A New Binary Particle Swarm Optimization Approach: Momentum and Dynamic Balance Between Exploration and Exploitation

Bach Hoai Nguyen, *Member, IEEE*, Bing Xue, *Member, IEEE*, Peter Andreae, and Mengjie Zhang, *Fellow, IEEE*

***Abstract*—Particle swarm optimization is a heuristic optimization algorithm generally applied to continuous domains. Binary particle swarm optimization is a form of particle swarm optimization, applied to binary domains, but uses the concepts of velocity and momentum from continuous particle swarm optimization, which leads to its limited performance. In our previous work, we reformulated momentum as a stickiness property and velocity as a flipping probability to develop sticky binary particle swarm optimization. The initial design provides a good base, but many key factors need to be investigated. In this work, we propose a new algorithm named dynamic sticky binary particle swarm optimization by developing a dynamic parameter control strategy based on an investigation of exploration and exploitation in binary search spaces. The proposed algorithm is compared with four state-of-the-art dynamic binary algorithms on two types of binary problems: knapsack and feature selection. The experimental results on knapsack datasets show that the new velocity and momentum assist sticky binary particle swarm optimization in evolving better solutions than the benchmark algorithms. On feature selection, the dynamic strategy takes the advantages of these two newly defined movement concepts to help the proposed algorithm produce smaller feature subsets with higher classification performance. This is the first time in binary particle swarm optimization, the four important concepts, i.e., velocity, momentum, exploration, and exploitation, are investigated systematically to capture properties of binary search spaces to evolve better solutions for binary problems.**

***Index Terms*—Binary Particle Swarm Optimization, Feature Selection, Knapsack, Classification**

I. Introduction

Evolutionary Computation (EC) includes a group of population-based heuristic approaches, in which individuals interact with each other to search for optimal or near-optimal solutions. One such EC algorithm, Particle Swarm Optimization (PSO) [1], has been extensively applied to many real-wold problems [2], [3] because of its simplicity, efficiency and effectiveness [4]. PSO was inspired by the social behavior of birds flocking, and was proposed to solve continuous problems. As an EC algorithm, PSO maintains a set of particles, which represent candidate solutions. The quality of each particle is measured by a fitness function. Each particle has its own position which is usually a numeric vector, in which each entry corresponds to a decision variable. In order to explore the search space, particles move around according to their velocities. Each particle records its personal best position, called *pbest (pb)* and its neighbors' best position, called *gbest (gb)*. The two bests contribute to the particle's velocity, which helps the particle to explore in more promising regions. Specifically, for a particle, the d^{th} component of its velocity is updated according to the following equation:

$$v_d^{t+1} = w \times v_d^t + c_1 \times r_1 \times (pb_d - x_d^t) + c_2 \times r_2 \times (gb_d - x_d^t) \quad (1)$$

where t means the t^{th} iteration in the whole evolutionary process, x is the current position, c_1 and c_2 are acceleration constants, and r_1 and r_2 are two random values uniformly distributed in [0,1]. At each step, the particle's position is updated by using Eq. (2).

$$x_d^{t+1} = x_d^t + v_d^{t+1} \quad (2)$$

The original PSO is continuous PSO (CPSO) being applied and extended to solve many continuous problems [5]. PSO has also been extended to cope with discrete problems [6]. Binary PSO (BPSO) was originally developed by Kennedy and Eberhart [7], to solve many combinatorial problems, e.g. job-shop scheduling [8] and feature selection [9]. In BPSO, the position is a binary vector, and the velocity is a continuous vector and is still updated by Eq. (1). But rather than adding the velocity to the position to get the new position, in BPSO, the velocity entry is used to determine the probability that the corresponding position entry takes the value 1, as shown in the position update equation in Eq. (3).

$$x_d^{t+1} = \begin{cases} 1, & \textbf{if } rand() \leq s(v_d^{t+1}) \\ 0, & \textbf{otherwise} \end{cases} \quad (3)$$

where $s(v_d^{t+1}) = \dfrac{1}{1 + e^{-v_d^{t+1}}}$, known as a transfer function.

Velocity plays a major role in the performance of a PSO algorithm. As shown in Eq. (1), the velocity consists of three components. The first component is momentum (wv_d^t) showing the influence of the current direction. Different particles usually have different momentums, which helps to maintain the diversity of the swarm, especially when all the particles share their best experience. Also, when the particle arrives the best position discovered by the swarm so far, momentum is the only factor which allows the particle to keep exploring better solutions. The other two are cognitive component and social component, which guide the particle toward its own best experience and neighbors' best experience.

In CPSO, particles can move smoothly in a direction, so defining a velocity as a vector of real numbers is meaningful. However, in a binary search space, a particle moves by

flipping its position entries. Such a movement is not correctly described as a velocity and directly applying the notions of speed, direction, and momentum to the binary domain is not valid. In [10], we argued that these concepts were misleading and being more precise about the nature of the movement led to a better formulation in terms of "stickiness" and a "flipping probability" that gave an algorithm with better results than standard BPSO. It is not appropriate to apply the velocity concept of CPSO directly to BPSO as in [7]. This inappropriate application leads to BPSO's limited performance compared with CPSO [11].

It is also important to control the contributions of the three components of velocity in PSO to balance between exploitation and exploration [12]. The exploration ability corresponds to a tendency to discover new search regions, while the exploitation ability corresponds to finding the best solution within the current region. The balance between exploration and exploitation relates directly to the inertia weight and the two acceleration parameters [13], which control the momentum, cognitive and social components, respectively. In CPSO, a larger inertia weight, which gives higher velocities, results in more exploration and a smaller inertia weight guides the swarm to focus more on exploitation. A control strategy that starts with a high inertia weight and gradually decreases it results in more exploration at the beginning and more exploitation at the end of each run, which has been widely used in CPSO.

However, the inertia weight in BPSO has an opposite effect, which is shown in [14] and theoretically proved in [15] (on the assumption that *pbest* and *gbest* are not changed). In contrast to CPSO, the velocity in BPSO does not directly determine the new position of a particle. It is used to derive the probability of the position entry being 1, which makes BPSO's movement very different from that of CPSO. Besides, since the movement to the new position ignores the previous location, one cannot say the new position is far or close to the previous location.

The velocity, momentum, exploration and exploitation in BPSO need appropriate formations so that the particles can move through a binary search space in a meaningful way. In our previous work [10], new definitions of velocity and momentum have been defined in a coherent way that allows BPSO to work directly on a binary search space. We will continue our previous work to investigate BPSO further and develop a new BPSO algorithm in which the binary movement is reflected more accurately, and the balance between exploration and exploitation is better controlled.

A. Goal

The overall goal of this paper is to develop a new BPSO algorithm, which can consider properties of binary search spaces to explore the search space more effectively and produce better solutions for binary problems. To achieve this goal, the key concepts, which are momentum and velocity, are revised so that the particles move around the search space by more effective mechanisms. Also, a dynamic parameter setting strategy is developed to further enhance the BPSO's search ability by considering the trade-off between exploration and exploitation. The proposed BPSO algorithm is compared with four well-known and state-of-the-art EC algorithms on two types of well-known binary problems: knapsack and feature selection. Specifically, we will investigate the following objectives:

- Investigate whether applying the two revised concepts can help BPSO to evolve better item subsets with higher profit in the knapsack problems;
- Investigate whether the new momentum and velocity can assist the particles to better explore the large and complex search space of feature selection, and can result in smaller feature subsets with higher classification performance;
- Investigate whether the proposed dynamic parameter setting strategy can balance between exploration and exploitation to further improve the search ability of BPSO on both knapsack and feature selection problems; and
- Analyse the effect of the two revised concepts and the dynamic strategy during the BPSO's search process.

II. Background

A. Related Work on BPSO

Binary PSO (BPSO) has been applied to many real-world problems. Sarath et al. [16] applied BPSO to generate association rules from transactional datasets. The results on a dataset from an Indian commercial bank showed that applying BPSO not only provided higher quality rules but also avoided redundant rules. Taha et al. [17] used BPSO to detect available frequencies in cognitive radio, which allowed to utilize system resources. The BPSO-based dynamic allocation achieved higher detection rates and lower false alarm rates with different noise ratios. Lin et al. [18] used BPSO to search for highly profitable item sets instead of frequent item sets in transactional databases. It was shown that the BPSO based algorithm was more efficient, more effective and converged faster than GAs.

In many other works, BPSO has also been modified to improve its performance. In the original BPSO [7], a sigmoid-function was used as a transfer function known as an S-shaped function. In [19], V-shaped transfer functions were proposed for BPSO. The position updating equation also considered the previous location. The experimental results showed that applying V-shaped functions while considering the previous location improved the performance of BPSO. However, it was not clear that which modification contributed more to the improvement. The performance of BPSO on different datasets heavily depended on the specific transfer functions, even when they were from the same family (V-shaped or S-shaped). Hence, it was not an easy task to select an appropriate transfer function for a particular task or dataset. Islam et al. [20] proposed a time-varying S-shaped transfer function for BPSO which aimed to provide a smooth shift from more exploration to more exploitation during the evolutionary process. Experimental results showed that the time-varying transfer function assisted BPSO to achieve better results than a normalized linear transfer function [21] and a well-known V-shaped transfer functions [19].

Zhang et al. [22] used BPSO for feature selection, which aimed to improve the performance in spam detection problems. In this work, a mutation operator was used to increase the diversity of the swarm and avoid premature convergence. The experimental results indicated that the proposed algorithm achieved better results than GAs and the standard PSO algorithm. Yang et al. [23] attempted to allocate workload to sensors in a network so that the system was more energy-efficient and the communication volume was reduced. The BPSO algorithm with a V-shape function, a new updating equation, and mutation operators was proposed. The BPSO algorithm outperformed GAs and standard BPSO. However, the modifications in the two proposed BPSO algorithms still ignored the previous location. Aiman et al. [24] modified BPSO to achieve a good state assignment on a finite state machine, which aimed to minimize the area of sequential circuits. The authors proposed an updating equation for velocity, which considered the correlation between $pbest$ and $gbest$. Although the experimental results showed that the modified BPSO algorithm was more effective than standard BPSO and GAs, it was potential that the swarm would be stuck at local optima when $pbest$ and $gbest$ had the same position. Zhai et al. [25] improved a BPSO-based instance selection algorithm by utilizing an immune mechanism. Before updating the swarm, some particles were modified based on the numbers of their good neighbors (vaccine value) and current positions. The experimental results on datasets with different numbers of instances showed that the immune BPSO outperformed both standard BPSO and INSIGHT, which was a deterministic instance selection algorithm, regarding the classification accuracy and the size of instance subsets. It could be seen that considering the previous position in BPSO has a positive effect. However, the proposed algorithm was designed specifically for instance selection.

In 2016, Liu et al. [15] provided a detailed analysis of the effect of the inertia weight parameter on the searching ability of BPSO. Particularly, it was shown that when the two bests were not changed, a larger inertia weight tended to enhance the exploration ability, which was opposite to CPSO. Based on this observation, an incremental strategy for the inertia weight was defined as follows:

$$w = \begin{cases} \underline{w} + \frac{\pi(\overline{w}-\underline{w})}{\rho\overline{\pi}}, & \textbf{if } \pi \leq \rho\overline{\pi} \\ \overline{w}, & \textbf{if } \rho\overline{\pi} < \pi \leq \overline{\pi} \end{cases} \tag{4}$$

where π and $\overline{\pi}$ represented for the current iteration and the maximum number of iterations, $\overline{w}$ and $\underline{w}$ were the upper and lower boundaries of w, ρ was a parameter in the interval [0,1], which was used to determine the number of iterations to increase w from $\underline{w}$ to $\overline{w}$. Based on the analysis, Liu et al. [15] proposed a BPSO algorithm named Up BPSO, which outperformed the standard BPSO algorithm with a constant and a linearly decreasing inertia weight.

Besides modifying standard BPSO to achieve a better trade-off between exploration and exploitation, there have been some studies proposing new metaphors for BPSO. For instance, Pampara et al. [26] proposed an Angle-modulated BPSO algorithm that essentially used a sine function with four parameters to generate a real-valued vector. The vector was then converted to a binary string using 0 as a threshold. The four parameters were optimized using a standard continuous PSO. Although the algorithm significantly reduced the number of original decision variables to only four parameters, it introduced strong correlations between original decision variables since they were generated from the same function. Jeong et al. [27] applied Quantum Computing to propose a Quantum-inspired BPSO algorithm. The main idea was that the j^{th} entry was represented by an angle θ_j. Then $sin(\theta_j)^2$ was used as a probability of the corresponding entry being 1. For each iteration, θ_j was updated with respect to a rotation angle θ and the differences between the current position and $pbest$ or $gbest$. In the proposed algorithm, the rotating angle θ was also dynamically updated as follows:

$$\theta = \theta_{max} - (\theta_{max} - \theta_{min}) \times \frac{t}{T} \tag{5}$$

where t is the current iteration number, and T is the maximum number of iterations.

Although many studies have attempted to improve the performance of BPSO, they are mainly based on $continuous$ velocity and momentum concepts borrowed from continuous PSO. Some new metaphors for BPSO have been proposed such as Quantum BPSO, but they do not consider characteristics of movements in binary search spaces, i.e., bouncing between points. To address this problem, we proposed a new velocity and momentum concept to let binary particles move in a more meaningful way [10]. The standard velocity was replaced by a probability vector, which determined the flipping probability of a position entry. Therefore, there was no need to worry about the transfer function as in the original BPSO. Following the new velocity concept, a new momentum was introduced, which was also known as $stickiness$ property. The main idea was that, if a bit has just flipped then it will try to $stick$ with the new value for a while. In the following iterations, if the bit's value was not changed, its $stickiness$ property decayed until the bit was flipped again. Although the proposed stickiness BPSO (SBPSO) [10] algorithm had good results on knapsack and feature selection problems, this is still a very initial work. We have not fully investigated its abilities such as the parameters of the algorithm are not optimized yet. Also, its performance has not been compared with recently developed binary algorithms such as Up BPSO [15], Time-varying BPSO [20], Quantum BPSO [27]. Besides momentum and velocity, exploration and exploitation are also two essential properties in a PSO algorithm, but they have not been defined in a systematic way with the other new concepts.

B. Knapsack Problems

Knapsack is a traditional binary optimization problem, which can be described as follows: given a set of n items and a knapsack; each i^{th} $(i = 1 \dots n)$ item has a positive profit p_i and a number of positive resource consumption $r_{i1}, r_{i2}, \dots, r_{im}$ corresponding to m resources; the knapsack has m capacities $C_j (j = 1 \dots m)$ for each resource; the task is to select an item subset so that the total profit of selected items is maximized while for all resources the total resource

consumption does not exceed the knapsack's resource capacity. The problem can be described using the following formula:

$$max \sum_{i=1}^{n} p_i * x_i,$$
$$s.t. : \sum_{i=1}^{n} r_{ij} * x_i \leq C_j, \forall j \in \{1, 2, \ldots, m\}$$
$$x_i \in 0, 1, \forall i \in \{1, 2, \ldots, n\}$$

where the i^{th} item is selected if and only if $x_i = 1$. Knapsack has been widely used to evaluate many evolutionary algorithms such as PSO [28], DE [29], GAs [30], and even multi-objective algorithms [31]. Checking and repairing operators [32], [33] can be incorporated to avoid infeasible item sets. However, utilizing such operators may compensate for or hide poor performance in the underlying BPSO, which makes it difficult to compare between different BPSO algorithms. Therefore, this work adopts a penalty function to handle the Knapsack constraint.

C. Feature Selection

Because of significant advancements in technology, real-world datasets are getting bigger with a large number of features. Due to the "curse of dimensionality", learning systems e.g. classification methods, may not work well with these high-dimensional datasets. The task of classification is to category instances based on their attribute or feature values. However, a large feature set usually contains irrelevant or redundant features, which can hide useful information from other relevant features [34]. As a result, the classification process requires a long training time and the performance deteriorates due to possible overfitting. In order to build a more accurate classifier in a shorter training time, feature selection is proposed to remove all irrelevant/redundant features.

However, feature selection is a difficult problem because of its large search space. Suppose there are n original features, there are 2^n possible feature subsets to consider. In addition, features can interact with each other, which makes feature selection even more difficult [35]. Many search techniques have been applied to solve feature selection problems. An exhaustive search guarantees to find an optimal feature subset but it is not practical to perform on a large number of features. Sequential searches [36] start with an empty (full) feature set and add (remove) features to (from) the selected features. However, they are usually trapped at local optima since once a feature is selected (removed), it can not be removed (selected) later. Recently, sparse learning based feature selection methods gain a lot of attention [37]. The idea is to find an optimal weight vector, which minimizes the fitting error along with some regularization terms. Because of the sparse regularizer, some learned weights will be very small, and their corresponding features are discarded. However, these methods usually require a predefined number of selected features. Among sparse learning based feature selection methods, the robust feature selection method (RFS) [38] is one of the most popular ones. Instead of using the ℓ_2-norm based loss function, $\ell_{2,1}$-norm is applied to avoid outliers in data points. In addition, $\ell_{2,1}$ is also cheaper to calculate. Experimental results on six datasets show the better performance of RFS over several popular traditional feature selection approaches.

EC techniques have been applied widely because of its global search ability. A comprehensive survey about EC-based feature selection algorithms has been done by Xue et al. [35]. In comparison with other EC techniques, PSO is preferred since it is efficient, has fewer parameters and a natural representation for feature selection [39].

III. Proposed Algorithm

This section presents the new momentum and velocity concepts for BPSO by using the notion of stickiness. It then defines exploration and exploitation capabilities to cope with the movement strategy of the particles. Based on definitions of the two capabilities, a dynamic strategy is developed to control the trade-off between these capabilities better.

A. Sticky BPSO (SBPSO)

In standard BPSO, the new position is determined without considering the previous location, which can be seen in Eq. (3). The particles do not move smoothly as in CPSO. Particles change their positions by flipping position entries either from 0 to 1 or from 1 to 0. This kind of probabilistic binary change cannot usefully be described as a continuous velocity. Rather it is better to describe the change in terms of the probability of flipping. Therefore, instead of using a velocity vector we use a flipping vector p , in which each entry shows the probability of flipping the corresponding position entry.

To guide particles toward promising regions, the PSO velocity vector consists of three main components: momentum, cognitive and social factors. All three factors need modification for the binary domain. Momentum is a fundamental continuous concept (like velocity) and needs to be replaced by a more appropriate concept that still captures the role of momentum in CPSO. In CPSO, the momentum corresponds to a tendency to keep moving in the current direction. However, in BPSO, instead of moving in a direction, a particle's movement is described as whether its entries are flipped or not. Therefore, in BPSO, we replace the momentum by a measure of the tendency to stick with the current position, which we call $stickiness(stk)$. The idea is that a high stickiness for an entry means that the particle should stick with the value for a while so that the particle can explore around the entry's value, rather than switching to a different region of the space. If the stickiness of an entry is set to a high value and never changes, the particle could get stuck in an unproductive region. Therefore, we need a strategy to control the stickiness. We currently set the stickiness of an entry to be high when the entry is flipped and then decay the stickiness over time until it is 0 or the entry flips again. We use a linear decay that reduces the stickiness from 1 to 0 over a fixed number of steps (*ustkS*). The stickiness property of the d^{th} bit is updated using:

$$stk_d^{t+1} = \begin{cases} 1, \textbf{ if } \text{the bit is just flipped} \\ \max(stk_d^t - \frac{1}{ustkS}, 0), \textbf{ otherwise} \end{cases} \quad (6)$$

where t means the t^{th} iteration. It can be seen that if one bit is not flipped for *ustkS* iterations, its stickiness value stk_d will become 0, which significantly increases the probability that this bit will be flipped (*un-stick*) in the next iteration.

The cognitive and social factors (based on $pbest$ and $gbest$) are still important – they guide the particle towards the regions containing $pbest$ and $gbest$. However, the acceleration factors (and the random multipliers) are only appropriate in the continuous domain. For the binary domain, we need important weights that increase the flipping probability when the current position is different from the $pbest$ and $gbest$. Using the $stickiness$ property in place of the momentum factor and the modified cognitive and social factors, the flipping probability of a particle's d^{th} position entry is given in Eq. (7).

$$p_d = i_s * (1 - stk_d) + i_p * |pb_d - x_d| + i_g * |gb_d - x_d| \quad (7)$$

where i_s is the importance of the $stickiness$ factor, and i_p, i_g are the importance of the cognitive and social factors, respectively. As shown in Eq .(7), if $gbest$ and $pbest$ are not changed, the smaller the $stickiness$ the more likely the d^{th} bit will be flipped, which gives a high flipping probability to the bit that is not changed for a large number of iterations.

According to the flipping probability vector, the new position is determined by Eq. (8), which does consider the previous location to determine the new position.

$$x_d^{t+1} = \begin{cases} 1 - x_d^t & \textbf{, if } rand() < p_d \\ x_d^t & \textbf{, otherwise} \end{cases} \quad (8)$$

B. Exploration and Exploitation in SBPSO

In CPSO, a velocity shows how far a particle is going to move from the current position to the new position. A large velocity facilities exploration while a small one leads to more exploitation. In a binary search space, Hamming distance can be used to measure the distance between two binary solutions, which is the number of bits that the two solutions are different. If a large number of bits are mutated, the particle is exploring the solution. On the other hand, a few bits being flipped means the particle exploits the space around the current position. However, this is not clearly reflected in the standard BPSO algorithm since the updating equation ignores the previous position. By contrast, the difference between the new position and previous position is naturally shown by the flipping operation in SBPSO. As can be seen in Eq. (8), p_d shows the probability of flipping the d^{th} position entry, which means the larger p_d, the more likely the entry is flipped. Therefore, more bits to be flipped results in the particle towards exploration and a smaller probability vector lets the particle focus more on exploitation. Based on the defined exploration and exploitation concepts in SBPSO, a dynamic strategy is proposed in the next section to balance between the two abilities in SBPSO.

C. Dynamic Strategy

As can be seen from Eq. (6) and Eq. (7), the flipping probability is affected by four main parameters in which i_s, i_p, i_g and $ustkS$. During the evolutionary process, there might be four possible relationships between a current position's bit and the corresponding bits in $pbest$ and $gbest$, in particularly:

$$p_d = \begin{cases} i_s * (1 - stk_d) & \textbf{if } x_d = pb_d = gb_d \\ i_s * (1 - stk_d) + i_g & \textbf{if } x_d = pb_d \neq gb_d \\ i_s * (1 - stk_d) + i_p & \textbf{if } x_d = gb_d \neq pb_d \\ i_s * (1 - stk_d) + i_p + i_g & \textbf{if } x_d \neq pb_d = gb_d \end{cases} \quad (9)$$

Since Eq. (9) represents the flipping probability of a single bit, it is not possible to state exactly whether a case happens at the beginning, the middle or at the end of the evolutionary process. However, it is more likely that $pb_d \neq gb_d$ when the searching process has just started and $pb_d = gb_d$ at the end of the evolutionary process when the swarm converges. The largest value of p_d is $(i_s + i_p + i_g)$ when a bit is different from the bit's value in both $pbest$ and $gbest$ for a number of iterations without any improvement. Therefore, $(i_s + i_p + i_g)$ is set to 1, which ensures the bit is flipped to match the values of $pbest$ and $gbest$. Suppose that α is the ratio between i_p and i_g, i.e., $\alpha = i_p/i_g$. i_p and i_g can be expressed as bellows:

$$i_p = \alpha \times \frac{1 - i_s}{\alpha + 1}, \quad i_g = \frac{1 - i_s}{\alpha + 1} \quad (10)$$

By substituting Eq. (10) to Eq. (9), Eq. (9) can be rewritten as bellows:

$$p_d = \begin{cases} i_s \times (1 - stk_d) & \textbf{if } x_d = pb_d = gb_d \\ i_s \times (1 - stk_d - \frac{1}{\alpha+1}) + \frac{1}{\alpha+1} & \textbf{if } x_d = pb_d \neq gb_d \\ i_s \times (1 - stk_d - \frac{\alpha}{\alpha+1}) + \frac{\alpha}{\alpha+1} & \textbf{if } x_d = gb_d \neq pb_d \\ 1 - i_s * stk_d & \textbf{if } x_d \neq pb_d = gb_d \end{cases} \quad (11)$$

where stk_d is calculated based on $ustkS$ using Eq. (6).

As can be seen in Eq. (11), for a given α value, the flipping probability now mainly depends on two parameters i_s and $ustkS$. For a specific value of i_s, a smaller $ustkS$ results in a larger p_d, more exploration, while a larger $ustkS$ guides the swarm towards exploitation because the flipping probability is smaller. On the other hand, suppose that $ustkS$ is not changed, except for the case $(x_d \neq pb_d = gb_d)$, decreasing i_s makes p_d smaller, which guides the swarm to exploit more. A static setting for i_s and $ustkS$ might just encourage either good exploration or exploitation during the evolutionary process. Therefore, a dynamic setting mechanism for i_s and $ustkS$ is proposed to allow the search process to change gradually from exploration to exploitation. Particularly, in the proposed mechanism, i_s is linearly decreased and $ustkS$ is linearly increased with respect to the number of iterations, which can be seen in Eq. (12).

$$\begin{aligned} ustkS^t &= ustkS^L + \frac{t}{T} * (ustkS^U - ustkS^L) \\ i_s^t &= i_s^U - \frac{t}{T} * (i_s^U - i_s^L) \end{aligned} \quad (12)$$

where t stands for the t^{th} iteration, T is the maximum number of iterations, $ustkS^U$ and $ustkS^L$ stand for the upper bound and the lower bound of $ustkS$, i_s^U and i_s^L are the upper bound and the lower bound of i_s.

The dynamic strategy is applied to propose a new SBPSO algorithm, called Dynamic SBPSO, which is shown in Fig. 1.

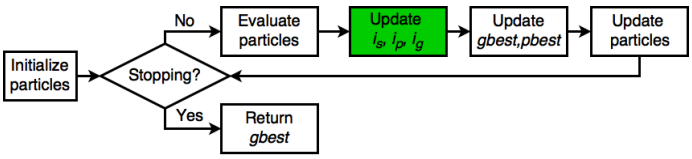

Fig. 1: Dynamic SBPSO Overview.

The standard SBSPO without the dynamic mechanism is called Static SBPSO. The difference between Static and Dynamic SBPSO is the green block, which updates the three important weights.

IV. Experiment Design

In this work, we perform three different experiments. The first experiment is to select a good setting for three parameters of the proposed SBPSO, i.e., i_s, α and $ustkS$, which is conducted only on some selected Knapsack datasets. Note that the first experiment does not aim to find a perfect parameter setting working on all problems. The aim is to have a good enough setting for SBPSO so that its comparisons with other well-known benchmark algorithms are relatively fair.

The second experiment is to compare Static and Dynamic SBPSO with four benchmark algorithms on all Knapsack datasets from three different Knapsack benchmark sets with varying difficulties. The four benchmark algorithms include three state-of-the-art BPSO algorithms, i.e., Quantum BPSO [27], Up BPSO [15], Time Varying BPSO [20], and a recently proposed Novel Modification Binary Differential Evolution (NMBDE) [40]. They have good theoretical analysis and outperform other standard binary algorithms such as GAs, binary DE [41], BPSO [7] that are not shown to reduce space. It should be noted that all the four benchmark algorithms have dynamic parameter settings to balance between exploration and exploitation.

The third experiment is conducted to compare the two versions of SBPSO with the four benchmark algorithms on feature selection that is more challenging than Knapsack due to its computationally intensive fitness function and the complex interactions between features.

A. Knapsack

In this work, three well-known Knapsack benchmark sets are used. The first one is SAC-94 [42] – a standard Knapsack library. SAC-94 contains six cases, called *hp, pb, pet, sento, weing,* and *weish* with n ranging from 10 to 105 and m ranging from 2 to 30. Each case has a number of datasets. The second benchmark set was provided by Glover and Kochenberger [43], named as "GK", which contains 11 datasets with much larger n (from 100 to 2500) and m (from 15 to 1500). The last benchmark set contains a large number of items (up to 5000) with complicated relationships between profits and resource consumptions of the items [44]. There are four relationship types: *uncorrelated instances* (UCI), *weakly correlated instances* (WCI), *strongly correlated instances* (SCI), and *inversely strongly correlated instances* (ISCI).

On the knapsack problems, each algorithm is run 30 independent times, and each run contains 1000 iterations. The population size is equal to the number of items, but it is bounded by 100 to avoid intensive computation costs.

TABLE I: Feature selection datasets.

Dataset	#Features	#Classes	#Instances
Wine	13	3	178
Australian	14	2	178
Zoo	17	7	101
Vehicle	18	4	846
German	24	2	1000
WBCD	30	2	569
Ionosphere	34	2	351
Sonar	60	2	208
Movementlibras	90	15	360
Hillvalley	100	2	606
Musk1	166	2	476
Arrhythmia	279	16	452
Madelon	500	2	4400
Multiple Features	649	10	2000

Regarding the representation, each position entry corresponds to one item, which means that a particle's length is the total number of items. The position entry is 1 or 0, which shows that the corresponding item is selected or discarded. The fitness function from [45] is used, which ensures that an infeasible solution always has a negative fitness value and a feasible solution always has a positive fitness value. The fitness function can be expressed as follows:

$$fitness_{KS} = profit - o \times s \times (maxProfit + 1) \quad (13)$$

where $profit = \sum_{i=1}^{n} p_i * x_i$ – total profits of selected items, $maxProfit = \max_{i=1}^{n} p_i$ – the largest profit of any of the items, o – the number of overfilled knapsack resources, s – the number of selected items.

B. Feature Selection

The six algorithms are compared on 15 feature selections datasets chosen from the UCI machine learning repository [46]. The datasets are different in the numbers of features, classes and instances, which can be seen in Table I. For each dataset, each algorithm is run 30 independent times, and each run contains 100 iterations. The swarm size is set to the number of features, and it is bounded by 100 [10].

The representation of the PSO algorithms for feature selection is the same as the representation for knapsack, in which each position corresponds to one original feature and indicates whether the feature is selected or not.

In feature selection, there are two main objectives, which are to maximize the classification performance and to minimize the number of selected features. The two objectives can be combined to form a fitness function as below:

$$fitness_{FS} = \gamma * ErrorRate + (1 - \gamma) * \frac{\#selected}{\#all} \quad (14)$$

where *ErrorRate* is the classification error of the selected features, *#selected* and *#all* represent the number of selected features and the total number of original features, respectively. γ is used to control the contribution of the two objectives. Since the classification performance is preferred over the number of selected features, γ is usually set to 0.9. The task is to find a feature subset to minimize the fitness function given in Eq. (14). In this work, KNN is used to calculate the classification error in Eq. (14). K is set as 5 to ensure that KNN can avoid noisy data while still maintaining its efficiency [47].

V. Parameter Study

As shown in Eq. (11), there are three important parameters: i_s is the importance of the stickiness property, α controls the contribution of $pbest$ and $gbest$, and $ustkS$ defines the number of steps or iterations to reduce stk from 1 to 0. This section studies the effect of the three parameters using six Knapsack datasets selected from the three different Knapsack benchmark sets (two for each). The six datasets are Weing1, Weish15, GK01, GK05, SCI500, and ISCI1000.

In case $x_d = pb_d = gb_d$, the maximum flipping probability of the d^{th} entry is i_s, i.e., the first case in Eq. (11), which shows a relationship between i_s and the number of bits being flipped. Therefore, we examine 11 values of i_s: 0/n, 1/n, 2/n,...,10/n, which corresponds to the 11 different numbers of bits possibly flipped, ranging from 0 to 10.

For $alpha = i_p/i_g$, we examine three values: 0.5, 1.0, and 2.0 representing three cases: $pbest$ contributes more, $pbest$ and $gbest$ have the same contribution, and $gbest$ contributes more.

$ustkS$ represents the number of iterations that a particle searches around a position. Given a large number of iterations, it is reasonable to give the particle more time to search around the position. Therefore, $ustkS$ should be based on the maximum number of iterations, T. Particularly, we examine ten different values of $ustkS$, which are $1\times T/100$, $2\times T/100$, $3 \times T/100$,..., $10 \times T/100$.

In summary, we examine the combination of 11 values for i_s, 3 values for α, and 10 values for $ustkS$, which forms 330 different settings for $(i_s, \alpha, ustkS)$. The maximum number of iterations, T, is set to 1000. Each setting is run ten independent times on each dataset. On a given dataset, each value of a parameter has an average profit calculated by averaging the profits obtained by all the settings given the parameter value. For example, the average profit of $i_s = 2/n$ is the average of profits obtained by all settings $(i_s = 2/n, \alpha, ustkS)$ with all possible values of α and $ustkS$.

The average profit values obtained by each setting of i_s on Weing1, GK05, and SCI500 are shown in Fig. 2. Similar patterns are obtained on the other three datasets. The results on all the six datasets are shown in Section II, Supplementary Material. It can be seen that the average profits are stable when i_s varies between 2/n and 10/n. An interesting pattern is that when the number of items increases, the most stable setting of i_s is smaller. The pattern indicates that it would be better for a particle to follow the best positions (more contribution for $pbest$ and $gbest$) when the search space is large and complex. From the results, the recommended value of i_s is 4/n, which is stable for both small and large datasets.

Fig. 3 illustrates the profit values obtained by three different settings of α. It can be seen that on most datasets, there is no significant difference between the three values. However, $\alpha = 2.0$ usually gives slightly better profits than other α values. The pattern illustrates that it would be better to allow $pbest$ contributes more than $gbest$ so that the population can maintain its diversity.

Fig. 4 shows the results of different values of $ustkS$. It can be seen that when the number of items increases, the best setting of $ustkS$ also increases. The pattern indicates

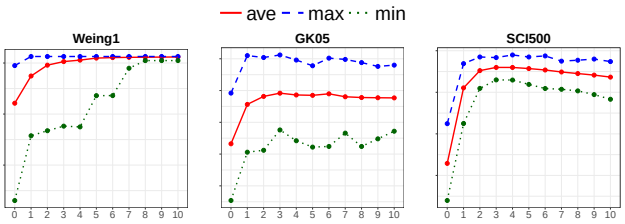

Fig. 2: Average profit values of i_s. (red – Average Profits, blue – Maximum Profits, green – Minimum Profits).

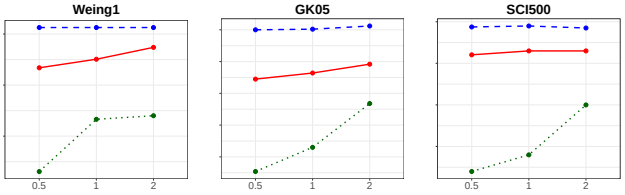

Fig. 3: Average profit values of α.

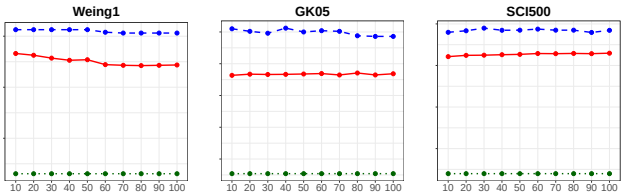

Fig. 4: Average profit values of $ustkS$.

that when the search space is large and complex, it would be better to search around promising search regions longer than to switch to other regions quickly. However, there is no significant difference between the different settings of $ustkS$. We recommend setting $ustkS$ to $8\times T/100$, which achieves the stable results on both small and large datasets.

The above parameter study gives a reasonable setting for Static SBPSO: $(i_s = 4/n, \alpha = 2.0, ustkS = 8 \times T/100)$. From the obtained pattern, the parameters of Dynamic SBPSO are set as: $i_s^L = 0/n, i_s^U = 10/n, ustkS^L = 1 \times T/100, ustkS^U = 10 \times T/100$ with an expectation that Dynamic SBPSO can cope well with both small and large datasets. This parameter study aims to have a generally good setting for SBPSO which works well not only on Knapsack but also on other binary optimization problems. Table II shows the parameter settings for both Static and Dynamic SBPSO.

VI. Results on Knapsack Datasets

This section presents the experimental results of the six algorithms on three Knapsack benchmark sets.

A. Profits

The SAC-94 benchmark set contains six different cases. Each case has a number of different datasets. Since the datasets have known optimal solutions, we compute the ratio between the number of times an algorithm evolves the optimal solutions and 30 – the total number of independent runs. The ratio is called a hit rate. Table III shows the average hit rates of each SAC-94 case. In the table, the best average hit rates are

TABLE II: Settings for two SBPSO versions: $i_g = (1 - i_s)/3, i_p = 2 \times i_g$, n – number of decision variables, T – maximum number of iterations.

Algorithm	i_s	$ustkS$
Static	$4/n$	$8 \times T/100$
Dynamic	$0/n - 10/n$	$1 \times T/100 - 8 \times T/100$

TABLE III: Hit rates on SAC-94 datasets. The top and second-best rates are marked in bold and underlined, respectively. #D is the number of datasets in each case.

Case	#D	NMBDE	Quantum	Up	TimeVarying	Stat	Dyn
hp	2	0.12	0.03	0.20	<u>0.27</u>	0.12	**0.40**
pb	6	0.17	0.07	0.19	<u>0.28</u>	0.11	**0.29**
pet	6	0.46	0.29	0.61	<u>0.62</u>	0.46	**0.68**
sento	2	**0.65**	0.10	0.07	0.18	0.08	<u>0.33</u>
weing	8	0.15	0.18	0.47	**0.53**	0.24	<u>0.52</u>
weish	30	<u>0.73</u>	0.46	0.40	0.52	0.53	**0.76**

marked in bold, and the second-best ones are underlined. It can be seen that among the six algorithms, Dynamic SBPSO achieves the highest hit rates on four out of the six cases. On the other two cases, Dynamic SBPSO is ranked as the second-best algorithm. Among the four benchmark algorithms, Time Varying BPSO is the most promising one, but its results are limited in comparison with Dynamic SBPSO. Time Varying BPSO achieves the best hit rate on only one case and the second-best hit rate on the four cases. Although Static SBPSO does not produce as good results as Dynamic SBPSO, its performance is stable across different cases of SAC-94. Particularly, Static SBPSO is ranked as the third/fourth best algorithm on four out of the six cases, and its hit rates are not the worst one in all the cases. Meanwhile, NMBDE, Quantum BPSO, Up BPSO obtain the worst performance on at least one case even though their parameters are dynamically controlled.

Although the two versions of SBPSO, especially the dynamic one, produce promising results on the six SAC-94 cases, SAC-94 is a relatively small Knapsack benchmark. Table IV presents the average profits obtained by the six algorithms on GK datasets that are larger and more complicated than SAC-94 datasets. Once more, Dynamic SBPSO is the best algorithm which produces the best profits on seven out of the eleven datasets and achieves the second-best result on the other four datasets. Time Varying BPSO can be considered the second-best algorithm since it produces the best profits on four GK datasets. Following Dynamic SBPSO and Time Varying, Static SBPSO is ranked as the second/third-best algorithm on nine out of the eleven datasets. The results show that Static SBPSO copes well with large and complex search spaces.

In contrast, NMBDE performs poorly on GK datasets. It results in the worst profit on all GK datasets. Among PSO-based algorithms, Up BPSO is the worst one while Quantum BPSO is slightly better than Up BPSO. The possible reason is that both Up and Quantum BPSO updates either the inertia weight or the rotation angle (similar to the inertia weight), which indirectly affects the flipping probability. Meanwhile, Time Varying BPSO and Dynamic SBPSO directly update the flipping probability via changing either the sigmoid function or parameters of the flipping probability equation.

The last Knapsack benchmark set is the most complicated one, which has a larger number of items (up to 5000 items) than GK datasets. More importantly, these high-dimensional datasets also have complex interactions between each the profit and the cost of each item. Table V presents the average profits obtained by the six algorithms on high-dimensional Knapsack datasets. The results illustrate the dominance of two SBPSO versions over the other benchmark algorithms. Although Static SBPSO does not use any dynamic mechanism to update its parameters, it still achieves the second-best profits on all high-dimensional datasets. Dynamic SBPSO completely dominates all the other algorithms with its best profits on all high-dimensional datasets. Similar to GK datasets, NMBDE produces the worst profits on all high-dimensional datasets. Among the three benchmark BPSO algorithms, Time Varying BPSO is still the most promising one. It can be seen that on the large and complex search spaces, SBPSO consistently evolves good solutions.

Table VI presents the comparisons between two versions of SBPSO and the other benchmark algorithm on all 79 Knapsack datasets from the three benchmark sets. The comparison is under the Wilcoxon significance test with a significance level of 0.05. As can be seen from the table, except for Time Varying BPSO, Static SBPSO is significantly better than the other benchmark algorithms on half of the datasets. These results are promising since the other algorithms update their parameters during their evolutionary processes while Static SBPSO uses a fixed parameter setting. Dynamic SBPSO is even better than Static SBPSO. Mainly, Dynamic SBPSO is significantly better than Static SBPSO on 60 out of the 79 datasets, and it is never significantly worse than Static SBPSO. These results show that the dynamic mechanism has a positive effect. In comparison with Quantum and Up BPSO, Dynamic SBPSO is significantly better than the two benchmark algorithms on most datasets. Similarly, Dynamic SBPSO is also significantly better than Time Varying BPSO on 44 datasets. Only on two datasets, Dynamic SBPSO is worse than Time Varying BPSO. An interesting fact is that although Dynamic SBPSO significantly outperforms NMBDE on 52 datasets, NMBDE can outperform Dynamic SBPSO on 12 datasets. All the 12 datasets are from the SAC-94 benchmark set, which means that NMBDE works quite well on small datasets but it does not scale well when the datasets are larger. On the knapsack problems, the two SBPSO algorithms also outperform three well-known modified BPSO algorithms: MBPSO [48], IBPSO [49], and PGBPSO (named by us) [50]. More details can be seen in Section IV of the supplementary material.

B. Further analysis of exploitation/exploration

To examine the search ability of the six algorithms, their convergence curves are recorded and shown in Fig. 5. The convergence curve is obtained by averaging the best fitness value achieved by the population in each iteration over the 30 independent runs. For example, if the number of iterations is 100, the evolutionary process has 100 average values.

As can be seen in Fig. 5, on the small datasets such as Pb1, the six algorithms have quite similar patterns. However, Quantum BPSO and Static SBPSO converge earlier and result in worse profits than other algorithms. On datasets with larger

TABLE IV: Average profits of the six algorithms on GK datasets.

Dataset	n	m	NMBDE	Quantum	Up	TimeVarying	Stat	Dyn
GK01	100	15	3.655E3 ± 8.3E0	3.711E3 ± 1.6E1	3.696E3 ± 1.2E1	3.713E3 ± 1.2E1	3.714E3 ± 1.1E1	**3.723E3 ± 1.1E1**
GK02	100	25	3.842E3 ± 1.1E1	3.902E3 ± 1.3E1	3.882E3 ± 1.4E1	3.898E3 ± 1.2E1	3.901E3 ± 1.2E1	**3.910E3 ± 1.3E1**
GK03	150	25	5.484E3 ± 1.0E1	5.556E3 ± 1.4E1	5.527E3 ± 1.5E1	5.559E3 ± 1.4E1	5.558E3 ± 1.2E1	**5.564E3 ± 9.7E0**
GK04	150	50	5.601E3 ± 8.9E0	5.667E3 ± 2.1E1	5.630E3 ± 1.4E1	5.665E3 ± 1.4E1	5.662E3 ± 1.5E1	**5.673E3 ± 1.5E1**
GK05	200	25	7.314E3 ± 1.2E1	7.411E3 ± 1.7E1	7.372E3 ± 1.9E1	7.411E3 ± 1.6E1	7.413E3 ± 2.1E1	**7.423E3 ± 2.1E1**
GK06	200	50	7.460E3 ± 8.1E0	7.535E3 ± 1.5E1	7.492E3 ± 1.8E1	**7.545E3 ± 1.6E1**	7.534E3 ± 1.4E1	7.543E3 ± 1.7E1
GK07	500	25	1.854E4 ± 1.7E1	1.872E4 ± 2.8E1	1.864E4 ± 3.9E1	1.878E4 ± 3.5E1	1.875E4 ± 3.6E1	**1.879E4 ± 3.4E1**
GK08	500	50	1.827E4 ± 1.4E1	1.838E4 ± 3.6E1	1.832E4 ± 3.5E1	**1.844E4 ± 2.3E1**	1.841E4 ± 3.0E1	1.842E4 ± 2.5E1
GK09	1500	25	5.605E4 ± 3.7E1	5.634E4 ± 7.0E1	5.622E4 ± 8.1E1	**5.652E4 ± 6.0E1**	5.642E4 ± 7.8E1	5.648E4 ± 7.6E1
GK10	1500	50	5.573E4 ± 2.5E1	5.593E4 ± 4.9E1	5.583E4 ± 5.3E1	**5.606E4 ± 5.3E1**	5.601E4 ± 6.3E1	5.604E4 ± 5.1E1
GK11	2500	100	9.318E4 ± 3.9E1	9.335E4 ± 3.8E1	9.328E4 ± 5.9E1	9.353E4 ± 4.7E1	9.349E4 ± 5.8E1	**9.354E4 ± 5.8E1**

TABLE V: Average profits of the six algorithms on high-dimensional Knapsack datasets.

Dataset	n	NMBDE	Quantum	Up	TimeVarying	Stat	Dyn
UCI500	500	1.838E5 ± 8.8E2	1.923E5 ± 9.0E2	1.870E5 ± 1.3E3	1.966E5 ± 5.2E2	1.978E5 ± 2.0E2	**1.979E5 ± 1.8E2**
UCI1000	1000	3.346E5 ± 2.3E3	3.738E5 ± 2.4E3	3.585E5 ± 3.9E3	3.929E5 ± 1.3E3	4.004E5 ± 5.5E2	**4.014E5 ± 5.6E2**
UCI2000	2000	6.188E5 ± 2.4E3	7.114E5 ± 7.5E3	6.788E5 ± 4.6E3	7.626E5 ± 4.6E3	7.997E5 ± 1.5E3	**8.030E5 ± 1.4E3**
UCI5000	5000	1.411E6 ± 5.3E3	1.596E6 ± 1.8E4	1.521E6 ± 1.2E4	1.711E6 ± 7.0E3	1.854E6 ± 3.9E3	**1.864E6 ± 5.6E3**
ISCI500	500	1.238E5 ± 1.7E2	1.271E5 ± 2.1E2	1.259E5 ± 2.8E2	1.278E5 ± 2.4E2	1.280E5 ± 1.9E2	**1.283E5 ± 1.5E2**
ISCI1000	1000	2.530E5 ± 2.4E2	2.587E5 ± 4.1E2	2.562E5 ± 3.7E2	2.602E5 ± 4.0E2	2.610E5 ± 4.2E2	**2.616E5 ± 3.4E2**
ISCI2000	2000	5.114E5 ± 3.5E2	5.201E5 ± 5.7E2	5.162E5 ± 8.2E2	5.224E5 ± 8.0E2	5.243E5 ± 5.9E2	**5.248E5 ± 7.2E2**
ISCI5000	5000	1.266E6 ± 4.9E2	1.281E6 ± 1.1E3	1.274E6 ± 9.6E2	1.285E6 ± 1.3E3	1.286E6 ± 1.2E3	**1.286E6 ± 1.4E3**
SCI500	500	1.544E5 ± 2.2E2	1.581E5 ± 2.4E2	1.567E5 ± 3.0E2	1.587E5 ± 2.4E2	1.590E5 ± 2.0E2	**1.593E5 ± 2.1E2**
SCI1000	1000	3.150E5 ± 2.4E2	3.212E5 ± 3.1E2	3.185E5 ± 4.2E2	3.227E5 ± 4.0E2	3.235E5 ± 3.5E2	**3.241E5 ± 3.5E2**
SCI2000	2000	6.136E5 ± 3.6E2	6.236E5 ± 7.0E2	6.188E5 ± 8.2E2	6.262E5 ± 8.4E2	6.282E5 ± 5.6E2	**6.290E5 ± 4.6E2**
SCI5000	5000	1.515E6 ± 6.2E2	1.532E6 ± 1.3E3	1.524E6 ± 9.7E2	1.537E6 ± 1.6E3	1.538E6 ± 1.0E3	**1.539E6 ± 1.2E3**
WCI500	500	1.317E5 ± 2.6E2	1.362E5 ± 2.6E2	1.345E5 ± 4.4E2	1.371E5 ± 2.7E2	1.376E5 ± 1.8E2	**1.379E5 ± 1.5E2**
WCI1000	1000	2.684E5 ± 4.2E2	2.764E5 ± 6.4E2	2.728E5 ± 5.5E2	2.786E5 ± 5.5E2	2.803E5 ± 3.3E2	**2.810E5 ± 4.4E2**
WCI2000	2000	5.184E5 ± 4.5E2	5.310E5 ± 9.2E2	5.257E5 ± 1.1E3	5.354E5 ± 1.1E3	5.387E5 ± 7.1E2	**5.400E5 ± 9.1E2**
WCI5000	5000	1.275E6 ± 8.4E2	1.296E6 ± 1.3E3	1.287E6 ± 1.4E3	1.303E6 ± 2.5E3	1.307E6 ± 1.2E3	**1.308E6 ± 1.3E3**

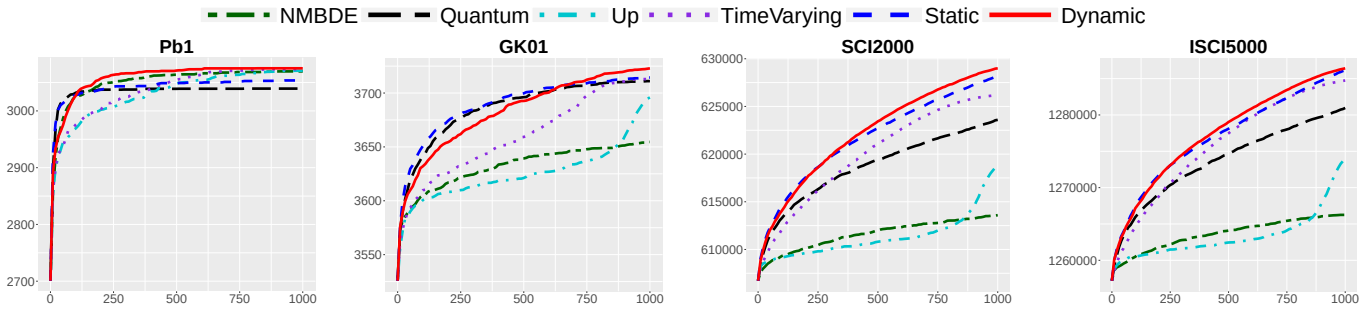

Fig. 5: Convergence curves of the six algorithms (the number of items increases from left to right).

TABLE VI: Better/Similar/Worse on 79 Knapsack datasets (under the Wilcoxon test with a significance level of 0.05).

	NMBDE	Quantum	Up	TimeVarying	Stat
Stat	39/18/22	32/47/0	43/21/15	17/39/23	
Dyn	52/15/12	72/7/0	58/21/0	44/33/2	60/19/0

numbers of items, the differences between the six algorithms are more significant. Up BPSO starts slower than the other algorithms. Its profit is significantly improved in the last 100 iterations. Although NMBDE achieves a better profit than Up BPSO in the first 900 iterations, its final profit is much worse than Up BPSO due to the significant improvement of Up BPSO in the last 100 iterations. The possible reason is that Up BPSO has a better control between exploration and exploitation. In the first 900 iterations, Up BPSO focuses more on exploration than NMBDE, so its profit is not as good as that of NMBDE. However, in the last 100 iterations, Up BPSO focuses more on exploiting the discovered promising regions, which results in its superiority over NMBDE. However, in comparison with the other four BPSO algorithms, Up BPSO has a worse performance despite its significant profit improvement at the end. The main reason is the other four BPSO algorithms steadily improves their profits, and they quickly leave Up BPSO behind.

In comparison with Static SBPSO, Dynamic SBPSO focuses more on exploration at the beginning, so its profit is slightly worse than of Static SBPSO. However, when Dynamic SBPSO focuses more on exploitation, its profit is improved more, which results in the fact that Dynamic SBPSO achieves better profit than Static SBPSO. Given 1000 iterations, some algorithms such as Up BPSO have a rapid increment in the later iterations, so we further compare the six algorithms with 3000 iterations. However, the pattern of 3000 iterations is similar to that of 1000 iterations since the algorithms can adapt with different maximum numbers of iterations because of their dynamic mechanisms. More details can be seen in Section III of the supplementary material.

To quantify the exploration/exploitation abilities of the six algorithms, we record their population diversities during

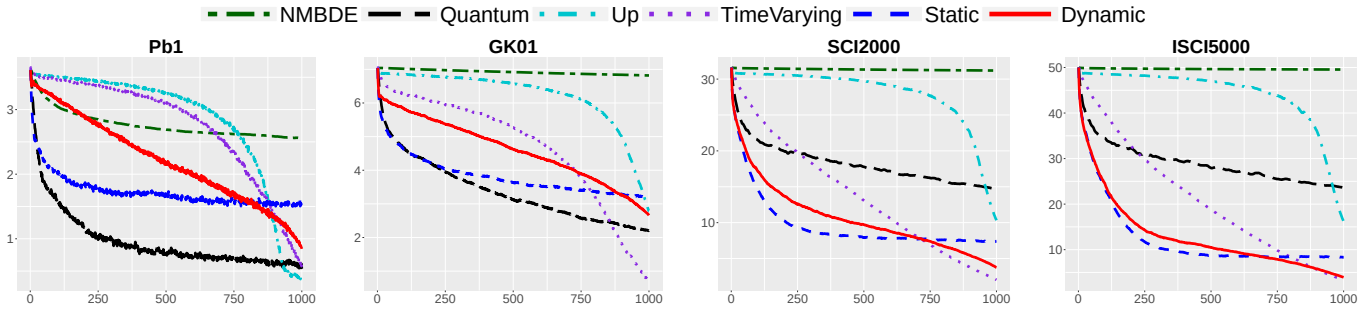

Fig. 6: Diversity profiles of the six algorithms.

their evolutionary processes. Particularly, the diversity at each iteration is calculated by the average distances between all population members. The diversity profiles of the six algorithms are shown in Fig. 6. It can be seen that, on the large datasets, NMBDE does not converge, which results in its poor performance. This result illustrates that NMBDE encounters troubles when dealing with large and complex search spaces. Among BPSO algorithms, Up BPSO has the worst control between exploration and exploitation. Its average distance between particles is significantly reduced in the last 100 iterations, which results in its sharp profit improvement at the end. However, this strategy should be avoided since its final profit is still not as good as other control strategies. Quantum BPSO and Static SBPSO converge quicker than Dynamic SBPSO and Time Varying BPSO. It is understandable since Static SBPSO is a static algorithm. Quantum BPSO indirectly controls the trade-off between exploration and exploitation through a *sine()* function. Meanwhile, both Dynamic SBPSO and Time Varying BPSO directly modifies the flipping probability to control the trade-off. In comparison between Dynamic SBPSO and Time Varying BPSO, it seems that Time Varying BPSO has a better dynamic mechanism since it starts with a higher diversity – more exploration and ends with a lower diversity – more exploitation. However, the final profit obtained by Dynamic SBPSO is still better than that of Time Varying BPSO.

To understand more about the search behavior of Time Varying BPSO and Dynamic SBPSO, we utilize a concept called "evolutionary factor" proposed in [51]. Firstly, the mean distance from each particle i to the other particles is calculated, called d_i. The evolutionary factor f is calculated as follows:

$$f = \frac{d_g - d_{min}}{d_{max} - d_{min}} \in [0, 1] \tag{15}$$

where d_g is the distance (d_i) of the globally best particle, d_{min} and d_{max} are the minimum and maximum values of all d_i. The evolutionary factors of Dynamic SBPSO and Time Varying BPSO on SCI2000 are shown in Fig. 7. It can be seen that for Dynamic SBPSO, there are many iterations where f = 0. According to Eq. (15), f=0 if $d_g = d_{min}$ or the globally best particle has the smallest average distance to other particles. Therefore, f=0 indicates that all the particles search around the globally best particle. A consistent pattern of Dynamic SBPSO is that f is equal to 0 for some iterations and then f is greater than 0 for some iterations. This pattern is repeated during the whole evolutionary process of Dynamic SBPSO. Thus, Dynamic SBPSO performs exploration and exploitation alternatively, i.e., exploring (finding) a promising search region and then exploiting the discovered region. This search behavior is very different from that of Time Varying BPSO, which sequentially focuses on exploration (large f values) at the beginning and exploitation (small f values) at the end.

Experimental results show that the alternating approach is more beneficial to large and complex search spaces. The exploration steps aim to discover promising regions in the search space; the exploitation steps aim to find the best point within a promising region. In the sequential approach, PSO may forget some discovered promising regions which might contain an optimal solution. The alternating approach can partially avoid that by exploiting discovered promising regions as they are found. Furthermore, even if PSO does not lose any promising regions, PSO usually splits its computation resources evenly on different promising regions in the exploitation step. In the sequential approach, the exploration may result in a large number of promising regions, so there will be only a little computation resource allocated to each region, which may lead to these regions not being well exploited, which does not result in good solutions. The alternating approach exploits the promising regions as they are found, so for each iteration, the number of promising regions exploited by the alternating approach is usually smaller than that of the sequential approach. Therefore, the alternating approach puts more computation resources in each region, which might result in better solutions. Most importantly, the exploitation process usually results in better quality solutions than the exploration process. Therefore, alternating between exploration and exploitation usually leads to better swarm quality during the evolutionary process, which is more likely to result in better final solutions than the sequential approach, given the same computation cost.

C. Computation time

The efficiency of the six algorithms is measured by computation time, which is shown in Table VII. In the table, the shortest computation time on each dataset is marked in bold, and the second-shortest one is underlined. Note that the table shows computation time only on GK datasets due to the page limit. The same results are obtained in the other datasets. Full average and standard deviation of computation time on all GK and high-dimensional datasets are shown in

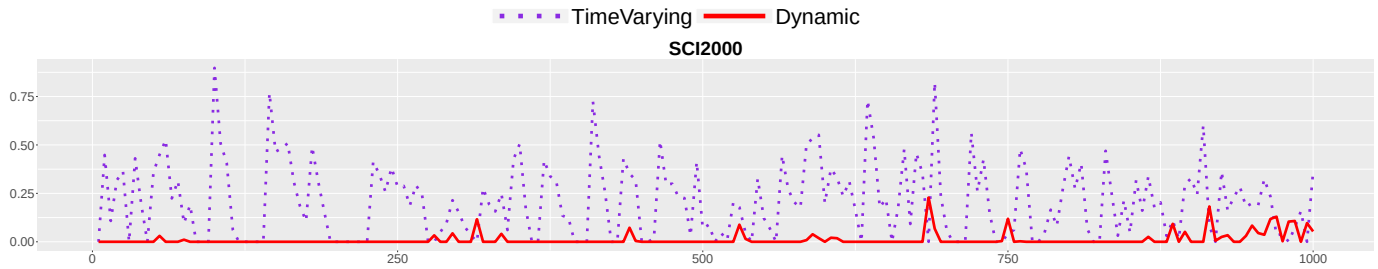

Fig. 7: Evolutionary factors of Dynamic SBPSO and Time Varying.

TABLE VII: Computation time on GK datasets (seconds).

Dataset	NMBDE	Quantum	Up	TimeVarying	Stat	Dyn
GK01	1.21	1	1.36	1.36	**0.6**	0.63
GK02	1.23	1.03	1.39	1.39	**0.64**	0.67
GK03	1.82	1.55	2.06	2.05	**0.94**	0.99
GK04	1.91	1.65	2.16	2.15	**1.04**	1.09
GK05	2.4	2.05	2.74	2.71	**1.26**	1.33
GK06	2.53	2.19	2.87	2.86	**1.39**	1.47
GK07	5.98	5.15	6.74	6.7	**3.63**	3.83
GK08	6.28	5.44	7.05	7.02	**4.03**	4.18
GK09	17.7	15.55	20.13	20.01	**12.37**	12.78
GK10	18.54	16.45	21.04	20.96	**13.39**	13.75
GK11	34.57	31.38	38.85	38.95	**24.16**	24.32

Supplementary Material. As can be seen from Table VII, Static SBPSO is the most efficient algorithm. The main reason is that SBPSO involves fundamental and efficient operators such as *addition, subtraction, multiplication*. Meanwhile, Quantum BPSO uses a $sine()$ function to convert from a real value to a value in the range [0,1], which makes it more expensive than SBPSO. NMBDE, Up BPSO, Time Varying BPSO are the three most computationally intensive algorithms since they use a logistic function to convert from a real value to a value in the range [0,1]. The logistic function is expensive since it involves an exponential calculation. The experimental results show that SBPSO is more effective and efficient than other binary benchmark algorithms. Compared with Static SBPSO, Dynamic SBPSO is a bit slower due to its dynamic mechanism. Therefore, Dynamic SBPSO provides a good trade-off between effectiveness and efficiency.

VII. Experimental Results on Feature Selection

In this section, the five algorithms are compared on 15 UCI feature selection datasets.

A. Feature Subsets

The comparisons on four terms: training accuracy, testing accuracy, the number of selected features and computation time are shown in Tables VIII and IX. In the tables, "All" means that all features are used in the classification process. The bold numbers/accuracies indicate that the corresponding algorithms achieve the highest performance. Static and dynamic SBPSO algorithms are compared with other algorithms using a Wilcoxon significance test with a significance level of 0.05.

As can be seen from Tables VIII and IX, on most datasets, the two SBPSO versions can successfully evolve small feature subsets with similar or better classification accuracy than using all features. For example on Arrhythmia, static and dynamic SBPSO select less than 28% of the original features while achieving almost 2% higher accuracy than using all features.

Regarding training accuracies, Dynamic SBPSO achieves the best classification performance on 10 out of the 15 datasets. Meanwhile, Time Varying BPSO can achieve the best training accuracies on only three out of the 15 datasets. Static SBPSO is worse than only Time Varying BPSO, and it achieves the best training accuracy on one dataset and the second-best accuracy on four other datasets.

Regarding feature subset sizes, the two versions of SBPSO completely dominates the other benchmark algorithms. While Static SBPSO selects the lowest number of features on most small and medium (less than 100 features) datasets, Dynamic SBPSO selects the lowest number of features on the datasets with large numbers of features.

Table IX shows the comparisons in regarding testing accuracies and computation times. Dynamic SBPSO achieves the best testing accuracy on five datasets and the second-best accuracy on three out of the 15 datasets, which is the same as Time Varying BPSO. Note that Dynamic SBPSO usually selects smaller numbers of features than Time Varying BPSO. It seems that testing accuracies of Dynamic SBPSO are not as good as its training accuracies, which indicate an overfitting problem on some datasets where the selected features are only good for the training set. In feature selection, the computation cost heavily depends on the number of selected features. Therefore, both SBPSO versions achieve good efficiency, especially on the large datasets where SBPSO algorithms select much smaller numbers of features than the other benchmark algorithms.

Table X shows the comparison results between two versions of SBPSO and the other benchmark algorithms. Both Static and Dynamic SBPSO are mostly similar or significantly better than the other benchmark algorithms regarding training accuracies, numbers of selected features and testing accuracies. In general, on all datasets, SBPSO can achieve at least the highest classification accuracy or the smallest number of selected features, which are the two main objectives of feature selection. However, since both objectives contribute to the fitness function, analyzing them separately cannot show the searching abilities of the algorithms. Therefore, the evolutionary process is investigated in the next subsection.

TABLE VIII: Number of selected features and training accuracies on Feature Selection.

Dataset	Number of selected features							Training accuracies						
	All	NMBDE	Quantum	Up	TV	Stat	Dyn	All	NMBDE	Quantum	Up	TV	Stat	Dyn
Wine	13.00	**5.33**	5.47	5.50	5.53	**5.33**	5.70	88.17	96.25	96.04	96.38	**96.58**	96.32	96.40
Australian	14.00	3.10	3.03	**3.00**	**3.00**	**3.00**	**3.00**	75.78	86.84	86.19	**86.96**	**86.96**	**86.96**	**86.96**
Zoo	17.00	**4.03**	4.43	4.13	4.20	**4.03**	4.07	86.72	97.62	**97.83**	97.64	97.72	97.62	97.62
Vehicle	18.00	7.87	8.03	8.00	8.17	**7.27**	7.60	88.51	89.78	89.91	89.86	**90.00**	89.78	89.80
German	24.00	11.23	11.00	12.20	10.17	**9.90**	10.23	80.14	**81.51**	81.36	81.43	81.15	80.88	81.06
WBCD	30.00	4.00	**3.33**	4.00	3.90	3.60	3.87	94.97	96.48	96.05	**96.55**	96.41	96.18	96.43
Ionosphere	34.00	5.60	5.60	6.20	5.30	4.87	**4.80**	85.77	94.07	94.11	93.81	94.16	94.00	**94.22**
Sonar	60.00	19.70	**15.07**	21.30	16.87	16.67	16.07	83.45	91.72	92.16	92.55	92.16	92.13	**92.62**
Movementlibras	90.00	26.63	23.37	26.20	23.57	**22.33**	22.77	97.88	97.91	97.87	97.86	97.92	97.92	**97.93**
Hillvalley	100.00	40.10	36.50	40.77	39.13	**35.13**	36.57	71.46	73.77	74.12	73.71	73.95	74.22	**74.23**
Musk1	166.00	77.83	73.53	76.33	73.67	71.37	**68.70**	92.19	94.63	95.16	94.57	95.28	95.34	**95.53**
Arrhythmia	278.00	106.13	93.13	108.73	89.47	82.03	**79.60**	94.35	95.43	95.79	95.56	95.85	95.81	**96.00**
Madelon	500.00	230.63	219.90	226.50	215.67	215.03	**210.70**	83.24	88.47	89.63	88.54	89.72	89.75	**90.14**
Isolet5	617.00	246.83	199.53	245.87	195.50	187.07	**185.93**	99.20	99.40	99.52	99.44	99.52	99.54	**99.55**
MultipleFeatures	649.00	252.37	201.93	248.60	193.63	135.67	**133.50**	99.33	99.57	99.61	99.61	99.62	99.62	**99.62**

TABLE IX: Computation times and testing accuracies on Feature Selection.

Dataset	Computation times (seconds)						Testing accuracies						
	NMBDE	Quantum	Up	TV	Stat	Dyn	All	NMBDE	Quantum	Up	TV	Stat	Dyn
Wine	3.28	3.26	3.26	**3.25**	3.28	3.29	76.54	97.53	97.00	97.78	**98.15**	97.74	97.82
Australian	53.27	**53.01**	53.76	53.66	54.14	54.47	70.05	**85.65**	84.87	85.51	85.51	85.51	85.51
Zoo	1.48	**1.45**	1.47	1.48	1.47	1.50	80.00	95.24	95.21	95.24	**95.30**	95.24	95.24
Vehicle	106.52	**105.56**	106.12	106.31	105.57	107.11	84.06	85.15	85.11	85.13	**85.23**	85.21	85.18
German	206.90	204.64	204.88	**204.00**	204.15	207.04	68.00	69.10	68.93	69.01	68.66	68.79	**69.15**
WBCD	73.80	74.82	**71.92**	76.70	75.99	78.30	92.98	93.06	**93.57**	93.22	93.10	93.33	93.10
Ionosphere	33.06	**32.35**	33.11	32.82	32.87	33.30	83.81	**88.67**	88.54	87.94	87.84	87.52	87.74
Sonar	20.84	**20.22**	20.93	20.65	20.97	20.92	76.19	80.16	79.89	79.89	80.48	80.37	**80.79**
Movementlibras	105.47	103.25	106.75	103.14	**101.77**	102.58	94.69	94.48	94.64	94.51	94.57	**94.66**	94.55
Hillvalley	1438.87	1422.43	1460.25	1425.26	**1410.05**	1416.21	56.59	57.33	57.80	58.05	**58.38**	57.99	58.03
Musk1	266.75	259.78	258.35	261.79	**253.14**	254.44	83.92	86.55	86.53	85.80	86.55	85.80	**86.99**
Arrhythmia	273.21	263.63	286.21	267.64	**254.00**	258.86	93.78	94.60	94.90	94.65	94.88	94.88	**95.08**
Madelon	13666.27	13061.01	13418.19	13170.90	13020.71	**12690.82**	70.90	78.59	79.74	78.76	79.55	79.39	**79.94**
Isolet5	4874.18	4557.99	5222.76	4659.38	4458.97	**4298.37**	98.36	98.69	98.89	98.76	**98.89**	98.85	98.87
MultipleFeatures	8308.84	7561.56	8641.53	7730.79	**6820.79**	7111.59	98.57	99.04	**99.10**	99.02	99.08	99.06	99.08

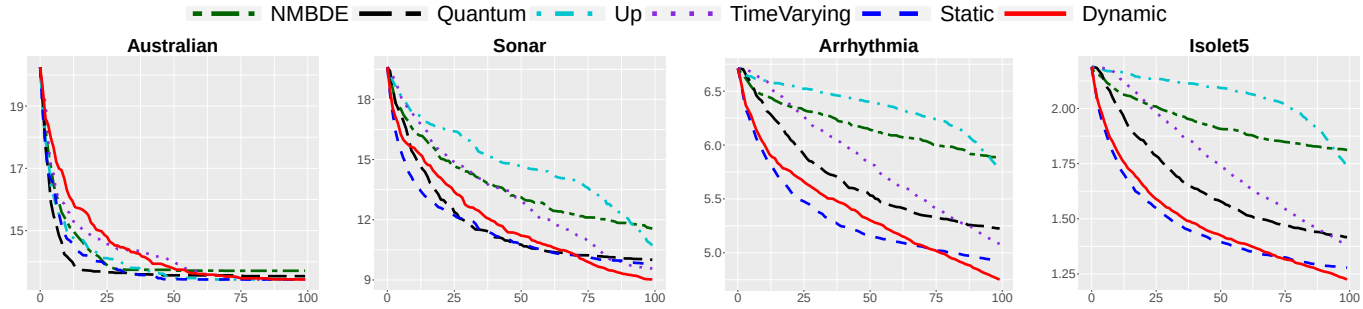

Fig. 8: Convergence curves of the six algorithms on Feature Selection.

TABLE X: Significance test results on Feature Selection.

Method	NMBDE	Quantum	Up	TimeVarying	Stat
Training Accuracy					
Stat	5/8/2	2/12/1	4/9/2	1/13/1	
Dyn	6/9/0	5/9/1	6/8/1	3/12/0	3/12/0
Number of features					
Stat	11/4/0	5/10/9	11/4/0	5/10/0	
Dyn	8/7/0	6/8/1	10/5/0	5/10/0	0/15/0
Testing Accuracy					
Stat	5/9/1	1/13/1	3/12/0	0/15/0	
Dyn	4/10/1	2/12/1	5/10/0	1/14/0	2/13/0

B. Evolutionary Processes

Similar to Knapsack, the best fitness value is recorded for each iteration. However, since each algorithm is run 30 independent times, the average of 30 best fitness values in each generation from the 30 runs is recorded. Therefore, for each algorithm, there will be 100 average fitness values corresponding to the 100 iterations. The 100 average values are used to draw an algorithm's convergence curve, which is shown in Fig. 8. In feature selection, the target is to minimize the fitness function so the lower the fitness value, the better the algorithm. In the figure, only four evolutionary processes are shown, since the processes are similar on the other datasets.

As can be seen from the figure, Up BPSO and NMBDE still perform worst on most datasets, especially on datasets with large numbers of features. In comparison with other benchmark algorithms, Static SBPSO evolves better feature subsets with smaller fitness values on large datasets such as Arrhythmia and Isolet5. On small and medium datasets, Static SBPSO is worse than at most one benchmark algorithm although Static SBPSO does not use any dynamic mechanism to control its parameters. This result illustrates the stability of Static SBPSO on different binary problems.

On most datasets, Dynamic SBPSO achieves the best fitness value. Similar to Knapsack, Static SBPSO usually generates better solutions than Dynamic SBPSO at the beginning. The main reason is that Dynamic SBPSO focuses on exploration

more than Static SBPSO at the beginning, it tends to discover more promising regions rather than focusing on specific regions. Therefore, once Dynamic SBPSO focuses more on exploitation, it can improve its fitness value over Static SBPSO.

VIII. Conclusions and Future Work

In this paper, new metaphors for BPSO are introduced to consider the properties of binary search spaces. Notably, the velocity of BPSO is redefined as a flipping probability vector, and the momentum is redefined as the tendency to "stick" with the current position. A primary advantage of the proposed algorithm is that the two new metaphors describe the movement of binary particles as bouncing between binary points in a binary search space. Therefore, the proposed algorithm reflects binary movements more naturally and accurately which makes it easier to define and control exploration and exploitation. Based on that, a dynamic strategy is also developed to manage the contributions of momentum, $pbest$ and $gbest$ to the movement of particles, which results in the balance between exploration and exploitation during the evolutionary process. The two versions of the proposed BPSO algorithm, i.e., Static and Dynamic SBPSO, are compared with four well-known/state-of-the-art dynamic algorithms NMBDE, Quantum BPSO, Up and Time Varying BPSO on two binary problems: knapsack and feature selection. The experimental results show that Dynamic SBPSO can achieve better performance than the other four benchmark algorithms on most of the cases. Although Static SBPSO is not as good as Dynamic SBPSO, it still evolves better solutions than the other four dynamic algorithms on large and complex search spaces. The superiority of Dynamic SBPSO over Static SBPSO illustrates that it is better to focus more on exploration at the beginning and gradually shift towards exploitation. More importantly, on a large and complex search space, it is better to take an alternating approach which alternates between exploration and exploitation than a sequential approach which performs all the exploration followed by all the exploitation. The alternating mechanism is a built-in property of SBPSO, and it is expected that SBPSO can be applied to other complex binary problems. Another advantage of the proposed algorithm is its simple computation consisting of basic mathematic operators, which leads to a high efficiency of the proposed algorithm.

Although SBPSO achieves promising results, there is future work can be done to further improve SBPSO. For example, Dynamic SBPSO performs well on the large and complex search spaces. However, the performance is not significantly improved on the small problems over the benchmark algorithms. This limitation might be tackled by developing more sophisticated updating mechanism for parameters, for instance, an adaptive/non-linear updating mechanism.

References

[1] J. Kennedy, R. Eberhart *et al.*, "Particle swarm optimization," in *Proceedings of IEEE international conference on neural networks*, vol. 4, no. 2, 1995, pp. 1942–1948.

[2] M. R. AlRashidi and M. E. El-Hawary, "A survey of particle swarm optimization applications in electric power systems," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 4, pp. 913–918, 2009.

[3] Z. Zhu, J. Zhou, Z. Ji, and Y. H. Shi, "DNA sequence compression using adaptive particle swarm optimization-based memetic algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 5, pp. 643–658, 2011.

[4] W. Hu and G. G. Yen, "Adaptive multiobjective particle swarm optimization based on parallel cell coordinate system," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 1, pp. 1–18, 2015.

[5] Y.-J. Gong, J.-J. Li, Y. Zhou, Y. Li, H. S.-H. Chung, Y.-H. Shi, and J. Zhang, "Genetic learning particle swarm optimization," *IEEE Transactions on Cybernetics*, vol. 46, no. 10, pp. 2277–2290, 2016.

[6] W. N. Chen, J. Zhang, H. S. H. Chung, W. L. Zhong, W. G. Wu, and Y. h. Shi, "A novel set-based particle swarm optimization method for discrete optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 2, pp. 278–300, 2010.

[7] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[8] M. Eddaly, B. Jarboui, and P. Siarry, "Combinatorial particle swarm optimization for solving blocking flowshop scheduling problem," *Journal of Computational Design and Engineering*, vol. 3, no. 4, pp. 295 – 311, 2016.

[9] H. Banka and S. Dara, "A hamming distance based binary particle swarm optimization (HDBPSO) algorithm for high dimensional feature selection, classification and validation," *Pattern Recognition Letters*, vol. 52, pp. 94 – 100, 2015.

[10] B. H. Nguyen, B. Xue, and P. Andreae, *A Novel Binary Particle Swarm Optimization Algorithm and Its Applications on Knapsack and Feature Selection Problems*. Cham: Springer International Publishing, 2017, pp. 319–332.

[11] C. Blum and X. Li, "Swarm intelligence in optimization," in *Swarm Intelligence*. Springer, 2008, pp. 43–85.

[12] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.

[13] F. van den Bergh and A. Engelbrecht, "A study of particle swarm optimization particle trajectories," *Information Sciences*, vol. 176, no. 8, pp. 937 – 971, 2006.

[14] M. A. Khanesar, M. Teshnehlab, and M. A. Shoorehdeli, "A novel binary particle swarm optimization," in *Mediterranean Conference on Control Automation(MED).*, 2007, pp. 1–6.

[15] J. Liu, Y. Mei, and X. Li, "An analysis of the inertia weight parameter for binary particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 5, pp. 666–681, 2016.

[16] K. Sarath and V. Ravi, "Association rule mining using binary particle swarm optimization," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 8, pp. 1832–1840, 2013.

[17] M. A. Taha and D. I. A. al Nadi, "Spectrum sensing for cognitive radio using binary particle swarm optimization," *Wireless personal communications*, vol. 72, no. 4, pp. 2143–2153, 2013.

[18] J. C.-W. Lin, L. Yang, P. Fournier-Viger, T.-P. Hong, and M. Voznak, "A binary PSO approach to mine high-utility itemsets," *Soft Computing*, pp. 1–19, 2016.

[19] 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.

[20] M. J. Islam, X. Li, and Y. Mei, "A time-varying transfer function for balancing the exploration and exploitation ability of a binary pso," *Applied Soft Computing*, vol. 59, pp. 182–196, 2017.

[21] J. C. Bansal and K. Deep, "A modified binary particle swarm optimization for knapsack problems," *Applied Mathematics and Computation*, vol. 218, no. 22, pp. 11 042–11 061, 2012.

[22] Y. Zhang, S. Wang, P. Phillips, and G. Ji, "Binary PSO with mutation operator for feature selection using decision tree applied to spam detection," *Knowledge-Based Systems*, vol. 64, pp. 22–31, 2014.

[23] J. Yang, H. Zhang, Y. Ling, C. Pan, and W. Sun, "Task allocation for wireless sensor network using modified binary particle swarm optimization," *IEEE Sensors Journal*, vol. 14, no. 3, pp. 882–892, 2014.

[24] A. H. El-Maleh, A. T. Sheikh, and S. M. Sait, "Binary particle swarm optimization (BPSO) based state assignment for area minimization of sequential circuits," *Applied soft computing*, vol. 13, no. 12, pp. 4832–4840, 2013.

[25] T. Zhai and Z. He, "Instance selection for time series classification based on immune binary particle swarm optimization," *Knowledge-Based Systems*, vol. 49, pp. 106–115, 2013.

[26] G. Pampara, N. Franken, and A. P. Engelbrecht, "Combining particle swarm optimisation with angle modulation to solve binary problems," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 1. IEEE, 2005, pp. 89–96.

[27] Y.-W. Jeong, J.-B. Park, S.-H. Jang, and K. Y. Lee, "A new quantum-inspired binary pso: application to unit commitment problems for power systems," *IEEE Transactions on Power Systems*, vol. 25, no. 3, pp. 1486–1495, 2010.

[28] H. bin Ouyang, L. qun Gao, S. Li, and X. yong Kong, "Improved global-best-guided particle swarm optimization with learning operation for global optimization problems," *Applied Soft Computing*, vol. 52, pp. 987 – 1008, 2017.

[29] M. F. Tasgetiren, Q. K. Pan, D. Kizilay, and G. Suer, "A differential evolution algorithm with variable neighborhood search for multidimensional knapsack problem," in *2015 IEEE Congress on Evolutionary Computation (CEC)*, 2015, pp. 2797–2804.

[30] J. P. Martins, H. Longo, and A. C. Delbem, "On the effectiveness of genetic algorithms for the multidimensional knapsack problem," in *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO Comp '14, 2014, pp. 73–74.

[31] H. Ishibuchi, N. Akedo, and Y. Nojima, "Behavior of multiobjective evolutionary algorithms on many-objective knapsack problems," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 2, pp. 264–283, 2015.

[32] M. Chih, "Self-adaptive check and repair operator-based particle swarm optimization for the multidimensional knapsack problem," *Applied Soft Computing*, vol. 26, pp. 378–389, 2015.

[33] ——, "Three pseudo-utility ratio-inspired particle swarm optimization with local search for multidimensional knapsack problem," *Swarm and evolutionary computation*, vol. 39, pp. 279–296, 2018.

[34] H. Zhao, A. P. Sinha, and W. Ge, "Effects of feature construction on classification performance: An empirical study in bank failure prediction," *Expert Systems with Applications*, vol. 36, no. 2, pp. 2633–2644, 2009.

[35] B. Xue, M. Zhang, W. N. Browne, and X. Yao, "A survey on evolutionary computation approaches to feature selection," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 4, pp. 606–626, 2016.

[36] J. Tang, S. Alelyani, and H. Liu, "Feature selection for classification: A review," *Data Classification: Algorithms and Applications*, p. 37, 2014.

[37] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *CoRR*, vol. abs/1601.07996, 2016.

[38] F. Nie, H. Huang, X. Cai, and C. H. Ding, "Efficient and robust feature selection via joint l2,1-norms minimization," in *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 1813–1821.

[39] H. Nguyen, B. Xue, I. Liu, and M. Zhang, "Filter based backward elimination in wrapper based pso for feature selection in classification," in *2014 IEEE Congress on Evolutionary Computation (CEC)*, 2014, pp. 3111–3118.

[40] L. Wang, X. Fu, Y. Mao, M. I. Menhas, and M. Fei, "A novel modified binary differential evolution algorithm and its applications," *Neurocomputing*, vol. 98, pp. 55 – 75, 2012.

[41] A. P. Engelbrecht and G. Pampara, "Binary differential evolution strategies," in *2007. IEEE Congress on Evolutionary Computation (CEC)*, 2007, pp. 1942–1947.

[42] S. Khuri, T. Bäck, and J. Heitkötter, "The zero/one multiple knapsack problem and genetic algorithms," in *Proceedings of the 1994 ACM symposium on Applied computing*. ACM, 1994, pp. 188–193.

[43] F. Glover and G. A. Kochenberger, "Critical event tabu search for multidimensional knapsack problems," in *Meta-Heuristics*. Springer, 1996, pp. 407–427.

[44] D. Pisinger, "Where are the hard knapsack problems?" *Computers and Operations Research*, vol. 32, no. 9, pp. 2271–2284, 2005.

[45] E. Özcan and C. Başaran, "A case study of memetic algorithms for constraint optimization," *Soft Computing*, vol. 13, no. 8, p. 871, Jul 2008.

[46] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[47] 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.

[48] J. C. Bansal and K. Deep, "A modified binary particle swarm optimization for knapsack problems," *Applied Mathematics and Computation*, vol. 218, no. 22, pp. 11 042–11 061, 2012.

[49] M. S. Mohamad, S. Omatu, S. Deris, and M. Yoshioka, "A modified binary particle swarm optimization for selecting the small subset of informative genes from gene expression data," *IEEE Transactions on information Technology in Biomedicine*, vol. 15, no. 6, pp. 813–822, 2011.

[50] S. Lee, S. Soak, S. Oh, W. Pedrycz, and M. Jeon, "Modified binary particle swarm optimization," *Progress in Natural Science*, vol. 18, no. 9, pp. 1161–1166, 2008.

[51] Z. H. Zhan, J. Zhang, Y. Li, and H. S. H. Chung, "Adaptive particle swarm optimization," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 6, pp. 1362–1381, 2009.