Using Reinforcement Learning for Tuning Genetic Algorithms

Jose Quevedo
Department of Industrial Engineering
University of Rhode Island
Kingston, Rhode Island, USA
jquevedo18@uri.edu

Marwan Abdelatti
University of Rhode Island
Kingston, Rhode Island
mabdelrazik@uri.edu

Farhad Imani
University of Connecticut
Storrs, Connecticut
farhad.imani@uconn.edu

Manbir Sodhi
Department of Industrial Engineering
University of Rhode Island
Kingston, Rhode Island
sodhi@uri.edu

ABSTRACT

Genetic algorithms (GAs) are a subclass of evolutionary algorithms often used to solve difficult combinatorial or non-linear problems. However, most GAs have to be configured for a particular problem type, and even then, the performance depends on many hyperparameters and reproduction operators. In this paper, a reinforcement learning (RL) approach is designed to adaptively set parameters for a GA used for solving a Capacitated Vehicle Routing Problem (CVRP). An RL agent interacts with the GA environment by taking actions that affect the parameters governing its evolution, starting from a given initial point. The results obtained by this RL-GA procedure are then compared with those obtained by alternate static parameter values. For a set of benchmark problems, the solutions obtained by the RL-GA are better (up to 11% improvement) than those obtained for the set as compared to the alternate approach. Examination of the results shows that the RL-GA maintains great diversity in the population pool, especially as the iterations accrue. Computational runs are traced to show how the RL agent learns from population diversity and solution improvements over time, leading to near-optimal solutions.

CCS CONCEPTS

• **Theory of computation** → **Reinforcement learning**; • **Computing methodologies** → **Genetic algorithms**; **Massively parallel algorithms**; • **Applied computing** → **Operations research**;

KEYWORDS

Reinforcement Learning, Genetic Algorithms,Vehicle Routing Problem, Heuristics, Optimization, Tuning, Design of Experiments, Parallelism, GPU, Acceleration

*Corresponding author: mabdelrazik@uri.edu

GECCO '21 Companion, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3463203

ACM Reference Format:
Jose Quevedo, Marwan Abdelatti, Farhad Imani, and Manbir Sodhi. 2021. Using Reinforcement Learning for Tuning Genetic Algorithms. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3449726.3463203

1 INTRODUCTION

The use of Evolutionary Algorithms (EA) for solving difficult combinatorial optimization problems is well documented in the literature [4, 7]. Most related heuristics or metaheuristics are tested with benchmark problems of established complexity such as the Traveling Salesman Problem (TSP) and variants, or Vehicle Routing Problems (VRPs) and variants, among others. Genetic algorithms (GAs) are subclass of EAs that mimic a number of reproduction features to emulate natural (genetic) growth processes. These features are: chromosome selection, mating, and mutation processes [8]. In nature, survival of the species is assumed to be the main reward. In computational methods, a typical GA algorithm generates a population of feasible solutions (i.e., individuals or chromosomes) each of which is evaluated by a fitness function - usually related to the objective and constraints of the problem being solved. In subsequent iterations, chromosomes with the greatest fitness are selected for breeding and a crossover mechanism is applied to obtain the next generation. A mutation process is also coded to introduce diversity within the population. Iterations continue until no improvements are recorded for several generations. What distinguishes GAs from other heuristic methods is that they operate on a population of potential solutions which increases the likelihood of maintaining a good feasible solution and eventually moving to a better one.

GAs can take many generations, and corresponding computation time, to converge. One approach to obtain good results in acceptable times is to parallelize the algorithm by either using multi-core CPU-based algorithms [21], or by utilizing recent hardware advances in graphics processing units (GPUs) [9, 11] that led to decreases in processing times by two to three dimensions of order [1]. One of the applications involving GAs to solve combinatorial optimization problems can be found in the well-known VRP problems that gain special relevance in modern logistics and supply chain operations. VRPs involve determining optimal delivery/pick up routes for multiple vehicles through a set of locations, subject to constraints [12]. Many variants of the VRP have been studied: the capacitated VRP

(CVRP), periodic VRP (PVRP), VRP with time windows (VRPTW) and others. The CVRP problem is a basic form of the VRP where the total demands of the customers serviced by a vehicle cannot exceed its capacity, and the customers must be visited only once.

GAs have been used effectively to solve many VRP scenarios [7, 13]. The use of GPUs for solving VRPs has also been reported in [21, 26]. [2] presented an improved GA incorporated with a local search algorithm for the CVRP. The proposed framework was entirely executed on a GPU and successfully provided high-quality solutions within reasonable computational times, and near-optimal solutions for smaller benchmark problems.

A common problem with GAs is the premature convergence, i.e., the inability to escape from locally optimal, but globally suboptimal, solutions. This can be addressed by preserving the population diversity, whereby a local optima can be escaped from by "jumping" to a mutant solution substantially different from the stagnant population. However, keeping the balance between speedy convergence and high diversity is not easy. The selection of appropriate parameter values for executing a GA for a particular class of problems is known as the parameter tuning problem. Correctly tuning a GA is essential for establishing a high-performing algorithm that can be used for yielding, it is a persistent challenge for developers because of the large number of options and the limited knowledge about the effect of the parameters on the performance [16].

Numerous techniques for tuning GA have been reported, including but not limited to: fuzzy logic [23], meta-EA [14] and design of experiments (DOE) [5]. [6] proposed a factorial DOE-based approach for setting GA parameters considering the interaction of the crossover and mutation rates. Multiple DOE methods have been studied in [5] for tuning GA applied to the single machine total weighted tardiness problem (TWTP). Although these methods resulted in near-optimal solutions, low efficiency remains a problem. First, solutions are obtained after many generations - this precludes their utilization as a means of finding solution for real-time systems. Second, the results obtained are sensitive to the values of several GA parameters such as mutation and crossover rates and changing these leads to instabilities of the algorithm [15]. Previous studies with self-adaptive controllers for online parameter tuning have improved solutions but suffer from premature convergence too[3].

RL is an effective learning-based controller that is used to determine parameter values for sequential stochastic decision problems. The multistage decision processes have been previously implemented in this domain for parameter selection by [10, 20, 22]. [19] utilized temporal difference (TD) learning to control the mutation step size for real-valued encodings. However, these decision models have focused on utilizing an off-the-shelf evolutionary algorithm and controlling a single parameter at a time (e.g., probability of permutation) to mitigate estimating a large state-action model. These deficiencies, in turn, cause patchwork issues, which are associated with ignoring the effect of parameter interactions when solving large-scale and dynamic real-world problems.

2 PROPOSED APPROACH

This paper introduces *RL* as a generic control mechanism to guide the GA to decide multiple parameters, aiming to achieve near-optimal results with reduced computational cost. We tune both

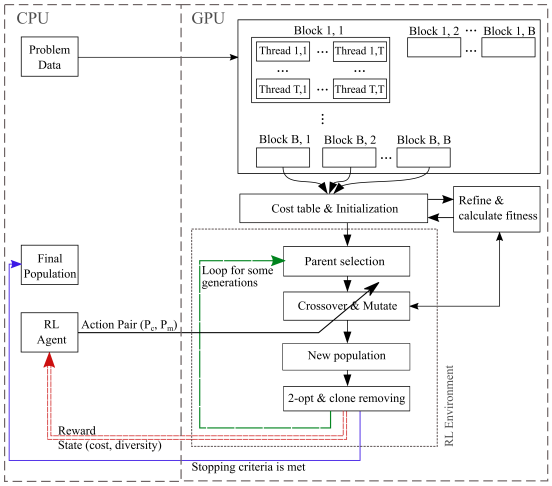

Figure 1: Proposed RL approach for adaptive tuning of a GA

the mutation and crossover probabilities (p_m and p_c respectively) simultaneously to employ decisions based on the feedback received from the evolutionary process. More specifically, the GA environment responds to the actions taken by the *RL* agent, namely the p_m and p_c values, and generates a new population with the associated rewards. This *RL*-based parameter controller improves the performance of the algorithm towards solving large-scale and practical problems. The proposed algorithm is tested on a number of common CVRP benchmark problems in the literature.

To evaluate the effectiveness of the proposed algorithm, we compare the results of our proposed approach with the results obtained by tuning the same GA using a design of experiments (DOE) on the same problems. The tuning settings by the DOE are the result of a two-level (2^k) full factorial design of experiment [18] pilot runs on small scale problems which are then reflected on medium and large problems.

3 RL-BASED PARAMETER SELECTION

The proposed framework is defined according to the Q-learning model [17, 24, 25], which iteratively improves the off-policy method by estimating transitions from one state-action pair to another. Therefore, to learn the value of the current policy and to improve it iteratively, we define *RL* to control GA operators as follows:

- **State space:** The state space (S) is defined as a tuple of two indices, namely, the change (C) in the value of the best cost from the older and the recent generation, and the diversity index (DI) that tells the rate of the number of unique chromosomes in the new generation. It's noted from Table 1 that the state space is of size 6×5 as detailed in the sequel.
- **Action space:** The action space defined as A, is the tuple (p_m, p_c), where p_m and p_c denote the mutation and the crossover rate respectively. The ranges of each of these two parameters are from 0 to 1 discretized separately into $d = 10$ intervals, and therefore, create $d^2 = 100$ possible action pairs.
- **Estimated state-action:** Q-learning is a values-based learning algorithm, which updates the value function (Q) for each

state-action pair ((S, A)) in Q-table using the following Bellman equation:

$$Q(S(t), A(t)) \leftarrow (1-\alpha)Q(S(t), A(t)) + \alpha(R(t) + \gamma \max_{A(t+1)} Q(S(t+1), A(t+1))) \quad (1)$$

The selection of an action is made based on the epsilon-greedy policy, and this policy chooses a random action either from the Q-table with a probability ϵ, or picks the greedy action with the highest Q-value.

- **Reward:** The reward R is determined based on the state S returned from the GA environment using two indices: the cost improvement C and the diversity index DI. Table 1 shows the returned reward values based on different C and DI levels. The rows in the table correspond to the possible values of the cost improvement (C); from very high change (VHC) to very low change (VLC) as well as a stalled state and an increase in cost; whereas the columns correspond to five possible values for the diversity index (DI) from very high diverse (VHD) to very low diverse (VLD).

	VHD	HD	MD	LD	VLD
VHC	200	150	100	50	25
HC	150	112.5	75	37.5	18.75
LC	100	75	50	25	12.5
VLC	50	37.5	25	12.5	6.25
Stalled	0	0	-10	-20	-30
Increased	-1000	-1500	-2000	-2500	-3000

Table 1: Rule table for reward values

As opposed to the conventional GA, where the mutation and crossover rates p_m and p_c are fixed throughout the run, our approach depicted in Figure 1 involves an adaptive change of these parameters. The population during the current state is denoted as Population(t), and its evolution to Population($t+1$) is governed by the values of p_c and p_m for the current state. The GA applies the selected values for parameters only for a fixed number of generations after which the reward is returned to the *RL* agent. The expected long-term impact and the influence of the action from this state are updated based on the reward value R. As a result, the proposed framework preserves the best chromosomes and changes the parameter values as the algorithm iterates. The learning rate (α) is used to control the influence of the target on the current Q-values. Since the target is the TD between two updates by the *RL*, it will be influenced by the improvement scale problem of GA. As the GA evolves, improvements to the fitness value will taper off. Consequently, we only consider the fitness value of the current generation. When GA is evolving, the learning rate of *RL* is controlled to decrease the influence of the target on the learned Q-values to prevent fitness decay. A complete pseudocode algorithm for the *RL* part of RL-GA is given in Algorithm 1. Interested readers can clone the functioning code from the GitHub repository[1].

[1]https://github.com/J-Que/RL-GA

Algorithm 1 *RL*-enabled GA for tuning the hyperparameters

Input: Population ($Population$): initial population at t=0
States (S): $\{C, DI\}$
Action (A): $\{(p_{c0}, p_{m0}), (p_{c0}, p_{m1}), \ldots, (p_{c99}, p_{99})\}$
Reward function (R): $S \times A \rightarrow \mathbb{R}$
Probabilistic transition function ($Transition$): $S \times A \rightarrow S$
Parameters: $\{\delta, \lambda, \gamma, \epsilon, \alpha\}$

```
function GARL(.)
    Q(S(t), A(t)) ← initialize
    while Q(S(t), A(t)) is not converged do
        Use State Set S(t)
        while t < T do
            p_c ← S(t).SetCrossoverProb()
            p_m ← S(t).SetMutationProb()
            Population(t) ← Population(t).SelectedGenes
            Population(t) ← Population(t).GA(p_c, p_m)
            R(t) ← Population(t).Reward()
            S(t+1) ← S(t).Transition(A(t))
            A(t) ← S(t).π(A(t))
            Q(S(t+1), A(t)) ← (1−α)Q(S(t), A(t)) + α(R(t)
                              +γ max_{A(t+1)} Q(S(t+1), A(t+1)))
            Population(t) ← Population(t+1)
            S(t) ← S(t+1)
            t ← t+1
        end while
    end while
```

Output: $Population(t = T), R(t = T)$

4 COMPUTATIONAL RESULTS

To test the performance of the RL-GA, several instances of benchmark problems from the literature were solved, and the results are reported in Table 2. Problems involving 40 to 70 customer nodes were selected, as well as a larger 200 node problem. Each problem was solved five times using the RL-GA algorithm. The first column on Table 2 is the sequence number. Column 2 is the problem label and it details the number of nodes and the number of vehicles. Column 3 is the best known solution as reported by the site[2]. Column 4 is the best solution obtained by the GA algorithm run only on a CPU. Column 5 reports the results obtained by solving the problem using a GPU version of the algorithm, as detailed in [2]. p_m and p_c for the results in Columns 4 and 5 were set based on a Design of Experiments approach, detailed in [1]. The sixth column reports the results of the RL-GA algorithm and the last two columns report the average value of the results obtained from the five runs and their respective standard deviation.

As can be seen from Table 2, RL-GA shows equal or better performances than the CPU-GA and the GPU-GA implementations for eleven of the twelve problems. For problem 8 (P-n22-k8), all the GA implementations improve the best known solution, and for problem 11 (P-n55-k15), the CPU-GA finds a better solution than the current best known solution. The averaged results also demonstrate equal or better performance than the GPU-GA for nine of the twelve problems, showing its consistency.

[2]http://vrp.atd-lab.inf.puc-rio.br/index.php/en/

	Problem	Published Optimal	GPU-GA	CPU-GA	RL-GA	Average RL-GA	Std Dev RL-GA
1	B-n31-k5	672	691	677	672*	675.6	4.41
2	B-n45-k5	751	840	762	751*	756	4.29
3	B-n50-k8	1312	1329	1327	1316	1324	4.15
4	M-n200-k17	1373	1473	1598	1434	1482.6	34.3
5	P-n16-k8	450	450	450	450*	450	0
6	P-n20-k2	216	216	216	216	216	0
7	P-n21-k2	211	211	211	211	211	0
8	P-n22-k8	603†	590	592	590*	596.2	5.04
9	P-n23-k8	529	529	529	529*	529	0
10	P-n40-k5	458	479	478	458*	465.2	9.37
11	P-n55-k15	989†	1058	975*	981	996	9.93
12	P-n70-k10	827	845	892	853	865.2	11.29

Table 2: Runs results and comparisons

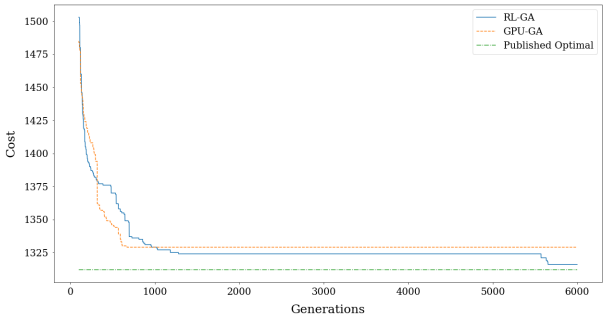

Figure 2: Solution trajectory for B-n50-k8

Figure 2 illustrates example trajectory (i.e., cost vs generations) taken whilst solving problem 3 (B-n50-k8). Here it can be seen that RL-GA finds a better solution than the DOE tuned GA. This also manifests when examining the solutions of the other problems tested, except for problem 12 where the GPU-GA ultimately attains a better solution.

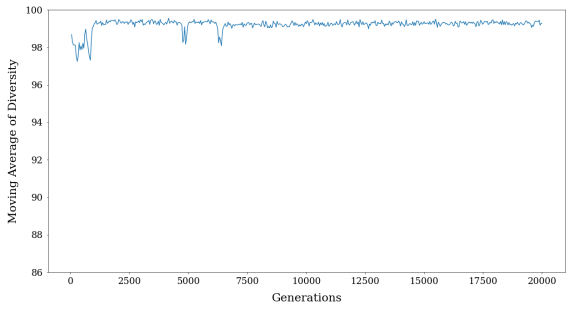

Figure 3: Solution diversity for B-n50-k8 with GPU-GA

Figures 3 and 4 show the diversity changes over the course of the entire run for problem (B-n50-k8). From the figures, it can be observed that RL-GA biases against population diversity when the objective solutions are improving rapidly, and towards increasing diversity when the solution stagnates. This is indeed the behavior that we wanted to inculcate - and is evidence that the reward function is therefore correctly defined.

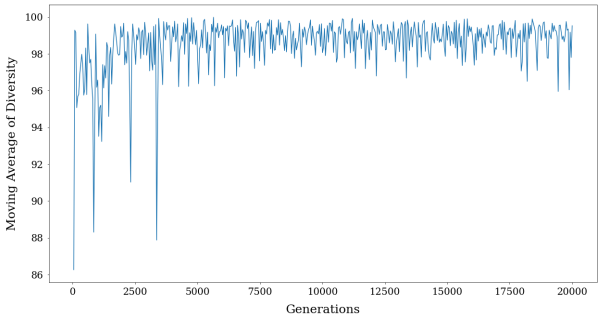

Figure 4: Solution diversity for B-n50-k8 with RL-GA

5 CONCLUSIONS

In this paper, a reinforcement learning (RL) approach has been designed to adaptively set parameters for a genetic algorithm (GA) used to solving a Capacitated Vehicle Routing Problem (CVRP). For a set of benchmark problems, the solutions obtained by the RL-GA are better than those obtained for the same set of problems using a static, Design of Experiments-based approach. The RL-GA appears to be promising in that it is able to maintain greater diversity in the population pool. Tests with problems involving a greater number of nodes for the CVRP are planned and these will require significant computing effort. Successful with larger problem sets, will add to the methods that can be used to obtain good solutions for complex practical problems using genetic methods with reasonable computing effort. Since the approach presented here acts on parameters that are fundamental to all genetic algorithms, they can widely be implemented for solving other combinatorial and nonlinear optimization problems utilizing GAs.

REFERENCES

[1] Marwan F Abdelatti, Abdeltawab Hendawy, and Manbir S Sodhi. 2021. Optimizing a GPU-Accelerated Genetic Algorithm for the Vehicle Routing Problem. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion)*.

[2] Marwan F Abdelatti and Manbir S Sodhi. 2020. An improved GPU-accelerated heuristic technique applied to the capacitated vehicle routing problem. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 663–671.

[3] Aldeida Aleti and Irene Moser. 2016. A systematic literature review of adaptive parameter control methods for evolutionary algorithms. *ACM Computing Surveys (CSUR)* 49, 3 (2016), 1–35.

[4] Tariq Alzyadat, Mohammad Yamin, and Girija Chetty. 2020. Genetic algorithms for the travelling salesman problem: a crossover comparison. *International Journal of Information Technology* 12, 1 (2020), 209–213.

[5] Arif Arin, Ghaith Rabadi, and Resit Unal. 2011. Comparative studies on design of experiments for tuning parameters in a genetic algorithm for a scheduling problem. *International Journal of Experimental Design and Process Optimisation* 2, 2 (2011), 102–124.

[6] Tapan P Bagchi and Kalyanmoy Deb. 1996. Calibration of GA parameters: the design of experiments approach. *Computer Science and Informatics* 26 (1996), 46–56.

[7] Barrie M Baker and MA Ayechew. 2003. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research* 30, 5 (2003), 787–800.

[8] Richard J Bauer Jr, Richard J Bauer, et al. 1994. *Genetic algorithms and investment strategies*. Vol. 19. John Wiley & Sons.

[9] A Benaini and A Berrajaa. 2018. Genetic algorithm for large dynamic vehicle routing problem on GPU. In *2018 4th International Conference on Logistics Operations Management (GOL)*. IEEE, 1–9.

[10] Fei Chen, Yang Gao, Zhao-qian Chen, and Shi-fu Chen. 2005. SCGA: Controlling genetic algorithms with Sarsa (0). In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, Vol. 1. IEEE, 1177–1183.

[11] IM Coelho, PLA Munhoz, LS Ochi, MJF Souza, C Bentes, and R Farias. 2016. An integrated CPU–GPU heuristic inspired on variable neighbourhood search for the single vehicle routing problem with deliveries and selective pickups. *International Journal of Production Research* 54, 4 (2016), 945–962.

[12] George B Dantzig and John H Ramser. 1959. The truck dispatching problem. *Management science* 6, 1 (1959), 80–91.

[13] Abel Garcia-Najera and John A Bullinaria. 2009. Comparison of similarity measures for the multi-objective vehicle routing problem with time windows. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. 579–586.

[14] John J Grefenstette. 1986. Optimization of control parameters for genetic algorithms. *IEEE Transactions on systems, man, and cybernetics* 16, 1 (1986), 122–128.

[15] Giorgos Karafotias, Mark Hoogendoorn, and AE Eiben. 2015. Evaluating reward definitions for parameter control. In *European Conference on the Applications of Evolutionary Computation*. Springer, 667–680.

[16] Giorgos Karafotias, Mark Hoogendoorn, and Ágoston E Eiben. 2014. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation* 19, 2 (2014), 167–187.

[17] P Read Montague. 1999. Reinforcement learning: an introduction, by Sutton, RS and Barto, AG. *Trends in cognitive sciences* 3, 9 (1999), 360.

[18] Douglas C Montgomery. 2017. *Design and analysis of experiments*. John wiley & sons.

[19] Sibylle D Muller, Nicol N Schraudolph, and Petros D Koumoutsakos. 2002. Step size adaptation in evolution strategies using reinforcement learning. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, Vol. 1. IEEE, 151–156.

[20] James E Pettinger and Richard M Everson. 2002. Controlling genetic algorithms with reinforcement learning. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. 692–692.

[21] Antón Rey, Manuel Prieto, José Ignacio Gómez, Christian Tenllado, and J Ignacio Hidalgo. 2018. A CPU-GPU parallel ant colony optimization solver for the vehicle routing problem. In *International Conference on the Applications of Evolutionary Computation*. Springer, 653–667.

[22] Yoshitaka Sakurai, Kouhei Takada, Takashi Kawabe, and Setsuo Tsuruta. 2010. A method to control parameters of evolutionary algorithms by using reinforcement learning. In *2010 Sixth International Conference on Signal-Image Technology and Internet Based Systems*. IEEE, 74–79.

[23] Arash Nasrolahi Shirazi, Meghan Steinhaus, Matthew Agostinelli, and Manbir Sodhi. 2016. Fuzzy cell genetic algorithm approach for flexible flow-line scheduling model. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 1–7.

[24] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.

[25] Christopher John Cornish Hellaby Watkins. 1989. Learning from delayed rewards. (1989).

[26] Mieczysław Wodecki, Wojciech Bożejko, Szymon Jagiełło, and Jarosław Pempera. 2015. Parallel Cost Function Determination on GPU for the Vehicle Routing Problem. In *International Conference on Artificial Intelligence and Soft Computing*. Springer, 778–788.