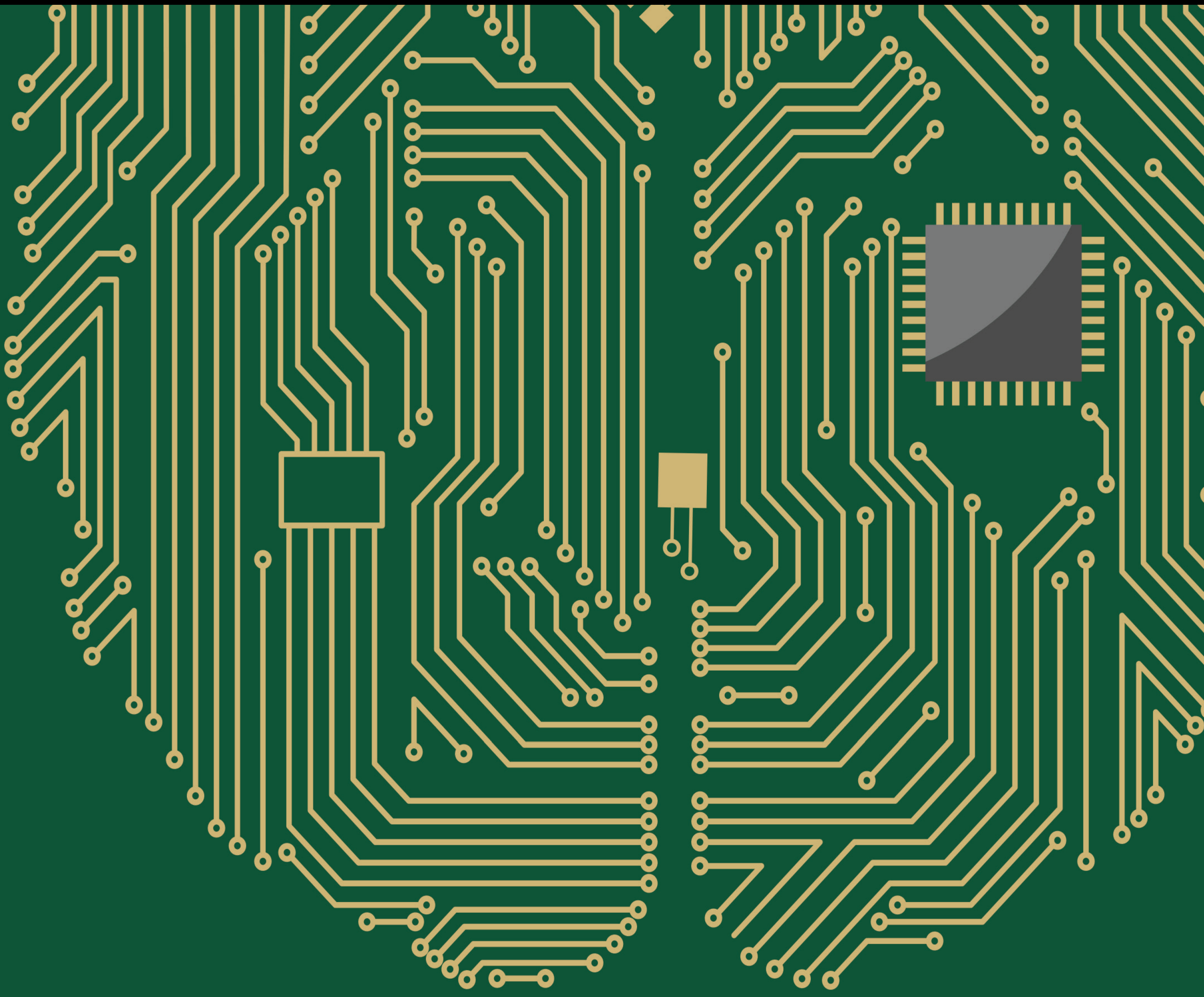
Computational Intelligence and Neuroscience

Nature-Inspired Computing Applied to Neuroscience

Lead Guest Editor: Wassim Ayadi
Guest Editors: Hend Bouziri, Wassila Aggoune-Mtalaa,, and Adelaide Valente Freitas



Nature-Inspired Computing Applied to Neuroscience

Nature-Inspired Computing Applied to Neuroscience

Lead Guest Editor: Wassim Ayadi
Guest Editors: Hend Bouziri, Wassila Aggoune-Mtalaa,, and Adelaide Valente Freitas

Contents

Hindawi
Computational Intelligence and Neuroscience
Volume 2020, Article ID 8873057, 10 pages
https://doi.org/10.1155/2020/8873057

Research Article

An Empirical Investigation of Transfer Effects for Reinforcement Learning

Jung-Sing Jwo,[1,2] Ching-Sheng Lin ,[1] Cheng-Hsiung Lee,[1] and Ya-Ching Lo[1]

[1]*Master Program of Digital Innovation, Tunghai University, Taichung 40704, Taiwan*
[2]*Department of Computer Science, Tunghai University, Taichung 40704, Taiwan*

Correspondence should be addressed to Ching-Sheng Lin; cslin612@gmail.com

Received 9 September 2020; Revised 28 October 2020; Accepted 5 December 2020; Published 16 December 2020

Academic Editor: Wassim Ayadi

Previous studies have shown that training a reinforcement model for the sorting problem takes very long time, even for small sets of data. To study whether transfer learning could improve the training process of reinforcement learning, we employ Q-learning as the base of the reinforcement learning algorithm, apply the sorting problem as a case study, and assess the performance from two aspects, the time expense and the brain capacity. We compare the total number of training steps between nontransfer and transfer methods to study the efficiencies and evaluate their differences in brain capacity (i.e., the percentage of the updated Q-values in the Q-table). According to our experimental results, the difference in the total number of training steps will become smaller when the size of the numbers to be sorted increases. Our results also show that the brain capacities of transfer and nontransfer reinforcement learning will be similar when they both reach a similar training level.

1. Introduction

Reinforcement learning (RL) aims at learning policies to map from states to actions for the purpose of maximizing the expected accumulated reward and reaching the goal. Compared with the supervised learning approaches where the models are trained on the input set and the given output set, the RL agent has to interact with the environment and learn from those experiences through trial and error to yield the optimal behaviour.

Mathematically, RL can be formulated as a Markov decision process (MDP) which is a framework to model decision-making problems [1]. An MDP is represented by the tuple $<S, A, T, R>$ where S denotes the state space in the environment and A is the action set to take in a given state. Function T is defined as $P(s\prime|st, na)$ which indicates the probability of the next state $s' \in S$ at time step $t+1$ given the current state $s \in S$ and the action $a \in A$ taken at time step t. Function R is a reward scheme used to assign the score for the action performed under the state s and is used as a guidance for the agent to produce suitable behaviours. Then, the objective of the RL agent is to learn a policy $\pi_\theta(a|s)$ which tells the agent what the best action $a \in A$ to perform is while the environment is in the state $s \in S$ with the parameter θ. In general, there are two main approaches to solving RL problems, model-based and model-free learning. In the model-based approaches, the goal is to learn the model of the environment and obtain the optimal policy relying on the past transitions. On the other hand, model-free approaches learn to directly acquire the optimal policy by the trial-and-error interactions without modelling the underlying environment. Model-based approaches are often sample-efficient, but the requirement of specifying the model of real-world tasks is often restrictive and difficult to satisfy. Therefore, model-free approach is commonly preferred over the model-based approach if it is not hard to sample the trajectories [2, 3]. Q-learning [4] and SARSA [5] are two well-known model-free RL algorithms which fit the optimized policy by learning the action-value (Q-value) function. Note that an action-value function is used to express the expectation of the reward for each state-action pair (s, a). In recent years, since the development of deep learning methods has gained significant attention and achieved innovations in many fields, it is common to adopt deep learning methods for RL algorithms in order to boost the performance. A combination of the convolutional neural network (CNN) [6] and Q-learning called deep Q-networks (DQN) [7, 8] is proposed to handle large state-action space. DQNs have been shown to reach at or even beyond human-level performance on many games. An alternative double estimator method, double Q-learning [9], is introduced to reduce the overestimations of the action values in the Q-learning algorithm. As double Q-learning was proposed in a tabular setting and DQN algorithm suffers from overestimations, double DQN is used for large-scale function approximation and to reduce the overestimations by combining double Q-learning and DQN [10–12].

RL algorithms usually require large amounts of trial-and-error and many learning iterations to determine an effective policy from very large-scale state-action space, making them very time-consuming. Recently, there has been a strong interest in the development of deep learning models with the ability to transfer experiences across similar tasks. The two representative types of methods are the transfer of trained models and transfer of learned knowledge [13]. The first methods transform the neural network layers from the pre-trained model to the target model [14, 15] whereas the second approaches aim at transferring learned knowledge from the trained network to the target network [16, 17]. A Q-learning-based approach has been applied for the sorting problem [18]. However, it takes large number of training steps to finish the training process, even for small sets of data. Since transfer learning has been widely adopted to speed up the training process, this motivates us to devise a transfer scheme and compare it with the nontransfer method in the training performance. In this paper, we conduct a series of experiments using the sorting problem as a case study. We transfer the knowledge learned from the task n to the task $n+1$ where n is the size of the numbers to be sorted and continuously use a Q-learning-based method to train the model. The total number of training steps and the size of the brain capacity, which denotes the knowledge in the Q-table, are two metrics to measure the impact of transfer learning techniques.

The rest of this paper is organized as follows. Section 2 reviews the background and related work of this paper. Section 3 describes our training strategies and detailed methodology. Experimental setup and results are presented and discussed in Section 4. In Section 5, we discuss conclusions and future work.

2. Background and Related Work

In this section, we first give an overview of Q-learning which is the base RL algorithm in this paper. The application of RL in the sorting problem is discussed as well.

Q-learning, a form of model-free method, is one of the most known RL algorithms initially designed for the use of Markov decision processes. It updates the Q-value with the following rule:

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha\left(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right), \tag{1}$$

where $Q(s_t, a_t)$ is the action-value function to compute the expected reward of a state-action pair at time step t, α is the learning rate, γ is the discount factor, and r_t is the reward obtained after selecting action a_t given state s_t. The max operator from the update rule indicates that the agent chooses the best action a by computing the maximum Q-value for the next state s_{t+1}. The mechanism to exploit the maximum Q-value while updating is called an off-policy algorithm, i.e., the choice of taking action a_t and a does not follow the same policy. On the contrary, the SARSA updates the Q-value based on the policy being followed by the following equation:

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \times Q(s_{t+1}, a) - Q(s_t, a_t)). \tag{2}$$

When the algorithm uses the same mechanism for the behaviour policy (i.e., $Q(s_t, a_t)$) and the estimation policy (i.e., $Q(s_{t+1}, a)$), it is called on-policy [19].

The sorting problem is a quintessential computer science task and has been applied to many fields since its emergence. Based on the analysis of all comparison-based sorting algorithms, the computation requires $O(n \log n)$ complexity. A RL-based approach, which applies stability and resiliency ideas from feedback controls, is proposed to overcome the errors and early program termination limitations for the traditional computing [20]. An empirical exploration compares the RL model with two traditional sorting algorithms and shows that the RL sorting model completes the task with less array manipulations. In order to investigate the effect of two different reward schemes, immediate reward and pure delayed reward, a Q-learning algorithm is implemented to compare the total number of training steps and average number of sorting steps [18]. A case study of the sorting problem is conducted and concludes that immediate reward takes much less steps to finish the task.

3. Methodology and Learning Design

In this section, we describe important features in our proposed methodology, which include training level and brain capacity. We also discuss how we designed the RL algorithm in order to formulate the sorting problem into RL settings.

3.1. RL-Based Setting for Sorting Problem. We model the initial state s_0 Step t, which consists of n elements, as the list of numbers to be sorted, and hence, there will be n factorial possible states denoted by S_n. For any state s at time, an action $a(i, j)$ is defined as the swap of values in position i and position j. Thus, there will be $C\binom{n}{2}$ possible actions in the action set A_n. Once the action $a(i, j)$ is chosen under state s_t, the next state s_{t+1} is determined by exchanging the element in position i with the element in position j of the state s_t. For example, assuming the initial state is $s_0 = [4, 5, 3, 2, 1]$ and an action $a(1, 4)$ is performed, this state-action pair will result in the next state $s_1 = [2, 5, 3, 4, 1]$.

As suggested by the previous study [18] that immediate reward performs better than pure delayed reward, we use immediate reward scheme in this research. We give the reward by considering whether the action actually improves the number of elements in the correct position. A similarity value is introduced to measure the similarity between the current state s_t and the goal state S_{goal} (i.e., the sorted list) as follows:

$$\text{sim}\left(s_t, S_{\text{goal}}\right) = \sum_{i=1}^{n} \text{Equal}\left(s_t[i], S_{\text{goal}}[i]\right), \tag{3}$$

where Equal function will return one if two states have the same value at position i and zero otherwise. We then compute the difference of $\text{sim}(s_t, S_{\text{goal}})$ and $\text{sim}(s_{t+1}, S_{\text{goal}})$ to assign the reward as follows:

$$\text{Reward} = \begin{cases} \text{reward\_better}, & \text{sim}(s_{t+1}, S_{\text{goal}}) > \text{sim}(s_t, S_{\text{goal}}), \\ \text{reward\_equal}, & \text{sim}(s_{t+1}, S_{\text{goal}}) = \text{sim}(s_t, S_{\text{goal}}), \\ \text{reward\_worse}, & \text{sim}(s_{t+1}, S_{\text{goal}}) < \text{sim}(s_t, S_{\text{goal}}). \end{cases} \tag{4}$$

In this paper, reward_better is 1, reward_equal is 0, and reward_worse is −1. For the aforementioned example, since $s_0 = [4, 5, 3, 2, 1]$ will receive a similarity value of 1 and $s_1 = [2, 5, 3, 4, 1]$ will receive a value of 2, the reward value of reward_better will be given.

3.2. Learning Algorithm. The objective of the learning algorithm is to sort a given example which consists of n numbers for a series of episodes until the success rate reaches a predefined threshold. Algorithm 1 (RL_Sort) represents how we executed the model training on one training instance based on the Q-learning algorithm. The algorithm gives a list S_{training} and a Q-table as inputs and then produces a new Q-table and the number of training steps as output. RL_Sort begins with the initialization of upper_bound, train_steps, and success_rate. The upper_bound is used to define the maximum allowed number of swaps for sorting and we set $n+1$ as the threshold. The variable train_steps is to store the number of episodes spent for training. The variable success_rate is the criterion to terminate the training process and is set to 0.75 in our experiments. S_{goal} is the correct sorting result. The experimental parameters are as follows: $\alpha = 0.05$, $\gamma = 0.9$, and $\varepsilon = 0.85$. In each episode from line 11 to line 31, the model chooses an action $a(i, j)$ given current state s based on ε-greedy [21] and receives a new state s' (lines 12~13). There are two conditions in which the episode will end. In one condition, s' is the S_{goal} and a positive reward (reward_win = 1) will be given (lines 16~18). In the other condition, the number of swapping times already exceeds upper_bound and a negative reward (reward_lose = −1) will be received. Since the first condition reaches a success state, we will examine the success rate for the latest 100 episodes to determine whether the training process should stop or a new episode should begin. For the cases that the current episode needs to continue (lines 23~28), the Q-table is updated based on the reward equation (4).

When the training task moves from the example of sorting n numbers to $n+1$ numbers, values in Q-table are usually set to zero or randomly initialized. In our transfer setting, the knowledge learned from sorting n numbers is migrated to solve the problem of sorting $n+1$ numbers. For the Q-table obtained from sorting n numbers (denoted as Q_source with size $(n!) \times C\binom{n}{2}$), we expand its state representation by appending a number $n+1$ at the end of each state to fit in the Q-table representation for sorting $n+1$ numbers (denoted as Q_target with size $(n+1)! \times C\binom{n+1}{2}$). Therefore, each state s in Q_source will become s.append$(n+1)$. We then are able to map the Q-value of the state-action pair from Q_souce to Q_target. In this way, as the number in position $n+1$ is already in the correct position, we try to encourage the model to exploit the prior knowledge from Q_souce and avoid touching the action related to the position $n+1$. For example, when n equals 3 and one of the state is [1, 3, 2] with actions $a(1, 2)$, $a(1, 3)$, and $a(2, 3)$, we will transfer these 3 Q-values in Q_source to Q_target where the corresponding state is [1, 3, 2, 4] with actions $a(1, 2)$, $a(1, 3)$, and $a(2, 3)$. Those nontransferable Q-values will be set to zero or randomly initialized. Figure 1 demonstrates how we transfer a Q-table from $n = 3$ to $n = 4$.

3.3. Performance Metrics. In this paper, we define three performance metrics which include training level, number of training steps, and brain capacity.

Training level is a performance-oriented indicator to measure how well the model can use the existing knowledge to perform the task during training. After finishing a training procedure of one instance for sorting n numbers, the model is scheduled to sort $n!$ tasks where each task is given by a permutation of those n numbers. Subsequently, we compute the average number of sorting steps for these $n!$ tasks as the model's training level. *Number of training steps*, which is denoted as train_steps in Algorithm 1, is the number of episodes that the model spends on training an example. It is an important factor to measure the effectiveness of the algorithm. *Brain capacity* is concerned with the status of Q-table and is an important measure to compare the knowledge usage between nontransfer and transfer methods. It is defined as the ratio of entries which have been updated in a Q-table.

3.4. Experimental Setup and Results. In order to compare the difference and efficacy between nontransfer and transfer methods, a case study in the sorting problem is presented. We illustrate a series of experiments for both nontransfer and transfer RL to investigate the difference of training speed and the contrast of knowledge requirement.

3.4.1. Experimental Setup. We design an experimental setting to train the model to sort lists of n numbers where each list is from a permutation of $\{1, 2, \ldots, n\}$. In order to provide an equitable comparison, we run nontransfer and transfer

```
input: $S_{\text{training}}$, $Q_n[S_n, A_n]$
 (1)  initialize
 (2)     upper_bound = $n + 1$
 (3)     train_steps = 0
 (4)     success_rate = 0.75
 (5)     $S_{\text{goal}} = [1, 2, \ldots, n]$
 (6)  repeat
 (7)     end = FALSE
 (8)     swap_times = 0
 (9)     $s = S_{\text{training}}$
(10)     current_rate = 0
(11)     repeat
(12)        Select an action a based on $\varepsilon$-greedy
(13)        Perform the action $a$ and observe $s'$ and the corresponding reward
(14)        swap_times = swap_times + 1
(15)        if ($s'$ is $S$_goal) then
(16)           $Q_n[s, a] \longleftarrow Q_n[s, a] + \alpha \times (\text{reward\_win} - Q_n[s, a])$
(17)           end = TRUE
(18)           Check the success rate for the latest 100 episodes and assign to current_rate
(19)        elseif (swap_times >upper_bound) then
(20)           $Q_n[s, a] \longleftarrow Q_n[s, a] + \alpha \times (\text{reward\_lose} + \gamma \times \max_{a'} Q_n[s', a'] - Q_n[s, a])$,
(21)           end = TRUE
(22)        else
(23)           if (dist($s'$, $S$_goal) > dist($s$, $S$_goal))
(24)              $Q_n[s, a] \longleftarrow Q_n[s, a] + \alpha \times (\text{reward\_better} + \gamma \times \max_{a'} Q_n[s', a'] - Q_n[s, a])$,
(25)           elseif (dist($s'$, $S$_goal) < dist($s$, $S$_goal))
(26)              $Q_n[s, a] \longleftarrow Q_n[s, a] + \alpha \times (\text{reward\_worse} + \gamma \times \max_{a'} Q_n[s', a'] - Q_n[s, a])$,
(27)           else
(28)              $Q_n[s, a] \longleftarrow Q_n[s, a] + \alpha \times (\text{reward\_equal} + \gamma \times \max_{a'} Q_n[s', a'] - Q_n[s, a])$,
(29)           $s \longleftarrow s'$
(30)     until end is TRUE
(31)     train_steps = train_steps + 1
(32)     until current_rate >= success_rate
(33)     return $Q_n$ , train_steps
```

Algorithm 1: The Q-learning based algorithm for the sorting task. RL_Sort.

RL in parallel and propose an algorithm, which is presented as pseudocode in Algorithm 2, to satisfy our needs.

The input of Algorithm 2 consists of a list S_{training} which is a permutation of $\{1, 2, \ldots, n\}$ and a Q-table ($TRQ_{n-1}[S_{n-1}, A_{n-1}]$) which is learned from sorting $n - 1$ numbers. A Q-table ($NRQ_n[S_n, A_n]$) of nontransfer RL is initialized to zero for all Q-values and a Q-table ($TRQ_n[S_n, A_n]$) of transfer RL is transferred from TRQ_{n-1} $[S_{n-1}, A_{n-1}]$ as the mechanism discussed in Section 3. B. A variable upper_bound is used as one of the constraints for the training level. The input list S_{training} is given to both S_{nt} and S_{tr} as the initial sorting list for both methods. Then, the algorithm starts iteratively to solve the sorting tasks. We will begin with the nontransfer RL. This process consists of training and evaluation. In the training part, we input the current Q-tables ($NRQ_n[S_n, A_n]$) and the list S_{nt} to Algorithm 1 to train the model (line 11). The number of training steps returned from Algorithm 1 is accumulated to the variable NonTrans_Tr_Steps (line 13). For the evaluation part, the returned $NRQ_n[S_n, A_n]$ of Algorithm 1 is then used to sort $n!$ lists from the permutation of $\{1, 2, \ldots, n\}$ and the average number of sorting steps is model's training level denoted as Avg_{nt}. We then select the list which takes the maximum number of steps to sort as the new S_{nt} (line 15). The same procedure is also applied for transfer RL as seen in lines 12, 14, and 16. The above process is repeated until two models reach a similar training level (i.e., Avg_{nt} and Avg_{tr} are very close or both of them are lower than upper_bound). This restriction is to ensure that both two methods exhibit comparable abilities to sort $n!$ lists and affirm that it is fair to conduct a further comparison of the total number of training steps and the brain capacity.

3.4.2. Experimental Results. As an empirical study, we illustrate our results for n equal to 5, 6, 7, and 8. In order to produce a more fair view of the comparison, we repeat Algorithm 2 for 30 episodes for each n. The total number of training steps and the brain capacity are two perspectives to measure the performance. The total number of training steps for the nontransfer method is abbreviated to

	(1, 2)	(1, 3)	(2, 3)
{1, 2, 3}	0	0	0
{1, 3, 2}	0	0	0.05
{2, 1, 3}	0.992	−0.272	0.331
{2, 3, 1}	0.05	0.05	0.691
{3, 1, 2}	0	0.265	0
{3, 2, 1}	−0.042	0	0

Append

	(1, 2)	(1, 3)	(2, 3)
{1, 2, 3, 4}	0	0	0
{1, 3, 2, 4}	0	0	0.05
{2, 1, 3, 4}	0.992	−0.272	0.331
{2, 3, 1, 4}	0.05	0.05	0.691
{3, 1, 2, 4}	0	0.265	0
{3, 2, 1, 4}	−0.042	0	0

Transfer

	(1, 2)	(1, 3)	(2, 3)	(1, 4)	(2, 4)	(3, 4)
{1, 2, 3, 4}	0	0	0			
{1, 3, 2, 4}	0	0	0.05			
{2, 1, 3, 4}	0.992	−0.272	0.331			
{2, 3, 1, 4}	0.05	0.05	0.691			
{3, 1, 2, 4}	0	0.265	0			
{3, 2, 1, 4}	−0.042	0	0			
{1, 2, 4, 3}						
{1, 3, 4, 2}						
{1, 4, 2, 3}						
{1, 4, 3, 2}						
{2, 1, 4, 3}						
{2, 3, 4, 1}						
{2, 4, 1, 3}						
{2, 4, 3, 1}						
{3, 1, 4, 2}						
{3, 2, 4, 1}						
{3, 4, 1, 2}						
{3, 4, 2, 1}						
{4, 1, 2, 3}						
{4, 1, 3, 2}						
{4, 2, 1, 3}						
{4, 2, 3, 1}						
{4, 3, 1, 2}						
{4, 3, 2, 1}						

FIGURE 1: An illustration of transferring the Q-table from $n = 3$ to $n = 4$.

```
input: $S_{\text{training}}$, $\text{TRQ}_{n-1}[S_{n-1}, A_{n-1}]$
(1)   initialize
(2)     new $\text{NRQ}_n[S_n, A_n]$
(3)     new $\text{TRQ}_n[S_n, A_n]$
(4)     $\text{TRQ}_n[S_n, A_n] \longleftarrow \text{TRQ}_{n-1}[S_{n-1}, A_{n-1}]$
(5)     upper_bound = $n + 1$
(6)     Assign $S_{\text{training}}$ to $s_{\text{nt}}$ and $s_{\text{tr}}$
(7)     finish = FALSE
(8)     NonTrans_Tr_Steps = 0
(9)     Trans_Tr_Steps = 0
(10)  repeat
(11)    $\text{NRQ}_n[S_n, A_n]$, Steps$s_{\text{nt}}$ = RL_Sort($s_{\text{nt}}$ , $\text{NRQ}_n[S_n, A_n]$)
(12)    $\text{TRQ}_n[S_n, A_n]$ , Steps$_{\text{tr}}$ = RL_Sort($s_{\text{tr}}$ , $\text{TRQ}_n[S_n, A_n]$)
(13)    NonTrans_Tr_Steps = NonTrans_Tr_Steps + Steps$s_{\text{nt}}$
(14)    Trans_Tr_Steps = Trans_Tr_Steps + Steps$s_{\text{tr}}$
(15)    Sort $n!$ lists in $S_n$ by $\text{NRQ}_n$, compute the average $\text{Avg}_{\text{nt}}$ and pick the list with max value as $s_{\text{nt}}$
(16)    Sort $n!$ lists in $S_{\text{n}}$ by $\text{TRQ}_n$, compute the average $\text{Avg}_{\text{tr}}$ and pick the list with max value as $s_{\text{tr}}$
(17)    if ($|\text{Avg}_{\text{nt}} - \text{Avg}_{\text{tr}}|/\text{Avg}_{\text{tr}} <= 0.1$) or ($\text{Avg}_{\text{nt}} <=$ upper_bound and $\text{Avg}_{\text{tr}} <=$ upper_bound)
(18)      finish = TRUE
(19)  until finish is TRUE
```

ALGORITHM 2: The algorithm for training the non-transfer and transfer RL methods.

Table 1: Detailed training results of nontransfer and transfer methods to solve sorting 5 numbers for 30 episodes.

			$n = 5$			
	NonTrans_Tr_Steps	Trans_Tr_Steps	Ratio_Tr_Steps	NonTrans_Br_Capacity	Trans_Br_Capacity	Ratio_Br_Capacity
0	167	20	8.35	0.1983	0.1500	1.32
1	215	94	2.29	0.1808	0.2342	0.77
2	964	365	2.64	0.2808	0.2142	1.31
3	207	42	4.93	0.2025	0.1533	1.32
4	94	120	0.78	0.1817	0.1717	1.06
5	189	110	1.72	0.1633	0.1825	0.89
6	361	22	16.41	0.2092	0.1783	1.17
7	146	22	6.64	0.1675	0.1642	1.02
8	94	335	0.28	0.1892	0.2083	0.91
9	382	118	3.24	0.2208	0.1742	1.27
10	230	118	1.95	0.1817	0.1850	0.98
11	78	32	2.44	0.1225	0.1775	0.69
12	276	48	5.75	0.1825	0.1517	1.20
13	130	60	2.17	0.2067	0.1525	1.36
14	320	64	5.00	0.2242	0.1658	1.35
15	241	96	2.51	0.1875	0.1883	1.00
16	286	38	7.53	0.2075	0.1633	1.27
17	246	128	1.92	0.2058	0.2042	1.01
18	140	114	1.23	0.1867	0.1567	1.19
19	249	46	5.41	0.3217	0.1575	2.04
20	130	532	0.24	0.1775	0.2183	0.81
21	154	10	15.40	0.0983	0.1000	0.98
22	10	36	0.28	0.0558	0.1042	0.54
23	456	175	2.61	0.2242	0.1842	1.22
24	101	12	8.42	0.0717	0.1175	0.61
25	400	84	4.76	0.1775	0.1183	1.50
26	117	109	1.07	0.0983	0.1283	0.77
27	241	113	2.13	0.1433	0.1633	0.88
28	184	50	3.68	0.2008	0.1858	1.08
29	102	56	1.82	0.1583	0.1617	0.98

Table 2: Detailed training results of nontransfer and transfer methods to solve sorting 6 numbers for 30 episodes.

			$n = 6$			
	NonTrans_Tr_Steps	Trans_Tr_Steps	Ratio_Tr_Steps	NonTrans_Br_Capacity	Trans_Br_Capacity	Ratio_Br_Capacity
0	936	417	2.24	0.0598	0.0603	0.99
1	1020	508	2.01	0.0878	0.0719	1.22
2	1203	684	1.76	0.1020	0.0725	1.41
3	1253	411	3.05	0.0801	0.0647	1.24
4	750	241	3.11	0.0620	0.0456	1.36
5	1446	1344	1.08	0.1035	0.1137	0.91
6	871	142	6.13	0.0612	0.0395	1.55
7	1386	476	2.91	0.0878	0.0650	1.35
8	972	565	1.72	0.0708	0.0717	0.99
9	1272	752	1.69	0.0874	0.0746	1.17
10	857	426	2.01	0.0697	0.0560	1.24
11	1175	3850	0.31	0.1052	0.1300	0.81
12	1199	563	2.13	0.0882	0.0673	1.31
13	945	543	1.74	0.0809	0.0710	1.14
14	1634	915	1.79	0.1110	0.0873	1.27
15	1281	944	1.36	0.0971	0.0956	1.02
16	970	628	1.54	0.0847	0.0780	1.09
17	1070	428	2.50	0.0719	0.0593	1.21
18	3918	4929	0.79	0.1569	0.1664	0.94
19	1578	955	1.65	0.1133	0.0908	1.25
20	857	203	4.22	0.0530	0.0437	1.21

Table 2: Continued.

	n = 6					
	NonTrans_Tr_Steps	Trans_Tr_Steps	Ratio_Tr_Steps	NonTrans_Br_Capacity	Trans_Br_Capacity	Ratio_Br_Capacity
21	1461	1008	1.45	0.1050	0.0975	1.08
22	743	364	2.04	0.0639	0.0534	1.20
23	1299	633	2.05	0.0866	0.0734	1.18
24	1665	686	2.43	0.1037	0.0734	1.41
25	4582	1216	3.77	0.1469	0.1098	1.34
26	945	695	1.36	0.0768	0.0680	1.13
27	4021	1201	3.35	0.1384	0.1086	1.27
28	942	474	1.99	0.0737	0.0597	1.23
29	1453	1276	1.14	0.1109	0.1165	0.95

Table 3: Detailed training results of nontransfer and transfer methods to solve sorting 7 numbers for 30 episodes.

	n = 7					
	NonTrans_Tr_Steps	Trans_Tr_Steps	Ratio_Tr_Steps	NonTrans_Br_Capacity	Trans_Br_Capacity	Ratio_Br_Capacity
0	7444	3725	2.00	0.0575	0.0476	1.21
1	17430	10013	1.74	0.0895	0.0761	1.18
2	10969	3716	2.95	0.0605	0.0494	1.22
3	11175	2908	3.84	0.0541	0.0420	1.29
4	9032	2744	3.29	0.0514	0.0417	1.23
5	3097	731	4.24	0.0257	0.0233	1.10
6	16747	15702	1.07	0.0830	0.0868	0.96
7	6947	4555	1.53	0.0566	0.0524	1.08
8	5964	3726	1.60	0.0502	0.0488	1.03
9	4137	1214	3.41	0.0290	0.0273	1.06
10	11132	12738	0.87	0.0710	0.0782	0.91
11	6983	9039	0.77	0.0594	0.0660	0.90
12	7727	1751	4.41	0.0408	0.0316	1.29
13	12421	28476	0.44	0.0877	0.1083	0.81
14	14832	25429	0.58	0.0920	0.1187	0.77
15	12450	7392	1.68	0.0769	0.0689	1.12
16	10787	6533	1.65	0.0539	0.0529	1.02
17	8659	22808	0.38	0.1045	0.1001	1.04
18	7670	2634	2.91	0.0428	0.0387	1.10
19	8086	9071	0.89	0.0615	0.0659	0.93
20	9687	6631	1.46	0.0556	0.0553	1.00
21	2474	580	4.27	0.0314	0.0296	1.06
22	10906	15964	0.68	0.0664	0.0851	0.78
23	11889	5882	2.02	0.0587	0.0514	1.14
24	5962	4259	1.40	0.0478	0.0493	0.97
25	19346	13054	1.48	0.0886	0.0767	1.16
26	5705	3114	1.83	0.0468	0.0396	1.18
27	7431	12660	0.59	0.0642	0.0762	0.84
28	3096	927	3.34	0.0249	0.0246	1.01
29	4668	953	4.90	0.0337	0.0257	1.31

NonTrans_Tr_Steps and Trans_Tr_Steps for the transfer method. These two variable names are used in Algorithm 2 as well. Subsequently, we apply similar abbreviations to the brain capacity and denote them by NonTrans_Br_Capacity and Trans_Br_Capacity. NonTrans_Br_Capacity is calculated as the ratio of Q-values which have been updated in the $NRQ_n[A_n, S_n]$ and Trans_Br_Capacity is the percentage of updated Q-values in the $TRQ_n[S_n, A_n]$. The detailed results are reported in Tables 1–4 for different n. Looking at the comparison of the total number of training steps, we can see that the values of NonTrans_Tr_Steps and Trans_Tr_Steps increase significantly when n increases. It is worth noting that some of these two values are less than 100 when n is 5. Therefore, instead of using the latest 100 episodes to check the success rate mentioned in Section 3. B, we opt for the latest 10 episodes to examine that. Regarding the comparison of the brain capacity, the values of NonTrans_Br_Capacity and Trans_Br_Capacity are generally smaller than 0.25 and their values are almost less than 0.1 while n is greater than 6. This implies that the knowledge requirement only occupies a small portion of the Q-table in order to solve the sorting task.

For each episode, we also calculate the ratio of the total number of training steps (Ratio_Tr_Steps) as the division of NonTrans_Tr_Steps by Trans_Tr_Steps and the ratio of the

Table 4: Detailed training results of nontransfer and transfer methods to solve sorting 8 numbers for 30 episodes.

	$n = 8$					
	NonTrans_Tr_Steps	Trans_Tr_Steps	Ratio_Tr_Steps	NonTrans_Br_Capacity	Trans_Br_Capacity	Ratio_Br_Capacity
0	82101	92246	0.89	0.0624	0.0710	0.88
1	77386	83553	0.93	0.0606	0.0689	0.88
2	32674	17731	1.84	0.0358	0.0452	0.79
3	19818	17490	1.13	0.0340	0.0451	0.75
4	24449	11835	2.07	0.0336	0.0443	0.76
5	34761	27067	1.28	0.0399	0.0487	0.82
6	30299	12635	2.40	0.0348	0.0448	0.78
7	26920	14774	1.82	0.0351	0.0448	0.78
8	53885	44233	1.22	0.0479	0.0546	0.88
9	21150	6778	3.12	0.0293	0.0439	0.67
10	56551	73505	0.77	0.0533	0.0650	0.82
11	47152	43085	1.09	0.0477	0.0544	0.88
12	57590	51508	1.12	0.0505	0.0569	0.89
13	21072	9521	2.21	0.0332	0.0457	0.73
14	57659	36219	1.59	0.0445	0.0500	0.89
15	64347	41492	1.55	0.0523	0.0546	0.96
16	31041	15146	2.05	0.0356	0.0443	0.80
17	72028	72386	1.00	0.0594	0.0678	0.88
18	42305	9869	4.29	0.0393	0.0441	0.89
19	29735	20833	1.43	0.0353	0.0468	0.75
20	50376	46719	1.08	0.0485	0.0559	0.87
21	29481	11004	2.68	0.0452	0.0439	1.03
22	37180	32229	1.15	0.0415	0.0500	0.83
23	42596	25663	1.66	0.0400	0.0473	0.85
24	30466	14245	2.14	0.0337	0.0456	0.74
25	62672	60769	1.03	0.0515	0.0593	0.87
26	57160	55132	1.04	0.0520	0.0538	0.97
27	27488	12747	2.16	0.0466	0.0447	1.04
28	28282	19243	1.47	0.0375	0.0410	0.91
29	34866	26506	1.32	0.0435	0.0477	0.91

brain capacity (Ratio_Br_Capacity) as the division of NonTrans_Br_Capacity by Trans_Br_Capacity. For the value of Ratio_Tr_Steps, there are nine numbers greater than or equal to 5.00 when n equals 5. But, as n increases, this phenomenon does not appear and the transfer effects diminish. For the value of Ratio_Br_Capacity, the range is much narrower and is largely concentrated between 0.75 and 1.25. As described in Algorithm 2, both nontransfer and transfer methods are required to have very close training levels in order to finish a training episode. Since close training level means that two methods have similar abilities and performance for sorting $n!$ lists, this could explain why the value of Ratio_Br_Capacity is around 1. In general, transfer method exhibits better performance in terms of training steps. However, in some cases, Ratio_Tr_Steps is smaller than 1, which means nontransfer method takes less steps to complete the training. Since both methods require similar size of the brain capacity to sort $n!$ lists, there may be possibilities that the transfer model exploits the transferred knowledge but does not explore enough to expand its knowledge. This will lead to take more training steps to finish the training process.

To explore the distribution of the Ratio_Tr_Steps and Ratio_Br_Capacity, boxplots are presented in Figures 2 and 3 to do the statistical analyses. A boxplot represents the minimum, 25th percentiles, median, 75th percentiles, and

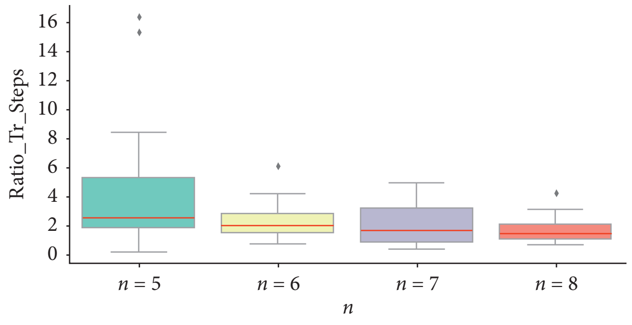


Figure 2: The boxplots of Ratio_Tr_Steps for different n.

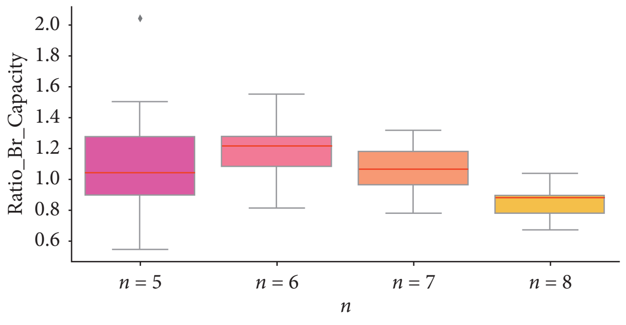


Figure 3: The boxplots of Ratio_Br_Capacity for different n.

Table 5: The averages of Ratio_Tr_Steps and Ratio_Br_Capacity for different n.

Average item	$n = 5$	$n = 6$	$n = 7$	$n = 8$
Ratio_Tr_Steps	4.12	2.26	2.07	1.65
Ratio_Br_Capacity	1.08	1.18	1.06	0.85

maximum of the given dataset. In Figure 2, we observe that the medians of the Ratio_Tr_Steps, which are the red lines inside the box, gradually decrease when n increases. This is in accordance with our previous observation that the growth of n may lower the transfer effects. In Figure 3, the medians of the Ratio_Br_Capacity all occur around 1.00 mostly aligning with our previous conjecture. In addition to the statistics in boxplots, we also compute the averages of Ratio_Tr_Steps and Ratio_Br_Capacity in Table 5. The average performance shows very similar trends as the boxplots.

4. Conclusions

It is reported from prior research that the Q-learning-based approach for the sorting problem requires a large number of training steps. Since the transfer learning method is able to share the knowledge learned from the source domains with the target domain, we devised a transfer scheme to investigate the time cost and knowledge usage issues between nontransfer and transfer models. The Q-table obtained from the prior task is served as the knowledge source to be transferred to the next task. We chose the sorting problem as our case study to analyse two important performance metrics, number of training steps and brain capacity. As a result of the experiment, the brain capacity for two models will be similar after reaching a similar training level. The difference of the total number of training steps between two models will be significant when n is smaller. However, as n increases, the proportion of the transferred knowledge will be smaller and the difference will become less pronounced, making the transfer effect insignificant.

As shown in Table 4, the maximum number of total training steps is close to 100,000 while n equals 8. It would be necessary to enable faster learning in order to handle larger n. Future work will therefore be concerned with the reduction of the state space. State abstraction [22, 23] with the ability to leverage the knowledge learned from prior experiences is worth the effort to improve the scalability of the current approach. Another area of future work is to extend the current tabular representation approach to the deep learning-based methods in order to improve the learning stability and computational efficiency.

Data Availability

No data were used to support the findings of the study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors are grateful to Timothy Kitterman for helpful discussions.

References

[1] R. Li, Z. Zhao, Q. Sun et al., "Deep reinforcement learning for resource management in network slicing," *IEEE Access*, vol. 6, pp. 74429–74441, 2018.

[2] S. Tschiatschek, K. Arulkumaran, J. Stühmer, and K. Hofmann, "Variational inference for data-efficient model learning in pomdps," 2018, https://arxiv.org/pdf/1805.09281.pdf.

[3] P. Malekzadeh, M. Salimibeni, A. Mohammadi, A. Assa, and K. N. Plataniotis, "MM-KTD: multiple model kalman temporal differences for reinforcement learning," *IEEE Access*, vol. 8, pp. 128716–128729, 2020.

[4] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[5] G. A. Rummery and M. Niranjan, *On-Line Q-Learning Using Connectionist Systems*, Department of Engineering, University of Cambridge, England, UK, 1994, https://www.researchgate.net/profile/Mahesan_Niranjan/publication/2500611_On-Line_Q-Learning_Using_Connectionist_Systems/links/5438d5db0cf204cab1d6db0f/On-Line-Q-Learning-Using-Connectionist-Systems.pdf.

[6] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1746–1751, Doha, Qatar, October 2014.

[7] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[8] J. Farebrother, M. C. Machado, and M. Bowling, "Generalization and regularization in DQN," 2018, https://arxiv.org/pdf/1810.00123.pdf.

[9] H. V. Hasselt, "Double Q-learning," in *Advances in Neural Information Processing Systems*, pp. 2613–2621, Massachusetts Institute of Technology Press, Cambridge, MA, USA, 2010, http://papers.nips.cc/paper/3964-double-q-learning.pdf.

[10] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, Phoenix, AZ, USA, 2016, March, https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/download/12389/11847.

[11] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan, and P. Li, "A double deep Q-learning model for energy-efficient edge scheduling," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 739–749, 2019.

[12] Y. Zhang, P. Sun, Y. Yin, L. Lin, and X. Wang, "Human-like autonomous vehicle speed control by deep reinforcement learning with double Q-learning," in *Proceedings of the 2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1251–1256, Changshu, China, June 2018.

[13] Y. Keneshloo, N. Ramakrishnan, and C. K. Reddy, "Deep transfer reinforcement learning for text summarization," in *Proceedings of the 2019 SIAM International Conference on Data Mining*, pp. 675–683, Calgary, Canada, May 2019.

[14] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in Neural Information Processing Systems*, pp. 3320–3328, MIT Press, Cambridge, MA, USA, 2014, http://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.pdf.

[15] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the NAACL-HLT*, Minneapolis, MN, USA, January 2019.

[16] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Advances in Neural Information Processing Systems*, pp. 4077–4087, MIT Press, Cambridge, MA, USA, 2017, http://papers.nips.cc/paper/6996-prototypical-networks-for-few-shot-learning.pdf.

[17] K. Fu, T. Zhang, Y. Zhang et al., "Meta-SSD: towards fast adaptation for few-shot object detection with meta-learning," *IEEE Access*, vol. 7, pp. 77597–77606, 2019.

[18] C. Lin, J. Jwo, C. Lee, and Y. Lo, "Empirical explorations of strategic reinforcement learning: a case study in the sorting problem," *Proceedings of the Estonian Academy of Sciences*, vol. 69, no. 3, pp. 186–196, 2020.

[19] H. Van Seijen, H. Van Hasselt, S. Whiteson, and M. Wiering, "A theoretical and empirical analysis of expected Sarsa," in *Proceedings of the 2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pp. 177–184, Nashville, TN, USA, March 2009.

[20] A. Faust, J. B. Aimone, C. D. James, and L. Tapia, "Resilient computing with reinforcement learning on a dynamical system: case study in sorting," in *Proceedings of the 2018 IEEE Conference on Decision and Control (CDC)*, pp. 5999–6006, Miami Beach, FL, USA, December 2018.

[21] E. Rodrigues Gomes and R. Kowalczyk, "Dynamic analysis of multiagent Q-learning with ε-greedy exploration," in *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 369–376, Montreal, Canada, June 2009.

[22] D. Abel, D. Hershkowitz, and M. Littman, "Near optimal behavior via approximate state abstraction," in *Proceedings of the International Conference on Machine Learning*, pp. 2915–2923, June 2016, http://proceedings.mlr.press/v48/abel16.pdf.

[23] D. Abel, D. Arumugam, L. Lehnert, and M. Littman, "State abstractions for lifelong reinforcement learning," in *Proceedings of the International Conference on Machine Learning*, pp. 10–19, July 2018, http://proceedings.mlr.press/v80/abel18a/abel18a.pdf.

Hindawi
Computational Intelligence and Neuroscience
Volume 2020, Article ID 8896570, 14 pages
https://doi.org/10.1155/2020/8896570

Research Article

Binary Political Optimizer for Feature Selection Using Gene Expression Data

Ghaith Manita [1,2] and Ouajdi Korbaa [1]

[1] *Laboratory MARS, LR17ES05, ISITCom, University of Sousse, Sousse, Tunisia*
[2] *ESEN, University of Manouba, Manouba, Tunisia*

Correspondence should be addressed to Ghaith Manita; gaieth.manita@gmail.com

Received 9 September 2020; Revised 2 November 2020; Accepted 16 November 2020; Published 29 November 2020

Academic Editor: Wassim Ayadi

DNA Microarray technology is an emergent field, which offers the possibility of obtaining simultaneous estimates of the expression levels of several thousand genes in an organism in a single experiment. One of the most significant challenges in this research field is to select high relevant genes from gene expression data. To address this problem, feature selection is a well-known technique to eliminate unnecessary genes in order to ensure accurate classification results. This paper proposes a binary version of Political Optimizer (PO) to solve feature selection problem using gene expression data. Two transfer functions are used to design a binary PO. The first one is based on Sigmoid function and will be noted as BPO-S, while the second one is based on V-shaped function and will be noted as BPO-V. The proposed methods are evaluated using 9 biological datasets and compared with 8 binary well-known metaheuristics. The comparative results show the prevalent performance of the BPO methods especially BPO-V in comparison with other techniques.

1. Introduction

Molecular biology research evolves through the development of technologies used to carry them out. It is not possible to investigate a countless number of genes using conventional strategies. DNA Microarray is a technology that allows researchers to investigate and treat problems that were once considered untraceable. The expression of many genes can be examined in a solitary response rapidly and productively. DNA Microarray technology is enabling the scientific community to understand the fundamental aspects underlying the growth and development of life, as well as to investigate the hereditary reasons for irregularities in the working of the human body.

Therefore, microarray technology remains to this day a useful asset for measuring of gene expression. Beyond the technology itself, the analysis of the data from microarrays is a complex statistical problem. And this is due to the large number of genes and the complexity of biological networks which increase the challenges of understanding and interpreting the resulting mass of data, which often consists of millions of measurements. Hence, extracting relevant biological knowledge from microarray data turns into a hard task due to the curse of dimensionality problem [1].

Generally, gene expression data are often redundant and noisy with large number of genes. In order to reduce the dimensionality of such datasets by selecting the most informative features, Feature Selection (FS) procedure seems to be an essential preprocessing phase before the implementation of machine learning classifiers in order to minimize training times and memory requirements [2].

Feature selection methods are classified into three categories based on the evaluation criteria used: filter, wrapper, and embedded [3]. This categorization depends on the involvement of a learning algorithm in the used approach.

The filter methods (Chi-Square [4], Information Gain [5], Gain Ratio [6], and ReliefF [7]) select a subset of variables by preprocessing the data from a model. The selection process is independent of the classification process. One of the advantages is that it is completely independent of the data model we are trying to build. It proposes a satisfactory subset of variables to explain the structure of the hidden data and that the subset is independent of the chosen learning algorithm. On the contrary, wrapper methods aim to generate representative subsets and evaluate them using a classification algorithm. This evaluation is carried out by calculating a score, e.g., a score of a set will be a compromise between the number of variables eliminated and the success rate of the classification on a test set. Therefore, wrapper methods are more exact than the filter approaches since they consider the relations among the features. Another advantage is its conceptual simplicity; we do not need to understand how induction is affected by the selection of variables, just generate and test. Nevertheless, the computational cost is significantly increased and depends on the used learning algorithm [8]. Finally, embedded methods integrate selection directly into the learning process, and decision trees are the most emblematic illustration. However, we classify in this group all techniques that evaluate the importance of a variable in coherence with the criterion used to evaluate the overall relevance of the model. They are generally known by their reasonable trade-off between efficiency and computing costs [9, 10].

FS is regarded as an NP-complete combinatorial optimization problem [11]. The search space size is strongly dependent to the increase of the number of features in the studied dataset. An exhaustive search for the optimal relevant feature often leads to stagnation in local optima [12]. Therefore, metaheuristic methods are potentially more suitable to deal with this problem because of their ability to find acceptable solutions in reasonable periods of time [13]. The objective function may be the accuracy of the classification or another criterion that could consider the best compromise between the computational burden of attribute extraction and efficiency [14]. Metaheuristics are stochastic approaches and fall into two categories: population-based approaches and single-solution approaches [14, 15]. Generally, they are inspired by nature, social behavior, biological behavior of animals or birds or insects, physical or chemical phenomena, etc.

In the literature, many works were introduced in order to implement stochastic methods to address the FS problem, such as Simulated Annealing (SA) [16], Tabu Search (TS) [17, 18], Genetic Algorithm (GA) [19–22], Particle Swarm Optimization (PSO) [23, 24], Ant Colony Optimization (ACO) [25, 26], Artificial Bee Colony (ABC) [27, 28], and Differential Evolution (DE) [29, 30].

Generally, these traditional methods suffer from a slow convergence rate, and they have a lot number of parameters to be tuned. Hence, a simple and efficient global search technique is needed. For that, during this work, we use the Policy Optimizer (PO) [31] as the main resolution technique since it is a newly introduced metaheuristic which is human behavior-based algorithm. Moreover, as mentioned in [31], PO produces better solutions for dealing with optimization problems than other well-known metaheuristics in the literature. In this paper, a novel binary version is proposed to find the most representative subset of a given dataset. The binary version introduced here is performed using two different transfer functions.

The structure of this paper is as follows: the standard (continuous) version of Political Optimizer (PO) is presented in Section 2. In Section 3, we introduce the binary version of the latter algorithm called BPO. The obtained results and conducted comparisons are reported in Section 4. Finally, the conclusion and several directions for future papers are stated in Section 5.

2. Overview of the Political Optimizer (PO)

Political Optimizer is a newly proposed metaheuristic based on human behavior and inspired by the multiphased political process. However, it should be noted that the proposed algorithm is not the first of this kind. In PO, the concept of politics is mapped from a different perspective and unlike the recent politics-inspired algorithms, and this is due to four reasons. First, PO tries to model all the important steps in politics such as party formation, party-ticket/constituency allocation, election campaign and party switching, interparty election, and parliamentary affairs after government formation. Second, PO introduces a novel position updating strategy called recent past-based position updating strategy (RPPUS). This latter represents the learning behavior of politicians from the previous election. Third, each individual solution assumes a double job: a party member and an election candidate. Using this concept, each solution can be updated according to two better solutions: the party leader and the constituency winner. Finally, to improve the results, intermediary solutions needs to cooperate and communicate via a phase named parliamentary affairs.

In PO, each party member is viewed as a candidate solution where its goodwill is considered the position in the search space. Moreover, the evaluation function is computed during the election phase where the number of votes obtained by each member party represents the fitness of the candidate solution.

Political Optimizer (PO) is formed by five main phases as follows: party formation and constituency allocation, election campaign, party switching, interparty election, and parliamentary affairs. It should be mentioned that the first phase (party formation and constituency allocation) is executed only one time to initialize and affect different variables. However, the remaining phases are running in loop, as detailed in Algorithm 1. The used variables in PO are summarized in Table 1.

2.1. Party Formation and Constituency Allocation. In the beginning, the population P is partitioned in N parties, where each party P_i includes N members (potential solution). Moreover, each jth member is noted as P_i^j and represented by a d-dimensional vector, where the value d is the number of input variables of the treated problem and $P_{i,k}^j$ is kth dimension of P_i^j.

As mentioned before, each member is considered as an election candidate besides its role as a party member. Hence, N constituencies are formed and contain jth member of each contesting party. This division is illustrated in Figure 1. Furthermore, the leader of the ith party after computing the

```
Input: $n$ (number of constituencies, political parties and party members), $\lambda_{\max}$ (upper limit of the party switching rate), $T_{\max}$ (total number of iterations)
Output: final population $\mathscr{P}(T_{\max})$
/* Initialization/*
Initialize $(n*n)$ candidate members $P$
compute the fitness of each member $p_i^j$
compute the set of the party leaders $P^*$ and the set of the constituency winners $C^*$, by using equation (3)
$t = 1$;
$P(t-1) = P$;
$F(P(t-1)) = f(P)$;
$\lambda = \lambda_{\max}$;
while $t \le T_{\max}$ do
  $P_{\text{temp}} = P$;
  $f(P_{\text{temp}}) = f(P)$
  foreach $P_i \in P$ do
    foreach $p_i^j \in P_i$ do
      $p_i^j = \text{ElectionCampaign}(p_i^j, p_i^j(t-1), p_i^j c_j^*)$;
    end
  end
  PartySwitching $(P, \lambda)$;
  /* Election phase */
  compute the fitness of each member $p_i^j$
  compute the set of the party leaders $P^*$ and the set of the constituency winners $C^*$, by using equation (3)
  Parliamentary Affairs $(C^*, P)$;
  $P(t-1) = P_{\text{temp}}$;
  $F(P(t-1)) = f(P_{\text{temp}})$;
  $\lambda = (\lambda - \lambda_{\max}/T_{\max})$;
  $t = t + 1$;
end
```

Algorithm 1: Pseudocode of PO.

Table 1: List of the used variables.

Variable	Description
P	Set of all political parties (whole population)
P_i	ith political party
P_i^j	jth member of ith party
$P_{i,k}^j$	kth dimension of jth member of ith political party
C	Set of all constituencies
C_j	jth constituency
P_i^*	Leader of ith political party
C_j^*	Winner of jth constituency
λ	Party switching rate
N	Number of parties, constituencies, and members in each party
$T_{\max}$	Total number of iterations

fitness of all member is noted as P_i^* and the set of all the party leaders is represented by P^*. On the contrary, after the election, C^* regroups the winners from all the constituencies named the parliamentarians, where C_j^* denotes the winner of jth constituency.

2.2. Election Campaign. During this phase, party members are trying to enhance their chances of being elected by changing their positions according to three aspects. First, they try to learn from previous experience using a novel position updating strategy called recent past-based position updating strategy (RPPUS), as formulated in equations (1) and (2). Second, each party member is trying to update his current position according to the party leader. Finally, candidate positions are updated with reference to the constituency winner:

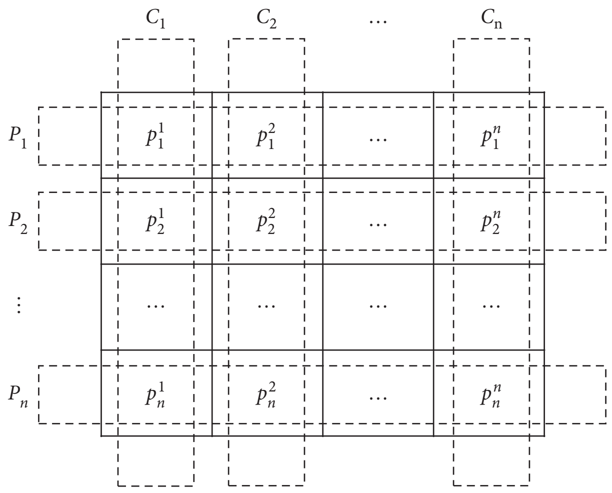


Figure 1: Illustration of the logical division of the population P in political parties and constituencies [31].

$$P_{i,k}^{j}(t+1)=\begin{cases} m^{*}+r\left(m^{*}-P_{i,k}^{j}(t)\right), & \text{if } P_{i,k}^{j}(t-1)\le P_{i,k}^{j}(t)\le m^{*} \text{ or } P_{i,k}^{j}(t-1)\ge P_{i,k}^{j}(t)\ge m^{*},\\ m^{*}+(2r-1)\left|m^{*}-P_{i,k}^{j}(t)\right|, & \text{if } P_{i,k}^{j}(t-1)\le m^{*}\le P_{i,k}^{j}(t) \text{ or } P_{i,k}^{j}(t-1)\ge m^{*}\ge P_{i,k}^{j}(t),\\ m^{*}+(2r-1)\left|m^{*}-P_{i,k}^{j}(t-1)\right|, & \text{if } m^{*}\le P_{i,k}^{j}(t-1)\le P_{i,k}^{j}(t) \text{ or } m^{*}\ge P_{i,k}^{j}(t-1)\ge P_{i,k}^{j}(t),\end{cases} \tag{1}$$

$$P_{i,k}^{j}(t+1)=\begin{cases} m^{*}+(2r-1)\left|m^{*}-P_{i,k}^{j}(t)\right|, & \text{if } P_{i,k}^{j}(t-1)\le P_{i,k}^{j}(t)\le m^{*} \text{ or } P_{i,k}^{j}(t-1)\ge P_{i,k}^{j}(t)\ge m^{*},\\ P_{i,k}^{j}(t-1)+r\left(P_{i,k}^{j}(t)-P_{i,k}^{j}(t-1)\right), & \text{if } P_{i,k}^{j}(t-1)\le m^{*}\le P_{i,k}^{j}(t) \text{ or } P_{i,k}^{j}(t-1)\ge m^{*}\ge P_{i,k}^{j}(t),\\ m^{*}+(2r-1)\left|m^{*}-P_{i,k}^{j}(t-1)\right|, & \text{if } m^{*}\le P_{i,k}^{j}(t-1)\le P_{i,k}^{j}(t) \text{ or } m^{*}\ge P_{i,k}^{j}(t-1)\ge P_{i,k}^{j}(t).\end{cases} \tag{2}$$

According to Algorithm 2, which describes the whole process of election campaign, the relationship between current fitness and the previous fitness is the main factor to choose between using equations (1) or (2).

2.3. Party Switching. In order to balance between exploration and exploitation, a phase called party switching is started after the election campaign phase. Using an adaptive parameter λ named party switching rate, each party member P_i^j can be selected and switched to some randomly chosen party P_r. Hence, it is swapped with the least fit member of the party P_r, as presented in Algorithm 3.

2.4. Election. This phase aims to evaluate the fitness of all candidates contesting in constituency. After that, the party leaders and constituency winners are updated as follows:

$$\begin{aligned} q &= \arg\min f\left(P_i^j\right), \quad 1\le i\le N,\\ C_j^{*} &= P_q^i,\\ P_j^{*} &= P_q^i. \end{aligned} \tag{3}$$

2.5. Parliamentary Affairs. After determining the party leaders and constituency winners (parliamentarians), each parliamentarian aims to improve his performance in order to mimic the interaction and cooperation of the winning candidates to run the government in the postelection phase. This process is presented in Algorithm 4, where each parliamentarian C_j^{*} updates its position in relation to randomly chosen parliamentarian C_r^{*}. It should be noted that the movement is applied only if the performance of C_j^{*} is enhanced.

3. Binary Political Optimizer (BOP)

As mentioned before, political member's goodwill is considered as a candidate position and moves in the search space towards continuous-valued positions. However, in binary optimization problems, such as feature selection, the search space is modelled as a n-dimensional Boolean lattice, and political member's goodwill needs to be represented by binary vectors.

In order to convert a continuous algorithm to a binary version, we should utilize transfer functions (TF), and it considered as the most efficient and convenient way [32]. Transfer functions are classified into two categories according to their shapes: S-shaped and V-shaped, as illustrated in Figure 2.

In this work, two versions are proposed, based on the transfer function used. In the first one, the political member's goodwill is updated using the Sigmoid function (S-shaped) and called BPO-S. While, in the second one, we used the Hyperbolic Tangent transfer function, called BPO-V.

Without any modification in the previously detailed phases, only two steps are integrated after the continuous computation. The first step is to calculate the probability of changing a position's element to 0 or 1 according to the following equation:

$$P\left(x_d^i(t)\right)=\mathrm{TF}\left(x_d^i(t)\right), \tag{4}$$

where TF is the used transfer function that could be Sigmoid (equation (5)) or Hyperbolic Tangent (equation (6)) and $x_d^i(t)$ is the ith political member in the dth in the iteration t:

$$\mathrm{TF}(x)=\frac{1}{1+e^{-x}}, \tag{5}$$

$$\mathrm{TF}(x)=|\tanh(x)|. \tag{6}$$

In the second step, the probability computed by equation (4) is then inserted in equation (7) in order to convert continuous value of each member position to 0 or 1:

$$x_d^i(t)=\begin{cases} 1, & \text{if } P\left(x_d^i(t)\right)\ge \text{rand},\\ 0, & \text{otherwise,}\end{cases} \tag{7}$$

where rand is a uniform random number between 0 and 1.

The flowchart of the proposed binary algorithm is presented in Figure 3.

3.1. Binary Political Optimizer Applied for Feature Selection. In this section, we exploited the proposed BPO in feature selection for classification problems. As mentioned before, the feature selection problem is an NP-hard combinatorial binary optimization problem. For a feature vector sized N, the different feature combinations would be $2N$ which increase exponentially the number of possible solutions where an exhaustive search is probably not practical. Therefore, we used the proposed BPO in order to find an acceptable solution with reasonable execution time. The main objective is to maximize the classification accuracy and minimize the

```
Result: $p_i^j(t+1)$ ▷ updated position of $p_i^j$
if $f(p_i^j(t)) \leq f(p_i^j(t-1))$ then
    for $k \longleftarrow 1$ to $d$ do
        $m^* \longleftarrow p_{i,k}^*$ ▷ where $p_i^*$ is the leader of $i$th party
        $r \longleftarrow$ random number from the interval [0, 1]
        ▷ Update the position with respect to the party leader
        $p_{\mathrm{i,k}}^j \longleftarrow$ update $p_{\mathrm{i,k}}^j(t)$ by using equation (1)
        $m^* \longleftarrow c_{j,k}^*$ ▷ where $c_j^*$ is the winner of $j$th constituency
        $r \longleftarrow$ random number from the interval [0, 1]
        ▷ Update the position with repect to the constituency winner
        $p_{i,k}^j(t+1) \longleftarrow$ update $p_{i,k}^j$ by using equation (1)
    end
else
    for $k \longleftarrow 1$ to $d$ do
        $m^* \longleftarrow p_{i,k}^*$
        $r \longleftarrow$ random number from the interval [0, 1]
        ▷ Update the position w.r.t the party leader
        $p_{i,k}^j \longleftarrow$ update $p_{i,k}^j(t)$ by using equation (2)
        $m^* \longleftarrow c_{j,k}^*$
        $r \longleftarrow$ random number from the interval [0, 1]
        ▷ Update the position w.r.t the constituency winner
        $p_{i,k}^j(t+1) \longleftarrow$ update $p_{i,k}^j$ by using equation (2)
    end
end
```

Algorithm 2: ElectionCampaign $(p_i^j, p_i^j(t-1), p_i^j, c_j^*)$.

```
foreach $P_i \in P$ do
    foreach $p_i^j \in P_i$ do
        $sp$ = random number from the interval [0, 1]
        if $sp < \lambda$ then
            $r$ = random integer from the range [1, $n$]
            $q = \arg\max f(p_r^j), 1 \leq j \leq n$
            swap $(p_r^q, p_i^j)$
        end
    end
end
```

Algorithm 3: PartySwitching (P, λ).

```
for $j \longleftarrow 1$ to $n$ do
    $r \longleftarrow$ random integer in the range 1 to $n$, where $r \neq j$
    $a \longleftarrow$ random number from the interval [0, 1]
    $c_{\mathrm{new}}^* \longleftarrow c_r^* + (2a-1)|c_r^* - c_j^*|$
    compute the fitness if $c_{\mathrm{new}}^*$
    if $f(c_{new}^*) \leq c_j^*$ then
        $c_j^* \longleftarrow c_{\mathrm{new}}^*$
        $f(c_j^*) \longleftarrow f(c_{\mathrm{new}}^*)$
        $i \longleftarrow$ party index of the winner of $j$th constituency $p_i^j \longleftarrow c_{\mathrm{new}}^*$
        $f(p_i^j) \longleftarrow f(c_{\mathrm{new}}^*)$
    end
end
```

Algorithm 4: PartySwitching (parliamentary affairs (C^*, P)).

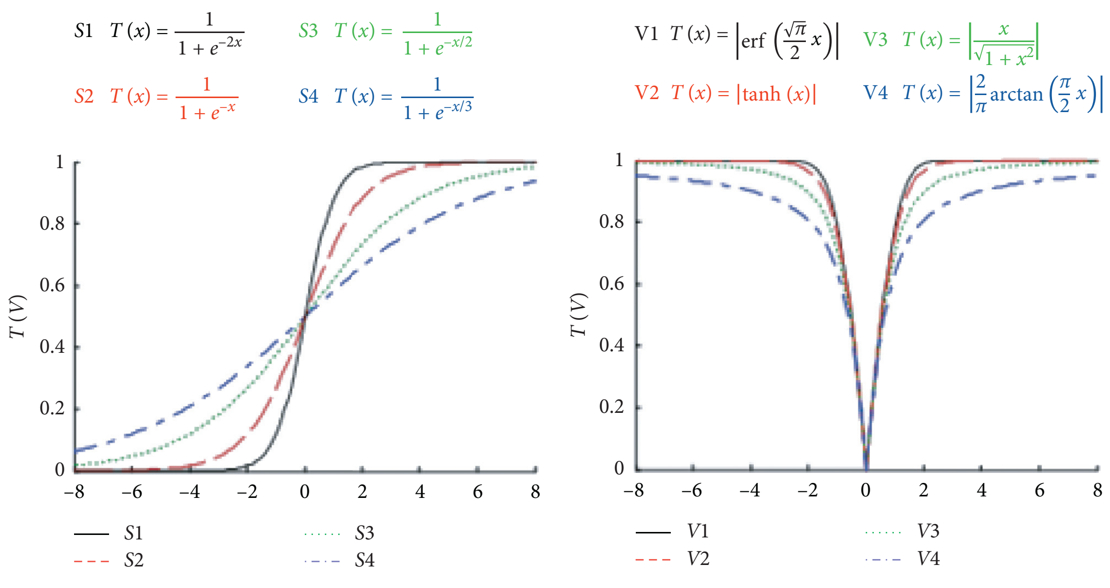


Figure 2: (a) S-shaped and (b) V-shaped family of transfer functions [32].

number of selected features. The used fitness function is presented in the following equation [33]:

$$\uparrow F = \mathrm{Acc} + \omega\left(1 - \frac{sf}{nf}\right), \tag{8}$$

where Acc is the classification accuracy given a chosen classifier, ω is the weight factor which is a value between 0 and 1, sf is the length of selected feature subset, and nf is the total number of features. In this study, we set ω to 0.5 for all the experiments in the next section. For the classifier, we chose to use k-Nearest Neighbor (k-NN) to compute the accuracy of selected subset. Moreover, to ensure the robustness of the obtained results, every used dataset is divided randomly into two different parts: training and testing set, according to 10-fold crossvalidation method.

4. Experimental Results

In this section, all experiments were repeated for 100 independent times to obtain statistically meaningful results. Furthermore, each algorithm was implemented using MATLAB R2020a and was run on an Intel Core i7 machine, 2.6 GHz CPU, and 16 GB of RAM.

4.1. Dataset. In this study, nine benchmark biological datasets are used to assess the performance of the proposed approach [34–44]. Table 2 outlines the datasets used in this work.

4.2. Parameter Settings. To evaluate the proposed model, several experiments were conducted to compare the BPO algorithm with seven different metaheuristic optimization algorithms: Binary Particle Swarm Optimization (BPSO) [45], Binary Genetic Algorithm (BGA) [46], Binary Bat Algorithm (BBA) [47], Binary Differential Evolution (BDE) [48], Binary Grey Wolf Optimizer (BGWO) [49], Binary Atom Search Algorithm (BASO) [50], Binary Harris Hawks Optimizer (BHHO) [51], and Binary Tree Growth Algorithm (BTGA) [52]. The parameters settings for all metaheuristic optimization algorithms are shown in Table 3.

4.3. Results and Discussion. In this section, we start to evaluate statically the performance of the two proposed version of BPO compared to other algorithms. Therefore, four different statistical measures are used to start the first step of evaluation. These measurements were the worst fitness value, the best fitness value, the mean fitness value (avg), and standard deviation (std). Table 4 outlines the obtained results using these measures where the best ones are highlighted in bold text. From the table, we assess the superiority of proposed algorithms, especially BPO-V, compared to others binary version of well-known algorithms. However, BPO-V and BPO-S can be described as unstable methods in most cases. This fact can be explained by the complexity of position update strategy adopted by PO. Furthermore, it can be observed that BASO is the most competitive algorithm with the two version of BPO. From these findings, it can be concluded that BPO-V is better than BPO-S, BGA, BGWO, BBA, BHHO, BDE, BASO, BPSO, and BTGA in extracting the most relevant feature of the tested datasets with the aim to maximize the classification performance and minimization of the number of selected features. This deduction was confirmed by applying a Wilcoxon Ranked Signed Test to the proposed algorithms compared in pairs with the other algorithms. This test is performed with a statistical significance value $\alpha = 0.05$. In Tables 5 and 6, the sign "+" in the winner lines designates that the null hypothesis is rejected and the proposed

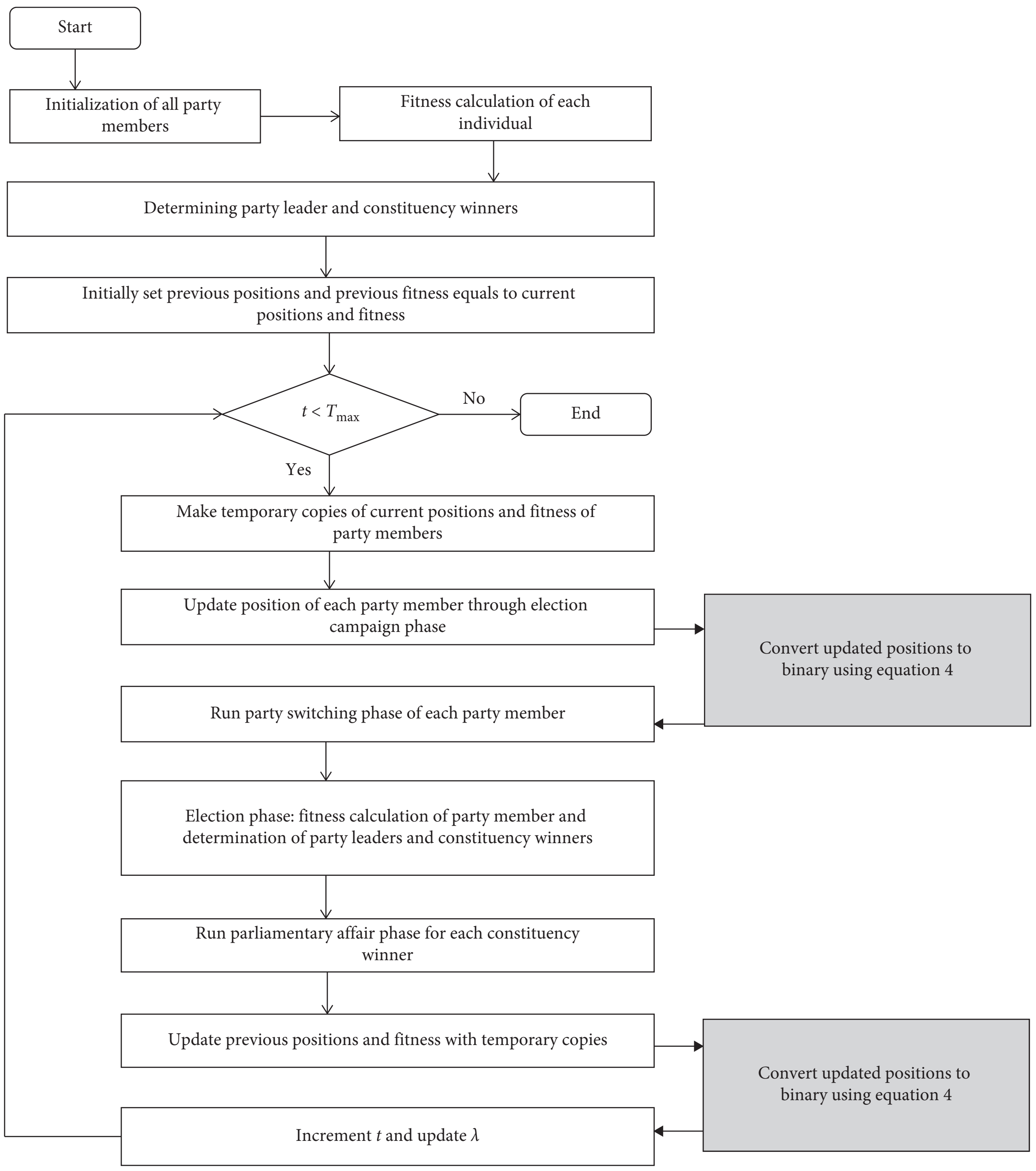

Start
Initialization of all party members
Fitness calculation of each individual
Determining party leader and constituency winners
Initially set previous positions and previous fitness equals to current positions and fitness
$t < T_{\max}$
No
End
Yes
Make temporary copies of current positions and fitness of party members
Update position of each party member through election campaign phase
Convert updated positions to binary using equation 4
Run party switching phase of each party member
Election phase: fitness calculation of party member and determination of party leaders and constituency winners
Run parliamentary affair phase for each constituency winner
Update previous positions and fitness with temporary copies
Convert updated positions to binary using equation 4
Increment t and update λ

FIGURE 3: Flowchart of the proposed algorithm.

TABLE 2: Details of datasets.

Dataset	No. of instances	No. of features	No. of classes	Type
CLL_SUB_111 [34]	111	11340	3	Continuous, multiclass
Colon [35]	62	2000	2	Discrete, binary
Leukemia [36]	72	7070	2	Discrete, binary
Lung [37]	203	3312	5	Continuous, multiclass
Lung_discrete [38]	73	325	7	Discrete, multiclass
Lymphoma [39]	96	4026	9	Discrete, multiclass
nci9 [40, 41]	60	9712	9	Discrete, multiclass
Prostate_GE [42, 43]	102	5966	2	Continuous, binary
SMK_CAN_187 [44]	187	19993	2	Continuous, binary

Table 3: Parameter settings for all used algorithms.

Algorithm	Parameter	Value
BPO	Parties (number of political parties)	5
	Lambda (max limit of party switching rate)	1
BPSO	c1 (cognitive factor)	2
	c2 (social factor)	2
	Vmax (maximum velocity)	6
	Wmax (maximum bound on inertia weight)	0.9
	Wmin (minimum bound on inertia weight)	0.4
BGWO	a	2
BBA	Goma	1
	Alpha	1
	Zigma	1
	Beta	1
	frequencyMin	20
	frequencyMax	50
BDE	CrossRate	0.9
BTGA	N1 (number of trees in first group)	3
	N2 (number of trees in second group)	5
	N4 (number of trees in fourth group)	3
	Tree reduction rate	0.8
	Parameter controls nearest tree	0.5
BHHO	Beta (levywalk)	1.5
BASO	Alpha (depth weight)	50
	Beta (multiplier weight)	0.2
	Vmax (maximum velocity)	6
BGA	crossoverRate	0.9
	mutationRate	0.1
All of them	SearchAgent(Bats, wolfs, particles, . . .)	30
	Maximum iterations	100

Table 4: Experimental result of the fitness function of the proposed algorithms compared to eight metaheuristics.

Dataset		BPO-S	BPO-V	BGA	BGWO	BBA	BHHO	BASO	BDE	BPSO	BTGA
CLL_SUB_111	Best	1.409	1.4217	1.2065	1.1769	1.2059	1.2547	1.4237	1.1254	1.2053	1.208
	Avg	1.2509	1.3254	1.1287	1.0926	1.1239	1.1433	1.2922	1.0498	1.1254	1.119
	Worst	1.0681	1.2528	1.0707	1.0368	1.0701	1.0894	1.2164	0.9753	1.0684	1.0685
	std	0.0698	0.0347	0.0262	0.0288	0.0264	0.0317	0.037	0.0268	0.0282	0.0296
Colon	Best	1.4995	1.4998	1.284	1.2632	1.2715	1.3807	1.4888	1.2637	1.275	1.2712
	Avg	1.4302	1.4922	1.2732	1.2551	1.2639	1.3433	1.4888	1.242	1.269	1.2633
	Worst	1.3308	1.433	1.2637	1.237	1.257	1.3093	1.45	1.2043	1.2635	1.258
	std	0.0527	0.0163	0.0036	0.0052	0.0026	0.0165	0.0086	0.0144	0.0025	0.0024
Leukemia	Best	1.4999	1.4999	1.276	1.2612	1.2627	1.3848	1.4914	1.2576	1.267	1.2637
	Avg	1.458	1.497	1.2708	1.2546	1.2593	1.3613	1.476	1.2526	1.2626	1.2595
	Worst	1.2772	1.4792	1.2649	1.2494	1.2567	1.3397	1.4628	1.2477	1.2597	1.2566
	std	0.0433	0.0039	0.002	0.0022	0.0012	0.0102	0.0064	0.0023	0.0012	0.0015
Lung	Best	1.4967	1.4953	1.2876	1.2634	1.2674	1.3818	1.4857	1.2655	1.2736	1.2714
	Avg	1.4491	1.4758	1.2791	1.2561	1.2631	1.361	1.471	1.254	1.2685	1.2635
	Worst	1.394	1.4502	1.2716	1.2462	1.2597	1.341	1.4576	1.2446	1.2654	1.2601
	std	0.0228	0.0101	0.003	0.0034	0.0016	0.0091	0.0064	0.0036	0.0019	0.002
Lung_discrete	Best	1.4892	1.4938	1.3292	1.2954	1.3062	1.3815	1.48	1.2862	1.3231	1.3062
	Avg	1.3954	1.4257	1.3127	1.2693	1.2866	1.354	1.4611	1.2563	1.3012	1.2881
	Worst	1.2631	1.3409	1.2954	1.2308	1.2754	1.32	1.4292	1.2108	1.2877	1.2754
	std	0.0562	0.0281	0.0072	0.0136	0.0063	0.0124	0.0096	0.0137	0.0067	0.0064
Lymphoma	Best	1.422	1.4389	1.2083	1.161	1.1616	1.2821	1.4268	1.1574	1.1696	1.1644
	Avg	1.3632	1.388	1.1992	1.1549	1.1583	1.2678	1.3785	1.1514	1.1656	1.1599
	Worst	1.1578	1.3489	1.1909	1.1501	1.1563	1.2532	1.3636	1.1465	1.1624	1.1567
	std	0.0358	0.0105	0.0036	0.0023	0.0012	0.006	0.0104	0.0022	0.0016	0.0016

Table 4: Continued.

Dataset		BPO-S	BPO-V	BGA	BGWO	BBA	BHHO	BASO	BDE	BPSO	BTGA
nci9	Best	1.3296	1.3267	1.1735	1.1629	1.1726	1.2093	1.3127	1.0949	1.1728	1.1663
	Avg	1.0929	1.1854	1.0703	1.0431	1.0681	1.0809	1.217	0.9878	1.0679	1.053
	Worst	0.9786	1.0738	1.0061	0.975	1.0022	0.993	1.1225	0.9168	1.0025	1.0006
	std	0.0729	0.0509	0.043	0.0401	0.0403	0.0428	0.0393	0.0375	0.0394	0.0422
Prostate_GE	Best	1.4972	1.4999	1.2674	1.2583	1.2614	1.3822	1.4828	1.2572	1.2649	1.2622
	Avg	1.4272	1.4751	1.2625	1.2458	1.2583	1.3452	1.4644	1.2413	1.2609	1.2584
	Worst	1.2551	1.4378	1.258	1.2182	1.2557	1.3098	1.446	1.2059	1.2575	1.255
	std	0.0452	0.0204	0.0019	0.0114	0.0014	0.0151	0.0089	0.0135	0.0015	0.0016
SMK_CAN_187	Best	1.3883	1.3916	1.1738	1.1665	1.1959	1.261	1.3986	1.1425	1.1969	1.1976
	Avg	1.2885	1.3294	1.1351	1.107	1.134	1.1691	1.3057	1.0722	1.1278	1.1305
	Worst	1.1754	1.276	1.1118	1.0616	1.089	0.0261	1.2642	1.0299	1.089	1.0889
	std	0.043	0.0231	0.017	0.0226	0.0203	0.0261	0.0261	0.0225	0.0197	0.024

Table 5: Pairwise statistical comparison of the BPO-S algorithm with other algorithms using the Wilcoxon signed-rank test ($\alpha = 0.05$).

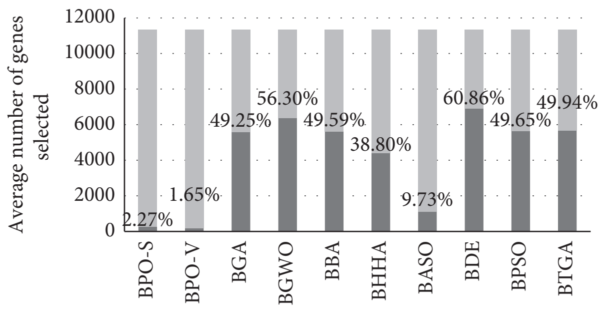
Dataset		BPO-S	BPO-V	BGA	BGWO	BBA	BHHA	BASO	BDE	BPSO	BTGA
CLL_SUB_111	p-value	—	$1.786E-04$	$1.827E-04$	$1.827E-04$	$1.786E-04$	$1.827E-04$	$1.827E-04$	$1.827E-04$	$1.827E-04$	$1.827E-04$
	Winner	—	−	+	+	+	+	−	+	+	+
Colon	p-value	—	$1.575E-04$	$1.766E-04$	$1.766E-04$	$1.746E-04$	$1.776E-04$	$1.776E-04$	$1.766E-04$	$1.776E-04$	$1.756E-04$
	Winner	—	−	+	+	+	+	−	+	+	+
Leukemia	p-value	—	$1.817E-04$	$1.817E-04$	$1.817E-04$	$1.806E-04$	$1.827E-04$	$1.827E-04$	$1.817E-04$	$1.806E-04$	$1.817E-04$
	Winner	—	−	+	+	+	+	+	+	+	+
Lung	p-value	—	$1.827E-04$	$1.827E-04$	$1.827E-04$	$1.827E-04$	$1.827E-04$	$1.827E-04$	$1.817E-04$	$1.796E-04$	$1.817E-04$
	Winner	—	−	+	+	+	+	−	+	+	+
Lung_discrete	p-value	—	$1.817E-04$	$1.806E-04$	$1.817E-04$	$1.766E-04$	$1.817E-04$	$1.776E-04$	$1.786E-04$	$1.806E-04$	$1.806E-04$
	Winner	—	−	+	+	+	+	−	+	+	+
Lymphoma	p-value	—	$1.827E-04$	$1.817E-04$	$1.817E-04$	$1.817E-04$	$1.817E-04$	$1.000E+00$	$1.817E-04$	$1.817E-04$	$1.796E-04$
	Winner	—	−	+	+	+	+	=	+	+	+
nci9	p-value	—	$3.600E-03$	$1.405E-04$	$2.730E-04$	$3.447E-04$	$1.817E-04$	$1.932E-03$	$1.706E-03$	$8.501E-04$	$5.708E-04$
	Winner	—	−	+	+	+	+	+	+	+	+
Prostate_GE	p-value	—	$4.600E-03$	$1.806E-04$	$1.806E-04$	$1.817E-04$	$7.650E-04$	$1.004E-03$	$1.817E-04$	$1.806E-04$	$1.806E-04$
	Winner	—	−	+	+	+	+	+	+	+	+
SMK_CAN_187	p-value	—	$2.100E-02$	$1.827E-04$	$1.827E-04$	$1.827E-04$	$1.827E-04$	$9.097E-01$	$1.827E-04$	$1.827E-04$	$1.817E-04$
	Winner	—	−	+	+	+	+	=	+	+	+

Table 6: Pairwise statistical comparison of the BPO-V algorithm with other algorithms using the Wilcoxon signed-rank test ($\alpha = 0.05$).

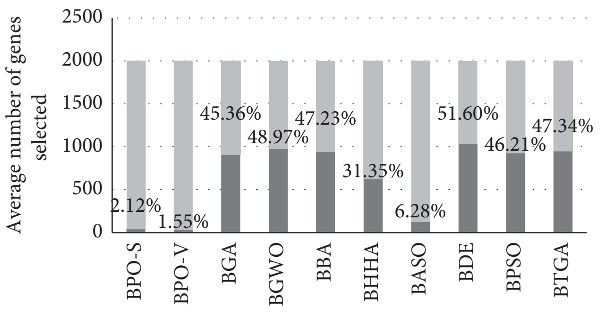
Dataset		BPO-S	BPO-V	BGA	BGWO	BBA	BHHA	BASO	BDE	BPSO	BTGA
CLL_SUB_111	p-value	$1.786E-04$	—	$1.786E-04$	$1.786E-04$	$1.786E-04$	$1.786E-04$	$5.354E-02$	$1.786E-04$	$1.786E-04$	$1.786E-04$
	Winner	+	—	+	+	+	+	=	+	+	+
Colon	p-value	$1.575E-04$	—	$1.612E-04$	$1.612E-04$	$1.593E-04$	$1.621E-04$	$3.954E-04$	$1.612E-04$	$1.621E-04$	$1.602E-04$
	Winner	+	—	+	+	+	+	+	+	+	+
Leukemia	p-value	$1.817E-04$	—	$1.806E-04$	$1.806E-04$	$1.796E-04$	$1.817E-04$	$2.821E-04$	$1.806E-04$	$1.796E-04$	$1.806E-04$
	Winner	+	—	+	+	+	+	+	+	+	+
Lung	p-value	$1.827E-04$	—	$1.827E-04$	$1.827E-04$	$1.827E-04$	$1.827E-04$	$4.274E-01$	$1.817E-04$	$1.796E-04$	$1.817E-04$
	Winner	+	—	+	+	+	+	=	+	+	+
Lung_discrete	p-value	$1.817E-04$	—	$1.796E-04$	$1.806E-04$	$1.756E-04$	$1.806E-04$	$1.238E-02$	$1.776E-04$	$1.796E-04$	$1.796E-04$
	Winner	+	—	+	+	+	+	=	+	+	+
Lymphoma	p-value	$1.827E-04$	—	$1.817E-04$	$1.817E-04$	$1.817E-04$	$1.817E-04$	$1.133E-02$	$1.817E-04$	$1.817E-04$	$1.796E-04$
	Winner	+	—	+	+	+	+	=	+	+	+
nci9	p-value	$3.600E-03$	—	$1.827E-04$	$3.298E-04$	$5.828E-04$	$1.827E-04$	$1.827E-04$	$1.827E-04$	$1.827E-04$	$5.828E-04$
	Winner	+	—	+	+	+	+	+	+	+	+
Prostate_GE	p-value	$4.600E-03$	—	$1.817E-04$	$1.817E-04$	$1.827E-04$	$1.827E-04$	$7.337E-01$	$1.827E-04$	$1.817E-04$	$1.817E-04$
	Winner	+	—	+	+	+	+	=	+	+	+
SMK_CAN_187	p-value	$2.100E-02$	—	$1.806E-04$	$1.806E-04$	$1.806E-04$	$1.806E-04$	$3.108E-02$	$1.806E-04$	$1.806E-04$	$1.796E-04$
	Winner	+	—	+	+	+	+	+	+	+	+

TABLE 7: The average number of selected features of the proposed algorithms compared to eight metaheuristics.

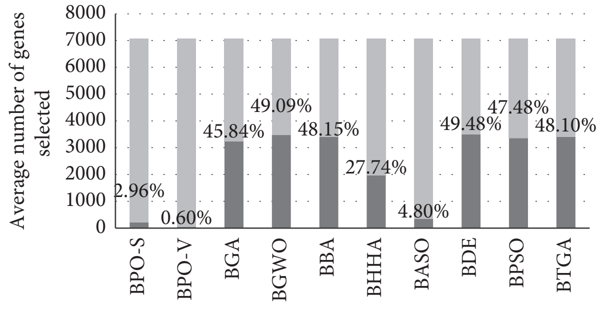
Dataset	BPO-S	BPO-V	BGA	BGWO	BBA	BHHO	BASO	BDE	BPSO	BTGA
CLL_SUB_111	257.3	187.62	5585.41	6384.14	5623.11	4399.46	1103.75	6901.91	5630.84	5662.63
Colon	42.34	31.03	907.2	979.4	944.51	626.94	125.54	1032.07	924.18	946.87
Leukemia	209.45	42.16	3240.73	3470.59	3404.03	1961.36	339.51	3498.41	3356.92	3400.45
Lung	83.61	109.25	1463.4	1615.38	1569.27	915.67	192.23	1629.41	1533.21	1566.4
Lung_discrete	45.69	31.56	121.74	149.94	138.7	94.9	25.31	158.4	129.2	137.73
Lymphoma	152.41	71.52	1574.2	1931.39	1903.45	1022.39	155.81	1959.52	1844.89	1890.76
nci9	1627.87	785.56	4720.81	5232.5	4811.22	3932.79	1062.24	6128.74	4799.88	4879.49
Prostate_GE	170.27	171.27	2833.86	3032.71	2883.41	1841.66	424.93	3086.76	2853.46	2883.27
SMK_CAN_187	222.56	371.68	9922.05	11069.31	9945.36	7199.64	1945.67	11519.65	9945.69	9977.48



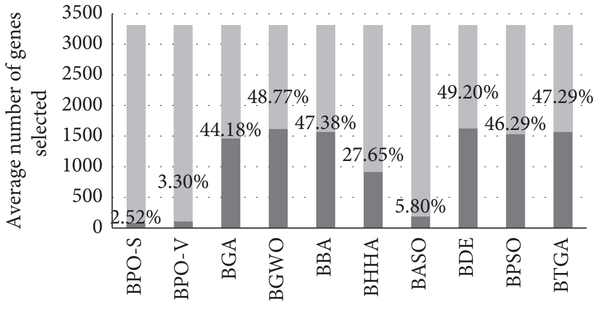
(a)



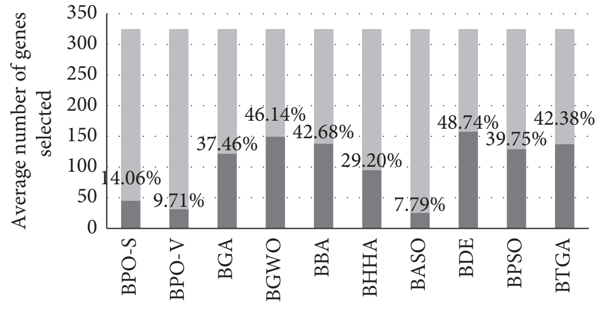
(b)



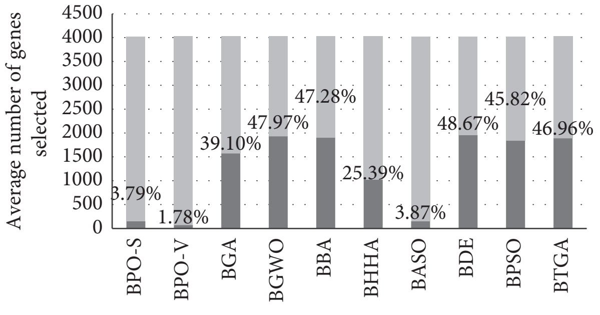
(c)



(d)



(e)



(f)

FIGURE 4: Continued.

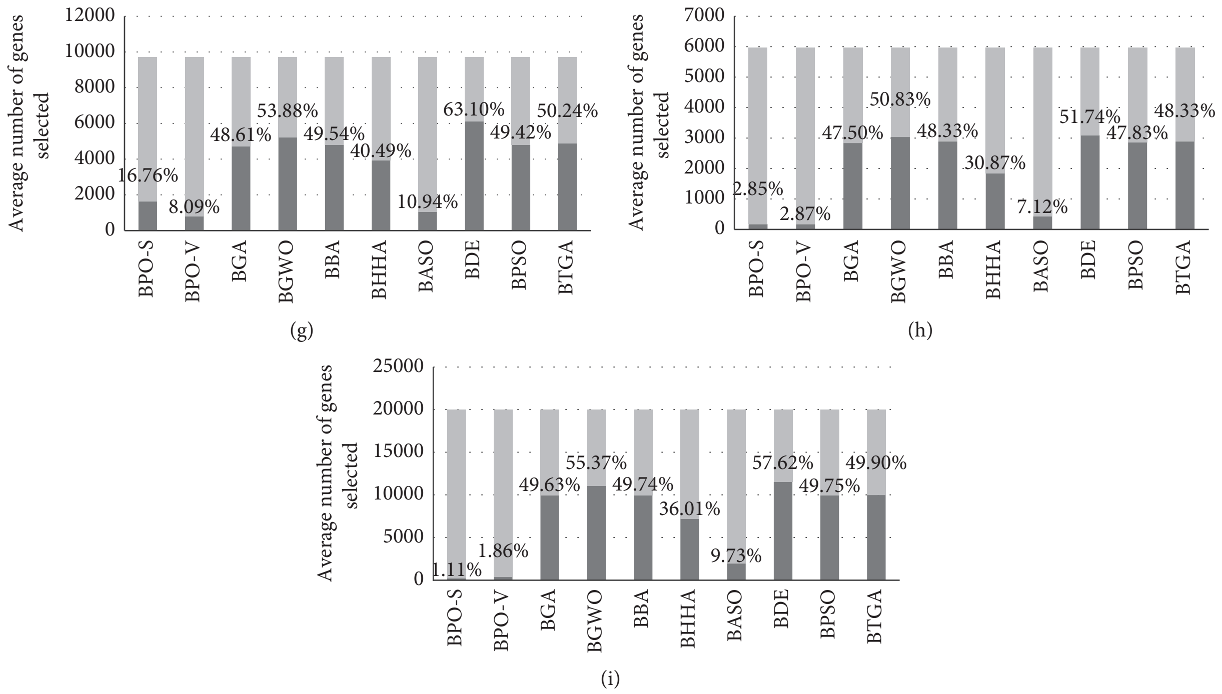


Figure 4: Average number of genes (features) selected for each of the 9 datasets (numbers on the bars indicate the percentage of selected genes).

Table 8: The average accuracy of the proposed algorithms compared to eight metaheuristics.

Dataset/time (s)	BPOV1	BPOV2	BGA	BGWO	BBA	BHHO	BASO	BDE	BPSO	BTGA
CLL_SUB_111	0.6864	0.6909	0.6818	0.5909	0.4545	0.5455	0.5000	0.6818	0.5000	0.5909
Colon	0.8667	0.8500	0.8333	0.8333	0.7500	0.7500	0.8167	0.7500	0.7500	0.7500
Leukemia	0.7714	0.9286	0.9286	0.8571	0.8571	0.8571	0.7143	0.7857	0.8571	0.9286
Lung	0.9200	0.9250	0.9150	0.9150	0.9150	0.9000	0.8750	0.9250	0.9150	0.9200
lung_discrete	0.8214	0.8571	0.7857	0.7857	0.8571	0.7143	0.6429	0.8571	0.7857	0.7143
Lymphoma	0.8263	0.8947	0.8947	0.7895	0.8421	0.8947	0.7895	0.8421	0.8947	0.8947
nci9	0.4417	0.5167	0.3333	0.3333	0.5000	0.5000	0.2500	0.5000	0.4167	0.5000
Prostate_GE	0.8650	0.9500	0.9000	0.9500	0.8500	0.9000	0.8500	0.7500	0.9500	0.9000
SMK_CAN_187	0.8949	0.8516	0.6216	0.8108	0.8649	0.7297	0.7027	0.5946	0.6757	0.7568

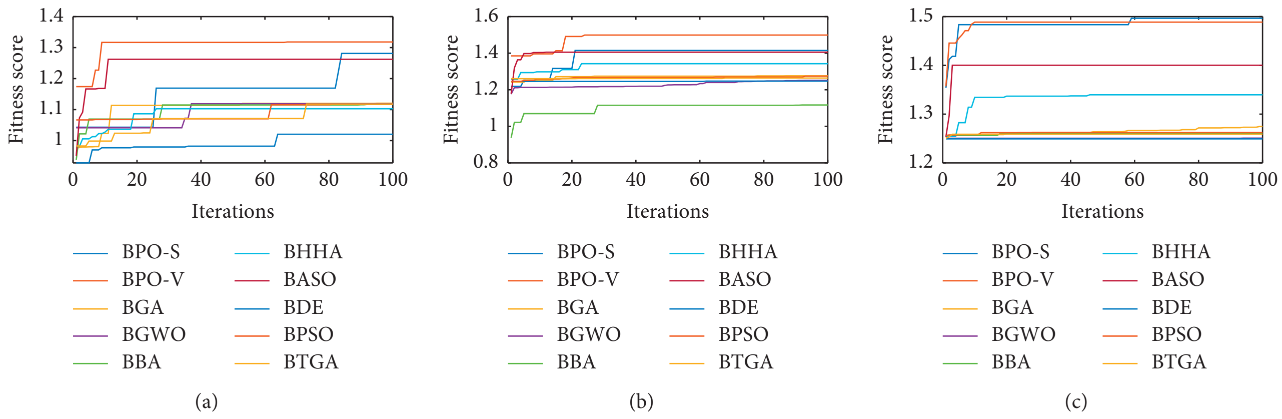


Figure 5: Continued.

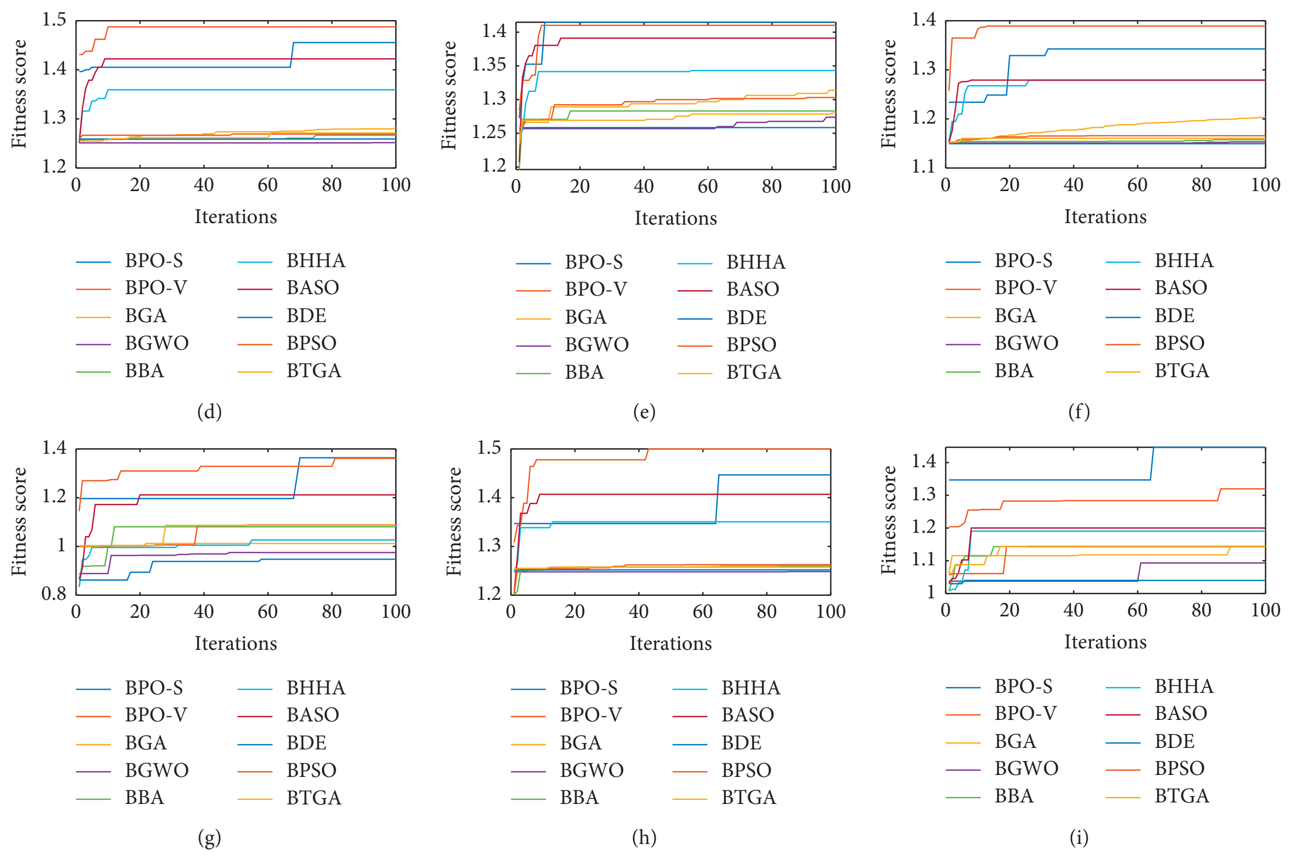


Figure 5: Convergence curves of the proposed approaches compared to 8 metaheuristics for each of the 9 datasets.

Table 9: The average execution time of the proposed algorithms compared to eight metaheuristics.

Dataset/time (s)	BPO-S	BPO-V	BGA	BGWO	BBA	BHHO	BASO	BDE	BPSO	BTGA
CLL_SUB_111	211.5163	166.0972	156.3265	156.1411	166.7361	269.3532	245.7208	172.91	235.5515	160.3811
Colon	103.2834	110.327	94.1061	96.418	94.6258	164.3746	105.777	89.9312	123.4293	93.9638
Leukemia	140.8718	117.0147	113.6496	127.0532	127.4182	195.2044	170.5102	115.6307	137.6324	110.3661
Lung	168.1855	135.1064	130.194	132.89	125.6319	227.7096	157.0562	139.6336	126.6119	116.7391
lung_discrete	111.4208	108.97	94.692	90.6665	82.4157	178.5376	101.8686	93.4482	80.0484	78.4514
Lymphoma	139.1397	117.5715	135.3171	109.7464	114.0983	202.2584	147.4745	116.1915	103.0365	100.1243
nci9	163.066	127.4642	141.5162	138.7996	137.8389	226.385	236.8451	130.9642	128.7595	116.3961
Prostate_GE	161.4066	131.6549	125.0905	130.5117	130.1564	211.7674	188.3833	134.1632	119.473	114.9097
SMK_CAN_187	519.7175	351.3269	253.5977	342.1177	332.1609	564.8705	423.7244	368.1337	342.0833	289.6829

algorithms (BPO-S or BPO-V) statistically outperform in pairs the other ones with 95% significance level ($\alpha = 0.05$). In case of inferiority, the sign "–" is used. From these tables, we can reaffirm in first place the superiority of BPO-S and BPO-V. Moreover, as mentioned before, the BASO algorithm is the most concurrent algorithm.

In the second step, to confirm this superiority, BPO-S and BPO-V are evaluated in terms of accuracy and average number of selected features. From Table 7, it can be concluded that BPO-S and BPO-V outperform in an inescapable way the other algorithms regarding the number of selected features. Hence, Figure 4 is drawn to better visualize the obtained results. One more time, BASO showed the most competitive behavior. On the contrary, Table 8 outlines the comparative results in term of accuracy, where it can be seen that BPO-V is the best algorithm. Therefore, the proposed algorithms strongly reduce the number of selected features without losing important information to deal with the problem treated by the dataset.

At the end of this evaluation, we compare BPO-V and BPO-S in terms of execution time and convergence. Regarding convergence speed and best fitness score obtained, Figure 5 shows that BPO-V also excels in this point. Generally, after 20 iterations, it reaches its optimum solution. On the contrary, despite the good results of BPO-S in terms of fitness score, this algorithm arrives at its best performance late, generally after 50 iterations. In the second term and which concerns the execution time, BPO-V and BPO-S showed poor results according to Table 9. This fact can be explained by the complexity of the algorithm proposed in [31] and its large number of functions to execute and large number of conditions to verify.

5. Conclusions

In this paper, we proposed two versions of binary PO algorithm and applied to feature selection problem on gene expression data. To assess the robustness of our work, we used 9 standard datasets characterized by their huge dimensionality. Obtained results are compared to 8 binary versions of well-known metaheuristics. Experimental results prove the excellence performance of proposed algorithm. The results are evaluated using different indicators assessing convergence, reduction size, accuracy, performance (fitness score), and runtime. In future work, BPO could be hybridized with other metaheuristic algorithms as well as another classifier instead of KNN such as SVM.

Data Availability

The data used to support the findings of the study are available at http://featureselection.asu.edu/datasets.php.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] A. Antoniadis, S. Lambert-Lacroix, and F. Leblanc, "Effective dimension reduction methods for tumor classification using gene expression data," *Bioinformatics*, vol. 19, no. 5, pp. 563–570, 2003.

[2] B. Cao, D. Shen, J. T. Sun, Q. Yang, and Z. Chen, "Feature selection in a kernel space," in *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 121–128, Corvallis, OR, USA, June 2007.

[3] E. Pashaei and N. Aydin, "Binary black hole algorithm for feature selection and classification on biological data," *Applied Soft Computing*, vol. 56, pp. 94–106, 2017.

[4] H. Liu and R. Setiono, "Chi2: feature selection and discretization of numeric attributes," in *Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence*, Herndon, VA, USA, November 1995.

[5] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[6] J. Quinlan, *C4. 5: Programs for Machine Learning*, Morgan Kaufmann, Burlington, MA, USA, 1993.

[7] M. Robnik-Łikonja and I. Kononenko, "Theoretical and empirical analysis of relieff and rrelieff," *Machine Learning*, vol. 53, pp. 23–69, 2003.

[8] R. Kohavi and G. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[9] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[10] Y. Saeys, I. Inza, and P. Larranaga, "A review of feature selection techniques in bioinformatics," *Bioinformatics*, vol. 23, no. 19, pp. 2507–2517, 2007.

[11] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery in databases," *AI Magazine*, vol. 17, p. 37, 1996.

[12] B. Xue, M. Zhang, and W. N. Browne, "Particle swarm optimization for feature selection in classification: a multi-objective approach," *IEEE Transactions on Cybernetics*, vol. 43, no. 6, pp. 1656–1671, 2013.

[13] E. G. Talbi, *Metaheuristics: from Design to Implementation*, Vol. 74, John Wiley & Sons, Hoboken, NJ, USA, 2009.

[14] R. Y. M. Nakamura, L. A. M. Pereira, K. A. Costa et al., "A binary bat algorithm for feature selection," in *Proceedings of the 25th Conference on Graphics, Patterns and Images (SIBGRAPI)*, pp. 291–297, Ouro Preto, Brazil, August 2012.

[15] F. W. Glover and G. A. Kochenberger, Eds., *Handbook of Metaheuristics*, Springer Science & Business Media, Vol. 57, Berlin, Germany, 2006.

[16] R. Meiri and J. Zahavi, "Using simulated annealing to optimize the feature selection problem in marketing applications," *European Journal of Operational Research*, vol. 171, no. 3, pp. 842–858, 2006.

[17] H. Zhang and G. Sun, "Feature selection using tabu search method," *Pattern Recognition*, vol. 35, no. 3, pp. 701–711, 2002.

[18] I. O. Oduntan, M. Toulouse, R. Baumgartner, C. Bowman, R. Somorjai, and T. G. Crainic, "A multilevel tabu search algorithm for the feature selection problem in biomedical data," *Computers & Mathematics with Applications*, vol. 55, no. 5, pp. 1019–1033, 2008.

[19] W. Siedlecki and J. Sklansky, "A note on genetic algorithms for large-scale feature selection," *Pattern Recognition Letters*, vol. 10, no. 5, pp. 335–347, 1989.

[20] J. Bala, K. D. Jong, J. Huang, H. Vafaie, and H. Wechsler, "Using learning to facilitate the evolution of features for recognizing visual concepts," *Evolutionary Computation*, vol. 4, no. 3, pp. 297–311, 1996.

[21] L. Jourdan, C. Dhaenens, and E. Talbi, "A genetic algorithm for feature subset selection in data-mining for genetics," in *Proceedings of the 4th Metaheuristics International Conference, MIC*, pp. 29–34, Porto, Portugal, July 2001.

[22] L. S. Oliveira, R. Sabourin, F. Bortolozzi, and C. Y. Suen, "A methodology for feature selection using multiobjective genetic algorithms for handwritten digit string recognition," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 17, no. 6, pp. 903–929, 2003.

[23] R. Sharkawy, K. Ibrahim, M. M. A. Salama, and R. Bartnikas, "Particle swarm optimization feature selection for the classification of conducting particles in transformer oil," *IEEE Transactions on Dielectrics and Electrical Insulation*, vol. 18, no. 6, pp. 1897–1907, 2011.

[24] Y. Y. Wang, H. Zhang, C. H. Qiu, and S. R. Xia, "A novel feature selection method based on extreme learning machine and fractional-order Darwinian PSO," *Computational Intelligence and Neuroscience*, vol. 2018, Article ID 5078268, , 2018.

[25] M. H. Aghdam, N. Ghasem-Aghaee, and M. E. Basiri, "Text feature selection using ant colony optimization," *Expert Systems with Applications*, vol. 36, no. 3, pp. 6843–6853, 2009.

[26] Y. Chen, D. Miao, and R. Wang, "A rough set approach to feature selection based on ant colony optimization," *Pattern Recognition Letters*, vol. 31, no. 3, pp. 226–233, 2010.

[27] M. Schiezaro and H. Pedrini, "Data feature selection based on artificial bee colony algorithm," *EURASIP Journal on Image and Video Processing*, vol. 2013, no. 1, p. 47, 2013.

[28] H. Rao, X. Shi, A. K. Rodrigue et al., "Feature selection based on artificial bee colony and gradient boosting decision tree," *Applied Soft Computing*, vol. 74, pp. 634–642, 2019.

[29] R. N. Khushaba, A. Al-Ani, and A. Al-Jumaily, "Feature subset selection using differential evolution and a statistical repair mechanism," *Expert Systems with Applications*, vol. 38, no. 9, pp. 11515–11526, 2011.

[30] E. Hancer, B. Xue, and M. Zhang, "Differential evolution for filter feature selection based on information theory and feature ranking," *Knowledge-Based Systems*, vol. 140, pp. 103–119, 2018.

[31] Q. Askari, I. Younas, and M. Saeed, "Political optimizer: a novel socio-inspired meta-heuristic for global optimization," *Knowledge-Based Systems*, vol. 195, Article ID 105709, 2020.

[32] S. Mirjalili and A. Lewis, "S-shaped versus v-shaped transfer functions for binary particle swarm optimization," *Swarm and Evolutionary Computation*, vol. 9, pp. 1–14, 2013.

[33] G. I. Sayed, A. Tharwat, and A. E. Hassanien, "Chaotic dragonfly algorithm: an improved metaheuristic algorithm for feature selection," *Applied Intelligence*, vol. 49, no. 1, pp. 188–205, 2019.

[34] C. Haslinger, N. Schweifer, S. Stilgenbauer et al., "Microarray gene expression profiling of b-cell chronic lymphocytic leukemia subgroups defined by genomic aberrations and VH mutation status," *Journal of Clinical Oncology*, vol. 22, no. 19, pp. 3937–3949, 2004.

[35] U. Alon, N. Barkai, D. A. Notterman et al., "Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays," *Proceedings of the National Academy of Sciences*, vol. 96, no. 12, pp. 6745–6750, 1999.

[36] T. R. Golub, D. K. Slonim, P. Tamayo et al., "Molecular classification of cancer: class discovery and class prediction by gene expression monitoring," *Science*, vol. 286, no. 5439, pp. 531–537, 1999.

[37] A. Bhattacharjee, W. G. Richards, J. Staunton et al., "Classification of human lung carcinomas by mRNA expression profiling reveals distinct adenocarcinoma subclasses," *Proceedings of the National Academy of Sciences*, vol. 98, no. 24, pp. 13790–13795, 2001.

[38] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1226–1238, 2005.

[39] A. A. Alizadeh, M. B. Eisen, R. E. Davis et al., "Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling," *Nature*, vol. 403, no. 6769, pp. 503–511, 2000.

[40] D. T. Ross, U. Scherf, M. B. Eisen et al., "Systematic variation in gene expression patterns in human cancer cell lines," *Nature Genetics*, vol. 24, no. 3, pp. 227–235, 2000.

[41] U. Scherf, D. T. Ross, M. Waltham et al., "A gene expression database for the molecular pharmacology of cancer," *Nature Genetics*, vol. 24, no. 3, pp. 236–244, 2000.

[42] D. Singh, P. G. Febbo, K. Ross et al., "Gene expression correlates of clinical prostate cancer behavior," *Cancer Cell*, vol. 1, no. 2, pp. 203–209, 2002.

[43] J. B. Welsh, L. M. Sapinoso, A. I. Su et al., "Analysis of gene expression identifies candidate markers and pharmacological targets in prostate cancer," *Cancer Research*, vol. 61, no. 16, pp. 5974–5978, 2001.

[44] A. Spira, J. E. Beane, V. Shah et al., "Airway epithelial gene expression in the diagnostic evaluation of smokers with suspect lung cancer," *Nature Medicine*, vol. 13, no. 3, pp. 361–366, 2007.

[45] L.-Y. Chuang, H.-W. Chang, C.-J. Tu, and C.-H. Yang, "Improved binary PSO for feature selection using gene expression data," *Computational Biology and Chemistry*, vol. 32, no. 1, pp. 29–38, 2008.

[46] O. H. Babatunde, L. Armstrong, J. Leng, and D. Diepeveen, "A Genetic Algorithm-Based Feature Selection," *International Journal of Electronics Communication and Computers Engineering*, vol. 5, pp. 899–905, 2014.

[47] S. Mirjalili, S. M. Mirjalili, and X.-S. Yang, "Binary bat algorithm," *Neural Computing and Applications*, vol. 25, no. 3-4, pp. 663–681, 2014.

[48] G. Pampara, A. P. Engelbrecht, and N. Franken, "Binary differential evolution," in *Proceedings of the 2006 IEEE International Conference on Evolutionary Computation*, pp. 1873–1879, IEEE, Vancouver, Canada, July 2006.

[49] E. Emary, H. M. Zawbaa, and A. E. Hassanien, "Binary grey wolf optimization approaches for feature selection," *Neurocomputing*, vol. 172, pp. 371–381, 2016.

[50] J. Too and A. Rahim Abdullah, "Binary atom search optimisation approaches for feature selection," *Connection Science*, vol. 32, pp. 1–25, 2020.

[51] T. Thaher, A. A. Heidari, M. Mafarja, J. S. Dong, and S. Mirjalili, "Binary Harris Hawks optimizer for high-dimensional, low sample size feature selection," in *Evolutionary Machine Learning Techniques*, pp. 251–272, Springer, Singapore, Asia, 2020.

[52] J. Too, A. Abdullah, N. Mohd Saad, and N. Mohd Ali, "Feature selection based on binary tree growth algorithm for the classification of myoelectric Signals," *Machines*, vol. 6, no. 4, p. 65, 2018.