

Orthogonal Bandit Learning for Portfolio Selection Under Cardinality Constraint

Mahdi Moeini(✉)

Chair of Business Information Systems and Operations Research (BISOR), Technische Universität Kaiserslautern, 67663 Kaiserslautern, Germany
mahdi.moeini@wiwi.uni-kl.de

Abstract. In this paper, we address the portfolio selection problem and solve it by means of a machine learning approach. More precisely, we study portfolio optimization under *cardinality constraint*, which limits the number of assets in a portfolio. This problem is known to be NP-hard and, consequently, difficult to solve for some settings of the cardinality parameter. In order to solve the problem, we introduce a hybrid approach that combines effectively an existing *bandit learning algorithm* with a *kernel search* heuristic. The bandit learning algorithm is used for conducting online portfolio selections and the kernel search manages the cardinality constraint. In order to investigate the performance of our algorithm, we carried out computational experiments on real-world market data sets. According to the numerical results, we observe that, despite the presence of cardinality constraint, our hybrid algorithm shows good performance in solving the test instances in reasonable computation time and, in many cases, the algorithm provides portfolios with higher cumulative wealth than those obtained through the existing bandit learning algorithm, which is applied on classical portfolio selection problem.

Keywords: Portfolio selection problem · Machine learning · Multi-armed bandit · Bandit learning · Cardinality constraint · Kernel search

1 Introduction

The portfolio selection is one of the classical optimization problems which arise in financial industry. By definition, the objective of the portfolio selection problem consists in finding an optimal way of diversifying a given amount of money among a set of available assets, while each combination of these assets defines a *portfolio* [5,21]. The seminal work of Nobel prize laureate Markowitz [16] provides a model for finding optimal portfolios. In the Mean-Variance (MV) model of Markowitz [16,17], each portfolio is assigned to an amount of return and a level of risk, which is defined by variance of the asset returns [1,5,12]. However, despite its simplicity, the MV model suffers from different aspects, e.g., poor performance in

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11621, pp. 232–248, 2019.
https://doi.org/10.1007/978-3-030-24302-9_17

out-of-sample settings and lack of real-world constraints. Hence, several research papers have focused on bridging the gap between the classical model and the real-world situations such as including *buy-in threshold* [1,12,15,19] or *cardinality constraint* [5–7,9,13,15]. Similarly, alternative optimization models have been introduced to address other shortcomings of the MV model [1,10].

Furthermore, in the age of globalization, the financial industry faces the challenge of processing massive amount of data in order to make the best decisions. To this end, machine learning is an effective tool that might be used and, in particular, the *multi-armed bandit learning* methods have already been applied for portfolio optimization. The bandit learning strategies can fit well in the context of portfolio selection as they use existing knowledge and exploitation-exploration policy to catch new information, to optimize the reward, and to select the most beneficial arms, where each arm is assigned to an asset [3,8]. In the same stream of research, *orthogonal bandit learning algorithm* has been developed, by Shen et al., as an effective tool for finding optimal portfolios [21]. In this approach, *orthogonal portfolios* are constructed, the *Sharp ratio* and reward of each arm is computed, and then the optimal arms are selected to compose the optimal portfolio. However, the orthogonal bandit learning is used for solving a very basic portfolio selection model, which permits also short selling of assets. Hence, in order to handle more practical restrictions, i.e., portfolio selection under cardinality constraint, we extend the orthogonal bandit learning by combining it with a *Kernel Search* (KS) method [6]. The KS is a heuristic framework that starts with an initial set of assets, as the kernel, and then, by solving a sequence of problems, identifies further assets to add them into the kernel. This approach is suitable for handling cardinality constraint. In order to evaluate the performance of our hybrid algorithm, we carried out computational experiments on several real-world test instances and report the numerical results, which confirm the ability of our approach in providing high-quality solutions within reasonable computation time. Furthermore, despite limits imposed by the cardinality constraint, on many instances, our approach provides better portfolio compositions than those obtained through the basic orthogonal bandit learning algorithm.

We organize the remainder of this paper as follows: In Sect. 2, we present the basic concepts that we are going to use throughout this paper. Then, Sect. 3 is devoted to the notation, the description of the portfolio optimization problem, and the bandit learning approach, which we call *basic bandit*, for the portfolio selection. In Sect. 4, we modify and extend the basic bandit to handle cardinality constraint in a portfolio selection problem. Section 5 reports the test settings of our computational experiments and their numerical results. Finally, some conclusions are drawn in Sect. 6.

2 Preliminaries

In this section, we present the basic concepts that the contributions of this article are based on.

2.1 Bandits

The Multi-Armed Bandit (MAB) problem was first stated by Thompson in 1933 [2,22], although in a more specific form. His motivation for studying the bandit problem were clinical trials, where different treatments should be allocated to patients. Later Robbins [20] was one of the first researchers who picked up and studied the model.

By definition, the bandit problem generally describes the situation of a gambler, who must decide which arm of a K-slot machine to pull in order to maximize his total rewards within a fixed number of trials. In order to present a formal description, we follow the notation of [21]. Assume that n arms representing n actions and time steps t_k, where $k = 1, \ldots, m$ are given such that if an arm i is pulled at time t_k, then we earn a bounded reward $r_i(t_k) \in \mathbb{R}$, where $i = 1, \ldots, n$. The objective of bandit learning consists in sequential (and one at a time) selection of suitable arms in order to maximize the total reward, earned by pulling arms, or to minimize the regret. In a stochastic context, where ν_i denotes the expected reward of arm i, the largest expected reward would be $\nu* := max_{1 \leq i \leq n} \nu_i$. If this expected reward takes place in m plays, then the maximum expected reward will be $m\nu^*$ and the *pseudo regret* after playing m times is computed as follows:

$$m\nu^* - \sum_{k=1}^{m} \mathbb{E}[r_{ik}(t_k)], \tag{1}$$

where i_k is used to indicate the index of the arm selected at time t_k and $r_{ik}(t_k)$ defines the corresponding reward (see [21] and references therein).

There are several approaches for solving the MAB problem. An easy and classical method is called the ϵ-greedy strategy: at a given time t, with a probability of $1 - \epsilon$, the arm with the best (highest) reward is pulled; otherwise, a random arm is played with a probability of ϵ. This two sided approach makes a trade-off between *exploitation* versus *exploration*. Another commonly accepted and utilized policy is called the *upper confidence bounds* (UCB) strategy [11,21]. The UCB algorithms establish guided exploration in contrast to simpler strategies like the ϵ-greedy method, which randomly choose the target of exploration. In the UCB, as soon as each arm is played once, the following function is used to select the best arm i^*, at time t_k:

$$i^*(t_k) := \arg\max_{1 \leq i \leq n} \left(\overline{r}_i(t_k) + \sqrt{\frac{2\ ln(k)}{k_i}} \right), \tag{2}$$

where, $\overline{r}_i(t_k)$ and k_i denote the mean reward and the played number of the arm i up to the time t_k [21].

The typical and classical assumptions in bandit learning consists in considering rewards as independent and identically distributed (i.i.d.) for each arm, selecting the best arms at each play, defining the reward mean as the objective, and finally ignoring presence of historical data. These hypotheses are not

respected in financial markets. In fact, if we assume each (financial) asset as an arm, it is well-known that assets are correlated, diversification is a priority in portfolio selection (to reduce risk of investment), the objective of portfolio optimization is minimizing risk as well as maximizing return, and the historical data are available and used. In order to bridge this gap, Shen et al. [21] developed a new bandit learning algorithm that we will discuss and extend in Sects. 3 and 4, respectively.

2.2 Portfolio Selection

Assume that we would like to invest a given capital, among a set of available assets, and in a discrete-time and finite horizon environment. The time horizon consists of time-steps $t_k = k\ \Delta t$, where k stands for the time-step index and Δt represents the duration between two time-steps. There is no specification with regard to which duration a time-step has to represent, i.e., it can be one day, one week, one month, or even one year, which basically depends on the underlying data. Given n risky assets, the prices of i-th asset at time t_k is $S_{k,i}$ (to simplify the notation, we might represent t_k by k, whenever it causes no ambiguity) and $R_{k,i}$ defines the return of asset i from time t_{k-1} to t_k, and computed by $R_{k,i} := S_{k,i}/S_{k-1,i}$. In the following, the whole set of assets is referred to as asset pool A, consisting of all n risky assets.

Let us denote the investment decision by a column vector $\omega_k = (\omega_{k1}, \ldots, \omega_{kn})^\top$ with the respective weight ω_{ki} (in percentage) of the i-th asset at time-step k. The sum over all weights must be equal to 1, which corresponds to the usage of the whole budget. Shen et al. [21] assume that *short selling* is permitted. Short selling means that the investor borrows the asset for sale as he presumes falling prices. If his prediction becomes true and the value decreases, he buys the same asset for a cheaper price and makes profit. In the opposite case, the investor will have loss. Therefore, $\omega_{ki} > 0$ refers to long- whereas $\omega_{ki} < 0$ indicates short-positions [21].

3 Bandit Learning with Orthogonal Portfolio

In this section, we present the algorithm of Shen et al. [21] that we call *Basic Bandit* (BB). It might be used for portfolio selection without cardinality constraint. Then, in Sect. 4, we will explain how we extend BB to take into account cardinality constraint.

Using return R_{ik} of each asset i at time t_k, we can compute the covariance matrix, denoted by $\mathbf{\Sigma_k}$, of the n assets. For this purpose, we might use a factor model as described in [4], which is suitable for computing large-scale covariance matrices. Then, we can decompose $\mathbf{\Sigma_k}$ into its principal components:

$$\mathbf{\Sigma_k} = \mathbf{H_k \Lambda_k H_k}^\top, \tag{3}$$

where $\mathbf{\Lambda_k}$ is a diagonal matrix containing the eigenvalues $(\lambda_{k,1}, \ldots, \lambda_{k,n})$ of the matrix $\mathbf{\Sigma_k}$, sorted in decreasing order. Furthermore, the columns of the matrix $\mathbf{H_k}$ are actually the eigenvectors of the corresponding eigenvalues of $\mathbf{\Sigma_k}$.

The basic bandit finds a portfolio allocation based on the asset pool A and provides the weights for the assets. However, in order to satisfy the condition of using the whole budget in percentage, the eigenvectors need to be normalized by their sum according to

$$\tilde{H}_{k,i} = \frac{\boldsymbol{H}_{k,i}}{\boldsymbol{H}_{k,i}^{\top}\mathbf{1}}, \tag{4}$$

where $\mathbf{1}$ is a vector with all entries equal to 1. In this way, we obtain n uncorrelated portfolios with the return of $\tilde{H}_{k,i}R_k$, and the covariance matrix of these returns is computed as follows:

$$\tilde{\boldsymbol{\Sigma}}_{\mathbf{k}} = \tilde{\mathbf{H}}_{\mathbf{k}}\boldsymbol{\Sigma}_{\mathbf{k}}\tilde{\mathbf{H}}_{\mathbf{k}}^{\top} = \tilde{\boldsymbol{\Lambda}}_{\mathbf{k}}. \tag{5}$$

The entries on the diagonal of $\tilde{\boldsymbol{\Lambda}}_{\mathbf{k}}$ represent the respective variances of the portfolio i at time-step k as $\tilde{\lambda}_{k,i} = \lambda_{k,i}/(\boldsymbol{H}_{k,i}^{\top}\mathbf{1})^2$. The new portfolios are called *orthogonal portfolios* [21]. According to [18], the orthogonal portfolios represent the risk factors in the market and we can use eigenvector directions to characterize market fluctuations [21].

Additionally, some empirical studies revealed that the covariance matrix can be decomposed into two groups of factors as follows:

$$\tilde{\boldsymbol{\Sigma}}_{\mathbf{k}} = \sum_{i=1}^{l} \tilde{\lambda}_{k,i}\tilde{H}_{k,i}\tilde{H}_{k,i}^{\top} + \sum_{i=l+1}^{n} \tilde{\lambda}_{k,i}\tilde{H}_{k,i}\tilde{H}_{k,i}^{\top}, \tag{6}$$

where the first summation is characterized as the *systematic* movement of the market and the second one represents the *idiosyncratic* risks [21]. While the first one should be followed (passive investment), the second one contains a potential, which can be exploited by the investor (active investment).

After dividing the arms into two groups, we need to determine the best arm out of each group. A way to define an order is to compute the Sharpe ratio $\bar{r}_{k,i}$ as it takes directly into account the return per risk unit. At time-step k, the Sharpe ratio of the i-th orthogonal portfolio, which is represented by the i-th column vector, is computed by

$$\bar{r}_{k,i} = \frac{\boldsymbol{H}_{k,i}\mathbb{E}[R_{k,i}]}{\sqrt{\lambda_{k,i}}}, \tag{7}$$

where $\mathbb{E}[R_{k,i}]$ can be estimated by the James-Stein Shrinkage Estimator [18].

After retrieving the order, based on the Sharpe ratio of each arm, the optimal arm of each subset is computed by taking the one-sided confidence bound into the reward function:

$$\begin{aligned} i_k^* &= \underset{i=1,\ldots,l}{\arg\max}\, \bar{r}_{k,i} + \sqrt{\frac{2\ln(k+\tau)}{\tau + k_i}}, \\ j_k^* &= \underset{i=l+1,\ldots,n}{\arg\max}\, \bar{r}_{k,i} + \sqrt{\frac{2\ln(k+\tau)}{\tau + k_i}}, \end{aligned} \tag{8}$$

where k_i is the number of times the i-th arm has been selected and τ stands for the length of the sliding time window. At this point, the arms of the bandit have been divided into two subsets and from each group, the best arms i_k^* and j_k^*, according to the Sharpe ratio, have been determined. Then, we compute a weighted average of the selected two best arms in order to minimize the total variance of the selected portfolio. More precisely, assume that $\tilde{H}_{k,i_k^*}$ and $\tilde{H}_{k,j_k^*}$ are the uncorrelated portfolios corresponding to the best selected arms i_k^* and j_k^*. Then, we minimize the total variance $\lambda_{k,p} = \theta_k^2 \tilde{\lambda}_{k,j_k^*} + (1-\theta_k)^2 \tilde{\lambda}_{k,i_k^*}$ to obtain the portfolio mixture weight θ_k^* as follows

$$\theta_k^* = \arg\min_{\theta_k} \lambda_{k,p} = \frac{\tilde{\lambda}_{k,i_k^*}}{\tilde{\lambda}_{k,j_k^*} + \tilde{\lambda}_{k,i_k^*}}. \tag{9}$$

The value of θ_k^* is then used to obtain the portfolio mixture through

$$\omega_k = (1 - \theta_k^*)\tilde{H}_{k,i_k^*} + \theta_k^* \tilde{H}_{k,j_k^*}. \tag{10}$$

The output of each iteration through the algorithm is the weights vector ω_k which contains the weights for each asset from the asset pool. In order to investigate the forecast quality of the algorithm, we compute the net return of the portfolio μ_k, from time t_{k-1} to t_k, by using the return R_{k+1} of the next time-step, i.e., t_{k+1}:

$$\mu_k = \omega_k^\top R_{k+1} - 1. \tag{11}$$

As a performance metric, we might use the achieved return to compute the *cumulative wealth* (CW), denoted by γ_k, as follows:

$$\gamma_k = \sum_{j=k-t_{tf}}^{k} \gamma_{j-1}(1 + \mu_j), \tag{12}$$

in which, for the very first iteration, $\gamma_{k-t_{tf}-1}$ is set to 1, representing a normalized initial investment of 1$, and the subscript $_{tf}$ stands for *time-frame* in the rolling horizon of the algorithm.

The above-described algorithm is summarized in Algorithm 1.

4 Extended Bandit Learning for Cardinality Constraint

We present *Extended Bandit* (ExtB) learning algorithm that is a modified and extended variant of the basic bandit with the objective of taking into account cardinality constraint, in absence of short selling possibility. The pseudo code of the ExtB is presented in Algorithm 2, which includes three parts: *basic bandit learning*, *asset limitation*, and *time restriction*. Since both the cardinality constraint and time limitation only reduce the selection and amount of assets, the other parts of the algorithm remain unaffected. Indeed, the basic bandit calculates the estimated covariance matrix, normalizes eigenvectors and eigenvalues,

Algorithm 1. Basic Bandit Learning Algorithm

1: **Inputs:** m,n, $l, \Delta t, \tau, \mathbf{R_k}$
2: **for** $k := \tau$ to m **do**
3: Calculate/estimate the returns $\mathbb{E}[\mathbf{R_k}]$ (James-Stein Shrinkage Estimator)
4: Calculate/estimate the covariance matrix $\mathbf{\Sigma_k}$
5: Perform principal component decomposition as in (3)
6: Normalize eigenvector and eigenvalues as shown in (4) and (5)
7: Compute sharpe ratio of each arm
8: Compute adjusted reward function according to (8)
9: Compute cutoff number l and select best arm from each subset
10: Compute optimal mixture weight θ_k^*
11: Compute optimal portfolio weight ω_k
12: Compute overall return and cumulative wealth γ_k
13: **end for**
14: **Outputs:** Portfolio weights and cumulative wealth for each time-step k

and computes an optimal portfolio weight vector. Then, asset limitation and time restriction parts, which are integrated into the basic bandit through the lines between 12 and 32, start handling cardinality constraint. Finally, prior to the last step, in order to evaluate the forecast quality and determine the cumulative value, we manipulate the weights to fit the cardinality constraint and replace the former weight vector.

In the following, we describe the two main complementary parts of the extended bandit learning algorithm, i.e., *asset limitation* and *time restriction*.

4.1 Asset Limitation with Kernel Search

The output of the basic bandit is a solution vector, which is computed by using orthogonal portfolios. More precisely, the solution vector, obtained through the basic bandit, shows the weight of each asset, i.e., the share that an asset receives from the total investment. This vector may contain a *large number* of assets with non-zero weights, which might be undesirable; in particular, if each asset from the former asset pool A is part of the final portfolio and every redistribution of the asset allocation requires transaction costs. Furthermore, a typical investor might be interested in having a given number C, named *cardinality parameter*, of assets in his/her portfolio just for the sake of having a better control on his/her portfolio. Therefore, a limitation in form of a *cardinality constraint* appears to be appropriate. This constraint is a mean to impose a limit on the number of assets composing the portfolio and is a well-studied topic in the financial literature [5–7,9,13,15].

Introduction of cardinality constraint adds a combinatorial aspect to the portfolio selection problem. In order to handle the situation of the new problem with the extended bandit approach, we need to make some additional assumptions. First, while the basic bandit allows short selling, assets may not be sold

Algorithm 2. Extended Bandit Learning Algorithm

Inputs: m,n, $l, \Delta t, \tau, \mathbf{R_k}, C, t_{ld}$
for $k = \tau$ to m **do**
  Calculate/estimate the returns $\mathbb{E}[\mathbf{R_k}]$ (using James-Stein Shrinkage Estimator)
  Calculate/estimate the covariance matrix $\mathbf{\Sigma_k}$(French and Fama 3-Factor Model)
  Perform principal component decomposition as in (3)
  Normalize eigenvector and eigenvalues as shown in (4) and (5)
  Compute Sharpe ratio of each arm
  Compute adjusted reward function according to (8)
  Compute cutoff number l and select best arm from each subset
  Compute optimal mixture weight θ_k^*
  Compute optimal portfolio weight ω_k
  if Cardinality Constraint selected **then**
    Determine adjusted asset pool $\hat{\mathbf{A}}$ with short positions removed
    Determine in decreasing order sorted $\bar{\mathbf{A}}$ based on their sharpe ratio $r_{k,i}$
    Set size of kernel K as C and fill with the $\lfloor C/2 \rfloor$ best assets from $\bar{\mathbf{A}}$
    Remove assets in K from $\bar{\mathbf{A}}$ and set basket size
    while size$(K) < C$ **do**
      Divide $\bar{\mathbf{A}}$ into baskets with the size from step 16
      for each basket b_i **do**
        Add temporarily b_i to K and calculate the expected return
        Calculate needed time $\Delta\hat{t}$
        if $t_{ld} < \Delta\hat{t}$ & timeRestriction.selected **then**
          break;
        end if
      end for
      Select the best basket b_i^* and add its assets to the kernel K
      Remove the assets from the best basket b_i^* from $\bar{\mathbf{A}}$
    end while
    For each asset that is not part of the selection: Set its weight to 0
    Adjust weights such that $\sum \omega_{k,i}^* = 1$
    $\omega_k = \omega_k^*$
  end if
  Compute the overall return and the cumulative wealth γ_k
end for
Outputs: Portfolio weights and cumulative wealth for each time-step k

short within the extended bandit. Furthermore, in order to have the most promising assets in the portfolio, we sort them through the combinatorial process of the extended bandit learning. For this purpose, we use the Sharpe ratio, which uses the simple mean of the i-th asset at time-step k:

$$\mu_{k,i} = \frac{1}{t_{tf}} \sum_{j=k-t_{tf}}^{k} R_{i,j},$$

and takes the last t_{tf} return data depending on the size of the preceding horizon.

We summarize the asset limitation process, which uses *kernel search heuristic* [6], in Algorithm 3. In the following, we describe different steps of this algorithm.

Algorithm 3. Asset Limitation Procedure for Handling Cardinality Constraint

Inputs: w_k, C
Remove the short soled assets to determine adjusted asset pool $\hat{\mathbf{A}}$
Use the Sharpe ratio to sort asset pool $\bar{\mathbf{A}}$ in decreasing order of ratios
Set size of kernel K as C and fill with the $\lfloor C/2 \rfloor$ best assets from $\bar{\mathbf{A}}$
Remove assets in K from $\bar{\mathbf{A}}$
Set basket size
while size(K) $<$ C **do**
    Divide $\bar{\mathbf{A}}$ into baskets with the size from step 6
    for each basket b_i **do**
        Unit K with b_i temporary and calculate the expected return
    end for
    Select the best basket b_i^* and add its assets to the kernel K
    Remove the assets from the best basket b_i^* from $\bar{\mathbf{A}}$
end while
Set each asset's weight to 0, that is not part of the selection
Adjust weights such that $\sum \omega_{k,i}^* = 1$
$\omega_k = \omega_k^*$
Output: Adjusted weights vector ω_k

The process starts with the adjustment of the asset pool A by removing all short positions. Then, we compute the Sharpe ratio for the remaining assets and sort them in decreasing order to obtain the sorted asset pool $\bar{\mathbf{A}}$.

The investor may set the value of the cardinality parameter C. Then, we define a set so-called *kernel* K that has the same size as C. At the beginning of the combinatorial process, the kernel K is filled to the half with the best assets from $\bar{\mathbf{A}}$. Then, these specific assets are removed from $\bar{\mathbf{A}}$.

Another concept in the kernel search heuristic is called *basket*, which is a set used to fill out the remaining parts of the kernel. The initial basket size is defined as $s_b = \lfloor C/4 \rfloor$. The basic idea of the kernel search is as follows: The available assets are divided into baskets of the same size. At each iteration of the heuristic, the value of the objective function is calculated, which is (in our case) the expected return, for the combination of the kernel and the respective basket. After evaluating each combination, we select the basket that leads to the best result and insert its assets into K. At the same time, we update the current asset pool by removing these assets from $\bar{\mathbf{A}}$. We repeat these steps until the kernel is filled with the same amount of assets as the cardinality constraint. It might happen that for the last iteration the basket size is too big. Hence, the new basket size $\hat{s}_b$ equals the remaining free spots inside the kernel K (for more details on the kernel search heuristic, see [6]).

As soon as the best assets have been found, the weights of the other positions are set to zero. The sum of all selected weights should be equal to 1; otherwise, parts of the budget remain unused. This adjustment is achieved by normalizing each weight by the sum of the weights of all remaining assets.

At a time step t_k, after finishing the process, the outcome is an weight-adjusted vector ω_k containing exactly C assets with $\omega_{k,i} > 0$ for $i \in \{1, \ldots, n\}$.

4.2 Optional Time Restriction

The performance of the extended bandit primarily depends on the size of the problem. While the calculation process takes about 3 s for 48 assets with historical data from the last 120 months, it might take about 15 min for almost 500 assets with equal historical information. Consequently, larger data sets result in higher computation times. Hence, in order to have a time management in the combinatorial process, we restrict it by means of a *time restriction* procedure (See lines 21–24 of Algorithm 2).

Let a maximum duration t_{max} be given, e.g., 15 min, which is 900 s. If we assume that the data set contains information for t_h time-steps, the overall recurrence number of the calculation process is given by the difference $t_h - t_{tf}$. For example, let us assume that the data set consists of 48 assets and $t_h = 500$, the cardinality parameter is equal to 10, and the selected time-frame (sliding window horizon) t_{tf} is 120. Then, the optimization process will be repeated 380 times.

Usually, the algorithm would just stop after a certain duration (set by the user), taking the solution, which has been found at that point in time. However, this approach might lead to a problem as the process might have to be aborted in the middle of the computation. This situation might be an issue; in particular, in case of the extended bandit as, due to the combinatorial process, the solution procedure might require long processing time. In order to handle this situation, we introduce the following mechanism. The maximum duration t_{max} is divided into equal parts for each calculation point, which is called *cycle duration* t_{cd}, incorporating the basic bandit part as well as the kernel search part of the extended bandit. The cycle duration t_{cd} is computed as follows:

$$t_{cd} := \frac{t_{max}}{t_h - t_{tf}}. \tag{13}$$

Additionally, the kernel search itself iterates through every possible basket to find the best combination. Because the number of iterations is known (which depends on the value of C and the basket size), the cycle duration can be divided by the number of iterations to obtain the *loop duration* t_{ld} for the kernel search:

$$t_{ld} = \frac{t_{cd}}{\left\lceil \frac{C/2}{s_b} \right\rceil}. \tag{14}$$

To continue the example from above, the respective cycle and loop duration are:

$$t_{cd} = \frac{t_{max}}{t_h - t_{tf}} = \frac{900\,\text{s}}{500 - 120} = 2.368\,\text{s}, \tag{15}$$

$$t_{ld} = \frac{t_{cd}}{\left\lceil \frac{C/2}{s_b} \right\rceil} = \frac{2.368\,\text{s}}{\left\lceil \frac{5}{2} \right\rceil} = 0.789\,\text{s}. \tag{16}$$

Using these values, we can have a better time management throughout the extended bandit learning algorithm. In fact, after each iteration, the algorithm

checks whether the allowed duration is exceeded (see lines 21–24 of Algorithm 2). The query appears after the first iteration. This guarantees that the combinatorial process takes place at least once, which means that the best basket taken is naturally the first one. Basically, the final kernel K would consist of the best C assets from $\bar{\mathbf{A}}$. In case of a big data set or a very small time limit, t_{cd} and t_{ld} might be too small even for a single iteration and, consequently, the time limit will not be sufficient to find a solution.

5 Computational Experiments and Numerical Results

In order to investigate the performance of the algorithms in solving the portfolio selection problem (under cardinality constraint), we carried out a set of computational experiments and report the results in this section.

5.1 Data and Test Settings

For the computational experiments, we used six real-world data sets, selected from the finance page of *Yahoo!* as well as the financial literature [6,7,21]. Table 1 sums up the main information regarding these data sets. In particular, test instances FF48 and S&P500, which are daily information of assets, have already been used in the literature, but the other ones have been collected from monthly prices on the finance page of *Yahoo!* In Table 1, the column of time-steps gives the length of the total horizon covered by the corresponding data set.

Table 1. Data sets.

Data set	Period	Time-steps
FF48	Daily	498
100 Assets	Monthly	119
200 Assets	Monthly	119
300 Assets	Monthly	119
400 Assets	Monthly	119
S&P500	Daily	1258

The basic bandit as well as the extended bandit learning algorithms require setting values to some parameters, which are given in Table 2. More precisely, in this table, we provide the different combinations for the parameters of the tested algorithms. In particular, at each experiment using the extended bandit, the value of the cardinality parameter C is selected from $\{5, 10, 15, 20\}$. These are selected values for which the portfolio optimization problem under cardinality constraint is known to be challenging and attract particular attention in the financial literature [5–7,9,13,15].

Table 2. Parameter settings for the experiments.

Parameter	Tested values
Sliding window time-frame	50% & 25% of respective time-steps
Asset limit (cardinality parameter C)	5, 10, 15, 20
Time limit	20 min

We implemented the algorithms in Java and ran the experiments on a computer with the following specifics: Windows 10 Professional, Intel i5-6600 CPU 3.30 GHz, 16 GB RAM.

5.2 Numerical Results

Shen et al. suggest in their paper to compute the cumulative wealth based on the returns of the last regarded time-step [21]. Basically, they take the return data for each asset from time-step $(k - t_{tf})$ until k, and calculate the realized return based on the same last returns at time-step k, which have already been taken into account to compute the asset allocation. Consequently, the *forecast* quality and potential of the algorithm is not considered. Therefore, in order to investigate whether the allocation leads to a yield, we suggest to compute the realized return or rather the cumulative wealth based on the return data at time-step $(k + 1)$. In order to verify the differences between these two separate cases, we tested the basic bandit and computed the cumulative wealth (CW) of the resulting portfolios at time-steps k and $k + 1$, i.e., for return vectors R_k and R_{k+1}, and report them in Table 3. This table contains also the computation time (CPU Time in seconds) of each experiment.

It can be seen that the settings with a smaller time-frame perform better, in terms of higher CW, except for the cases of R_{k+1} of 200 Assets, R_k of 300 Assets, and R_k of 400 Assets. This might be explained by the fact that, in every time-step, a new asset allocation is computed and the portfolio converted appropriately. Therefore, the information and also the potential in the long-term development of the allocation will not be used, since the investor only holds the portfolio for one period. A smaller time-frame, on the other hand, makes a better use of the short-term development of the assets and the algorithm applies the market fluctuations into the asset allocation.

Further, the increase of computation time depends on the number of assets (in the instances) and their available time horizon. For example, while the algorithm takes only 0.88 s to run through a small data set consisting of 48 assets with available data for 498 time-steps and a time-frame of 125, it nearly takes 16 min to compute the overall cumulative wealth for the S&P500 data set, consisting of 500 assets, 1258 time-steps and a time-frame of 315. Finally, the differences between CW values for R_k versus R_{k+1} reveal the forecast errors that we might run in an investment.

Table 3. Results for basic bandit

Data set	Time-frame	CPU time (s.)	CW R_k	CW R_{k+1}
FF48	125	0.90	65.03	67.18
	249	1.08	7.44	30.73
100 Assets	30	1.02	5.67	10.99
	60	1.00	4.24	3.17
200 Assets	30	6.70	3.67	1.86
	60	5.14	3.11	7.10
300 Assets	30	21.59	0.17	6.43
	60	15.73	4.17	2.73
400 Assets	30	51.81	0.50	2.39
	60	34.92	2.48	1.68
S&P500	315	965.29	1.89	1.55
	629	695.66	1.29	1.11

In addition, we tested the extended bandit learning algorithm with 50% as well as 25% time-frames, and for different values of the cardinailty parameter C. The results are reported in Tables 4 and 5. In these tables, we present the cumulative wealth (CW) of the solutions provided by the basic bandit (BB) and extended bandit (ExtB) for vectors R_{k+1}. It is important to note that the cardinality constraint is taken into account only by the ExtB and the results of the BB ignores the presence of this constraint (hence, the results of the BB are repeated in all sections of Tables 4 and 5). However, the BB permits short selling, whereas this is not the case of the ExtB. Hence, the ExtB solves a more restricted version of the portfolio selection problem. Nevertheless, according to the results shown in Tables 4 and 5, we observe that the quality of CW for the solutions provided by the ExtB are quite competitive to those obtained through the BB. In many cases, the ExtB provides better performance, in terms of CW, than the basic bandit.

Regarding the influence of the cardinality constraint, we observe, from Tables 4 and 5, that the presence of this constraint does not have big impact either on the quality of the solutions or on the computation time of the extended bandit learning algorithm. This might mean that our ExtB learning algorithm has a good performance in handling the cardinality constraint. In addition, we see that a higher value of C comes along with a reduction in the CW, which might be explained as follows: a higher value of C permits presence of a larger number of assets in the portfolio. Since additional assets have lower quality than the firstly added ones (e.g., in the case of $C = 5$); hence, their imposed presence will have a negative impact on the whole portfolio. Consequently, a wise selection of a suitable value for the cardinality parameter C is a matter of importance.

Comparing the results shown in Tables 4 versus 5, we observe that both algorithms, overall, provide solutions with higher CW-values when a smaller sliding horizon is used. However, using a smaller sliding horizon comes along with a higher computation effort. In fact, although the used data for each time-step becomes smaller, there are more time-steps for which an allocation has to be computed. Consequently, the whole computation process takes a bit longer.

Table 4. Results for extended bandit and basic bandit learning algorithms with R_{k+1} and 50% as time-frame.

Data set	Time-frame	Cardinality parameter C	Computation time in s	CW ExtB R_{k+1}	CW BB R_{k+1}
FF48	249	5	1.16	22.90	**30.73**
100 Assets	60	5	1.04	**8.01**	3.17
200 Assets	60	5	5.05	2.65	**7.10**
300 Assets	60	5	16.01	1.30	**2.73**
400 Assets	60	5	37.48	0.82	**1.68**
S&P500	629	5	705.89	**1.69**	1.11
FF48	249	10	0.73	16.41	**30.73**
100 Assets	60	10	4.31	**4.31**	3.17
200 Assets	60	10	4.46	5.31	**7.10**
300 Assets	60	10	14.63	**3.39**	2.73
400 Assets	60	10	35.35	1.50	**1.68**
S&P500	629	10	701.82	**1.56**	1.11
FF48	249	15	0.71	13.70	**30.73**
100 Assets	60	15	0.71	2.77	**3.17**
200 Assets	60	15	4.31	5.68	**7.10**
300 Assets	60	15	14.45	2.58	**2.73**
400 Assets	60	15	35.00	**3.25**	1.68
S&P500	629	15	691.10	**1.22**	1.11
FF48	249	20	0.67	17.18	**30.73**
100 Assets	60	20	0.67	1.96	**3.17**
200 Assets	60	20	4.35	5.70	**7.10**
300 Assets	60	20	14.36	**5.02**	2.73
400 Assets	60	20	35.29	**3.87**	1.68
S&P500	629	20	698.24	**1.14**	1.11

Table 5. Results for extended bandit and basic bandit learning algorithms with R_{k+1} and 25% as time-frame

Data set	Time-frame	Cardinality parameter C	Computation time in s	CW ExtB R_{k+1}	CW BB R_{k+1}
FF48	125	5	1.36	**88.06**	67.18
100 Assets	30	5	1.50	4.38	**10.99**
200 Assets	30	5	7.18	**2.38**	1.86
300 Assets	30	5	23.31	0.73	**6.43**
400 Assets	30	5	53.44	0.14	**2.39**
S&P500	315	5	984.32	**2.4**	1.55
FF48	125	10	0.95	57.95	**67.18**
100 Assets	30	10	1.15	9.05	**10.99**
200 Assets	30	10	6.47	**6.66**	1.86
300 Assets	30	10	22.24	1.82	**6.43**
400 Assets	30	10	52.02	1.05	**2.39**
S&P500	315	10	965.05	**2.09**	1.55
FF48	125	15	0.86	57.94	**67.18**
100 Assets	30	15	1.06	6.78	**10.99**
200 Assets	30	15	6.31	**9.22**	1.86
300 Assets	30	15	21.96	**9.90**	6.43
400 Assets	30	15	52.14	1.53	**2.39**
S&P500	315	15	953.74	**1.89**	1.55
FF48	125	20	0.85	54.265	**67.18**
100 Assets	30	20	1.03	8.72	**10.99**
200 Assets	30	20	6.29	**7.92**	1.86
300 Assets	30	20	22.03	5.14	**6.43**
400 Assets	30	20	52.73	2.17	**2.39**
S&P500	315	20	953.83	1.55	1.55

6 Conclusion

In this paper, we investigated a bandit learning algorithm introduced by Shen et al. [21] for solving basic portfolio selection problems. Further, we modified and extended this algorithm and combined it with the kernel search heuristic in order to handle cardinality constraint in a portfolio selection problem. To evaluate the algorithm, we conducted computational experiments on real-world instances and reported the results. According to the numerical results, our extended bandit learning algorithm can perform well in handling combinatorial aspect of the portfolio optimization under cardinality constraint and provides portfolios with a competitive quality to those obtained by the basic bandit learning algorithm.

An interesting future research direction might consist in enhancing our algorithm to take into account different kinds of transaction cost functions [10,14].

Acknowledgments. The author wishes to thank the Chair of Business Information Systems and Operations Research (BISOR) at the Technische Universität Kaiserslautern (Germany), where the paper was written, for the financial support.

References

1. Bartholomew-Biggs, M.C.: Nonlinear Optimization with Financial Applications, 1st edn. Kluwer Academic Publishers, Dordrecht (2005)
2. Berry, D.A., Fristedt, B.: Bandit Problems: Sequential Allocaton of Experiments. University of Minnesota: Chapman and Hall Ltd., London (1985)
3. Chen, W., Wang, Y., Yuan, Y.: Combinatorial multi-armed bandit: general framework and applications. In: Proceedings of the 30th International Conference on Machine Learning, pp. 151–159 (2013)
4. Fan, J., Fan, Y., Lv, J.: High dimensional covariance matrix estimation using a factor model (2007). http://arxiv.org/pdf/math/0701124v1
5. Fernández, A., Gómez, S.: Portfolio selection using neural networks. Comput. Oper. Res. **34**, 1177–1191 (2007)
6. Guastaroba, G., Speranza, M.G.: Kernel search: an application to the index tracking problem. Eur. J. Oper. Res. **217**(1), 54–68 (2012)
7. Gulpinar, N., Le Thi, H.A., Moeini, M.: Robust investment strategies with discrete asset choice constraints using DC programming. Optimization **59**(1), 45–62 (2010)
8. Hoffman, M.D., Brochu, E., de Freitas, N.: Portfolio allocation for bayesian optimization. In: The Conference on Uncertainty in Artificial Intelligence, pp. 327–336 (2011)
9. Jobst, N., Horniman, M., Lucas, C., Mitra, G.: Computational aspects of alternative portfolio selection models in the presence of discrete asset choice constraints. Quant. Finance **1**, 1–13 (2001)
10. Konno, H., Yamamoto, R.: Global optimization versus integer programming in portfolio optimization under nonconvex transaction costs. J. Glob. Optim. **32**, 207–219 (2005)
11. Lai, T.L., Robbins, H.: Asymptotically efficient adaptive allocation rules. Adv. Appl. Math. **6**(1), 4–22 (1985)
12. Le Thi, H.A., Moeini, M.: Portfolio selection under buy-in threshold constraints using DC programming and DCA. In: International Conference on Service Systems and Service Management (IEEE/SSSM 2006), pp. 296–300 (2006)
13. Le Thi, H.A., Moeini, M., Pham Dinh, T.: Portfolio selection under downside risk measures and cardinality constraints based on DC programming and DCA. Comput. Manag. Sci. **6**(4), 477–501 (2009)
14. Le Thi, H.A., Moeini, M., Pham Dinh, T.: DC programming approach for portfolio optimization under step increasing transaction costs. Optimization **58**(3), 267–289 (2009)
15. Le Thi, H.A., Moeini, M.: Long-short portfolio optimization under cardinality constraints by difference of convex functions algorithm. J. Optim. Theory Appl. **161**(1), 199–224 (2014)
16. Markowitz, H.M.: Portfolio selection. J. Finance **7**(1), 77–91 (1952)
17. Markowitz, H.M.: Portfolio Selection. Wiley, New York (1959)

18. Meucci, A.: Risk and Asset Allocation. Springer, Dordrecht (2009)
19. Moeini, M., Wendt, O., Krumrey, L.: Portfolio optimization by means of a χ-armed bandit algorithm. In: Nguyen, N.T., Trawiński, B., Fujita, H., Hong, T.-P. (eds.) ACIIDS 2016. LNCS (LNAI), vol. 9622, pp. 620–629. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49390-8_60
20. Robbins, H.: Some aspects of the sequential design of experiments. Bull. Am. Math. Soc. **58**, 527–535 (1952)
21. Shen, W., Wang, J., Jiang, Y.-G., Zha, H.: Portfolio choices with orthogonal bandit learning. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015), pp. 974–980 (2015)
22. Thompson, W.R.: On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika **25**, 275–294 (1933)