA, converged even at matrix size 10×10, which showed their ability to handle more complicated systems. Their difference in convergence rate is around 12% and almost constant throughout the different matrix sizes, with modified GA having the higher rate of success. SA on the other hand is highly recommended for much simpler systems but its rate of convergence decreases significantly as complexity of the system being handled increases. Of the three metaheuristic computations, the modified GA showed superior performance and is consistent all throughout the different matrix sizes. Note that the modified GA exhaustively searched through all the possible chromosome combination from the random chromosomes of the initial population.

The norm of error graph is shown in Fig. 5. This has an inverse relationship with the rate of convergence. The shown error norm is the average value for the corresponding method of computation shown, including the non-convergent cases. SA has the highest error norm of the three methods shown because of its non-convergence on matrix sizes 8×8 and 10×10. Note

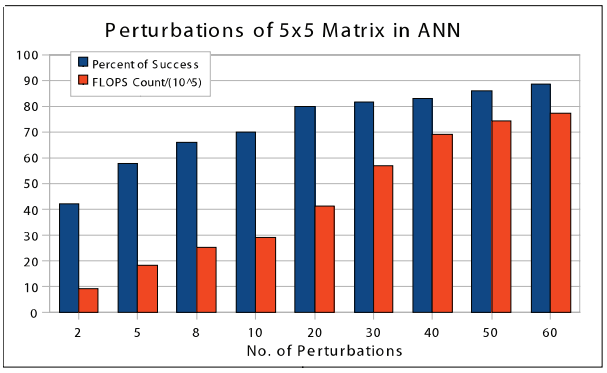

Fig. 7. The rate of convergence and flops count for ANN where the number of perturbations is varied. For each number of perturbations, 100 randomly generated 5×5 matrices are generated and the average data is shown.

that for matrix size 8×8, probabilistic ANN, which has 60% rate of convergence, is close to the error norm of SA, which has zero convergence. This result seemed anomalous at the first glance. However, recall that an error norm of less than tolerance $\varepsilon = 0.01$, is considered convergent. In the case of probabilistic ANN, this simply means that the nonconvergent cases of 40% has higher error norms that pulled the 60% convergent cases to its final average value of around 0.077. For SA, the final average error norm of around 0.081 is a representative of all the nonconvergent cases higher than 0.01.

Computations using modified GA has lower error norm than probabilistic ANN because of the former's higher rate of convergence. This is consistent through all the matrices considered. The difference in the error norm between the two methods slightly increases as the matrix size increases, with probabilistic ANN having the higher error norm. On the other hand, QR factorization always converged. Its error norm is always very small and is not shown in the graph.

The superiority of the modified GA is hampered by the fact the the number of flops involved in computing is much higher than probabilistic ANN and SA. The higher computational expense is inherent to GA. This is more true to the modified GA presented in this work, because of the exhaustive search through the pairing of each individual to the rest of the population. However, this may result into higher rate of convergence as compared to the unmodified GA. In Fig. 6 the numbers showed GA flops/100 to scale down the values of flops for GA, and SA is shown at flops/10. The number of flops for QR factorization is given as $2n^2m - \frac{2}{3}n^3$ [35]. This is negligible compared to the metaheuristic results and is not shown in the graph. Based on these results, one may choose between probabilistic ANN, modified GA, and SA depending on the requirement of computation. If the given system is not very complex and speed of convergence is important, SA may be the recommended method to use. For more complex systems, either the probabilistic ANN or modified GA may be used. Between the two, if speed is of bigger importance, then probabilistic ANN is a recommended. But if accuracy of results is the major concern, then GA is the better choice.

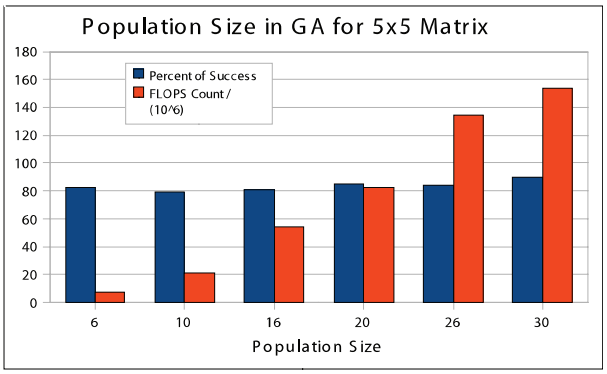

Fig. 8. The population size of GA is varied for a fixed matrix size of 5×5. Population size were chosen to be an even number. Rate of convergence and number of flops are shown, where 100 randomly generated matrices are considered for each population size.

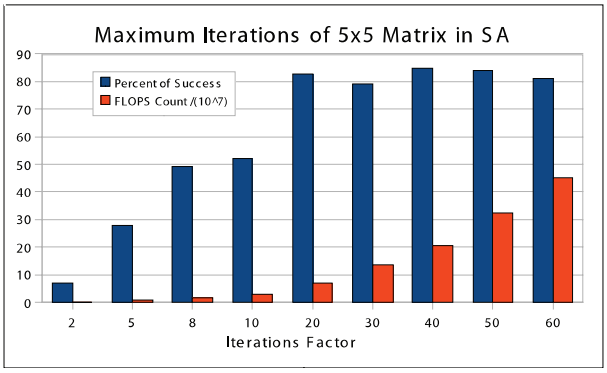

Fig. 9. The rate of convergence and flops count for SA where the maximum number of iterations is varied for a fixed matrix size of 5×5. The base number of iteration is 1,000 and is increased by multiplying the iteration factor.

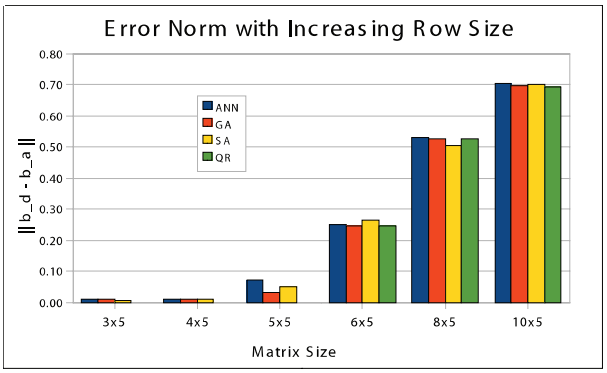

Fig. 10. The norm of error where matrix row size in varied while the column size is fixed at 5. The systems of equations started from an underdetermined system to an overdetermined system as the row size increased.

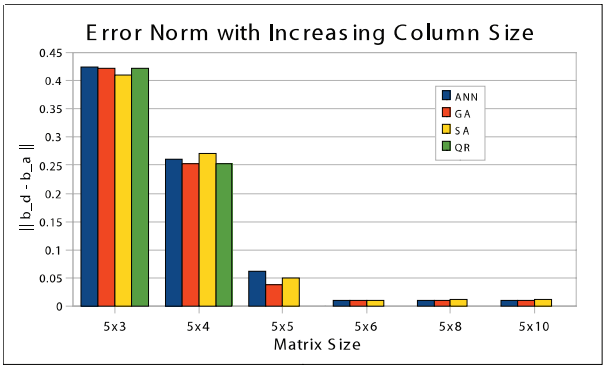

Fig. 11. The norm of error when the column size of the matrix is varied. The row size remains fixed at 5. The linear systems of equations started from an overdetermined system to an underdetermined system as the column size is increased.

B. Number of Iterations vs. Rate of Convergence

In the previous experiment presented, the maximum number of iterations for probabilistic ANN and SA was fixed at 10,000, and modified GA was fixed at 1,000 all throughout the different matrix sizes. We postulated that increasing the maximum number of iterations, for all methods discussed will increase the rate of convergence. An additional experiment was performed to test this postulate to gain an insight on the appropriate value of the maximum number of iterations. The matrix size 5×5 was chosen for this setup. The results of the experiments are shown in Figs. 7-9 where the rate of convergence is plotted together with the number of flops.

For probabilistic ANN shown in Fig. 7 to achieve a rate of convergence of around 80% and become comparable to the modified GA, the maximum number of random walks or perturbations must be increased to 20. A perturbation is performed for every 1,000 iterations, thus the maximum number of iterations at 20 perturbations is 20,000. Its corresponding total number of flops is around 40×10^5 compared to the modified GA which is around five times more. A further increase in the maximum number of iterations would increase the rate of convergence of probabilistic ANN until it reaches around 90% at 60,000. At this points its total flops is around 78×10^5, which is still lower than the modified GA.

The results of varying the population size of modified GA for matrix size 5×5 is shown in Fig. 8. Even at population size of six, its rate of convergence still showed at around 80% and its total flops is around 10×10^6 which is already less than three times that of probabilistic ANN. Increasing further the population size to around 30, with a corresponding flops count of around 150×10^6, does not significantly increase its rate of convergence. In this case, probabilistic ANN proved superior to modified GA.

The results in varying the maximum number of iterations for SA is shown in Fig. 9 with a 5×5 matrix size. The base maximum number of iterations is 1,000 and this is subsequently increased by multiplying it with the iteration factor. The figure showed that by further increasing the maximum number of iterations to 20,000 the rate of convergence will increase to around 85%, as compared to 50% when the maximum iteration is 10,000. With this rate of convergence, the number of flops is around 80×10^6 which is around four times that of the modified GA. The significant decrease in the rate of convergence for SA as the search space becomes bigger is because it lacks the parallel computation that is inherent to GA and ANN. In this case, modified GA proved superior to SA.

C. Underdetermined and Overdetermined Systems

Figs. 10 and 11 showed the values of the norm of the error when the matrices are allowed to be non-square. In this way, the linear system of equations is allowed to become an underdetermined, determined, or overdetermined system. The figures show that for overdetermined systems where convergence to an error norm of less than 0.01 is not possible, the norms of error of the metaheuristic computations are comparable to that of the QR factorization. Thus all four methods of computations outputted the minimum-norm residual solution, which is the best possible solution to the given system of equations. We can therefore state that when a true solution does not exist, a metaheuristic approach approximates the best solution that is comparable to the solution found by a non-heuristic computation.

VII. CONCLUSION AND RECOMMENDATIONS

This work has shown that linear systems of equations for determined, underdetermined, and overdetermined systems can be solved using metaheuristic computations, namely, probabilistic artificial neural network, modified genetic algorithm, and simulated annealing. A new version of genetic algorithm was also presented with a novel method of parent matching and mutation strategy. This work will serve as a tool of understanding the basic methodologies of metaheuristic computations in computing root-finding or optimization problems for a given set of input parameters, objective function, and desired results. The methods of computations searched the minimum-norm residual solution and are compared against QR factorization. For the case of overdetermined systems where no true solution exists, the approximated solutions found are comparable to that of QR factorization. The computations presented will be tested in the actual humanoid robot dynamics parameter identification which will be the continuation of this work.

VIII. ACKNOWLEDGEMENT

We wish to acknowledge the support given by the University Research Council, De La Salle University - Manila.

REFERENCES

[1] K. H. Chon and R. J. Cohen, "Linear and nonlinear ARMA model parameter estimation using an artificial neural network," *IEEE Transactions on Biomedical Engineering*, vol. 44, no. 3, pp. 168–174, March 1997.

[2] O. Sebakhy, H. Kader, W. Youssef, and S. Deghiedi, "Identification of linear discrete time systems using linear recurrent neural networks," in *Proceddings of the IEEE International Symposium on Industrial Electronics*, vol. 1, June 17-20 1996, pp. 374–379.

[3] G. Campa, M. Sharma, A. J. Calise, and M. Innocenti, "Neural network augmentation of linear controllers with application to underwater vehicles," in *Proceedings of the American Control Conference*, vol. 1, no. 6, Chicago, Illinois, U.S.A., June 28-30 2000, pp. 75–79.

[4] C. Perez, G. Gonzalez, and C. Salinas, "Neural versus difference equation modelling for 2d pattern recognition problem," in *2000 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 4, Oct. 8-11 2000, pp. 2851–2856.

[5] Y. Zhang, Z. Li, K. Chen, and B. Cai, "Common nature of learning exemplified by BP and hopfield networks for solving online a system of linear equations," in *IEEE International Conference on Networking, Sensing and Control, 2008*, Apr. 6-8 2008, pp. 832–836.

[6] Y.-N. Zhang and H.-F. Peng, "Zhang neural network for linear time-varying equation solving and its robotics application," in *Proceedings of the Sixth International Conference on Machine Learning and Cybernetics*. Hongkong, China: IEEE, Aug. 19-22 2007, pp. 3543–3548.

[7] X. Wu, Y. Xia, J. Li, and W. Chen, "A high-performance neural network for solving linear and quadratic programming," *IEEE Transactions on Neural Networks*, vol. 7, no. 3, pp. 643–651, May 1996.

[8] L. Rutkowski, "Adaptive probabilistic neural networks for pattern classification in time-varying environment," *IEEE Transactions on Neural Networks*, vol. 15, no. 4, pp. 811–827, July 2004.

[9] W.-B. Hu, K.-C. Li, and D. Zhao, "A novel probabilistic neural network system for power quality classification based on different wavelet transform," in *International Conference on Wavelet Anaylysis and Pattern Recognition*, vol. 2, Nov 2-4 2007, pp. 746–750.

[10] T. Song, M. Jamshidi, R. Lee, and M. Huang, "A modified probabilistic neural network for partial volume segmentation in brain MR image," *IEEE Transactions on Neural Networks*, vol. 18, no. 5, pp. 1424–1432, Sept. 2007.

[11] S.-H. Lin, S.-Y. Kung, and L.-J. Lin, "Face recognition/detection by probabilistic decision-based neural network," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 114–132, Jan. 1997.

[12] D. Gerbec, S. Gasperic, I. Smon, and F. Gubina, "Allocation of the load profiles to consumers using probabilistic neural network," *IEEE Transactions on Power Systems*, vol. 20, no. 2, pp. 548–555, May 2005.

[13] D. Srinivasan, X. Jin, and R. Cheu, "Evaluation of adaptive neural network models for freeway incident detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 5, no. 1, pp. 1–11, Mar. 2004.

[14] V. Pasias, D. Karras, and R. Papademetriou, "Traffic engineering in multi-service networks comparing genetic and simulated annealing optimization techniques," in *Proceedings 2004 International Joint Conference on Neural Networks*, vol. 3, July 25-29 2004, pp. 2325–2330.

[15] H. Nakayama, T. Iwata, and T. Yamauchi, "Learning and structuring of neural networks using genetic algorithm and linear programming," in *Proceedings of 1993 International Joint Conference on Neural Network*, vol. 3, Oct. 25-29 1993.

[16] M. Buckley, "Linear array synthesis using a hybrid genetic algorithm," in *Antennas and Propagation Society International Symposium*, vol. 1, July 21-26 1996, pp. 584–587.

[17] K. L. Lee and F. F. Yang, "Optimal reactive power planning using evolutionary algorithms: A comparative study for evolutionary programming, evolutionary strategy, genetic algorithm, and linear programming," *IEEE Transactions on Power Systems*, vol. 13, no. 1, pp. 101–108, Feb. 1998.

[18] J. Pittman and C. Murphy, "Fitting optimal piecewise linear functions using enetic algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 7, pp. 701–718, July 2000.

[19] G. Wang, X. Wang, Z. Wan, and Y. Chen, "Genetic algorithms for solving linear bilevel programming," in *Proceedings of the Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies*. IEEE Computer Society, Dec. 05-08 2005, pp. 920–924.

[20] P. Flores, C. Anaya, H. M. Ramirez, and L. B. Morales, "Automated linear modeling of time series with self adaptive genetic algorithm," in *Proceedings of International Joint Conference on Neural Network*. Orlando, Florida, U.S.A.: IEEE, Aug 2007, pp. 1389–1396.

[21] F. Wen and C. Chang, "A probabilistic approach to alarm processing in power systems using a refined genetic algorithm," in *Proceedings 1996 International Conference on Intelligent Systems Applications to Power Systems*, Jan. 28 - Feb. 2 1996, pp. 14–19.

[22] C. Srisuwanrat and P. Ioannou, "Optimal scheduling of probabilistic repetitive projects using completed unit and genetic algorithms," in *Winter 2007 Simulation Conference*, Dec. 9-12 2007, pp. 2151–2158.

[23] F. Wong, S. Chao, D.-C. Hu, and Y.-H. Mao, "Interpolated probabilistic tagging model optimized with genetic algorithm," in *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*, vol. 4, Aug. 26-29 2004, pp. 2569–2574.

[24] R. Guo, B. Li, Y. Zou, and Z. Zhuang, "Hybrid quantum probabilistic coding genetic algorithm for large scale hardware-software co-synthesis of embedded systems," in *2007 IEEE Congress on Evolutionary Computation*, Sept. 25-28 2007, pp. 3454–3458.

[25] S. Kirkpatrick, C. Gelatt, Jr., and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[26] R. Romero, R. Gallego, and A. Monticelli, "Transmission system expansion planning by simulated annealing," *IEEE Transactions on Power Systems*, vol. 11, no. 1, pp. 364–369, Feb. 1996.

[27] F. Lin, C. Kao, and C. Hsu, "Applying the genetic approach to simulated annealing in solving some NP-hard problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 6, pp. 1752–1767, Nov./Dec. 1993.

[28] R. Marion, R. Michel, and C. Faye, "Atmospheric correction of hyperspectral data over dark surfaces via simulated annealing," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 44, no. 6, pp. 1566–1574, June 2006.

[29] D. Khosla, M. Singh, and M. Don, "Spatio-temporal EEG source localization using simulated annealing," *IEEE Transactions on Biomedical Engineering*, vol. 44, no. 11, pp. 1075–1091, Nov. 1997.

[30] K.-P. Wong, S. R. Meikle, and D. Feng, "Estimation of input function and kinetic parameters using simulated annealing: Application in a flow model," *IEEE Transactions on Nuclear Science*, vol. 49, no. 3, pp. 707–713, June 2002.

[31] K. Bryan, P. Cunningham, and N. Boshakova, "Application of simulated annealing to the bioclustering of gene expression data," *IEEE Transactions on Information Technology in Biomedicine*, vol. 10, no. 3, pp. 519–525, July 2006.

[32] R. R. Tan, "An adaptive swarm-based simulated annealing algorithm for process optimization," in *Proceedings 11th Conference on Process Integration, Modelling and Optimisation for Energy Saving and Pollution Reduction*, Prague, Czech Republic, 24-28 Aug. 2008 2008.

[33] N. Metropolis, A. Rosenbluth, M. Bosenbluth, A. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, June 1953.

[34] L. N. Trefethen and D. Bau, III, *Numerical Linear Algebra*. PA, U.S.A.: Society of Industrial and Applied Mathematics (SIAM), 1997.

[35] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra, "Parallel tiled qr factorization for multicore architectures," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 3, pp. 1573–1590, June 2008.