AMO - Advanced Modeling and Optimization, Volume 16, Number 3, 2014

PSO and Harmony Search Algorithms for Cardinality Constrained Portfolio Optimization Problem

M. Salahi, M. Daemi, S. Lotfi, A. Jamalian

Department of Applied Mathematics, Faculty of Mathematical Sciences

University of Guilan, Rasht, Iran

salahim@guilan.ac.ir

Abstract: Markowitz cardinality constraint mean-variance (MCCMV) model is a well studied and important one in the portfolio optimization literature. It is formulated as mixed integer quadratic programming problem (MIQP) which belongs to class of NP-hard problems, thus various heuristic and meta-heuristic algorithms are applied to solve it. In this paper, two modified versions of particle swarm optimization (PSO) and harmony search (HS) algorithms are applied to solve the underlying problem. In the proposed PSO algorithm, modifications in inertia weight and learning coefficients are done and in the modified HS algorithm, modifications in harmony memory consideration rate, pitch adjustment rate, and bandwidth are done. Experimental results on five data sets that includes 31 assets up to 225 assets show that the modified HS algorithm is much faster than the modified PSO, specially on large data sets.

***Key words*:** Markowitz mean-variance model, cardinality constrained, portfolio optimization, efficient frontier, particle swarm optimization, harmony search.

JEL Classification:G11, C60.

1. Introduction

One of the most important problems in finance is selecting portfolio such that improve tradeoff between risk and return. For this purpose, Markowitz in 1952 introduced mean-variance model which is one of the well-known models for solving the portfolio selection problem [1-2]. This model can be described as a two-objective nonlinear programming problem such that the goal is to minimize risk and maximize return. This model can be equivalently reformulated as a quadratic programming (QP) problem [3]. The Efficient frontier for the standard mean-variance model as defined in [2] can be easily obtained by solving the corresponding QP for all tradeoffs between risk and return.

Many variants have been proposed to make the Markowitz model more realistic. These refinements come from real-world applications, in which constructing a portfolio made up of a large number of assets possibly with very small holdings for some of them, is clearly not desirable. This is because of transactions costs, minimum lot sizes, complexity of management and policy of the asset management companies [1-4]. In fact, limiting the number of different assets included in the portfolio decreases transaction cost. Also the monitoring of news and firm's results is easier and so less costly with few assets in the portfolio [5]. One of the most important variants of mean-variance model is "cardinality constraint" which includes limitation on the number of assets to be held in an efficient

portfolio, and prescribes lower and upper bounds on this limitations. "Quantity constraint" or "bounding constraint" is another limitation that limits the amount of an asset in portfolio. This makes sense in real life, because portfolio managers usually put a limit on amount of investment in a single security in their portfolio. Moreover, this constraint prevents having portfolios with small or large shares. However imposing such constraint in portfolio optimization makes it complex, and consequently finding efficient frontier becomes difficult. By adding cardinality and quantity constraints in mean-variance model, the QP formulation of the problem converts to mixed-integer QP (MIQP) model and QP solvers or traditional mathematical method cannot deal with it efficiently and might lead to local solutions [1-4]. Therefor, solving this problem with heuristic or meta-heuristic algorithms is of interest.
Many heuristics and meta-heuristic algorithms have been applied to obtain the efficient frontier of the corresponding MIQP problem, for example, Chang et al. (2001) used Genetic algorithm (GA), simulated annealing (SA) and Tabu Search (TS) to draw efficient frontier [4], Fernandez and Gomez (2007) applied neural network (NN) [2] and Cura (2009) proposed basic particle swarm optimization (PSO) heuristic [6] to tackle the problem. In another study, Anagnostopoulos et al. (2011) applied five multi-objective evolutionary algorithms (MOEAs), compared them and traced efficient frontier related to each algorithm [3]. Mozafari et al. (2011) proposed a hybrid algorithm of SA procedure and improved PSO for the problem [7]. Moreover, Deng at al. (2012) compared PSO with different modifications and suggested a mutation in PSO heuristic for maximizing diversification [8].

In this paper, we use a modified version of PSO by Eberhart and Shi [9] to solve the Markowitz MIQP model. This modification uses a simplified method of constriction factor and dynamic inertia weight. This version of PSO is referred to here as ICPSO. Also, we use a version of harmony search algorithm proposed by Mahdavi et al [10] with some modifications on parameters. This modification is called improved harmony search algorithm (IHS). The two algorithms are compared on five sets of test problem. Our comparison shows that IHS algorithm is much faster than ICPSO, especially on large data sets.

The rest of the paper is organized as follows. In Section 2 we introduce cardinality constrained portfolio optimization for the standard mean-variance model. To solve this problem, first we review PSO and HS algorithms in Section 3 and introduce some modifications on the algorithms parameters to have better performance. In Section 4 we present computational results and finally Section 5 gives the conclusions.

2. Cardinality constrained portfolio optimization

Having cardinality and quantity constraint in the standard mean-variance model leads us to the following model:

$$\begin{aligned} min \quad & x^t Q\, x && (1)\\ s.t \quad & \mu^t x = R && (2)\\ & 1^t x = 1 \qquad p(1) && (3)\\ & 1^t \delta = k && (4)\\ & l_i\delta_i \le x_i \le u_i\delta_i && (5)\\ & \delta_i \in \{0,1\} && (6) \end{aligned}$$

where Q is the variance-covariance matrix, μ is the vector of expected returns, R is the desired expected return, k is the number of desired assets in basket (portfolio) and l_i and u_i are lower and upper bound for asset i, respectively. Because variance is a criteria for measuring risk, we minimize variance of a portfolio in the objective. Constraint (2) ensures that we have return R. Constraint (3) makes proportion of a portfolio unique. Constraint (4) determines that we must have k assets in the portfolio and constraint (5) give limitation on proportion of a portfolio between lower and upper bound. δ_i's are binary variables, which if there is an asset in a portfolio then it is equal to 1, otherwise it is 0.

One usual way to solve this problem is to use trade-off parameter between risk and return of portfolio. Thus the cardinality constrained mean-variance model (CCMV) can be formulated as follows [2-4].

$$min \quad \lambda x^t Q\, x - (1-\lambda)\mu^t x \tag{7}$$
$$s.t \quad 1^t x = 1 \tag{8}$$
$$1^t \delta = k \qquad p(2) \tag{9}$$
$$l_i \delta_i \le x_i \le u_i \delta_i \tag{10}$$
$$\delta_i \in \{0,1\}. \tag{11}$$

If $\lambda = 0$, then we can reach to maximum return and if $\lambda = 1$ we can reach to minimum risk. By moving from $\lambda = 0$ to $\lambda = 1$, we can move from portfolios with maximum return toward portfolios with minimum risk. By solving the aforementioned problem for λ between 0 and 1 and obtaining respective optimal portfolios, we can compute coordinate of the corresponding mean and variance of returns. Linking these coordinates, result in a curve that is called efficient frontier. From this curve, we can see that how portfolios risk increase as desired return increases [1]. This frontier can be different from the one obtained by standard Markowitz model [4].

3. PSO and HS Algorithms

Finding algorithms with reasonable run time and good quality of solution for problems are two main goals in computer science. For this purposes, heuristic and meta-heuristic algorithms might be very useful, specially for problems belong the class of NP-hard problems [11-12]. In this section we present two meta-heuristic algorithms, namely PSO and HS algorithms to solve the MCCMV.

3.1. PSO meta-heuristic

PSO is a population based algorithm that fountain from social acting of a swarm of birds, bees or fishes which introduced by Eberhart and Kennedy in 1995. Each swarm has number of particles that their positions depend on their velocities and both are vectors in multi dimensional search spaces. Current particle's velocity also depends on its previous velocity, best position founded by it and best positions founded by entire swarm.

To picture algorithm better, imagine a group of birds that randomly searching for a food and one of them is near the food and there is just one food. Let's call this bird the leader. Other birds by following the leader, try to improve their position and try to be nearer to the food. In other words, by imitation from the leader and its knowledge, one of them can reach the food [13].

Using PSO method for selecting a basket, vector of particle's position is the basket of assets and vector of best particle's position in entire swarm is the efficient basket of assets.

3.1.1. ICPSO Method

For finding best particle in whole swarm we need information about particles distance from each other, from the best one and also we need information about their velocities. Let vector *x* denotes particle's location. Let index *b* determine best particle in the swarm and index g determine global best particle in entire swarm. So velocity and position of particle *i* at iteration t+1 can be updated by the following equations, respectively:

$$v_i(t+1) = w \times v_i(t) + c_1 \times rand \times (x_b - x_i) + c_2 \times rand \times (x_g - x_i), \tag{12}$$
$$x_i(t+1) = x_i(t) + v_i(t+1). \tag{13}$$

In equations (12), w is the inertia weight, c_1 and c_2 are personal and global learning coefficients, respectively and $rand$ is a random number between 0 and 1. Adjusting these coefficients make convergence of the algorithm better and faster [6-9].

In this paper, for tracing efficient frontier by PSO algorithm, we use the approach suggested in [6] that considers every particle in 2×N dimensional search space. Suppose that x_i include the share i on a portfolio and δ_i be binary

variable that shows share i is in our portfolio or not. In dimension x and dimension δ velocity vector will be updated as follows, respectively.

$$v_i(t+1) = k \times \left(w \times v_i(t) + c_1 \times rand \times (x_b - x_i) + c_2 \times rand \times (x_g - x_i)\right), \quad (14)$$

$$v\delta_i(t+1) = k \times \left(w \times v\delta_i(t) + c_1 \times rand \times (\delta_b - \delta_i) + c_2 \times rand \times (\delta_g - \delta_i)\right), \quad (15)$$

$$\mathrm{k} = 2/|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}| \quad , \quad \varphi = \mathrm{c}_1 + \mathrm{c}_2 \ , \quad \varphi > 4, \quad (16)$$

$$w_{new} = w_{old} \times 0.9, \quad (17)$$

where k is constriction factor and w is inertia weight. These formulas were derived from Clerc's work in 1999 where he noticed that manipulation of constriction factor could be essential for particles convergence [9].

The PSO with modification on inertia weight and constriction factor is referred to here as ICPSO. Moreover, for constraints satisfaction, we used arrangement algorithm in [6] that makes every solution (particle) feasible.

3.2. HS meta-heuristic

HS algorithm is a meta-heuristic method that introduced by Geem et al. (2001) and uses musician creation of music for producing harmonies. It has three rules that link creation of music to optimization method. These three rules are presented in Table 1 [14].

Table 1: analogues improvisation with optimization

Musician creation of music rules	Related rules in HS method	Called
1. Playing every sound from listener memory	Selecting every value from harmony search memory	memory consideration
2. Playing familiar sound to listener memory	Selecting value near to harmony search memory	pitch adjustment
3. Playing random sound from possible sound range.	Selecting random value from possible value range.	Randomization

Using HS method for selecting a portfolio, every harmony vector is related with a portfolio, each player is related with a decision variable, listener satisfaction is related with objective function, player's creation of music is related with local and global search procedure and range of music sounds is like interval of variables. Analogous to the other meta-heuristic methods, this method has some parameters that adjusting them might improve its performance. These parameters include HMS, HMCR, PAR, BW and NI that are harmony memory, harmony memory consideration rate, pitch adjusting rate, bandwidth and number of improvisation (creation of music), respectively.

3.2.1. IHS algorithm

Mahdavi et al [10], proposed dynamic parameters for HS algorithm and called the new version of HS algorithm improved HS (IHS). In this paper, we utilize this version of HS with modifications on parameters. We describe IHS procedure as follows:

- Initialization of the problem and algorithm parameters

Let $x_i = (x_i^1, \dots, x_i^m)\ \forall i = 1, \dots, N$ denote harmony vectors and N denotes number of harmony vectors. Also let $l_i \le x_i \le u_i\ \forall i = 1, \dots, N$ where l_i and u_i are given lower and upper bounds on the harmony vector i, respectively.

In general, optimization problem can be described as the following:

$$\min_x f(x) \quad st\ \ x_i \in [\, l_i, u_i] \ \forall i = 1, \dots, N$$

Three main parameters that have most effect on performance of IHS are HMCR, PAR and BW [9-11]. In this paper, we set theses parameters as following:

$HMCR_{max} = 0.99, \quad HMCR_{min} = 0.8,$

$PAR_{max} = 0.1, \quad PAR_{min} = 10^{-6},$

$BW_{max} = 0.1, \quad BW_{min} = 10^{-6}.$

and

$$HMCR = HMCR_{min} + \frac{HMCR_{max} - HMCR_{min}}{iter_{max}} \times iter, \tag{18}$$

$$PAR = PAR_{min} + \frac{PAR_{max} - PAR_{min}}{iter_{max}} \times iter, \tag{19}$$

$$BW = BW_{max} - \frac{BW_{max} - BW_{min}}{iter_{max}} \times iter, \tag{20}$$

where *iter* is the current iteration and $iter_{max}$ is the maximum number of iterations.

- Initialization of the harmony memory

Let HM denotes harmony memory and HMS denotes harmony memory size. HM is a matrix that filled up with harmony vectors in size of HMS as follows:

$$\mathrm{HM} = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_{N-1}^1 & x_N^1 \\ x_1^2 & x_2^2 & \dots & x_{N-1}^2 & x_N^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^{HMS-1} & x_2^{HMS-1} & \dots & x_{N-1}^{HMS-1} & x_N^{HMS-1} \\ x_1^{HMS} & x_2^{HMS} & \dots & x_{N-1}^{HMS} & x_N^{HMS} \end{bmatrix}. \tag{21}$$

- Creation of new harmony

Let NHM, NHMS denote the new harmony memory and new harmony memory size, respectively. Also NHM is a matrix of size NHMS. NHM includes $x_i^{new} = \left(x_i^{j\,new}, \dots, x_i^{j\,new}\right)$ for $j = 1, \dots, NHMS$ that are produced randomly. To improve these harmonies, we must use three rules as were mentioned in Table 1.

1. Memory consideration:

$$x_i^{j\,new} \leftarrow \begin{cases} x_i^{j\,new} \in \{x_i^1, x_i^2, \dots, x_i^{HMS}\} & with\ prob\ (HMCR) \\ x_i^{j\,new} \in [l_i, u_i] & with\ prob\ (1 - HMCR) \end{cases} \tag{22}$$

Here $x_i^{j\,new}$ is improved harmony and selected from HM with probability HMCR or otherwise from possible value range with probability 1-HMCR.

2. Pitch adjustment: for new produced harmony we must check that whether it needs to be adjusted or not.

$$pitch\ adjustment\ decision\ for\ x_i^{j\ new} \leftarrow \begin{cases} yes & with\ prob\ (PAR) \\ no & with\ prob\ (1-PAR) \end{cases} \quad (23)$$

3. Randomization: if pitch adjustment decision is yes then use Eq (24) as follow.

$$x_i^{j\ new} \leftarrow x_i^{j\ new} \pm rand \times Bw, \quad (24)$$

where *rand* is a random number and *Bw* is Bandwidth. If this is no then there is no need to adjust $x_i^{j\ new}$ [10, 14-15].

- Update harmony memory

To compare harmonies (candidate solutions), we must a have a metric. One obvious and simple criteria is to use objective function to evaluate the harmonies and find best harmony. In other words, we can use

$$f_H(x) = \lambda[x^T Q x] - (1-\lambda)[\mu^T x], \quad (25)$$

where $f_H(x)$ is value of fitness function for harmony *x*, λ is trade-off parameter, Q is variance-covariance matrix and μ is the vector of expected returns [6].

Now we can summarize IHS algorithm as it follows.

1. Initialize the problem and algorithm parameters: HMS, HMCR, PAR, BW and NHMS (NI).

2. Initialize harmony memory (HM): a matrix in HMS size.

3. a. Produce new harmony randomly in size of NHMS.

b. Check equation (22) and (23).

4. Find best new harmony and delete worse one based on f_H value.

5. Check stopping criteria. If the algorithm reaches maximum number of iterations, then it will be terminated. Otherwise steps 3 and 4 will be repeated [10-11, 14-15].

Table 2: Pseudo Code of IHS Meta-Heuristic

```
For i=1:maxiter
Compute HMCR,PAR,BW with related equations.
   For i=1:NI
      NHM(i).position=unifrnd(0.01,1, [1 number_of_variable]);
    For j=1:number_of_variables
      if rand<=HMCR
        k=randi([1 HMS]);
        NHM(i).position(j)=HM(k).position(j);
      else
        NHM(i).position(j)=rand;
      end
      if rand<=PAR
        NHM(i).position(j)=NHM(i).position(j)+ BW*unifrnd(0.01,1);
      end
   end
  Make NHM(i).position feasible;
```

Compute f_H ;
end
find best harmony and delete worse one;
end

In this algorithm, like ICPSO, we used arrangement algorithm in [6] to make every solution feasible.

4. Computational Results

To test the performance of ICPSO and IHS algorithm, we use data sets include the weekly prices from March 1992 to September 1997, for indices Hang Seng in Hong Kong, DAX 100 in Germany, FTSE 100 in UK, S&P 100 in USA and Nikkei in Japan. Numbers of different assets for these indices are 31, 85, 89, 98 and 225, respectively. The set of mean returns for assets, covariance between these assets and 2000 points from efficient frontier are available in http://people.brunel.ac.uk/%7Emastjjb /jeb/orlib/portinfo.html.

Let (v_i^s, r_i^s)s for i=1 to 2000 be the set of mean and variance of portfolio return points on the standard efficient frontier and (v_j^h, r_j^h) for $j = 1, \dots, \xi$ (that in here ξ is 51) be the set of mean and variance of portfolio return points on heuristic efficient frontier. Let (v_e^s, r_e^s) be the point on standard frontier that has minimum Euclidean distance from the heuristic point (v_j^h, r_j^h) where e the corresponding index can be found as follows:

$$e = arg \ \ min_{i=1,\dots,2000} \left(\sqrt{\left(v_i^s - v_j^h\right)^2 + \left(r_i^s - r_j^h\right)^2} \right) \qquad j = 1, \dots, 51. \tag{26}$$

So mean Euclidean distance, variance of return error and mean return error can be defined as follow:

$$\text{mean Euclidean distance} = \sum_{j=1}^{\xi} \sqrt{\left(v_e^s - v_j^h\right)^2 + \left(r_e^s - r_j^h\right)^2} \times \frac{1}{51}, \tag{27}$$

$$\text{variance of return error} = \left(\sum_{j=1}^{\xi} 100 \left| \frac{\left(v_e^s - v_j^h\right)}{v_j^h} \right| \right) \times \frac{1}{51}, \tag{28}$$

$$\text{mean return error} = \left(\sum_{j=1}^{\xi} 100 \left| \frac{\left(r_e^s - r_j^h\right)}{r_j^h} \right| \right) \times \frac{1}{51}, \tag{29}$$

respectively. For 51 different values of trade-off parameter, equally spaced in [0, 1], we run our algorithms for CCMV using the five test data sets. In our study, number of desired assets in our portfolio is 10 and lower and upper bound for all of these assets is 0.01 and 1, respectively. We implemented our algorithms in MATLAB version 8.0.0.783 and ran it on a Core CPU 2.70GHz machine with 1GB of RAM.

HMCR parameter in each step is increased linearly from 0.8 to 0.99 in the IHS parameter setting with Eq (18). Lowest value of PAR and BW both is taken to be equal to 10^{-6} and highest values of these parameters equal to 10^{-1}. Then, we linearly increased PAR while simultaneously decreased BW with Eq (19) and Eq (20), respectively. Maximum number of iterations which is used as stopping criteria is set to 50. Moreover, we set Harmony memory size equal to 10 and number of improvisation equal to 30. Also, For ICPSO parameters, we set number of populations equal to 50, maximum number of iterations equal to 100. Then, we linearly decreased inertia weight with Eq (17). Initial value of w is set to 1. Figures 1 to 5 illustrate the efficient frontier obtained from ICPSO and IHS algorithms as well as the standard efficient frontier. Table 4 shows the result of computation. It can be figured out that IHS algorithm outperform ICPSO algorithm. In Table 4 , MED is mean Euclidean distance, VRE is variance of return error and MRE is mean return error.

Table 3: Parameters setting for two meta-heuristics

ICPSO Parameter		IHS Parameter	min	max
Initial of w	1	**HMCR**	0.8	0.99
Damping rate of w	0.9	**PAR**	10^{-6}	0.1
No.population	50	**BW**	10^{-6}	0.1
Max.iteartion	100	**Max.iteartion**	50	
Constriction factor	0.729	**Harmony memory size**	10	
Global learning coefficient	2.05	**Number of improvisation**	30	
Personal learning coefficient	2.05			

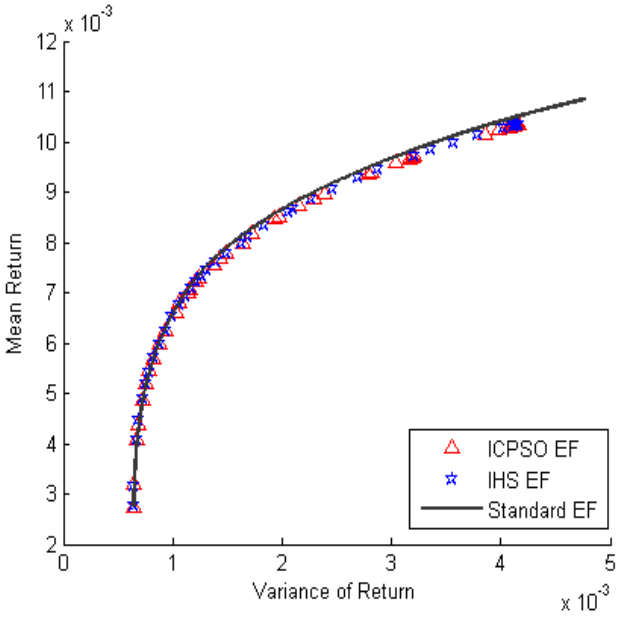

Figure 1: Efficient Frontiers for Hang Seng data set

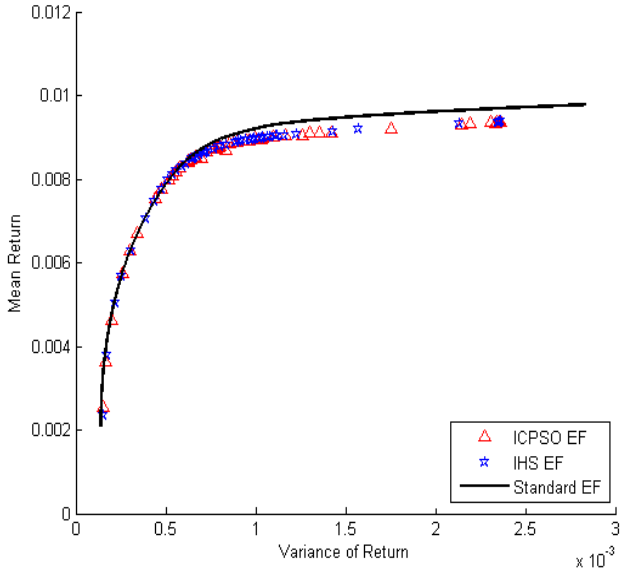

Figure 2: Efficient Frontiers for DAX data set

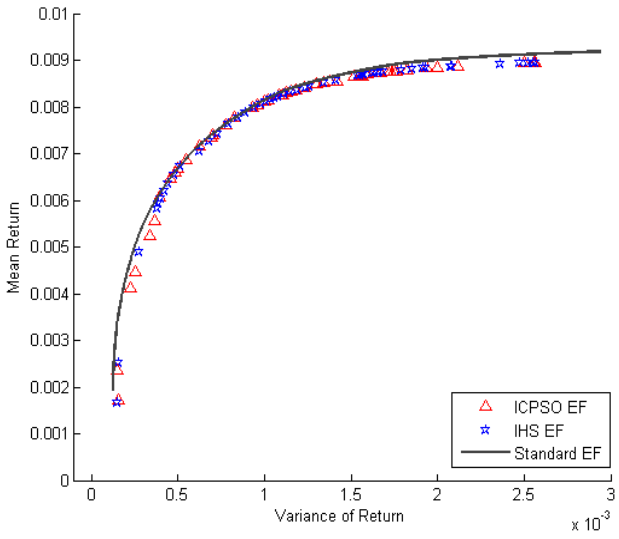

Figure 4: Efficient Frontiers for S&P 100 data set

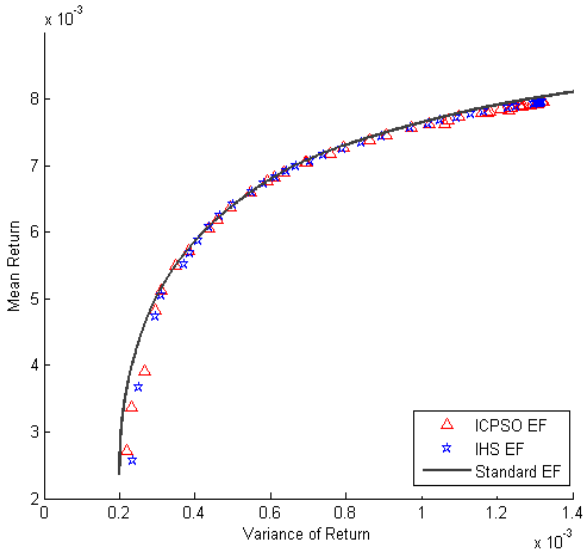

Figure 3: Efficient Frontiers for FTSE 100 data set

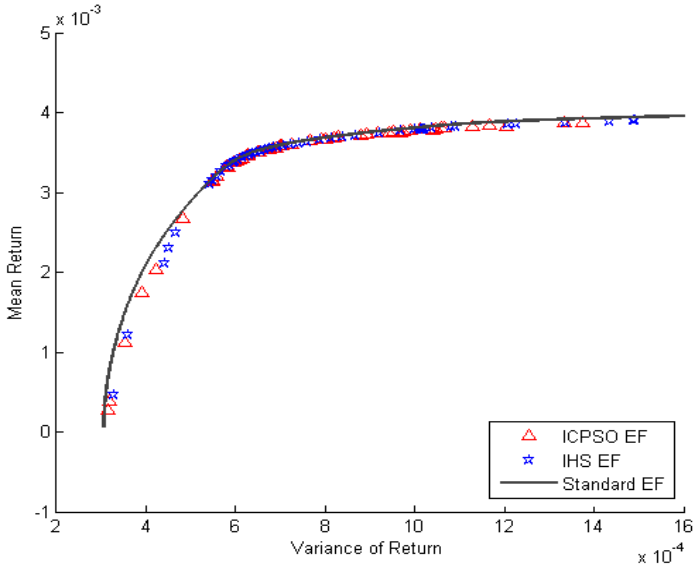

Figure 5: Efficient Frontiers for Nikkei data set

Table 4: Experimental Results for two Heuristics

Index	Number of Assets	Errors	ICPSO	IHS
HangSeng	31	MED	0.0000824	0.0000820
		VRE(%)	1.9003836	1.8044041
		MRE (%)	0.6409340	0.6483910
		Time (sec)	57	55
DAX100	85	MED	0.0001298	0.0001296
		VRE (%)	7.2061918	7.3849655
		MRE (%)	1.1876764	1.0492997
		Time (sec)	254	159
FTSE100	89	MED	0.0000408	0.0000400
		VRE (%)	3.3812060	3.2479355
		MRE (%)	0.3239293	0.3202507
		Time (sec)	269	168
S&P 100	98	MED	0.0000756	0.0000746
		VRE (%)	4.5894404	3.9023695
		MRE (%)	0.8963859	0.9480060
		Time (sec)	323	186
Nikkei	225	MED	0.0000220	0.0000206
		VRE (%)	1.8414523	1.6020636
		MRE (%)	0.4328426	0.4036726
		Time (sec)	2676	659

5. Conclusions

In this paper, we have studied Markowitz mean-variance model with limitation on the number and quantity of assets. For finding efficient portfolio and tracing efficient frontier, we proposed new modifications of PSO and HS algorithms. The two algorithms are compared on five benchmark data sets from 31 up to 225 assets. Results show that IHS algorithm outperforms ICPSO in both time and accuracy, specially on large data sets.

References:

1. G. Cornuejols, R. Tutuncu, Optimization Methods in Finance, Cambridge University Press, New York, (2007).

2. A. Fernandez, S. Gomez, Portfolio Selection Using Neural Networks, Computers & Operations Research, (2007), 34, 1177-1191.
3. K.P. Anagnostopoulos, G. Mamanis, The Mean–Variance Cardinality Constrained Portfolio Optimization Problem: An Experimental Evaluation of Five Multi-Objective Evolutionary Algorithms, Expert Systems with Applications, (2011), 38, 14208-14217.
4. T.J. Chang, N. Meade, J.E. Beasley, Y.M. Sharaiha, Heuristics for Cardinality Constrained Portfolio Optimization, Computers & Operations Research, (2000), 27, 1271-1302.
5. A. Kresta, K. Slova, Solving Cardinality Constrained Portfolio Optimization Problem by Binary Particle Swarm Optimization Algorithm, Department of Mathematical Methods in Economics, Faculty of Economics, VŠB-Technical University of Ostrava, Sokolská třída, 33(701), 21.
6. T. Cura, Particle Swarm Optimization Approach to Portfolio Optimization, Nonlinear Analysis: Real World Applications, (2008), 10, 2396-2406.
7. M. Mozafari, S. Tafazzoli, F. Jolai, A New IPSO-SA Approach for Cardinality Constrained Portfolio Optimization, International Journal of Industrial Engineering Computations, (2011), 2, 249-262.
8. G.F. Deng, W.T. Lin, C.C. Lo, Markowitz-Based Portfolio Selection with Cardinality Constraints Using Improved Particle Swarm Optimization, Expert Systems with Applications, (2012), 39, 4558-4566.
9. R.C. Eberhart, Y. Shi, Particle Swarm Optimization: Developments, Applications and Resources, In Evolutionary Computation, Proceedings of the 2001 Congress on IEEE, (2001), 1, 81-86.
10. M. Mahdavi, M. Fesanghary, E. Damangir, An Improved Harmony Search Algorithm for Solving Optimization Problems, Applied Mathematics and Computation, (2007), 188, 1567-1579.
11. Z.W. Geem, J.H. Kim, G.V. Loganathan, A New Heuristic Optimization Algorithm: Harmony Search, Simulation, (2001), 76, 60-68.
12. Nozim Komilov, Particle Swarm Intelligence: an Alternative Approach in Portfolio Optimization, Master Thesis, University Ca'Foscary, Venezia, (2008).
13. M. Salahi, A. Jamalian, A. Taati, Global Minimization of Multi-Funnel Functions Using Particle Swarm Optimization, Neural Computing and Applications, (2013), 23, 2101-2106 .
14. K. Luo, a Novel Self-Adaptive Harmony Search Algorithm, Journal of Applied Mathematics, (2013), 2013, Article ID 653749, 16 pages, doi:10.1155/2013/653749.
15. A. Kaveh, H. Nasr, Solving the Conditional and Unconditional p-center Problem with Modified Harmony Search: A Real Case Study, Scientia Iranica, (2011), 18, 867-877.

MATLAB Codes of HS and PSO

HS Method

```
clear all
%% parameters setting
global   Q  mo
load  cov1
load  mo1
Q=A; mo=mo1;
[~,nvar]=size(Q);
varsize=[1 nvar];     % Number of Variables
lambda=0;  deltalambda=0.02;
HMS=10;                % Harmony Memory Size
NHMS=30;               % New Harmony Memory Size
maxiter=50;            % Minimum Iteration
PARmin=0.000001; % Minimum Pitch Adjustment Rate
PARmax=0.1;         % Maximum Pitch Adjustment Rate
```

```
FWmin=0.000001;   % Minimum BandWith
FWmax=.1;         % Maximum BandWith
k=10;             % Number of Desired Asset
xglobal=zeros(51,nvar);
i2=1;
tic;
%% initialization
while (lamda<1.001)
empty.pos=[];
empty.cost=[];
HM=repmat(empty,HMS,1);
for i=1:HMS
HM(i).pos=unifrnd(0.01,1,[1 nvar]);
[HM(i).pos]=Harmoneyarrange2...
   (Q,mo,HM(i).pos,nvar,lamda,k);
HM(i).cost=Cost20(Q,mo,HM(i).pos,lamda);
end
[value,index]=sort([HM.cost]);
HM=HM(index);
GHM=HM(1);          % Global Harmony
%% main loop
best=zeros(maxiter,1);
MEAN=zeros(maxiter,1);
for iter=1:maxiter
   HMCR=.8+.19*iter/maxiter;  % Harmony Memory Consideration Rate
   PAR=PARmin+(PARmax-PARmin)*iter/maxiter; % Pitch Adjustment Rate
   FW=FWmax-(FWmax-FWmin)*iter/maxiter; % BandWith
   NHM=repmat(empty,NHMS,1);
  for i=1:NHMS
     NHM(i).pos=unifrnd(0.01,1,[1 nvar]);
     for j=1:nvar
     if rand<=HMCR
        kk=randi([1 HMS]);
        NHM(i).pos(j)=HM(kk).pos(j);
     else
        NHM(i).pos(j)=rand/5;
     end
     if rand<=PAR
        delta=FW*unifrnd(0.01,1)*(0.99);
        NHM(i).pos(j)=NHM(i).pos(j)+delta;
     end
     end
     [NHM(i).pos]=Harmoneyarrange2...
        (Q,mo,NHM(i).pos,nvar,lamda,k);
     NHM(i).cost=Cost20(Q,mo,NHM(i).pos,lamda);
  end
  [HM]=[HM;NHM];
  [value,index]=sort([HM.cost]);
  HM=HM(index(1:HMS));
  if HM(1).cost<GHM.cost
     GHM=HM(1);
  end
  best(iter)=GHM.cost;
```

```
  MEAN(iter)=mean([HM.cost]);
end
xglobal(i2,:)=GHM.pos;
fprintf('\n lambda(%d)= %2.2f\n  ',i2-1,lamda);
lambda=lambda+deltalambda;
i2=i2+1;
end
%% results
toc;
ResultHS (Q,xglobal,mo,nvar)
figure(3)
plot(best(1:iter),'r','LineWidth',2);
hold on
plot(MEAN(1:iter),'b','LineWidth',2);
xlabel('t')
ylabel(' fitness')
legend(' BEST' , 'MEAN')
title('HS')
```

PSO Method

```
clear all;
k=10; phi1=2.05; phi2=2.05;
phi=phi1+phi2;
chi=2/(phi-2+sqrt(phi^2-4*phi));
w1=chi*phi1;  w2=chi*phi2;
w=chi;  wdamp=.9;
global  Q  mo
load cov1
load  mo1
Q=A;
mo=mo1;
[~,nvar]=size(Q);
varsize=[1 nvar];
lamda=0;
deltalamda=0.02;
ret=zeros(1,50);
i5=1;
R=Q';
npop=50;
xglobal=zeros(npop,nvar);
maxiteration=100;
p=zeros(1,npop);
v=zeros(npop,nvar);
tic
while lamda<1.001
xp=zeros(npop,nvar);
xpbefor=xp;
zp=zeros(npop,nvar);
xpb=zeros(npop,nvar);
zpb=zeros(npop,nvar);
vx=zeros(npop,nvar);
vxbefor=vx;
```

```
vz=zeros(npop,nvar);
cost=zeros(1,npop);
bestcost=zeros(1,npop);
globalbestcost=inf;
for i=1:npop
   xp(i,:)=unifrnd(0,1,varsize);
   xp(i,:)=xp(i,:)/sum(xp(i,:));
   xpbefor(i,:)=xp(i,:);
   zp(i,:)=floor(unifrnd(0,1,varsize)+.5);
   vx(i,:)=zeros(varsize);
   vxbefor(i,:)=vx(i,:);
   vz(i,:)=zeros(varsize);
   cost(i)=rand;
   xpb(i,:)=xp(i,:);
   zpb(i,:)=zp(i,:);
   bestcost(i)=cost(i);
   if bestcost(i)<globalbestcost
      globalbestcost=bestcost(i);
      Gxpb=xpb(i,:);
      Gzpb=zpb(i,:);
   end
end
for i=1:npop
   v(i,:)=xp(i,:)+w1*(Gxpb-xp(i,:))+w2*(xpb(i,:)-xp(i,:));
   [xp(i,:),zp(i,:),vx(i,:)]=...
   arrangenewer(Q,mo,xp(i,:),xpbefor(i,:),zp(i,:),vx(i,:),vxbefor(i,:),v(i,:),nvar,lamda,k);
  cost(i)=Cost21(Q,mo,xp(i,:),zp(i,:),lamda);
   if cost(i)<bestcost(i)
      bestcost(i)=cost(i);
      xpb(i,:)=xp(i,:);
      zpb(i,:)=zp(i,:);
      if bestcost(i)<globalbestcost
        globalbestcost=bestcost(i);
        Gxpb=xpb(i,:);
        Gzpb=zpb(i,:);
      end
   end
end
gamma=globalbestcost;
for counter=1:maxiteration
   t=gamma;
    for i=1:npop
       for j=1:nvar
          r1=rand;r2=rand;
          vz(i,j)=(w*vz(i,j)+w1*r1*(zpb(i,j)-zp(i,j))+w2*r2*(Gzpb(j)-zp(i,j)));
          kesi=zp(i,j)+vz(i,j);
          alpha=0.06;
          zp(i,j)=round(1/(1+exp(-kesi))-alpha);
          if zp(i,j)==1
              vxbefor(i,:)=vx(i,:);
              xpbefor(i,:)=xp(i,:);
              vx(i,j)=(w*vx(i,j)+w1*r1*(xpb(i,j)-xp(i,j))+w2*r2*(Gxpb(j)-xp(i,j)));
             if vx(i,j)+xp(i,j)>=0
```

```
                xp(i,j)=vx(i,j)+xp(i,j);
            else
                zp(i,j)=0;
                xp(i,j)=0;
            end
        else
            xp(i,j)=0;
        end
        v(i,:)=(xp(i,:)+w1*(xpb(i,:)-xp(i,:))+w2*(Gxpb-xp(i,:)));
  [xp(i,:),zp(i,:),vx(i,:)]=...
    arrangenewer(Q,mo,xp(i,:),xpbefor(i,:),zp(i,:),vx(i,:),vxbefor(i,:),v(i,:),nvar,lamda,k);
  cost(i)=Cost21(Q,mo,xp(i,:),zp(i,:),lamda);
  if cost(i)<bestcost(i)
     bestcost(i)=cost(i);
     xpb(i,:)=xp(i,:);
     zpb(i,:)=zp(i,:);
     if bestcost(i)<globalbestcost
       globalbestcost=bestcost(i);
       Gxpb=xpb(i,:);
       Gzpb=zpb(i,:);
     end
  end
 if globalbestcost<gamma
     gamma=globalbestcost;
 end
      end
    end
    if counter>5
     if norm(t-gamma)<1e-6
          break
     end
    end
     w=w*wdamp;
end
xglobal(i5,:)=Gxpb;
ret(i5)=gamma;
fprintf('\n lamda(%d)= %2.2f\n  ',i5-1,lamda);
lambda=lambda+deltalambda;
i5=i5+1;
end
fprintf('\n');
toc
Result (Q,xglobal,mo,nvar)
figure (3);
plot(ret,'Linewidth',2)
xlabel('Iteration')
ylabel('Best Cost')
globalbestpositionx=Gxpb;
[~,pp]=size(globalbestpositionx);
PrinteResult(globalbestpositionx,gamma,pp)
```