Optimization Letters (2020) 14:747–758
https://doi.org/10.1007/s11590-019-01398-w

ORIGINAL PAPER

The effect of velocity sparsity on the performance of cardinality constrained particle swarm optimization

Kris Boudt[1,2] · **Chunlin Wan**[3]

Received: 16 February 2018 / Accepted: 29 January 2019 / Published online: 13 February 2019

Abstract
The Particle Swarm Optimization (PSO) algorithm is a flexible heuristic optimizer that can be used for solving cardinality constrained binary optimization problems. In such problems, only K elements of the N-dimensional solution vector can be non-zero. The typical solution is to use a mapping function to enforce the cardinality constraint on the trial PSO solution. In this paper, we show that when K is small compared to N, the use of the mapped solution in the velocity vector tends to lead to early stagnation. As a solution, we recommend to use the untransformed solution as a direction in the velocity vector. We use numerical experiments to document the gains in performance when K is small compared to N.

Keywords Binary particle swarm optimization · Cardinality mapping · Portfolio optimization

Numerous optimization problems in operations research and finance can be framed as a cardinality constrained binary optimization problem (CCBOP). It involves optimizing a possibly non-linear objective function in terms of a binary solution vector under the so-called cardinality constraint that there are at most a given number K of non-zero elements in the N-dimensional solution vector. Because of the cardinality constraint, standard gradient-based optimization methods are not applicable. When the solution vector and the number of non-zero elements are large, the brute force solution of examining all possible combinations becomes quickly impossible in practice. A solution is therefore to use heuristic optimization methods, such as differential evolution, genetic

✉ Chunlin Wan
chunlinwan@foxmail.com

Kris Boudt
kris.boudt@vub.be

1 Solvay Business School, Vrije Universiteit Brussel, Brussels, Belgium

2 School of Business and Economics, Vrije Universiteit, Amsterdam, The Netherlands

3 School of Economics, Sichuan University, 24, South Section 1, Yihuan Road, Chengdu, China

optimization, tabu search, simulated annealing, and the particle swarm optimization (PSO) approach (see e.g., [1,3,4,7,8]). Inspired by animal flocking behaviour, the PSO solves the problem based on a population of candidate solutions, called particles, that, in each iteration, move around in the search-space as a function of their position and velocity.

The velocity is a crucial component in the PSO as it determines how the local position can learn from local and global optimal solutions. More precisely, in the original PSO algorithm proposed by Kennedy and Eberhart [10], the velocity of a current solution is a linear function of the past velocity, the distance between the locally best solution and the current solution, and the distance between the globally best and the current solution. The original PSO fails to take the cardinality constraint into account. In order to handle cardinality constrained optimization problem, Cura [5] and Deng et al. [6], among others, recommended to use a mapping function at each iteration of the PSO algorithm to transform the candidate solution into a solution that belongs to the feasible space. An outstanding question is how this mapping should affect the velocity equation.

In Cura [5] and Deng et al. [6] the velocity is a function of the mapped solution. This leads to a sparse specification of the velocity. We call this approach the sparse velocity PSO (SV-PSO) approach. An alternative is to set the velocity as a function of the continuous (not-mapped) solution. We call this the continuous velocity PSO (CV-PSO) approach. We test the performance of the SV-PSO and CV-PSO algorithms for solving the portfolio optimization problem of finding the cardinality constrained portfolio that maximizes the mean-variance utility function. Our tests using both artificially generated returns and real-world stock returns show that, for small values of K compared to N, the SV-PSO algorithm can reach a good solution, but often is caught in stagnation with long iterations compared to the CV-PSO algorithm. This happens especially when the number of non-zero elements is small. In contrast, the CV-PSO algorithm reaches a high value more gradually, but with an apparently lower risk of early stagnation.

The remainder of the paper proceeds as follows. Section 1 introduces the cardinality constrained binary optimization problem and the traditional PSO algorithm, and also introduces the SV and CV versions of the PSO algorithm. Section 2 summarizes the results of the performance evaluation. Finally, Sect. 3 concludes the paper.

1 Definitions and pseudo-code

1.1 The cardinality constrained binary optimization problem

The decision maker needs to optimize the function f that depends on a N-dimensional binary decision vector x. The optimization is restricted by the cardinality constraint that at most K elements of x can be equal to one. The corresponding cardinality constrained binary optimization problem is:

$$\max_x \quad f(x) \quad \text{subject to} \quad x \in \{0,1\}^N, \quad x'\iota \leqslant K, \tag{1}$$

with ι the N-dimensional vector of ones. Note that this optimization problem is in essence a combinatorial selection problem, which is a widespread problem in decision making.

In this paper, we focus on the more specific case of portfolio selection based on the so-called mean-variance utility function. We thus consider the problem of optimizing the allocation to N financial assets with vector of expected returns (in excess of the risk free rate) μ and covariance matrix Σ. The portfolio is assumed to be fully invested and the portfolio weights are given by $x/(x'\iota)$. The scaling ensures that the portfolio is fully invested. As in the modern portfolio theory of Markowitz [11], we assume investors have mean-variance preferences with risk aversion parameter $\gamma > 0$, leading to the following objective function:

$$f(x) = \frac{x'\mu}{x'\iota} - \frac{\gamma}{2}\frac{x'\Sigma x}{(x'\iota)^2}. \tag{2}$$

In the absence of cardinality constraints, a solution to this problem follows from solving the first order conditions. In case of linear constraints, the solution is easily obtained by using quadratic programming. Our interest is in solving the problem under the cardinality constraint.

1.2 Traditional particle swarm optimization

The original PSO algorithm of Kennedy and Eberhart [10] uses an iterative solution-finding rule that is based on the observations of the social behaviour of animals. It assumes that a number of particles coexist and cooperate to get to the optimal solution.

Each particle represents a potential solution to the optimization problem. At each iteration t (for $t = 1, \ldots, T$), with T the maximum number of iterations, each particle i flies from its current position

$$x_i(t) = (x_{i1}(t), x_{i2}(t), \ldots, x_{iN}(t))',$$

with a changing velocity

$$v_i(t) = (v_{i1}(t), v_{i2}(t), \ldots, v_{iN}(t))',$$

to a better position in the search space in accordance to its own experience and the best experience of the adjacent particle swarm. This happens through the following updating equations:

$$\begin{aligned} v_i(t) &= wv_i(t-1) + c_1 r_1(t)[x_{li}(t-1) - x_i(t-1)] + c_2 r_2(t)[x_g(t-1) - x_i(t-1)] \\ x_i(t) &= x_i(t-1) + v_i(t), \end{aligned} \tag{3}$$

where $w, c_1, c_2 \geq 0$, $x_{li}(t)$ is the personal best experience (or local best) of particle i until iteration t, and $x_g(t)$ is the global best experience of the whole swarm up to iteration t. The values $r_1(t)$ and $r_2(t)$ are two independent random draws from the [0, 1] uniform distribution. We further have that the tuning parameter w is called the

inertia weight that is associated with the previous velocity. The scalars c_1 and c_2 are the acceleration factors (or learning factors) determining the particle's maximum step size to fly to the best position of individual particle and the best position of the whole swarm.

The updated velocity in (3) is composed of three terms. The first term is proportional to the previous velocity and provides momentum to the particle movement. The second term is driven by the deviation of the particle position with respect to the personal best experience [*i.e.*, $x_{li}(t-1) - x_i(t-1)$]. This represents the personal thinking of each particle and is known as the cognitive component. The cognitive component plays a key role by encouraging the particles to move toward the local best. The last item $(x_g(t-1) - x_i(t-1))$ is the social component, which helps the particles to find the global optimal solution by pulling the particles to the global best. The corresponding pseudo-code is given in Algorithm 1.

Algorithm 1: Pseudo-code of the traditional PSO algorithm.

1. Set n to the total number of N-dimensional particles in the swarm and T to the maximum number of iterations;
2. Initialize the current positions and velocities of the n particles as random draws from the [0,1] and [-1,1] uniform distribution respectively;
3. Set the local best position x_{li} of particle i to its current position ;
4. Set $x_g = x_{lg}$ to the global best position, where $g = \arg \max_{1 \leqslant i \leqslant n} f(x_{li})$;
5. For $t = 1 : T$
   - 5.1. For each particle $i = 1, \ldots, n$
     - Set $v_i(t) = w v_i(t-1) + c_1 r_1 (x_{li}(t-1) - x_i(t-1)) + c_2 r_2 (x_g(t-1) - x_i(t-1))$;
     - Set $x_i(t) = x_i(t-1) + v_i(t)$;
     - End
   - 5.2. For each particle $i = 1, \ldots, n$
     - If $f(x_i(t)) > f(x_{li})$, then $x_{li} = x_i(t)$;
     - If $f(x_{li}) > f(x_g)$, then $x_g = x_{li}$;
     - End
   - End
6. Set $x^* = x_g$.

1.3 Binary PSO

The traditional PSO in Algorithm 1 cannot solve the cardinality constraint. A common solution is to use a mapping function to deterministically force the candidate solution into an acceptable solution (see e.g., [5,6]). Given our cardinality constraint, this means setting the K largest values in x to unity, and the remainder to zero. The corresponding mapping function at iteration t, denoted by $h_t(\cdot)$, is as follows:

$$h_t(x) \equiv I[x > x_{(N-K)}(t)] \tag{4}$$

where, $I[.]$ is the vector-valued indicator function, for which element k is one if the condition is fulfilled for element k, and zero otherwise. The variable $x_{(N-K)}(t)$ is the $N - K$ largest value in $x(t)$.

We discuss now two implementations of the PSO algorithm that differ in how they impose the binary cardinality constraint. The traditional implementation is the so-called "Sparse Velocity" PSO (SV-PSO) algorithm as described in Cura [5] and Deng et al. [6]. As an alternative, we recommend to use a "Continuous Velocity" PSO (CV-PSO) algorithm.

In the SV-PSO algorithm described in Algorithm 2, the velocity is updated using the sparse vector $\tilde{x}$ as driver. The vector of velocity is thus also sparse, and included only K non-zero elements. Since it can only change at most $2K$ components in x, the SV-PSO algorithm may stagnate.

Algorithm 2: Pseudo-code of the SV-PSO algorithm.

1. Set n to the total number of N-dimensional particles in the swarm and T to the maximum number of iterations;
2. Initialize the current positions and velocities of the n particles as random draws from the [0,1] and [-1,1] uniform distribution respectively;
3. Set $\tilde{x}_i = h(x_i)$;
4. Set the local best position $\tilde{x}_{li} = \tilde{x}_i$ of particle i to its current position;
5. Set $x_g = x_{lg}$ to the global best position, where $g = \arg \max_{1 \leqslant i \leqslant n} f(\tilde{x}_{li})$;
6. For $t = 1 : T$
   - 6.1. For each particle $i = 1, \ldots, n$
     - Set $v_i(t) = w v_i(t-1) + c_1 r_1 (\tilde{x}_{li}(t-1) - \tilde{x}_i(t-1)) + c_2 r_2 (\tilde{x}_g(t-1) - \tilde{x}_i(t-1))$;
     - Set $x_i(t) = \tilde{x}_i(t-1) + v_i(t)$;
     - End
   - 6.2. Set $\tilde{x}_i(t) = h(x_i(t))$;
   - 6.3. For each particle $i = 1, \ldots, n$
     - If $f(\tilde{x}_i(t)) > f(\tilde{x}_{li})$, then $\tilde{x}_{li} = \tilde{x}_i(t)$;
     - If $f(\tilde{x}_{li}) > f(\tilde{x}_g)$, then $\tilde{x}_g = \tilde{x}_{li}$;
     - End
   - End
7. Set $x^* = \tilde{x}_g$.

In contrast with the SV-PSO algorithm, the proposed CV-PSO algorithm in Algorithm 3 uses the continuous solution vector x as the driver of the changes in velocity from one iteration to the other. In this heuristic implementation, the velocity and position still contain all the information of the particle, so it has more flexibility to improve in each iteration and thus avoid early stagnation. However, the vector of velocity and position of the CV-PSO algorithm may include some redundant information because the theoretically optimal solution only contains K non-zero elements.

Both algorithms have thus their pros and cons, which leads us to do numerical experiments to compare the performance of the two algorithms in practice.

2 Numerical experiments and analysis

First, we describe the parameter setup of the implementation of the PSO algorithms. Then we test the performance of both the SV-PSO and CV-PSO algorithms using

Algorithm 3: Pseudo-code of the CV-PSO algorithm.

1. Set n to the total number of N-dimensional particles in the swarm and T to the maximum number of iterations;
2. Initialize the current positions and velocities of the n particles as random draws from the [0,1] and [-1,1] uniform distribution respectively;
3. Set $\tilde{x}_i = h(x_i)$;
4. Set the local best position $x_{li} = x_i$ of particle i to its current position;
5. Set $x_g = x_{lg}$ to the global best position, where $g = \arg \max_{1 \leqslant i \leqslant n} f(x_{li})$;
6. For $t = 1 : T$
   6.1. For each particle $i = 1, \ldots, n$
      Set $v_i(t) = wv_i(t-1) + c_1 r_1(x_{li}(t-1) - x_i(t-1)) + c_2 r_2(x_g(t-1) - x_i(t-1))$;
      Set $x_i(t) = x_i(t-1) + v_i(t)$;
      End
   6.2. Set $\tilde{x}_i(t) = h(x_i)$;
   6.3. For each particle $i = 1, \ldots, n$
      If $f(\tilde{x}_i(t)) > f(\tilde{x}_{li})$, then $x_{li} = x_i(t)$;
      If $f(\tilde{x}_{li}) > f(\tilde{x}_g)$, then $x_g = x_{li}$;
      End
   End
7. Set $x^* = h(x_g)$.

stylized and real data experiments. Throughout the analysis, we set the risk aversion parameter γ in (2) equal to 10.

2.1 Parameter setting

The implementation of the PSO algorithm is as usual in the literature (see e.g., [9,12,13, 15]). The swarm size n is set to 1000 particles, and the maximum number of iterations is 1000. The position of particles $x_{i,j}$ is limited to [0, 1] at each iteration, because $x_{i,j}$ represents the weight in a long-only portfolio in our experiments. Similarly, we box constraint the velocity parameter by limiting its value to be less than one in absolute values. As usual, the parameter of inertia weight w is set to the fixed value of 0.4. We consider both fixed values for the acceleration factors c_1 and c_2, namely $c_1 = c_2 = 1.5$ and $c_1 = c_2 = 2$, as well as the following iteration-dependent setup, as described in Ratnaweera et al. [14] and Tripathi et al. [16]:

$$c_1(t) = (c_{1\,\min} - c_{1\,\max})(t/T) + c_{1\,\max} \tag{5}$$

$$c_2(t) = (c_{2\,\max} - c_{2\,\min})(t/T) + c_{2\,\min}, \tag{6}$$

where $c_{1\,\max} = c_{2\,\max} = 2.5$ and $c_{1\,\min} = c_{2\,\min} = 0.5$ and T is the maximum number of iterations. The results of all experiments are aggregated as the median value over 100 independent runs.

2.2 Evaluation on a stylized simulation setup

We first consider a stylized setup in which the solution is known. We assume there are $N = 500$ stocks available in the investment universe, while the number of assets in

the portfolio K is 10, 15, 20, 30, 40 and 50 stocks, respectively. The vector of mean excess returns μ is 3% for the first K stocks, and 0 for the remaining ones. All 500 stocks have an annualized standard deviation of 15% and the correlations among them are zero. The optimal solution is thus to equally weight the first K stocks.

Table 1 shows the results obtained for the two proposed PSO algorithms. The table shows the percentage of correctly selected stocks in percentage points. In this stylized setting, the global optimum is known and equals 100. For each algorithm, we indicate the best results (out of the various implementations considered) in italics. It can be observed that the CV-PSO algorithm could get the best result only by adopting $c_1 = c_2 = 2$, whereas for the SV-PSO algorithm, it depends on K. For larger values of K, it performs best by adopting the time variant c. For small values of K, the best performance is achieved when $c_1 = c_2 = 1.5$.

For each optimization problem characterized by the choice of K, we put the best performing algorithm in boldface. For small values of K, namely $K = 10$, 15 and 20, the CV-PSO algorithm outperforms the traditional SV-PSO algorithm. To explain this better performance for small values of K, we show for both the CV-PSO and SV-PSO, their best trajectory in Fig. 1 for the same cardinality constraint as in Table 1. We find that the SV-PSO algorithm reaches the optimal value quickly when $K = 30$, 40 and 50, but is caught in early stagnation when $K = 10$, 15 and 20. The CV-PSO algorithm does not suffer from this type of early stagnation, as it reaches higher values of the objective function in a rather smooth manner, at the cost of a slower speed of convergence when K is larger.

2.3 Real data experiments

For the performance evaluation on real data, we use the Nikkei 225 publicly available test data of Chang et al. [3]. It consists of the mean (μ) and covariance matrix (Σ) of the weekly returns for the 225 components of the Nikkei 225 stock market index over the period March 1992 to September 1997.[1] We set the cardinality number K equal to 5, 10, 15, 20, 30 and 50, respectively. All the experimental results of two proposed PSO algorithms with various parameters on the real-life data are listed in Table 2, where we have used similar notations as in Table 1. In particular, we indicate for each optimization problem, the best performing algorithm in boldface. We find that, for small values of K, namely $K = 5$, 10, 15 and 20, the CV-PSO algorithm outperforms the traditional SV-PSO algorithm.[2]

Figure 2 shows the trajectory of the two algorithms for $K = 5$, $K = 10$, $K = 15$, $K = 20$, $K = 30$ and $K = 50$. Note that, the SV-PSO is caught in early stagnation

[1] We are grateful to the referee for recommending us to use publicly available test data. The data is available at http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/port5.txt.

[2] For this real-data setup, the best value of the objective function is unknown in practice. As a sanity check recommended by the reviewer, we have also optimized the function using an alternative heuristic, namely Differential Evolution, as implemented in the R package DEoptim ([2]), and using the sparse mapping function described at https://github.com/R-Finance/PortfolioAnalytics/blob/master/sandbox/testing_DEoptim_cardinality_constraint.R. The results obtained are similar as those found for the SV-PSO, both in terms of best value for the objective function and the shape of the trajectory.

Table 1 Effect of velocity specification on the accuracy of selection (in %) for the portfolio invested in $K = 10$, $K = 15$, $K = 20$, $K = 30$, $K = 40$ and $K = 50$ out of $N = 500$ stocks for various calibrations of c

Method	Iteration	K = 10			K = 15			K = 20		
		$c = 1.5$	$c = 2$	$c = c(t)$	$c = 1.5$	$c = 2$	$c = c(t)$	$c = 1.5$	$c = 2$	$c = c(t)$
SV-PSO	50	*30*	30	20	20	*26.7*	20	*20*	30	25
	250	*30*	30	20	26.7	*26.7*	20	*20*	30	25
	500	*30*	30	20	26.7	*26.7*	26.7	*30*	30	90
	750	*30*	30	20	26.7	*26.7*	80	*80*	30	90
	1000	*90*	90	90	86.7	*86.7*	86.7	*90*	90	90
CV-PSO	50	70	***80***	70	80	***73.3***	80	55	***65***	75
	250	90	***100***	100	80	***93.3***	93.3	70	***95***	95
	500	90	***100***	100	80	***100***	100	75	***95***	95
	750	90	***100***	100	80	***100***	100	75	***100***	95
	1000	90	***100***	100	80	***100***	100	75	***100***	95
Method	Iteration	K = 30			K = 40			K = 50		
		$c = 1.5$	$c = 2$	$c = c(t)$	$c = 1.5$	$c = 2$	$c = c(t)$	$c = 1.5$	$c = 2$	$c = c(t)$
SV-PSO	50	20	26.7	***23.3***	30	25	***35***	52	24	***68***
	250	50	26.7	***90***	92.5	27.5	***95***	98	60	***100***
	500	93.3	26.7	***100***	97.5	40	***100***	98	74	***100***
	750	96.7	26.7	***100***	97.5	52.5	***100***	98	88	***100***
	1000	96.7	93.3	***100***	97.5	95	***100***	98	94	***100***
CV-PSO	50	53.3	*60*	73.3	60	*62.5*	70	58	*66*	62
	250	60	*86.7*	86.7	62.2	*95*	80	62	*92*	78
	500	63.3	*93.3*	93.3	62.5	*97.5*	85	62	*94*	78
	750	63.3	*93.3*	93.3	62.5	*97.5*	85	62	*98*	80
	1000	63.3	*96.7*	93.3	62.5	*97.5*	85	62	*98*	80

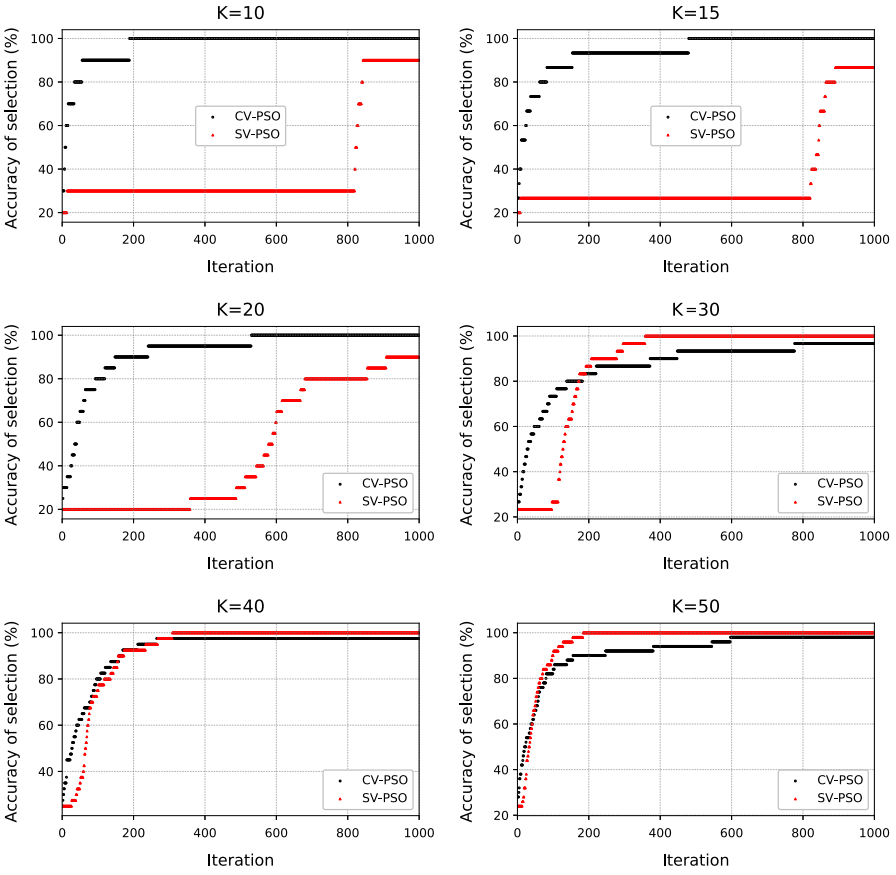

Fig. 1 Accuracy of selection at each iteration of the two proposed PSO algorithms. *Note*: We show the trajectory for the best performing SV-PSO and CV-PSO algorithms for each value of K. The corresponding numbers are indicated in italics in Table 1

when K is small. For small values of K, the CV-PSO algorithm is therefore more reliable as it gradually improves to achieve the best value.

3 Conclusion

This work contributes to the use of particle swarm optimization (PSO) for solving cardinality constrained binary optimization problems in which at most K elements in the N-dimensional solution vector can be non-zero. We investigate whether the velocity of particles should be driven by a sparse or continuous solution vector. The traditional solution in Cura [5] and Deng et al. [6] is to use sparsity. We call this the SV-PSO algorithm. We find that it suffers from early stagnation when K is small compared to N. As a solution, we propose to set the velocity as a function of the continuous solution vector. The convergence of the CV-PSO algorithm is slower, but

Table 2 Effect of velocity specification on the average mean variance utility (in %) for the portfolio invested in $K = 5$, $K = 10$, $K = 15$, $K = 20$, $K = 30$ and $K = 50$ out of $N = 225$ stocks for various calibrations of c

Method	Iteration	K = 5			K = 10			K = 15		
		$c = 1.5$	$c = 2$	$c = c(t)$	$c = 1.5$	$c = 2$	$c = c(t)$	$c = 1.5$	$c = 2$	$c = c(t)$
SV-PSO	50	−0.165	−0.149	−0.136	−0.243	−0.266	−0.269	−0.164	−0.328	−0.147
	250	−0.148	−0.149	−0.035	−0.070	−0.266	−0.007	−0.042	−0.285	−0.012
	500	−0.057	−0.149	0.012	−0.001	−0.204	0.003	−0.015	−0.135	−0.012
	750	*0.003*	−0.149	0.012	*0.003*	−0.143	0.003	−0.015	−0.043	−0.012
	1000	*0.023*	0.021	0.012	*0.003*	−0.001	0.003	−0.015	−0.017	−0.012
CV-PSO	50	0.006	***0.031***	0.031	−0.031	***−0.006***	−0.014	−0.073	***−0.040***	−0.048
	250	0.006	***0.031***	0.031	−0.007	***0.009***	0.007	−0.070	***−0.010***	−0.031
	500	0.006	***0.031***	0.031	−0.007	***0.012***	0.007	−0.070	***−0.010***	−0.031
	750	0.006	***0.031***	0.031	−0.07	***0.012***	0.007	−0.061	***−0.010***	−0.031
	1000	0.006	***0.031***	0.031	−0.007	***0.012***	0.007	−0.061	***−0.010***	−0.031
Method	Iteration	K = 20			K = 30			K = 50		
		$c = 1.5$	$c = 2$	$c = c(t)$	$c = 1.5$	$c = 2$	$c = c(t)$	$c = 1.5$	$c = 2$	$c = c(t)$
SV-PSO	50	−0.111	−0.378	−0.102	−0.113	−0.219	***−0.087***	−0.163	***−0.183***	−0.161
	250	−0.053	−0.059	−0.035	−0.082	−0.096	***−0.081***	−0.156	***−0.156***	−0.156
	500	−0.037	−0.047	−0.035	−0.082	−0.091	***−0.081***	−0.156	***−0.156***	−0.156
	750	−0.037	−0.047	−0.035	−0.082	−0.091	***−0.081***	−0.156	***−0.156***	−0.156
	1000	−0.037	−0.047	−0.035	−0.082	−0.083	***−0.081***	−0.156	***−0.156***	−0.156
CV-PSO	50	−0.061	***−0.107***	−0.096	−0.157	−0.126	−0.166	−0.232	−0.211	−0.238
	250	−0.061	***−0.039***	−0.065	−0.157	−0.090	−0.140	−0.231	−0.165	−0.226
	500	−0.061	***−0.034***	−0.062	−0.157	−0.086	−0.140	−0.231	−0.157	−0.221
	750	−0.061	***−0.034***	−0.062	−0.154	−0.083	−0.136	−0.231	−0.158	−0.221
	1000	−0.061	***−0.034***	−0.062	−0.154	−0.081	−0.136	−0.231	−0.157	−0.221

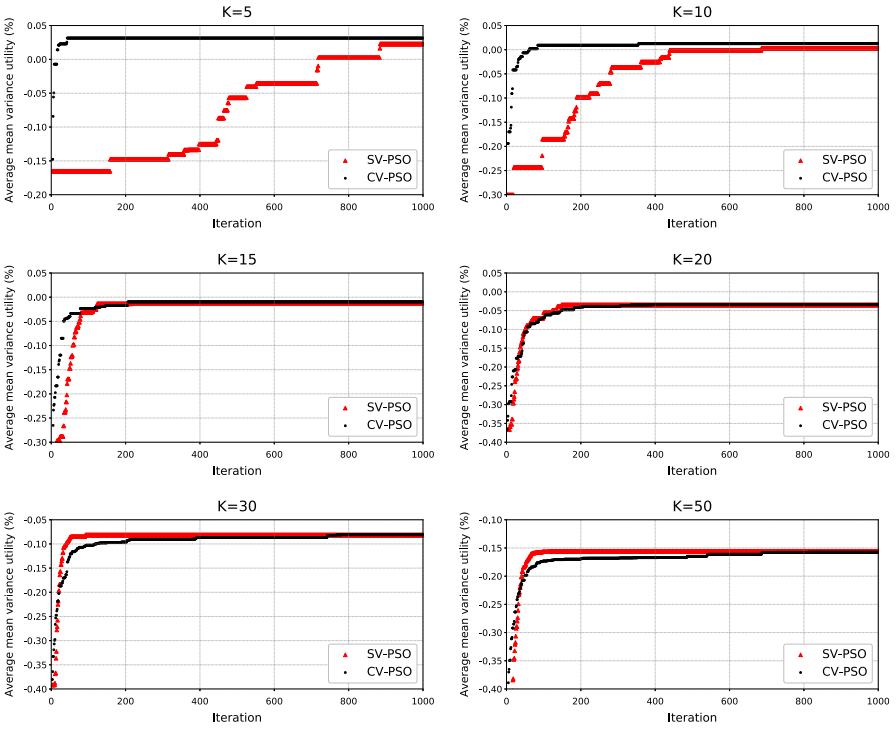

Fig. 2 The average mean variance utility at each iteration of the two proposed PSO algorithms. *Note*: We show the trajectory for the best performing SV-PSO and CV-PSO algorithms for each value of K. The corresponding numbers are indicated in italics in Table 2

more effective and thus more reliable in solving the cardinality constrained binary optimization problems when K is small compared to N.

The scope of our findings goes beyond the implementation of the particle swarm heuristic for cardinality constrained portfolio optimization. In fact, many heuristics use iterations in which the improved solution is found based on a function of the local best values. A notable example is Differential Evolution, as implemented in the R package DEoptim [1,2]. Our initial exploration shows that the use of sparsity in the updating equation used in Differential Evolution leads to similar results as for PSO. We therefore expect similar gains for those heuristics when avoiding to impose sparsity in the updating equations. A complete comparison across heuristics is an interesting avenue for further research, which falls beyond the scope of this letter.

Acknowledgements This paper was written while Chunlin Wan was visiting the finance department of the Solvay Business School at Vrije Universiteit Brussel. We are grateful to the Editor (Oleg Prokopyev), the associate editor, an anonymous referee, Xiang Deng, Paola Festa, Jiayin Gu, Nanjing Huang, Joao Miranda, Giang Nguyen, Wajid Raza and Marjan Wauters for helpful comments and suggestions. Financial support from the Chinese Scholarship Council and the National Natural Science Foundation of China (Grant Number 71742004) is gratefully acknowledged.

References

1. Ardia, D., Boudt, K., Carl, P., Mullen, K.M., Peterson, B.G.: Differential evolution with DEoptim: an application to non-convex portfolio optimization. R J. **3**(1), 27–34 (2011)
2. Ardia, D., Mullen, K., Peterson, B., Ulrich, J.: DEoptim: differential evolution optimization in R. R package version 2.2-4 (2007)
3. Chang, T.-J., Meade, N., Beasley, J.E., Sharaiha, Y.M.: Heuristics for cardinality constrained portfolio optimisation. Comput. Oper. Res. **27**(13), 1271–1302 (2000)
4. Chang, T.-J., Yang, S.-C., Chang, K.-J.: Portfolio optimization problems in different risk measures using genetic algorithm. Expert Syst. Appl. **36**(7), 10529–10537 (2009)
5. Cura, T.: Particle swarm optimization approach to portfolio optimization. Nonlinear Anal. Real World Appl. **10**(4), 2396–2406 (2009)
6. Deng, G.-F., Lin, W.-T., Lo, C.-C.: Markowitz-based portfolio selection with cardinality constraints using improved particle swarm optimization. Expert Syst. Appl. **39**(4), 4558–4566 (2012)
7. Fernández, A., Gómez, S.: Portfolio selection using neural networks. Comput. Oper. Res. **34**(4), 1177–1191 (2007)
8. Garg, H.: A hybrid pso-ga algorithm for constrained optimization problems. Appl. Math. Comput. **274**, 292–305 (2016)
9. Jarboui, B., Damak, N., Siarry, P., Rebai, A.: A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. Appl. Math. Comput. **195**(1), 299–308 (2008)
10. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Paper Presented at IEEE International Conference on Neural Networks, Perth, Australia. IEEE Service Center, Piscataway, NJ (1995)
11. Markowitz, H.: Portfolio selection. J. Finance **7**(1), 77–91 (1952)
12. Poli, R., Kennedy, J., Blackwell, T.: Particle swarm optimization. Swarm Intell. **1**(1), 33–57 (2007)
13. Pouya, A.R., Solimanpur, M., Rezaee, M.J.: Solving multi-objective portfolio optimization problem using invasive weed optimization. Swarm Evol. Comput. **28**, 42–57 (2016)
14. Ratnaweera, A., Halgamuge, S.K., Watson, H.C.: Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. IEEE Trans. Evol. Comput. **8**(3), 240–255 (2004)
15. Sedighizadeh, D., Masehian, E.: Particle swarm optimization methods, taxonomy and applications. Int. J. Comput. Theory Eng. **1**(5), 486–502 (2009)
16. Tripathi, P.K., Bandyopadhyay, S., Pal, S.K.: Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients. Inf. Sci. **177**(22), 5033–5049 (2007)