Computers & Operations Research 39 (2012) 1555–1565

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Solving the stochastic dynamic lot-sizing problem through nature-inspired heuristics

G.S. Piperagkas [a], I. Konstantaras [b], K. Skouri [c], K.E. Parsopoulos [a,*]

[a] Department of Computer Science, University of Ioannina, GR-45110 Ioannina, Greece
[b] Department of Business Administration, University of Macedonia, GR-54006 Thessaloniki, Greece
[c] Department of Mathematics, University of Ioannina, GR-45110 Ioannina, Greece

ARTICLE INFO

Available online 12 September 2011

Keywords:
Dynamic lot-size problem
Wagner–Whitin algorithm
Particle swarm optimization
Differential evolution
Harmony search

ABSTRACT

We investigate the dynamic lot-size problem under stochastic and non-stationary demand over the planning horizon. The problem is tackled by using three popular heuristic methods from the fields of evolutionary computation and swarm intelligence, namely particle swarm optimization, differential evolution and harmony search. To the best of the authors' knowledge, this is the first investigation of the specific problem with approaches of this type. The algorithms are properly manipulated to fit the requirements of the problem. Their performance, in terms of run-time and solution accuracy, is investigated on test cases previously used in relevant works. Specifically, the lot-size problem with normally distributed demand is considered for different planning horizons, varying from 12 up to 48 periods. The obtained results are analyzed, providing evidence on the efficiency of the employed approaches as promising alternatives to the established Wagner–Whitin algorithm, as well as hints on their proper configuration.

1. Introduction

The deterministic lot-size (DLS) problem consists of determining the quantity of products to order or produce in each time period over a finite discrete planning horizon, in order to satisfy the demand for each period while minimizing the summation of setup and inventory holding costs. This model was first introduced by Wagner and Whitin in 1958 [1], who developed an $O(H^2)$ forward algorithm for a general dynamic version of the uncapacitated economic lot-size model, where H stands for the number of time periods.

DLS is embedded within many practical production planning problems. The zero-inventory ordering principle, which imposes that no production is undertaken if inventory is available, constitutes a key contribution for the uncapacitated cases. Zangwill [2] extended the Wagner–Whitin model by allowing complete backlogging of unsatisfied demand. This was the first work to highlight the importance of using networks to represent some production planning problems. More specifically, the problem was represented as a minimum cost flow problem in a network with concave arc costs and a single source.

Although many alternative algorithms have been proposed, the dynamic programming method still remains the major analytical tool for solving lot-size problems. Federgruen and Tzur [3] presented a simple forward algorithm which solves the general dynamic lot-size model in $O(H \log H)$ time or in $O(H)$ under some assumptions on the cost data. This was the key improvement to the previously recommended well-known shortest path algorithm solution, which required $O(H^2)$ time.

Wagelmans et al. [4] extended the range of allowable cost data to permit coefficients with unrestricted signs. They developed an alternative algorithm to solve the resulting problem in $O(H \log H)$ time. Aggarwal and Park [5] developed an algorithm with complexity $O(H \log H)$, which solved the problem of H periods by solving two sub-problems of $H/2$ periods. Two recent review papers on the dynamic lot-size problem are those by Karimi et al. [6] and Jans and Degraeve [7]. The first one reviews single-level lot-size problems, their variants and solution approaches. The second one presents an overview of recent developments on the deterministic dynamic lot-size problem, focusing on the modeling of various industrial extensions rather than solution approaches.

All the above models assume that relevant data, such as the demand, are known and deterministic. However, this assumption is unrealistic in many situations. Guan and Miller [8] studied the stochastic version of the deterministic lot-size problem and proposed a polynomial time algorithm to obtain the optimal solution. Guan [9] studied a more general setting of the stochastic lot-size problem, assuming varying capacities and backlogging of unsatisfied demand.

Recently, Vargas [10] investigated the problem of planning dynamic order quantities, extending the Wagner–Whitin algorithm

* Corresponding author. Tel.: +30 2651008839; fax: +30 2651008890.
E-mail addresses: gpiperag@cs.uoi.gr (G.S. Piperagkas), ikonst@uom.gr (I. Konstantaras), kskouri@uoi.gr (K. Skouri), kostasp@cs.uoi.gr (K.E. Parsopoulos).

0305-0548/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2011.09.004

to the case of stochastic, time-varying demand with known density function. Safety stock requirements were implicitly included in planned order quantities whereby the objective was to minimize the sum of expected setup, backorder and inventory holding costs. A particularly elegant solution procedure was developed for the case of normally distributed periodic demands. In his analysis, Vargas [10] stated that his work

"...may serve as a basis for the development of improved production scheduling heuristics for the stochastic case and new heuristics can be directly incorporated in production scheduling systems."

Motivated by this proposal, we selected three heuristic optimization algorithms, still unstudied on the specific problem. The algorithms were selected on the basis of popularity, number of interdisciplinary applications, easy implementation and verified efficiency, which imply a prevailing position among similar approaches. Particle swarm optimization (PSO) [11], differential evolution (DE) [12] and harmony search (HS) [13] are three algorithms that perfectly matched our criteria. Although they were not the only candidate approaches, the ongoing interest of the research community on their properties and applications was the motive for our choice in the present study.

The engaged algorithms have been shown to be very efficient in various scientific and engineering problems. Their development was inspired by the evolution and self-organization properties of living entities. PSO roughly resembles the swarming behavior observed in bird-flocks or fish schools, which is also intimately related to physical laws that characterize more fundamental systems such as gases in particle physics. On the other hand, DE is closely associated with evolutionary algorithms, resembling recombination and mutation procedures. HS performs a procedure similar to the musical improvisation process, although its structure and operation have many commons with state-of-the-art evolutionary algorithms such as evolution strategies.

The aforementioned algorithms have been successfully applied also in operations research. For instance, we can refer to flow shop and machine scheduling problems [14–16], optimal scheduling of multiple dam systems and fluid-transport network design [17,18], inventory optimization [19], dynamic lot-sizing problems [20], etc.

The rest of the paper is organized as follows: in Section 2, the model is clarified and an analytical mathematical formulation is provided. Section 3 is devoted to the description of the employed algorithms. The experimental setup is presented in Section 4, along with the obtained results and, finally, the paper concludes in Section 5.

2. Problem formulation

The mathematical description of the stochastic lot-size problem is deployed in the following paragraphs, closely following the presentation of Vargas [10].

2.1. Assumptions and notation

The formal representation of the problem requires the following assumptions for the stochastic version of the Wagner–Whitin lot-size problem:

(a) The planning horizon is composed of H time periods.
(b) Demand in each period, t, is non-negative, independent and stochastic with known density.
(c) The production capacity is unlimited.
(d) Unsatisfied demand is fully backlogged and a backlogging cost is assessed at the end of each period per unit backlogged.
(e) A fixed lead time, L, is assumed and no disposal of inventory is allowed.
(f) At the conclusion of the horizon, holding or backlogging costs are assessed and any backlogged demand is left unfilled.
(g) The first scheduled production lot arrives at the start of period 1, and there are no pipeline production lots.

Additional notation used in the problem formulation, follows below:

d_t: demand in period $t=1,2,\ldots,H$, with known density function, $\phi_t(x)$
h_t: holding cost, assessed at the end of each period per unit held
b_t: backlogging cost, assessed at the end of each period per unit backlogged, assumed to be proportional to the holding cost, i.e., $b_t=ph_t$, for some $p>0$
A_t: fixed production setup for each time period
S_t: cumulative sum of all production lots to arrive up to and including period t (initially $S_0=0$)
$k(x)$: binary decision variable defined as

$$k(x)=\begin{cases}1, & x\neq 0,\\ 0, & x=0,\end{cases} \tag{1}$$

where x is the production volume in a time period.

Now we can give the general mathematical formulation of the problem, in the following section.

2.2. Existing formulation and solution approach

The expected cost incurred in period $t=1,2,\ldots,H$ is given by the following expression:

$$E_t(S_1,S_2,\ldots,S_t)=A_{t-L}k(S_t-S_{t-1})+h_t\int_0^{S_t}(S_t-q)f_t(q)\,dq + b_t\int_{S_t}^{\infty}(q-S_t)f_t(q)\,dq, \tag{2}$$

where $f_t(q)$ is the convolution of the demand density functions ϕ_i for periods $i=1-L$ to t. The corresponding optimization problem is to specify the cumulative production amounts, S_t, $t=1,2,\ldots,H$, that produce the minimum total expected setup, holding and backlogging costs [10], i.e.

$$\min_{0<S_1\le S_2\le\cdots\le S_H}\sum_{t=1}^{H}E_t(S_1,S_2,\ldots,S_t). \tag{3}$$

Actually, the problem is solved in two stages. At first the optimal replenishment quantities for any sequence of epochs is determined and, afterwards, the optimal sequence of replenishment epochs is identified.

More specifically, if S is the cumulative production quantity received up to period i, then the expected cost incurred in periods $i,i+1,\ldots,j-1$, is computed as

$$K(S,i,j)=A_{i-L}+\sum_{t=i}^{j-1}h_t\int_0^{S}(S-q)f_t(q)\,dq + \sum_{t=i}^{j-1}b_t\int_{S}^{\infty}(q-S)f_t(q)\,dq, \tag{4}$$

where i and j are periods in which a production lot is available, with

$$1\le i<j\le H+1, \tag{5}$$

while no replenishment occurs for $j=H+1$.

As stated in [10], Eq. (4) is differentiable and convex in S so that we obtain the optimal solution by setting its derivative equal

to zero. Consequently, if

$$F_t(x) = \int_0^x f_t(q)\, dq, \tag{6}$$

then

$$\sum_{t=i}^{j-1}(h_t + b_t)F_t(S) = \sum_{k=i}^{j-1} b_t, \tag{7}$$

which can be extended as [10]

$$\sum_{t=i}^{j-1}\left\{\frac{(h_t+b_t)}{\sum_{k=i}^{j-1}(h_k+b_k)}F_t(S)\right\} = \frac{\sum_{t=i}^{j-1} b_t}{\sum_{t=i}^{j-1}(h_t+b_t)} = \frac{p}{(1+p)}. \tag{8}$$

The first optimization stage is completed by computing the root, $S^*(i,j)$, of Eq. (8), which represents the optimal cumulative production amount to be received up to period i while no subsequent production lot is received before period j, and p is the parameter for which, $b_t = ph_t$ holds.

The second optimization stage consists of the computation of the optimal sequence of replenishment epochs, i.e., the sequence that gives the minimum value of the objective function defined in Eq. (3). This can be done by using different techniques and it constitutes the point of our interference in the solution process with the heuristic algorithms.

Exhaustive search is the most trivial algorithm for solving the problem and may even be effective in small problem instances. However, it becomes exponentially laborious with the number of time periods, since the number of all possible sequences becomes very large, requiring prohibitive computation time for their assessment. Nevertheless, it may still be useful for small problem instances.

Vargas [10] proposed a sophisticated technique that builds a tree-like structure and translates the problem to an equivalent one of finding the shortest path of the resulting acyclic network. The corresponding arcs of the network represent the options of replenishment occurring in period i with no subsequent replenishment until period j, and they are labeled with non-negative arc costs

$$C_{ij} = K(S^*(i,j),i,j), \quad 1 \le i < j \le H+1. \tag{9}$$

Further analysis on this technique can be found in [10].

In the current work, we propose the solution of the second optimization stage by using the heuristic optimization algorithms mentioned in the previous sections. Details of the proposed approaches are provided in the following sections, after a brief presentation of the employed algorithms.

3. Employed heuristic optimization algorithms

The main features of the employed algorithms, PSO, DE and HS, are presented in this section. In our presentation, we assume that the optimization problem under consideration is defined as

$$\min_{\mathbf{x} \in X \subset \mathbb{R}^n} G(\mathbf{x}), \tag{10}$$

where X is the search space. No assumptions on the objective function $G(\mathbf{x})$ are required, except the availability of its value at any give point in X.

3.1. Particle swarm optimization

The original PSO algorithm was introduced in 1995 by Eberhart and Kennedy [11]. The method utilizes a *swarm*, i.e., a population of search points that iteratively probe the search space for the global minimizer. The search points are called *particles*, and they concurrently move with an adaptable velocity (position shift) to new positions.

Moreover, each particle has a *memory* where it retains the best position it has ever visited, i.e., the position with the lowest function value. This can be considered as experience storage for the particle, which exploits this information to guide its search towards the most promising regions of the search space. The search stops as soon as a stopping criterion is achieved, usually related to the quality of the best solution found so far or to the number of function evaluations spent by the algorithm.

Apart from the personal memory, each particle has a neighborhood, i.e., a set of indices of other particles that shares their memories (best positions) with it. Thus, the particle decides on its next move by aggregating its own discoveries with the best findings of its neighboring mates.

Based on the concept of neighborhood, two main PSO schemes were proposed. The first one is called the *global* PSO (also known as *gbest*) model on account of the underlying global information-sharing scheme. According to this, all particles assume the whole swarm as their neighborhood and each particle takes into consideration its own memory as well as the overall best memory, i.e., the best position ever discovered by the whole swarm.

The second model is the *local* PSO (also known as *lbest*), where each particle is assigned a neighborhood usually consisting of a few particles. The organization of particles in neighborhoods rises the concept of *neighborhood topologies*, which refers to abstract representations of information-flow channels among the particles. Usually, the topologies are represented as graphs consisting of nodes (particles) and interconnections (communication channels) among them. The most common topology is the *ring*, where all particles are assumed to lie on a ring ordered according to their indices, such that each particle has two immediate neighbors with adjacent indices. Then, for a given particle, its neighborhood is completely defined by determining a *radius*, i.e., the number of particles with adjacent indices that constitute the neighborhood. More information on the effect of neighborhoods can be gained in [21,22]. Fig. 1 illustrates the aforementioned ring topology (left) as well as another popular scheme called *star* (right).

In general, the search procedure of heuristic algorithms such as PSO consists of two phases, namely *exploration* and *exploitation*. In the first, the swarm attempts to detect the most promising regions of the search space. In the latter, it promotes the faster convergence to the most promising regions detected so far. It has been experimentally verified that the neighborhood topology can influence the swarm's convergence dynamic. As can be easily inferred, the gbest model promotes the search around the overall best position in favor of exploitation. On the other hand, the lbest model with its regional and gradual information transmission between particles, leans effectively to exploration.

To put it formally, let

$$S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \tag{11}$$

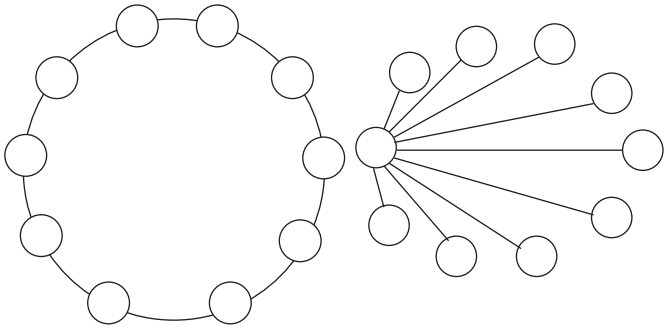

Fig. 1. The ring (left) and star (right) neighborhood topologies of PSO.

be a swarm of N particles, $\mathbf{x}_i \in X \subset \mathbb{R}^n$, $i=1,2,\ldots,N$. The i-th particle has a velocity (position shift), $\mathbf{v}_i$, and retains in memory the best position, $\mathbf{p}_i \in X$, it has ever visited. A ring neighborhood of radius m for the particle $\mathbf{x}_i$, is defined as the set of indices

$$B_i = \{i-m, i-m+1, \ldots, i, \ldots, i+m-1, i+m\}. \tag{12}$$

The ring is assumed to recycle at its end, i.e., the particles $\mathbf{x}_N$ and $\mathbf{x}_2$ are the immediate neighbors of $\mathbf{x}_1$.

Assume that g_i is the index of the best position found so far in the neighborhood of $\mathbf{x}_i$, i.e.

$$g_i = \arg\min_{j \in B_i} G(\mathbf{p}_j), \tag{13}$$

and let t denote the iteration counter. Then, according to the *constriction coefficient* version of PSO [23], the swarm is updated as follows:

$$v_{ij}^{(t+1)} = \chi[v_{ij}^{(t)} + \varphi_1(p_{ij}^{(t)} - x_{ij}^{(t)}) + \varphi_2(p_{gj}^{(t)} - x_{ij}^{(t)})], \tag{14}$$

$$x_{ij}^{(t+1)} = x_{ij}^{(t)} + v_{ij}^{(t+1)}, \tag{15}$$

where $i=1,2,\ldots,N$, and $j=1,2,\ldots,n$. The parameter χ is the constriction coefficient and it is used as a means to control the magnitude of the velocities. The other two parameters are defined as

$$\varphi_1 = c_1 r_1, \quad \varphi_2 = c_2 r_2, \tag{16}$$

where c_1 and c_2 are positive constants, also called the *cognitive* and the *social* parameters, respectively, and r_1, r_2, are random variables uniformly distributed in [0,1], assuming different values for each i, j and t.

The constriction coefficient is needed to restrict the magnitude of the velocities, promoting convergence and alleviating the "swarm explosion" effect that has been shown to be detrimental for the search procedure [23]. In early PSO versions, the parameters were empirically determined based on trial runs. In more recent versions, the PSO stability analyses by Clerc and Kennedy [23] and Trelea [24] imply that parameters are selected such that the following equation holds:

$$\chi = \frac{2}{|2-\varphi-\sqrt{\varphi^2-4\varphi}|}, \tag{17}$$

where $\varphi > 4$ and $\varphi = c_1 + c_2$. Following this result, the values $\chi = 0.729$, $c_1 = c_2 = 2.05$ are considered as the default parameter set.

A full iteration of PSO is completed with the best positions update

$$\mathbf{p}_i^{(t+1)} = \begin{cases} \mathbf{x}_i^{(t+1)} & \text{if } G(\mathbf{x}_i^{(t+1)}) < G(\mathbf{p}_i^{(t)}), \\ \mathbf{p}_i^{(t)} & \text{otherwise.} \end{cases} \tag{18}$$

PSO was primarily designed to operate in continuous search spaces. However, experimental evidence has shown signs of efficiency also in integer and mixed-integer problems, without the need of radical modifications in the algorithm [15,25]. The most straightforward approach to achieve this is the use of its standard form with the particles being rounded to the closest integer prior to each function evaluation, while their position updates are performed in the continuous domain. This is the approach adopted also in the present paper. A typical example is provided in a later section.

3.2. Differential evolution

The DE algorithm was introduced by Storn and Price [12] as a population-based, stochastic optimization algorithm for numerical optimization problems. Similar to PSO, DE employs a population, $S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, of individuals to probe the search space. The population is randomly initialized, usually following a uniform distribution over the search space. At each iteration, N competitions are held to determine the members of the population for the next iteration. This is achieved by iteratively applying three operations on each individual: *mutation*, *crossover* and *selection*.

The mutation operator produces a new vector, $\mathbf{v}_i$, for each individual, $\mathbf{x}_i$, $i=1,2,\ldots,N$, by combining it with some of the rest. Different operators have been proposed for this task. The following five operators are among the most common DE mutation schemes:

$$\text{DE1}: \ \mathbf{v}_i^{(t+1)} = \mathbf{x}_g^{(t)} + F(\mathbf{x}_{r_1}^{(t)} - \mathbf{x}_{r_2}^{(t)}), \tag{19}$$

$$\text{DE2}: \ \mathbf{v}_i^{(t+1)} = \mathbf{x}_{r_1}^{(t)} + F(\mathbf{x}_{r_2}^{(t)} - \mathbf{x}_{r_3}^{(t)}), \tag{20}$$

$$\text{DE3}: \ \mathbf{v}_i^{(t+1)} = \mathbf{x}_i^{(t)} + F(\mathbf{x}_g^{(t)} - \mathbf{x}_i^{(t)} + \mathbf{x}_{r_1}^{(t)} - \mathbf{x}_{r_2}^{(t)}), \tag{21}$$

$$\text{DE4}: \ \mathbf{v}_i^{(t+1)} = \mathbf{x}_g^{(t)} + F(\mathbf{x}_{r_1}^{(t)} - \mathbf{x}_{r_2}^{(t)} + \mathbf{x}_{r_3}^{(t)} - \mathbf{x}_{r_4}^{(t)}), \tag{22}$$

$$\text{DE5}: \ \mathbf{v}_i^{(t+1)} = \mathbf{x}_{r_1}^{(t)} + F(\mathbf{x}_{r_2}^{(t)} - \mathbf{x}_{r_3}^{(t)} + \mathbf{x}_{r_4}^{(t)} - \mathbf{x}_{r_5}^{(t)}), \tag{23}$$

where t denotes the iteration counter; F is a fixed user-defined parameter; g denotes the index of the best individual in the population, i.e.

$$g = \arg\min_{j=1,\ldots,N} G(\mathbf{x}_j), \tag{24}$$

and $r_i \in \{1,2,\ldots,N\}$, $i=1,2,\ldots,5$ are mutually different, randomly selected indices that differ also from the index i. Obviously, in order to enable all mutation operators, it must hold that $N > 5$. All vector operations in Eqs. (19)–(23) are performed componentwise.

After mutation, the recombination operator is applied on the generated vector, $\mathbf{v}_i$, producing a *trial vector*

$$\mathbf{u}_i = (u_{i1}, u_{i2}, \ldots, u_{in}), \tag{25}$$

which is defined as follows:

$$u_{ij}^{(t+1)} = \begin{cases} v_{ij}^{(t+1)} & \text{if } R_j \le CR \quad \text{or} \quad j = RI(i), \\ x_{ij}^{(t)} & \text{otherwise,} \end{cases} \tag{26}$$

where $j=1,2,\ldots,n$; R_j is the j-th evaluation of a uniform random number generator in the range [0,1]; $CR \in [0,1]$ is a user-defined crossover constant; and $RI(i)$ is an index randomly selected from the set $\{1,2,\ldots,n\}$.

Finally, the produced trial vector, $\mathbf{u}_i$, is compared against the corresponding individual and the best between them is included in the population of the next generation, i.e.,

$$\mathbf{x}_i^{(t+1)} = \begin{cases} \mathbf{u}_i^{(t+1)} & \text{if } G(\mathbf{u}_i^{(t+1)}) < G(\mathbf{x}_i^{(t)}), \\ \mathbf{x}_i^{(t)} & \text{otherwise.} \end{cases} \tag{27}$$

Apparently, DE does not require a separate memory as PSO, since it operates directly on the best solutions found so far (the corresponding best positions in PSO). This renders DE a greedier algorithm than PSO. Also, there is no sound theoretical evidence on the proper parameter setting of the algorithm. Several different settings have been used in the literature [26] but their performance appears to be strongly dependent on the operator and problem at hand. Nevertheless, by their nature, the mutation operators that involve the best individual of the population are expected to be more exploitation-oriented than those that use randomly selected individuals.

3.3. Harmony search

HS was proposed by Geem et al. in 2001 [13]. Inspiration was drawn from musical performance processes that occur when a musician searches for a better state of harmony, improvising

the instrument pitches towards a better aesthetic outcome. In a similar manner, the algorithm seeks for the global optimum, maintaining and iteratively updating a memory, called the *harmony memory* (HM), of candidate solutions (harmonies). HM contains the best harmonies considered so far, i.e., candidate solutions with the smallest objective function values.

HM consists of N randomly initialized vectors (memory size is user-defined) along with their function values [27]

$$HM = \left[\begin{array}{ccc|c} x_{11} & \cdots & x_{1n} & G(\mathbf{x}_1) \\ x_{21} & \cdots & x_{2n} & G(\mathbf{x}_2) \\ \vdots & \ddots & \vdots & \vdots \\ x_{N1} & \cdots & x_{Nn} & G(\mathbf{x}_N) \end{array}\right] \tag{28}$$

The harmonies in HM are ordered in ascending order with respect to their values, i.e.

$$G(\mathbf{x}_1) \le G(\mathbf{x}_2) \le \cdots \le G(\mathbf{x}_N). \tag{29}$$

Also, two additional parameters must be defined. The first one is the *harmony memory considering rate* (*HMCR*), which stands for the probability of selecting a vector component value among those already stored in HM. The second parameter is called the *pitch adjusting rate* (*PAR*) and it defines the mutation probability of a selected value from HM. The role of these parameters is clarified below.

The algorithm works iteratively, exploiting the stored information in HM for the production of one or many new solutions at each iteration. The new solutions are built component by component, selecting at each step either a stored component or a random value. More specifically, let $\mathbf{x} = (x_1, x_2, \ldots, x_n)^\top$ be a new solution to be built, with components

$$x_j \in X_j, \quad j = 1, 2, \ldots, n, \tag{30}$$

where $X_j \subset \mathbb{R}$ is the subspace of the search space X that corresponds to the j-th component. Then, x_j is probabilistically selected according to the scheme

$$x_j = \begin{cases} x_{sj} \in \{x_{1j}, x_{2j}, \ldots, x_{Nj}\} & \text{if } r_j \le HMCR, \\ y \in X_j & \text{otherwise,} \end{cases} \tag{31}$$

where r_j is a random variable uniformly distributed in $[0, 1]$; s is an index selected from the set $\{1, 2, \ldots, N\}$; and y is a random value in X_j.

The selection of s is probabilistic and can be either uniform over the set of indices or it can be a linear ranking selection scheme [28] of the stored harmonies in HM. In the latter, high selective pressure can impose strong selection bias towards the indices of the best solutions, in contrast to uniform selection which assigns equal selection probabilities to all indices. Consequently, uniform selection is expected to be more diversity-preserving in the produced harmonies.

After the construction of the new solution, $\mathbf{x} = (x_1, x_2, \ldots, x_n)^\top$, each component is mutated as follows:

$$x_j = \begin{cases} x_j + qw & \text{if } R_j \le PAR, \\ x_j & \text{otherwise,} \end{cases} \tag{32}$$

where R_j is a uniformly distributed random variable in [0,1]; q is a uniformly distributed random variable in $[-1, 1]$; and w is a user-defined mutation magnitude.

The aforementioned operations can be repeated to produce a number, M_{prod}, of new harmonies. Each new harmony is evaluated with the objective function, and the best $M_{\text{rep}} \le M_{\text{prod}}$ of them replace the worst M_{rep} harmonies stored in HM, if they improve them. Both parameters M_{rep} and M_{prod} are user defined. In the simplest case, $M_{\text{rep}} = M_{\text{prod}} = 1$ is used.

In contrast to PSO and DE, the HS operators with proper parameter setting can directly handle integer and mixed-integer problems without any modification. Also, we shall notice that HS has shown many structural similarities with the established evolution strategies (ES) approaches [29]. In fact, HS can be considered as an alternative ES variant, although with a different motivation and inspiration source. In view of this similarities, it is anticipated that the performance of HS can provide evidence also on the performance of standard ES variants.

4. Experimental settings and results

In this section, we expose the exact settings used in our experiments as well as the obtained results, followed by the corresponding discussion.

4.1. Solution representation

As mentioned in Section 2, a production schedule for the entire planning horizon that minimizes the total cost defined in Eq. (3) can be determined through a two-stage optimization procedure. In the first stage, the optimal replenishment quantities for any sequence of replenishment epochs are analytically computed using Eq. (8) [10].

The second stage consists of a binary optimization problem that aims at identifying the optimal sequence of replenishment epochs. For each epoch, a decision of placing an order (corresponding to 1) or not (corresponding to 0) must be made. If the decision is to place an order then the optimal quantity is already known from the first optimization stage and it is directly used for the computation of the total cost.

For a problem of H epochs, the employed algorithms need to work on the n-dimensional binary space $X = \{0, 1\}^n$ with $n = H$. However, as we already mentioned, the employed algorithms are primarily destined to work on real variables (with the exception of HS, which can alleviate this problem). For this reason, the tried-and-true rounding technique was adopted in the present study. More specifically, the algorithms were let to operate on the real search space $X = [0, 1]^n$ but, whenever the function evaluation of a particle (or individual) was required, its components were rounded to the nearest integer (0 or 1) without substituting the real components with the integers in the vectors. For example, the candidate solution

$$\mathbf{x} = (0.31,\ 0.74,\ 0.56,\ 0.91,\ 0.22)^\top \tag{33}$$

would be mapped to the binary vector

$$\overline{\mathbf{x}} = (0,\ 1,\ 1,\ 1,\ 0)^\top \tag{34}$$

and

$$G(\mathbf{x}) = G(\overline{\mathbf{x}}). \tag{35}$$

In contrast to other approaches that introduce new operators to tackle integer variables at the cost of radically changing the algorithms' operation and performance, this approach imposes only minimal intervention in the algorithms' dynamics. Only for the case of HS, the real components of the harmonies were replaced by their nearest integers both in HM and in the new harmonies produced by mutation.

4.2. The case of normally distributed demand

Without restricting the applicability of the considered algorithms, stochastic demand was assumed to follow a normal distribution in our study, in accordance to the case study reported in [10]. Under this assumption, Eqs. (4) and (8) can be simplified to avoid multiple numerical integrations for determining the optimal cumulative production quantities.

In this manner, two functions are involved: *cumulative normal distribution* and the *standard normal loss integral*. Let σ_t and μ_t denote the mean and standard deviation of cumulative demand in period t, with density function $f_t(x)$. Let also $\phi(x)$ denotes the standard normal density function with cumulative distribution function $\Phi(x)$, and let

$$I_N(x) = \int_{z=x}^{\infty} (z-x)\phi(z)\,dz \tag{36}$$

be the normal loss integral. Then, Eq. (4) is simplified as follows:

$$K(S,i,j) = A_{i-L} + \sum_{t=i}^{j-1} h_t \sigma_t [z_t + (1+p) I_N(z_t)], \tag{37}$$

where

$$z_t = \frac{S-\mu_t}{\sigma_t}. \tag{38}$$

A proof of Eq. (37) is extensively presented in [10].

4.3. Test problems

Our aim in the present study was the investigation of the employed algorithms on various instances of the problem with respect to its dimension. The main interest behind this is the scaling of the run-time required to find the optimal sequence of replenishment epochs. As we already mentioned in previous sections, the exhaustive inspection of all possible combinations requires exponentially increasing time with the problem dimension (number of epochs), which becomes prohibitive even for modern computer systems. Therefore, efficiency with respect to both solution accuracy and run-time was in the center of our investigation.

For consistency, the test problems used in our experiments were based on the 12-dimensional problem presented in [10]. The data provided in this source includes the setup cost and cumulative demand for 12 epochs. We used this data and extended them for up to 48 epochs. For this purpose, we fitted a Gaussian distribution on the provided data and generated new setup cost and cumulative demand values by sampling the fitted distribution. The obtained values are reported in Table 1.

We considered the problem for dimensions 12, 18, 24, 30, 36, 42 and 48. The optimal sequence of the replenishment epochs for each instance was initially determined through exhaustive search. The total number of binary sequences per case as well as the required run-time[1] for their evaluation are reported in Table 2. As we can see, problem instances with more than 36 epochs need excessive computation time due to the huge number of sequences, which becomes larger than 10^{10}.

It must be emphasized that the actual number of binary variables for a problem instance of H epochs is equal to $H-1$, due to the model assumption that there is always an order decision in the first epoch [10], which implies that the corresponding binary variable is always fixed to 1. Thus, a problem with H epochs corresponds to 2^{H-1} binary sequences.

4.4. Experimental setup

We performed extensive experiments with the three employed algorithms under different parameter settings and variants. PSO was considered in both its global and local variant with ring neighborhoods of radius 1 and the default parameter set given in Section 3.1. DE was considered in its five basic operators and all possible combinations of its parameters, $F,CR \in \{0.3, 0.5, 0.7\}$. HS was considered under various harmony memory sizes and for both the uniform and the linear ranking selection schemes. Preliminary experiments provided clear evidence that for harmony memory size equal to N, the values $M_{prod} = N/2$ and $M_{rep} = N/5$ for the produced and replaced new harmonies (see Section 3.3), respectively, as well as $HMCR = 0.9$ and $PAR = 0.3$, constitute appropriate choices.

The swarm size in PSO (equivalently the population size in DE and harmony memory size in HS) was set to $10 \times n$ for all problem instances, where n is the corresponding problem dimension. For each algorithm, 100 independent experiments were performed per problem instance. The stopping condition was the determination of the optimal sequence of replenishment epochs within a prescribed maximum number of function evaluations. For the smallest problem instances, this number was equal to the total number of sequences. For larger instances, it was limited to the value $T_{max} = 5 \times 10^6$ as reported in Table 3.

Besides the employed heuristic optimization algorithms, the COMSOAL [30] approach was also applied on the problem under

Table 1
Setup cost and cumulative demand ($f_t(q)$) used in the test problems.

Time period	Setup cost	Cumulative demand	
		Mean	StD
1	85	69	7.7
2	102	98	8.3
3	102	134	9.2
4	101	195	11.4
5	98	256	13.3
6	114	282	13.6
7	105	316	14.1
8	86	383	16.0
9	119	428	16.7
10	110	495	18.3
11	98	574	20.3
12	114	630	21.2
13	108	657	25.1
14	122	718	25.6
15	79	767	26.8
16	111	816	27.7
17	106	894	28.6
18	89	952	29.9
19	98	1008	30.9
20	106	1089	31.7
21	140	1127	33.0
22	132	1192	33.6
23	89	1259	34.5
24	135	1307	35.5
25	110	1363	36.2
26	102	1395	36.9
27	110	1427	37.3
28	101	1481	37.8
29	102	1546	38.5
30	119	1644	39.3
31	118	1685	40.5
32	118	1741	41.0
33	110	1792	41.7
34	90	1810	42.3
35	110	1855	42.5
36	120	1876	43.1
37	108	1943	43.3
38	114	1980	44.1
39	110	2034	44.5
40	100	2077	45.1
41	106	2135	45.6
42	95	2177	46.2
43	112	2238	46.6
44	91	2304	47.3
45	92	2387	48.0
46	94	2436	48.9
47	72	2450	49.4
48	118	2488	49.5

[1] The time refers to an Intel I7 (c) machine with 8 GB of memory.

Table 2
Total number of sequences and run-time required for the exhaustive search per problem instance. The symbol "$\sim$" stands for "order of" and "$>$" denotes "higher than".

Dim.	Sequences	Time
12	2048	0.1 s
18	131 072	0.5 s
24	$\sim 10^6$	~ 5 s
30	$\sim 10^8$	~ 312 s
36	$> 10^{10}$	> 7 h
42	$> 10^{10}$	> 2 days
48	$> 10^{10}$	> 5 days

Table 3
Maximum function evaluations (T_{max}) and swarm/population/harmony memory size (N) per problem instance.

Dim.	T_{max}	N
12	2048	120
18	131 072	180
24	5×10^6	240
30	5×10^6	300
36	5×10^6	360
42	5×10^6	420
48	5×10^6	480

consideration, with the same computational resources as for the rest of the algorithms. COMSOAL is a randomized algorithm for sequencing operations in assembly lines [31–33]. Its operation is based on the generation of sequences by randomly picking a task and constructing subsequent tasks, while concurrently keeping a record of the best detected sequence. If during its construction a sequence is proved to be worst than the best one (i.e., it corresponds to a higher cost) then it is abandoned and another sequence is built. Given the binary nature of the problem at hand, such a naive algorithm could prove to be promising with a minimal effort.

All algorithms were extensively tested. The total number of independent experiments for all problem instances was higher than 26 000. For each algorithm, variant, parameter set and problem instance, we recorded the success rate, i.e., the number of experiments (out of 100) where it succeeded to reach the optimal solution within the maximum number of function evaluations. Also, the mean, standard deviation, minimum and maximum values of the expected number of function evaluations, as well as the average required run-time (in s) were recorded per algorithm and problem instance.

4.5. Presentation of results and discussion

In view of the huge amount of the obtained results, it was necessary to make a selection of only the most interesting cases to report in our presentation. For this reason, we identified the most promising variant of each algorithm. For PSO, the lbest model was far the most successful variant. The gbest model was prone to get stuck in suboptimal solutions, even for the low-dimensional problem instances. This can be attributed to its exploitation orientation in combination with the rounding scheme. Very often, the particles in the gbest model were rapidly clustered in very small ranges around the best solutions, also assuming very small velocities. This effect, combined with the fact that components in the range [0,0.5] were mapped to 0 while components in (0.5,1] were mapped to 1, offers a reasonable explanation for the low efficiency of gbest PSO. On the other hand, lbest PSO is clearly more exploration-oriented than gbest. Thus, the particles were able to retain sufficiently higher velocities that allowed them to move from the one-half of the search space to the other, thereby exploring a higher number of binary sequences.

Regarding the DE algorithm, two operators were clearly distinguished among the five presented in Section 3.2, namely DE2 and DE5, while the most successful parameter values were $F=0.7$ and $CR=0.3$. A closer look at these two operators reveals that they both use only randomly selected individuals from the population, in contrast to the rest operators that exploit the best individual. This can be interpreted as an evidence that, on average, operators with higher diversity-preserving properties are related to the best observed performance. This is aligned with our observations reported above for the PSO algorithm.

Finally, for the HS algorithm, the uniform selection scheme was more efficient than the linear ranking scheme. It must be underlined that, since uniform selection represents the most fair scheme (equal selection probability for all vectors stored in memory), the linear ranking scheme was assigned high selective pressure, representing the completely biased selection towards the best harmonies. Obviously, uniform selection is more diversity-preserving than the linear ranking. Hence, its superiority aligns with the observations for the previous two algorithms, i.e., diversity-preserving variants are more successful in the specific problem.

In fact, this is the first interesting conclusion of the present work and it can be attributed to the nature of the binary optimization problem, with local minimizers that differ slightly (in one or two components) from the global one while their function values differ less than 0.6% from the global minimum.

Table 4 reports the detailed results for the aforementioned most successful algorithmic variants. More specifically, for each algorithm and problem instance, the success rate (successful experiments out of 100), mean, standard deviation, minimum, and maximum values of the required function evaluations for the successful experiments, as well as the required run-time (in s) are reported. The column that corresponds to the best algorithm is boldfaced. As best algorithm, we considered the one that primarily had the largest success rate and the smallest mean, and secondarily the smallest run-time. Although COMSOAL was inferior than most of the population-based approaches, it is included in Table 4 for comparison purposes (although excluded from the figures).

In addition to Table 4, the results are also graphically illustrated in Figs. 2–10 to provide intuitive evidence of their performance and facilitate visual comparisons among them. Fig. 2 illustrates the success rate per algorithm, with different colors denoting the different problem instances in ascending dimension order. In Figs. 3–9 boxplots are used to illustrate the distribution of the number of function evaluations required for each algorithm in the 100 independent experiments. On each box, the central mark is the median, the edges of the box are the 25-th and 75-th percentiles, the whiskers extend to the most extreme values, and the outliers are plotted individually (denoted with crosses). The notches define comparison intervals between medians. Two medians are significantly different at the 5% level if the corresponding intervals do not overlap. The interval endpoints are the extremes of the notches. Finally, Fig. 10 illustrates the scaling of the required run-time per algorithm as dimension increases.

A first inspection of Table 4 provides some immediate conclusions. Firstly, there is an undoubtful superiority of the DE variants over the rest of the algorithms. Evidently, DE2 is the overall best performing variant, followed by DE5, PSO and HS, while COMSOAL has a clearly declining performance as problem dimension increases. Secondly, the performance differences among them exhibit an increasing pattern with the problem's dimension. As we

Table 4
Results for all algorithms and problem instances.

Dim.	Stat.	PSO	HS	DE2	DE5	COM
12	Succ.	86	99	100	**100**	63
	Mean	911.16	556.36	823.20	**778.80**	900.60
	StD	454.33	291.81	363.19	**373.54**	627.22
	Min	240	180	240	**240**	3
	Max	2040	1740	1800	**1800**	2028
	Time	0.00023	0.00000	0.00000	**0.00010**	0.11000
18	Succ.	100	100	**100**	100	71
	Mean	7524.00	10 693.80	**3556.80**	3997.80	58 300.61
	StD	6736.47	9927.87	**1277.40**	1417.39	40 082.74
	Min	540	540	**360**	720	740
	Max	46 440	53 730	**7560**	8100	131 004
	Time	0.007	0.014	**0.005**	0.007	1.587
24	Succ.	100	100	**100**	100	46
	Mean	20 846.40	29 565.60	**10 022.40**	11 714.40	2 196 825.00
	StD	16 223.65	23 897.46	**2440.96**	3581.99	1 218 299.43
	Min	4080	1920	**3840**	3600	87 747
	Max	92 160	126 600	**16 560**	20 880	4 489 500
	Time	0.026	0.063	**0.036**	0.053	70.945
30	Succ.	100	100	**100**	100	1
	Mean	61 653.00	63 703.50	**25 302.00**	31 272.00	1 479 775.00
	StD	92 675.79	33 648.22	**6189.35**	8393.15	0.00
	Min	8100	7200	**10 800**	4800	1 479 775
	Max	899 700	219 600	**37 800**	52 500	1 479 775
	Time	0.119	0.191	**0.123**	0.152	251.590
36	Succ.	100	99	**100**	100	0
	Mean	122 878.80	198 676.36	**41 648.40**	56 170.80	–
	StD	79 122.15	159 591.75	**8280.73**	11 729.42	–
	Min	11 160	21 780	**19 440**	29 880	–
	Max	383 040	801 360	**63 000**	84 960	–
	Time	0.274	0.756	**0.245**	0.354	–
42	Succ.	100	99	**100**	100	0
	Mean	277 708.20	607 986.06	**74 991.00**	103 286.40	–
	StD	198 676.70	308 601.75	**14 390.06**	21 287.47	–
	Min	16 380	28 560	**44 520**	53 760	–
	Max	1 086 120	1 339 380	**106 680**	156 660	–
	Time	0.756	2.772	**0.562**	0.735	–
48	Succ.	100	88	**100**	100	0
	Mean	607 920.00	1 199 640.00	**130 032.00**	203 716.80	–
	StD	387 796.66	452 624.87	**21 837.47**	36 113.55	–
	Min	63 360	105 360	**72 480**	98 880	–
	Max	1 770 240	2 167 440	**189 120**	291 360	–
	Time	1.913	6.611	**1.149**	1.835	–

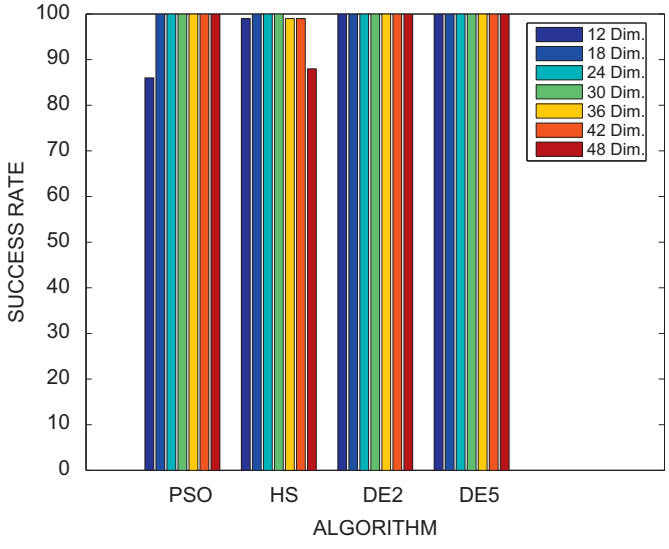

Fig. 2. Success rate for each algorithm. Different colors denote the different problem instances. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

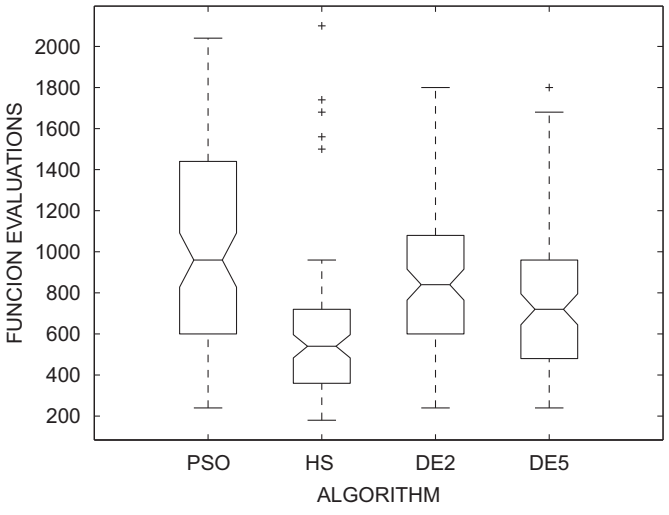

Fig. 3. Boxplots of the mean number of function evaluations required for each algorithm in 100 independent experiments for the 12-dimensional problem instance.

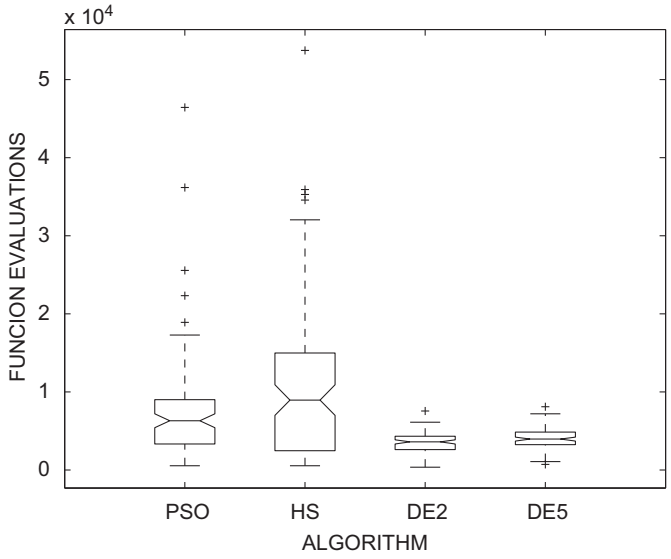

Fig. 4. Boxplots of the mean number of function evaluations required for each algorithm in 100 independent experiments for the 18-dimensional problem instance.

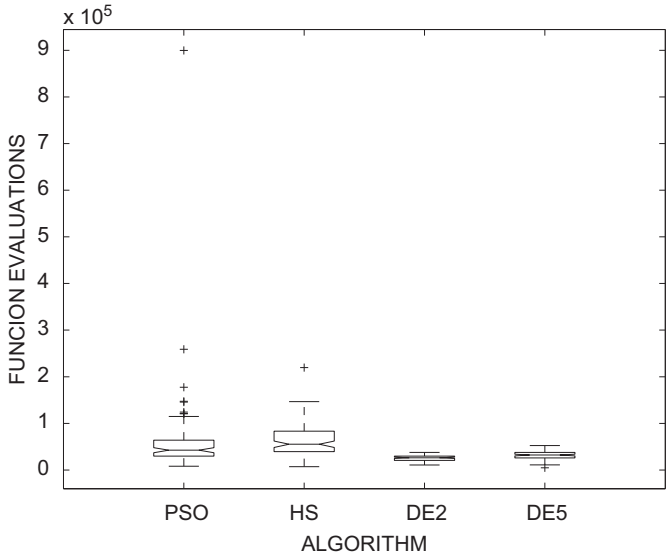

Fig. 6. Boxplots of the mean number of function evaluations required for each algorithm in 100 independent experiments for the 30-dimensional problem instance.

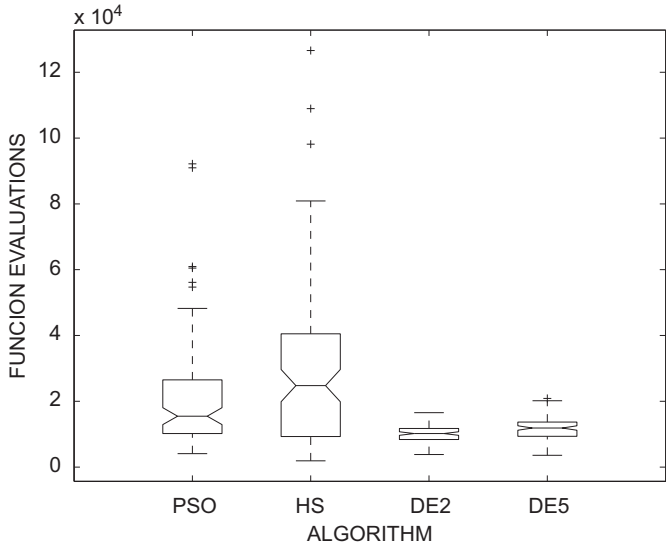

Fig. 5. Boxplots of the mean number of function evaluations required for each algorithm in 100 independent experiments for the 24-dimensional problem instance.

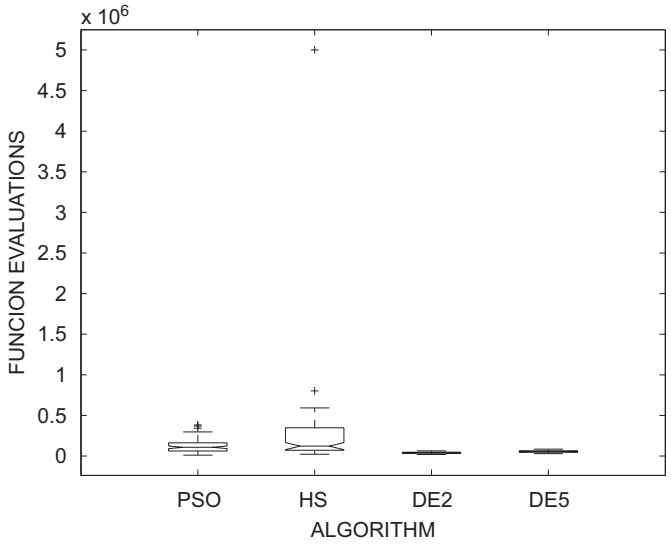

Fig. 7. Boxplots of the mean number of function evaluations required for each algorithm in 100 independent experiments for the 36-dimensional problem instance.

observe, even DE5 grows an exponentially increasing difference with DE2, although it has a slightly better mean in the 12-dimensional case. However, the latter problem instance needs special attention. A closer inspection of the reported data reveals that, despite the slightly lower mean of DE5, DE2 has slightly smaller standard deviation, which may suggest statistical insignificance between them. This is also visible in Fig. 3 with the overlapping comparison intervals between their medians in the boxplots. Also, we can observe that HS had also very satisfactory performance. In fact, it had the smallest mean in the successful experiments but with a slightly worse success rate, which is the reason for not considering it as the best algorithm in the 12-dimensional case. The same question as previously for the DE variants rises also here, i.e., how (statistically) crucial is the observed difference.

In order to answer this question, we performed a statistical significance test for each pair of algorithms. For this purpose, the two-sided rank-sum test of the hypothesis that the performance samples of each pair of algorithms come from distributions with equal medians was used. Each pair was tested against the null hypothesis that the samples have the same median in a 95% level of significance. The outcome of the tests is reported in Table 5, where the existence of statistical significance is denoted with the symbol "∗" and the lack is denoted with the symbol "–".

The statistical tests revealed that, as anticipated, the two DE algorithms had essentially the same performance in the 12-dimensional problem. Also, the visual evidence from Fig. 3, which suggests that HS has noteworthy performance in the specific problem instance, is verified by the statistical significance of HS against the rest of the algorithms. However, the performance of HS exhibits a rapid decline as dimension increases. The picture becomes clearer in higher-dimensional instances, where all algorithms are statistically different, with a single exception between PSO and HS in the 18-dimensional case, which can be attributed to their large standard deviations.

Regarding the running time illustrated in Fig. 10, we observe an anticipated superiority of the computationally cheapest approach,

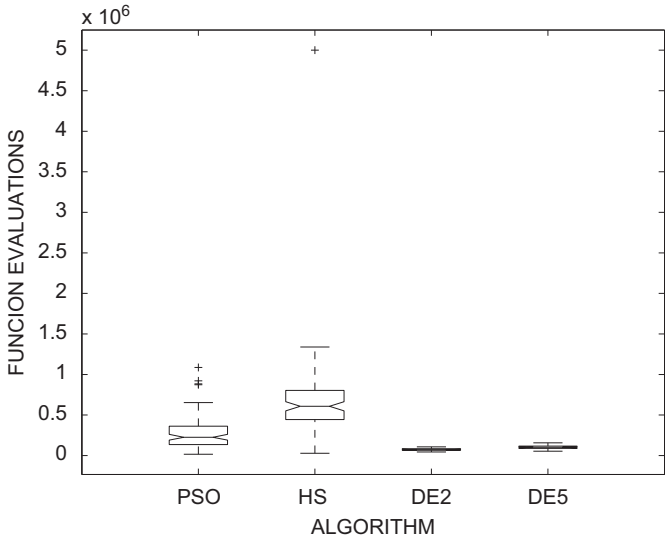

Fig. 8. Boxplots of the mean number of function evaluations required for each algorithm in 100 independent experiments for the 42-dimensional problem instance.

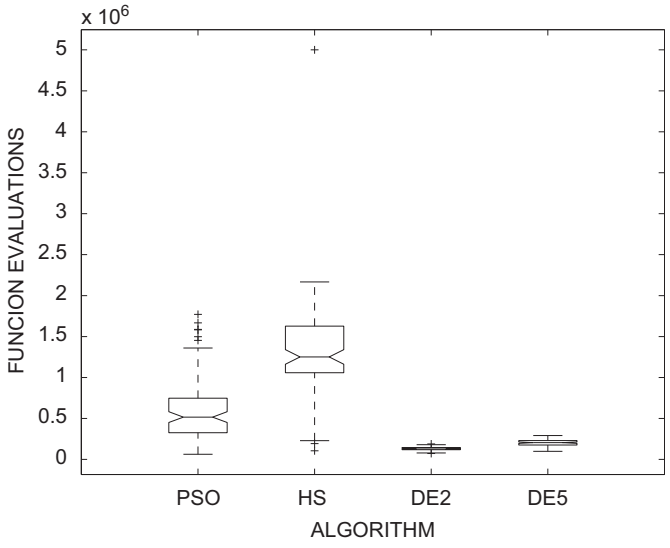

Fig. 9. Boxplots of the mean number of function evaluations required for each algorithm in 100 independent experiments for the 48-dimensional problem instance.

i.e., the DE2 variant, while PSO and DE5 had virtually the same run-time requirements. HS remained the more computationally demanding algorithm, evidently due to the excessive required number of function evaluations and its reduced success rate in the higher-dimensional cases.

As a closing remark, we must underline that notwithstanding their differences, the employed heuristic algorithms offered immense improvement against the exhaustive search, which is the trivial baseline for addressing such problems.

5. Conclusion

The Wagner–Whitin dynamic lot-size problem has been widely studied in the literature. The original deterministic model recently has been extended by considering stochastic demand. This was the main problem tackled in the present paper. Our approach was based on PSO, DE and HS, three established algorithms with an ongoing increasing popularity in research community. Proper modifications were introduced in the algorithms to address the most controversial part of problem, which consists of a binary optimization task. Special attention was paid to avoid radical modifications of the algorithms' dynamics.

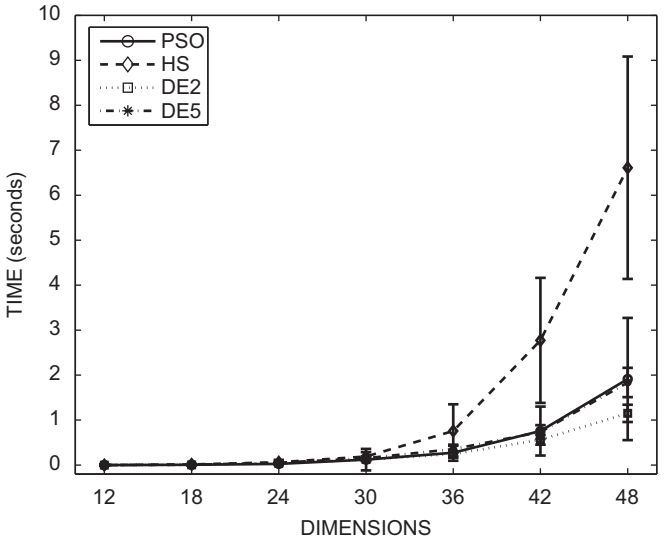

Fig. 10. The required running time per algorithm and problem instance.

Table 5
Wilcoxon rank-sum tests between the algorithms.

Dim.		PSO	HS	DE2	DE5	COM
12	PSO	–	*	*	*	*
	HS		–	*	*	*
	DE2			–	–	*
	DE5				–	*
	COM					–
18	PSO	–	–	*	*	*
	HS		–	*	*	*
	DE2			–	*	*
	DE5				–	*
	COM					–
24	PSO	–	*	*	*	*
	HS		–	*	*	*
	DE2			–	*	*
	DE5				–	*
	COM					–
30	PSO	–	*	*	*	*
	HS		–	*	*	*
	DE2			–	*	*
	DE5				–	*
	COM					–
36	PSO	–	*	*	*	*
	HS		–	*	*	*
	DE2			–	*	*
	DE5				–	*
	COM					–
42	PSO	–	*	*	*	*
	HS		–	*	*	*
	DE2			–	*	*
	DE5				–	*
	COM					–
48	PSO	–	*	*	*	*
	HS		–	*	*	*
	DE2			–	*	*
	DE5				–	*
	COM					–

Experimental results on a previously used test case with normally distributed demand manifest that the employed algorithms, especially DE and PSO, can be very efficient even in high-dimensional problems, with respect to both solution accuracy and time efficiency. Also, they were able to outperform other randomized algorithms such as COMSOAL. The next step in our research will consider problems with different distributions of demand as well as different heuristic optimization approaches.

Acknowledgment

The authors would like to thank the anonymous reviewers for their useful comments and suggestions.

References

[1] Wagner HM, Whitin TM. Dynamic version of the economic lot size model. Management Science 1958;5:89–96.
[2] Zangwill W. A backlogging model and a multi-echelon model of a dynamic economic lot size production system—a network approach. Management Science 1969;15:506–27.
[3] Federgruen A, Tzur M. A simple forward algorithm to solve general dynamic lot sizing models with n periods in $o(n \log n)$ or $o(n)$ time. Management Science 1991;37:909–25.
[4] Wagelmans A, Van Hoesel S, Kolen A. Economic lot sizing: an $o(n \log n)$ algorithm that runs in linear time in the Wagner–Whitin case. Operations Research 1992;40:145–56.
[5] Aggarwal A, Park JK. Improved algorithms for economic lot size problems. Operations Research 1993;41:549–71.
[6] Karimi B, Ghomi SMTF, Wilson JM. The capacitated lot sizing problem: a review of models and algorithms, Omega. The International Journal of Management Science 2003;31:365–78.
[7] Jans R, Degraeve Z. Modeling industrial lot sizing problems: a review. International Journal of Production Research 2008;46:1619–43.
[8] Guan Y, Miller AJ. Polynomial-time algorithms for stochastic incapacitated lot-sizing problems. Operations Research 2008;56:1172–83.
[9] Guan Y. Stochastic lot-sizing with backlogging: computational complexity analysis. Journal of Global Optimization 2010:1–28.
[10] Vargas V. An optimal solution for the stochastic version of the Wagner–Whitin dynamic lot-size model. European Journal of Operational Research 2009;198(2):447–51.
[11] Kennedy J, Eberhart RC. Particle swarm optimization. In: Proceedings of IEEE international conference on neural networks, vol. 4. Piscataway, NJ: IEEE Service Center; 1995. p. 1942–8.
[12] Storn R, Price K. Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 1997;11:341–59.
[13] Geem ZW, Kim JH, Loganathan G. A new heuristic optimization algorithm: harmony search. Simulation 2001;76(2):60–8.
[14] Pan Q-K, Tasgetiren MF, Liang Y-C. A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. Computers and Operations Research 2008;35(9):2807–39.
[15] Parsopoulos KE, Vrahatis MN. Studying the performance of unified particle swarm optimization on the single machine total weighted tardiness problem. In: Sattar A, Kang BH, editors. Lecture notes in artificial intelligence (LNAI), vol. 4304. Springer; 2006. p. 760–9.
[16] Tasgetiren M, Liang Y-C, Pan Q-K. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. Computers & Industrial Engineering 2008;55(4):795–816.
[17] Geem Z. Optimal scheduling of multiple dam system using harmony search algorithm, Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), vol. 4507, 2007. p. 316–23.
[18] Geem Z. Novel derivative of harmony search algorithm for discrete design variables. Applied Mathematics and Computation 2008;199(1):223–30.
[19] Parsopoulos K, Skouri K, Vrahatis M. Particle swarm optimization for tackling continuous review inventory models. In: Lecture notes in computer science, vol. 4974. Springer; 2008. p. 103–12.
[20] Gaafar LK, Aly AS. Applying particle swarm optimisation to dynamic lot sizing with batch ordering. International Journal of Production Research 2009;47(12):3345–61.
[21] Kennedy J. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In: Proceedings of the IEEE congress on evolutionary computations. Washington, DC, USA: IEEE Press; 1999. p. 1931–8.
[22] Suganthan PN. Particle swarm optimizer with neighborhood operator. In: Proceedings of the IEEE congress on evolutionary computations, Washington, DC, USA, 1999. p. 1958–61.
[23] Clerc M, Kennedy J. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computations 2002;6(1):58–73.
[24] Trelea IC. The particle swarm optimization algorithm: convergence analysis and parameter selection. Information Processing Letters 2003;85:317–25.
[25] Laskari EC, Parsopoulos KE, Vrahatis MN. Particle swarm optimization for integer programming. In: Proceedings of the IEEE 2002 congress on evolutionary computation. Hawaii (HI), USA: IEEE Press; 2002. p. 1582–7.
[26] Epitropakis MG, Tasoulis DK, Pavlidis NG, Plagianakos VP, Vrahatis MN. Enhancing differential evolution utilizing proximity-based mutation operators. IEEE Transactions on Evolutionary Computations 2011;15(1):99–119.
[27] Lee KS, Geem ZW. A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice. Computer Methods in Applied Mechanics and Engineering 2005;194:3902–33.
[28] Chakraborty UK, Deb K, Chakraborty M. Analysis of selection algorithms: a Markov chain approach. Evolutionary Computation 1996;4:133–67.
[29] Beyer H-G, Schwefel H-P. Evolution strategies: a comprehensive introduction. Natural Computing 2002;1(1):3–52.
[30] Arcus AL. COMSOAL: a computer method of sequencing operations for assembly lines. International Journal of Production Research 1966;4:259–77.
[31] DePuya GW, Whitehouse GE. Applying the COMSOAL computer heuristic to the constrained resource allocation problem. Computers & Industrial Engineering 2000;38:413–22.
[32] Özcan U, Toklu B. Balancing two-sided assembly lines with sequence-dependent setup times. International Journal of Production Research 2010;48(18):5363–83.
[33] Becker C, Scholl A. A survey on problems and methods in generalized assembly line balancing. European Journal of Operational Research 2006;168:694–715.