MEMETIC ALGORITHMS FOR NEURAL NETWORK TRAINING IN BIOINFORMATICS

Y.G. Petalas and M.N. Vrahatis
Department of Mathematics,University of Patras, GR–26110 Patras, Greece,
University of Patras Artificial Intelligence Research Center(UPAIRC)
Phone:+302610997374, Fax:+302610992965
email:{petalas,vrahatis}@math.upatras.gr

ABSTRACT: Bioinformatics is a new, rapidly growing, scientific area that exploits computational techniques to study DNA and protein sequences. A particularly interesting task in this context is to predict the structure of proteins. Artificial Neural Networks can efficiently handle classification and prediction tasks. On the other hand, Memetic Algorithms belong to the class of heuristic methods that have been developed to address difficult optimization problems. Since training Artificial Neural Networks is a hard optimization problem, Memetic Algorithms can be used to address this task. In this paper we propose a Local-Search-Based Memetic Algorithm, and study its performance as a neural network training method. The resulting Neural Network is applied for the prediction of the cellular localisation sites of two proteins. The performance of the proposed method is compared to that of alternative memetic and global optimization algorithms.

KEYWORDS: Bioinformatics, Memetic Algorithms, Neural Networks, Differential Evolution

1 INTRODUCTION

Modern Biology has rendered it possible to determine and store huge amounts of information about DNA sequences including the amino acid sequences of a diverse set of proteins. Analyzing these proteins is critical because proteins comprise both the "building blocks" and the "architects" of life. Since a protein's structure determines its biological function, the task of protein structure prediction has drawn considerable attention. Solving this problem will contribute to our understanding of the gene expression and it will also facilitate the design of new products, including diagnostic tests and drugs. In this work we examine the clasification problems associated with the E.coli and yeast proteins [1], [2]. A short description of the data sets that we will be used in our analysis follows.

E.Coli Data Set: The objective of this data set is to predict the cellular localisation sites of E.coli proteins [3]. There are 8 different cellular sites, namely, cytoplasm (cp), inner membrane without signal sequence (im), periplasm (pp), inner membrane with uncleavable signal sequence (imU), outer membrane (om), outer membrane lipoprotein (omL), inner membrane lipoprotein (imL) and inner membrane with cleavable signal sequence (imS). The attributes are signal sequence recognition methods (specifically those of McGeoch and von Heijne), the presence of charge on N-terminus of predicted lipoproteins and 3 different scoring functions on the amino acid contents whether predicted as a outer membrane or inner membrane, cleavable or uncleavable sequence signal.

Yeast Data Set: Similarly to the E.coli data set, the objective is to determine the cellular localisation of the yeast proteins [3]. There are 10 different sites, which include: CYT (cytosolic or cytoskeletal); NUC (nuclear); MIT (mitochondrial); ME3 (membrane protein, no N-terminal signal); ME2 (membrane protein, uncleaved signal); ME1 (membrane protein, cleaved signal); EXC (extracellular); VAC (vacuolar); POX (peroxisomal) and ERL (endoplasmic reticulum lumen). The attributes are similar to the E.coli data set with the addition of nuclear localisation information.

To address the classification problems associated with the aforementioned data sets, we employ Artificial Feedforward Neural Networks (FNNs). FNNs are parallel computational models comprised of densely interconnected, simple, adaptive processing units, characterized by an inherent propensity for storing exponential knowledge and rendering it available for use. FNNs resemble the human brain in two fundamental respects; firstly, knowledge is acquired by the network from its environment, through a learning process, and secondly, inter-neuron connection strengths, known as synaptic weights, are employed to store the acquired knowledge [4].

The efficient supervised training of FNNs is the subject of considerable ongoing research and numerous algorithms have been proposed to this end. Supervised training amounts to the global minimization of the network error function E. The rapid computation of a set of weights that minimizes this error is a difficult task since, in general, the number of network weights is large and the resulting error function generates a complex surface in the weight space, characterized by multiple local minima and broad flat regions adjoined to narrow steep ones. As neural network training algorithm, we propose a new Memetic Algorithm (MA). MAs are population–based heuristic search methods for global optimization.

The rest of the paper is organized as follows: Section 2 is devoted to MAs, in section 3 we give a short description of our method, in Section 4 we present experimental results, and the paper ends with concluding remarks in Section 5.

2 MEMETIC ALGORITHMS

Memetic Algoriths (MAs) comprise a family of population–based, heuristic, search algorithms designed to perform global optimization. MAs bear a similarity to genetic algorithms (GAs) [5]. While GAs rely on the concept of biological evolution, MAs mimic cultural evolution. In nature genes, are usually not modified during an individual's lifetime, whereas memes are. Most MAs can be interpreted as a cooperative-competitive strategy of optimizing agents. Their success can be attributed to the synergy of the different search approaches incorporated. Local-Search-based Memetic Algorithms (LS-based MAs) have been successfully applied in combinatorial optimization and especially for the approximate solution of NP optimization problems [6]. For completeness purposes we outline the general description of an LS-based MA as it appears in [6]:

```
Begin
InitializePopulation Pop using FirstPop()
Foreach individual i ∈ Pop do i ← Local-Search-Engine(i)
Foreach individual i ∈ Pop do i ← Evaluate(i)
Repeat

    parfor j = 1 to # recombinations do
    selectToMerge a subset S_par of Pop
    offspring ← Recombine(S_par, x);
    offspring ← Local-Search-Engine(offspring);
    Evaluate(offspring);
    addInPopulation individual offspring to Pop;
    endparfor

    parfor j = 1 to # mutations do
    selectTomutate an individual i ∈ Pop;
    i_m ← Mutate(i);
    i_m ← Local-Search-Engine(i_m);
    i_m ← Evaluate(i_m);
    addInPopulation individual offspring to Pop;
    endparfor

Pop ← SelectPop(Pop);
if Pop has converged then Pop ← RestartPop(Pop);
until termination-condition=True;
End
```

Next we briefly explain the above pseudo-code. InitializePopulation initializes the population with some random values. Local-Search-Engine() receives as input an individual and executes a local search method for a few iterations. The Evaluate() function plays the role of the objective function. After the initial population has been created, the recombination process takes place for selected individuals. The term parfor in the pseudo-code indicates operations that can be executed in parallel.

A new individual is created by recombining the selected individuals according to the Recombine() function. The Mutate() function performs the mutation operation to some individuals of the population. The SelectPop() function chooses the individuals that will survive in the next population. The convergence of the population is determined by the similarity of the individuals in the population. When the population has converged a RestartPop() function is used. In general, the best individual is retained and a new population is created using some randomized method. All individuals in the population are evaluated and the whole process is repeated. The termination-condition can be implemented in various ways, including time-expiration and/or generation-expiration criteria.

From the above description of the LS-based MA we see that it is an evolutionary algorithm that incorporates a local search in each iteration. A number of hybrid algorithms that exploit GAs as an evolution algorithm and a local search at each iteration (GA-LS) have been proposed [7], [8], [9], [10]. The GA-LS hybrid scheme is interesting because the local and the global search methods can influence each other's behavior. An important example of this phenomenon is the Baldwin effect [8], [11] in which learning in natural systems speeds up the rate of evolutionary change. Similar effects have been observed by a number of authors that used GA-LS hybrids [7], [8], [9]. GA-LS hybrid methods have been used for neural network training [7], [10]. In this contribution, we propose an LS-based MA which uses the Differential Evolution (DE) algorithm [12] as the global search method and in each iteration we apply a Random Walk with Direction Exploitation as the local search method.

3 METHOD DESCRIPTION

The proposed algorithm is based on the DE algorithm with the difference that it performs a local search in each iteration before applying the evolutionary operators of DE. As a local search method we use Random Walk with Direction Exploitation. Random walk with Direction Exploitation can be applied even if the objective function is discontinuous and non differentiable. It has been shown to be effective in cases where other methods fail due to local difficulties such as sharply varying functions and shallow regions [13]. Below we give a short description of DE as well as the Random Walk with Direction Exploitation.

3.1 DIFFERENTIAL EVOLUTION

DE [12] is a population-based, direct-search algorithm for the global optimization of multimodal functions. DE exploits a population of individuals, the ith individual denoted as u_i. A description of the algorithm follows:

Step 1. Initialize the individuals of the population with random values.

Step 2. (Mutation Step) Mutate each individual u_i (called the target individual), of the population to form a trial individual v_i, by applying one of the following operators,

$$\begin{aligned}
v_i &= u_{r1} + \alpha(u_{r2} - u_{r3}) \quad (DE1) \\
v_i &= u_{r1} + \alpha(u_{r1} - u_{r2}) \quad (DE2) \\
v_i &= u_{best} + \alpha(u_{r1} - u_{r2}) \quad (DE3) \\
v_i &= u_i + \alpha(u_i - u_{best}) + \alpha(u_{r1} - u_{r2}) \quad (DE4) \\
v_i &= u_{best} + \alpha(u_{r1} - u_{r2}) + \alpha(u_{r3} - u_{r4}) \quad (DE5)
\end{aligned}$$

where α is a parameter assuming values in the interval $[0, 1]$, and $r1, r2, r3, r4$ are random integers satisfying $r1 \neq r2 \neq r3 \neq r4 \neq i \neq best$. The index $best$ is used to represent the individual with the lowest function value in the current population.

Step 3. (Crossover step) For each element of the trial individual, v_i, obtain a random value r in the interval $[0, 1]$. If $r \leqslant \beta$, then the jth component of the trial individual remains unchanged, where $\beta \in [0, 1]$ is called the crossover factor.Otherwise the jth component of the trial vector assumes the value of the jth component of the vector u_i.

Step 4. (Selection step) For each trial individual of the population evaluate its function value. If this value is lower compared to that of the target individual, then the trial individual replaces the target individual in the next generation. Otherwise the target individual is retained in the next generation. Go to step 2.

DE is a very efficient and effective algorithm in practice and like other evolutionary algorithms, it can be easily parallelized [14].

3.2 RANDOM WALK WITH DIRECTION EXPLOITATION

In the proposed algorithm Random Walk with Direction Exploitation is used as a local search algorithm [13]. Random Walk (RW) is an iterative method that generates a sequence of approximations to the minimizer by assuming a random vector as a search direction. Thus, if x_{i-1} is the approximation to the minimizer obtained in the $(i-1)$th iteration, the new value of x in the ith iteration, x_i, is computed through,

$$x_i = x_{i-1} + \lambda u_i$$

where λ is a prescribed scalar step–length, and u_i is a unit–length, random vector. The workings of the method are summarized in the following,

Step 1. Start with an initial point x_1 and a scalar step length, λ. Compute the function value $f_1 = f(x_1)$.

Step 2. Set the iteration number, $i = 1$.

Step 3. Generate a set of n random numbers and use them to form the unit–length random vector u_i.

Step 4. Find the value of the objective function at the point:

$$f = f(x_1 + \lambda u)$$

Step 5. Compare the values of f and f_1. If $f < f_1$, set $x_i = x_1 + \lambda u$, $f_1 = f$ and goto Step 4. Else compute $f = f(x_1 - \lambda u)$. If $f < f_1$, set $x_i = x_1 - \lambda u$, $f_1 = f$ and goto Step 4. Else if $f \geqslant f_1$ reduce the scalar step length λ and repeat Steps 3 through 5.

Step 6. If an improved point could not be generated even after reducing the value of λ below a small number ϵ, stop the procedure and return as output the current point, x_1.

3.3 THE PROPOSED METHOD

The resulting memetic algorithm combining the aforementioned methods is,

Step 1. Initialize the individuals of the population with random values.

Step 2. For each individual u_i of the population perform a Random Walk with Direction Exploitation for some iterations.

Step 3. Perform mutation step of the DE.

Step 3. Perform crossover step of DE.

Step 4. Perform selection step of DE and go to Step2.

The termination criterion can be a predetermined number of iterations, or a criterion based on the value of the objective function.

4 EXPERIMENTAL RESULTS

The problems used were E.coli and yeast dataset from the UCI repository [2].

In the experiments apart for the DE algorithm and the proposed method (DERW), we implemented a memetic algorithm that uses a GA as the global search method, and

BackPropagation as as the local search method, denoted as GABP. Moreover, a GA equipped with Random Walk with Direction Exploitation (GARW) as a local search method was considered, as described in Section 3. The GAs used exploit a real–valued encoding and the slotted wheel roulette as selection mechanism [5]. We performed crossover by selecting with a probability 0.7 each individual from the population. Taking random pairs of the selected individuals, called *parents*, a child is produced by taking the mean value of the parents. During the mutation process, for each component of each individual, we add a random number (from a normal distribution with zero mean and variance 0.1) with probability 0.4.

We executed 30 simulations for the E.coli problem and 30 simulations for the yeast problem taking each time 70% of the data set for training and retaining the remaining 30% as a test set (in each simulation a different partitioning of the data was imposed).

For the E.coli problem the architecture used was 7-5-8 yielding an 88–dimensional optimization task. For the yeast problem the architecture used was 8-5-10 resulting in an 105–dimensional problem. The population for the standard DE algorithm was set to 80 and 55 for the E.coli and the yeast problem, respectively. The population for the RWDE algorithm was set to 7 in the E.coli problem and was set to 10 for the yeast problem. In the hybrid GA algorithms the population was set to 10. Population sizes were selected so as to achieve the same average number of function evaluations for all the algorithms and thus be able to compare their performance more effectively. In the yeast problem only the two individuals with the minimum function values were allowed to perform a Random Walk with Direction Exploitation. Each call to the Random Walk method executed 10 iterations, while a call to BP executed 5 iterations. For the DE and the DERW the parameters α and β were set to 0.5 and 0.7 respectively. The termination criterion for all the algorithms was set to 300 iterations. We measured the classification error and calculated the average (Mean), standard deviation (Stdev), maximum (Max) and minimum (Min) values. These results are presented in Tables 1 and 2 for the E.coli and the yeast problem respectively.

Table 1: E.coli problem

Algorithm	**Mean**	**Stdev**	**Max**	**Min**
DE1	28.54	5.39	39.29	21.43
DE2	41.04	6.68	58.93	23.21
DE3	17.82	3.68	24.11	9.82
DE4	44.91	9.33	58.93	26.79
DE5	31.10	5.25	41.96	22.32
GABP	19.64	3.10	25.00	10.71
GARW	17.47	4.26	28.57	11.61
DE1RW	16.67	4.70	28.57	9.82
DE2RW	14.40	3.58	20.54	7.14
DE3RW	16.88	3.69	23.21	10.71
DERW4	15.54	3.56	22.32	8.93
DERW5	16.22	4.45	25.00	8.93

From the results reported in Table 1, for the E.coli problem, we observe that GARW outperforms the GABP algorithm, while the DE2RW algorithm achieved the minimum classification error among all the tested algorithms. From the standard Diffrential Evolution only the third mutation operator, DE3 produced a performance close to the DERW algorithms.

Table 2: Yeast problem

Algorithm	Mean	Stdev	Max	Min
DE1	63.77	4.38	75.35	56.36
DE2	69.77	2.85	73.94	64.20
DE3	47.89	3.65	57.58	42.83
DE4	69.52	3.58	84.44	65.86
DE5	58.94	3.03	67.27	53.53
GABP	50.36	3.14	53.54	45.86
GARW	59.11	3.55	63.64	52.12
DE1RW	48.06	3.99	59.39	41.82
DE2RW	47.01	3.99	55.96	41.62
DE3RW	46.81	3.08	56.57	41.01
DE4RW	46.76	3.40	56.57	41.41
DE5RW	47.28	2.07	54.14	43.23

Table 2 exhibits the results obtained for the yeast problem. GABP exhibited worse performance compared with GARW and DERW and DE4RW produced the minimum classification error. The standard DE algorithms exhibited relatively poor performance with the exception of DE3.

5 CONCLUDING REMARKS

A new Local-Search based Memetic algorithm has been introduced. The algorithm exploits the Differential Evolution algorithm as a global search method, and the Random Walk with Direction Exploitation as a local search method. It has been applied on two problems from the field of bioinformatics and the obtained results are promising. The proposed method improve the performace of the standard DE algorithm as it manages to achieve better classification results while performing the same average number of function evaluations. In a future work, we intend to investigate the efficiency of the proposed method on other test problems. Moreover, we aim to try variations of the proposed algorithm using different local and global search methods.

Acknowledgment

We thank the referees for their useful remarks. We thank the European Social Fund (ESF), Operational Program for Educational and Vocational Training II (EPEAEK II), and particulary the Program PYTHAGORAS, for funding the above work.

References

[1] A.C. Tan; D. Gilbert, 2003, "An empirical comparison of supervised learning techniques in bioinformatics", In *Proceedings of the 1st Asia Pacific Bioinformatics Conference (APBC 2003)*, pp. 219–222.

[2] C.L. Blake; C.J. Merz. "UCI repository of machine learning databases", 1998.

[3] P.. Horton; K. Nakai, 1996, "A probabilistic classification system for predicting the cellural localization sites of proteins", In *Proceedings of 4th International Conference on ISMB*, pp. 109–115.

[4] S. Haykin, 1999, "Neural networks: A comprehensive foundation", New York: Macmillan College Publishing Company.

[5] D. Goldberg, 1989, "Genetic Algorithms in search, optimization and machine learning", Addison-Wesley Publishing Company, Inc.

[6] D. Corne; M. Dorigo; F. Glover, 1999, "New ideas in optimization", McGraw-Hill.

[7] R.K. Belew; J. McInerny; N.N Schraudolph. "Evolving networks:using the genetic algorithm with connectionist learning". In C.G. Langton; C. Taylor; J.D. Farmer; S. Rasmussen, editors, *Proceedings of the Second Conference in Artificial Life*, pp. 511–548. Addison-Wesley, 1991.

[8] G.E. Hinton; S.J Nowlan, 1987, "How learning can guide evolution", *Complex Systems*, 1, pp. 495–502.

[9] R. Geesing; D.G. Stork. "Evolution and learning in neural networks:the number and distribution of learning trials affect the rate of evolution". In R.P. Lippmann; J.E. Moody; D.S Touretzky, editors, *NIPS 3*, pp. 804–810. Morgan Kaufmann, 1991.

[10] W.E. Hart. "Adaptive global optimization with local search". PhD thesis, University of California, San Diego,USA, 1994.

[11] R.K Belew, 1990, "Evolution, learning and culture:computational metaphores for adaptive algorithms", *Complex Systems*, 4, pp. 11–49.

[12] R. Storn; K. Price, 1997, "Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces", *Journal of Global Optimization*, 11, pp. 341–359.

[13] S.S. Rao, 1992, "Optimization theory and applications", Wiley Eastern Limited.

[14] V.P. Plagianakos; M.N. Vrahatis, 2002, "Parallel evolutionary training algorithms for 'hardware–friendly' neural networks", *Natural Computing*, 1, pp. 307–322.