Variable neighborhood search for the economic lot sizing problem with product returns and recovery

Angelo Sifaleras*

Department of Applied Informatics, School of Information Sciences, University of Macedonia, 156 Egnatia Str., Thessaloniki 54636, Greece

Ioannis Konstantaras

Department of Business Administration, School of Business Administration, University of Macedonia, 156 Egnatia Str., Thessaloniki 54636, Greece

Nenad Mladenović

LAMIH, University of Valenciennes, France

Abstract

The economic lot sizing problem with product returns and recovery is an important problem that appears in reverse logistics, and has recently been proved to be NP-hard. In this paper, we suggest a Variable Neighborhood Search (VNS) metaheuristic algorithm for solving this problem. It is the first time that such an approach has been used for this problem in the literature. Our research contributions are threefold: first, we propose two novel VNS variants to tackle this problem efficiently. Second, we present several new neighborhoods for this combinatorial optimization problem, and an efficient local search method for exploring them. The computational results, obtained on a recent set of benchmark problems with 6480 instances, demonstrate that our approach outperforms the state-of-the-art heuristic methods from the literature, and that it achieved an average optimality gap equal to 0.283% within average 8.3 seconds. Third, we also present a new benchmark set with the largest instances in the literature. We demonstrate the robustness of the

*Corresponding author. Tel: +302310 891884, Fax: +302310 891881

Email addresses: sifalera@uom.gr (Angelo Sifaleras), ikonst@uom.gr (Ioannis Konstantaras), nenad.mladenovic@univ-valenciennes.fr (Nenad Mladenović)

Preprint submitted to International Journal of Production Economics October 27, 2014

Please cite this paper as:

Sifaleras A., Konstantaras I., and Mladenović N., "Variable neighborhood search for the economic lot sizing problem with product returns and recovery", International Journal of Production Economics, Elsevier Ltd., Vol. 160, pp. 133-143, 2015.

The final publication is available at Elsevier via http://dx.doi.org/10.1016/j.ijpe.2014.10.003

proposed VNS approach in this new benchmark set compared with Gurobi optimizer.

Keywords: Inventory, Variable Neighborhood Search, Mathematical Programming, Lot Sizing, Remanufacturing
2010 MSC: 90B05, 90C59, 65K05, 90-08

---

1. Introduction

Reverse logistics stands for all operations related to the return of products and materials. It is a process of planning, implementing, and controlling the efficient, cost effective flow of raw materials, in-process inventory, finished products and related information from the point of consumption to the point of origin for the purpose of recovery value or proper disposal. Reusing product, or material returns have gained considerable attention in industry and academia because of economical, environmental and legislative reasons. Balancing economic development with environmental protection is a key challenge to sustain manufacturing companies. Conventional manufacturing is unsustainable because of its significant adverse environmental impacts. Remanufacturing can help companies to achieve sustainable manufacturing by saving costs via reductions in consumption of natural resources. Remanufacturing can also help reduce environment burden by decreasing landfill wastes and reclaim resources and energy already consumed in the original manufacturing of the products. Besides the environmental benefits, remanufacturing also provides economic incentives to companies by selling the remanufactured products and extending the life cycles of the products [13].

Remanufacturing transforms used products into like new ones. After disassembly, sorting and cleaning, modules and parts are extensively inspected and problematic parts are repaired, or if not possible, replaced by new parts [16]. These operations allow a considerable amount of value incorporated in the used product to be regained. Remanufactured products have usually the same quality as the new products and are sold for the same price, but they are less costly. The significance of remanufacturing is that it would allow manufacturers to respond to environmental and legislative pressure by enabling them to meet waste legislation while maintaining high productivity for high-quality, lower-cost products with less landing filling and consumption of raw materials and energy [19].

Inventory management and control is one of the key decision making areas

while managing product returns and remanufacturing. A scientific literature review of the existing quantitative models on inventory control with product returns and remanufacturing can be found in the recent work of Akcali & Cetinkaya [1]. The Dynamic or Economic Lot Sizing Problem (DLSP or ELSP) is one of the most extensively researched topics in inventory control literature. ELSP considers a warehouse or retailer facing a dynamic and deterministic demand for a single item over a finite horizon [40]. A lot of research has been done on dynamic lot sizing since late 1950s after the seminal paper [40]. For a general review of the economic lot sizing problem, readers may refer to [6, 7]. The ELSP with remanufacturing options (ELSRP) is an extension of the classical Wagner Whitin model. The additional feature is that in each period known quantities of used products enter the system. These returns can be remanufactured to satisfy demand besides regular manufacturing. This means that there are two types of inventory: the inventory of returns and the inventory of serviceables, where a serviceable is either a newly manufactured item or a remanufactured returned item. In ELSR problem, the traditional trade-off between set-up and holding costs is extended with remanufacturing set-up cost and holding cost for returns.

Many different variants of the ELSR problem, described above, have been studied. Richter & Sombrutzki [28] extended the classical Wagner-Whitin model by introducing remanufacturing process. They assumed that the dynamic demand could be satisfied from two sources: newly produced items, and the used ones, returned to the system, stored and remanufactured. They presented a dynamic programming algorithm to determine the periods in which products are manufactured and remanufactured. Richter & Weber [29] introduced a reverse Wagner-Whitin model with variable manufacturing and remanufacturing costs. They also investigated the behaviour of the system with a disposal option. Golany et al. [14] studied a variant of ELSRP with deterministic demand and return setting in the presence of disposal options and provided a polynomial solution algorithm. Yang et al. [44] extended the work by Golany et al. [14] on the concave cost functions. Based on a special structure of the extreme-point optimal solutions for the minimum concave cost problem, they developed a polynomial-time heuristic algorithm. Beltran & Krass [5] studied the case where demand can be satisfied by new items and unprocessed returned items, and the returned items can also be disposed. They proposed some useful properties of the optimal solution (manufacturing and disposal decisions) which led to a dynamic programming algorithm. Pineyro & Viera [23] proposed and evaluated a set of inventory policies de-

signed for the ELSRP under the assumption that remanufacturing the used items is more suitable than disposing them and producing new items. Teunter et al. [36] presented two variants of the basic dynamic lot sizing model with product returns. In the first model variant, it is assumed that there is a joint set up cost for manufacturing and remanufacturing when the same production line is used for both processes, and the second model variant assumed separate set up costs for manufacturing and remanufacturing when separate production lines are used. For these two models, several heuristic algorithms were proposed and compared with the computational performance of modified versions of three well-known heuristics, namely Silver-Meal (SM), Least Unit Cost, and Part Period Balancing. Teunter et al. [38] later advanced the study of the ELSRP by developing fast but simple heuristics that can provide near-optimal solutions. Schulz [30] proposed a generalization of the Silver-Meal based heuristic introduced by Teunter et al. [36] for the separate set up cost setting by using known results of the static lot sizing problem.

Recently, Nenes et al. [22] studied some inventory control policies for inspection and remanufacturing and proposed alternative policies when both demand of new products and returns of used products are stochastic; Helmrich et al. [27] added some valid inequalities to the original formulation of the ELSRP to improve its formulation, and also presented reformulations based on the shortest path problem for the ELSR problem. Also, Piñeyro and Viera [24] extended the ELSR problem for the case where substitution is allowed for remanufactured items.

Tang & Teunter [35] studied the multi product dynamic lot sizing problem for a hybrid production line with manufacturing new products and remanufacturing the returned products. They considered one manufacturing and one remanufacturing lot for each product during a common cycle time and formulated a Mixed Integer Linear Programming (MILP) problem to find an exact solution. The multi-product dynamic lot sizing problem with two production sources, manufacturing and remanufacturing, for which operations are performed on separate dedicated lines was studied in [37]. The authors proposed a mixed integer programming model to solve the problem for a fixed cycle time, which can be combined with a cycle time search to find an optimal solution. In [45], the multi-product ELSRP was further analyzed, extending the scheduling policy from the common cycle to a basic period policy. They relaxed the constraint of one manufacturing lot and one remanufacturing lot for each product during a common cycle time studied in [35], and they proposed an algorithm to solve their model.

Recently, both trajectory-based and population-based metaheuristics were used to tackle the ELSR problem. Piñeyro and Viera [23] suggested a Tabu Search procedure for solving the problem tackled, even with a more general cost structure and final disposing of returns. Li et al. [18] also developed a Tabu Search (TS) algorithm based on an alternative mixed-integer linear programming formulation. The TS algorithm solves the problem of several small linear sub-problems of the original model. Moustaki et al. [21] studied the behavior of Particle Swarm Optimization (PSO) algorithm on the ELSR problem. The most suitable variants of the algorithm were identified, and the necessary modifications in the formulation of the corresponding optimization problem were provided. The performance of the above two metaheuristic algorithms were compared with the Silver-Meal based heuristics, proposed by Schulz [30], in each of the above papers, and showed that the proposed algorithms can be considered as a promising alternative for solving ELSR problem. In a very recent paper [4], the authors proved that the ELSR problem is NP-hard, and they constructed a heuristic method that uses dynamic programming and the Wagner-Whitin algorithm to solve the problem.

In this work, we propose two novel variants of the Variable Neighborhood Search (VNS) algorithm to find the optimal solution for the ELSR problem, and also we assess their performance first on the test suite used by Schulz [30], and second on a new benchmark set with more than four times larger instances. The proposed VNS approach is also compared with the established SM-based variants from [30] and other metaheuristics optimization algorithms. Although VNS has also been applied to other inventory problems [2, 3, 41, 42, 43], this is the first time that it is used for this particular problem.

1.1. Research contributions

The research contributions of this paper are as follows:

- Two novel VNS schemes are proposed for the first time and tested for this combinatorial optimization ELSR problem. The methodological contribution is based on the fact that the proposed VNS approaches use new strategies for the local search and also for the shaking phase.
- Several new neighborhoods for this combinatorial optimization problem are presented and an efficient local search method for exploring them is described.

- The computational results obtained on an established set of benchmark problems with 6480 instances show that our VNS metaheuristic algorithm outperforms the state-of-the-art heuristic methods from the literature, and that it is able to achieve an average optimality gap equal to 0.283% within average 8.3 seconds.
- Our approach does not depend on any other, commercial or not, solver for either computing a starting solution or an intermediate computation. Thus, it is a self-contained solver without any link to other callable library API.
- The proposed VNS solver is quite fast and requires only 8.3 and 30 seconds to solve all the instances of this set of 6480 benchmark problems on average and maximum case, respectively.
- A new benchmark set with the currently largest instances (52 periods) in the literature have been developed and is made publicly available. Computational results obtained on this new data set prove the robustness of the proposed VNS approach.

1.2. Outline

The rest of the paper is organized as follows: Section 2 provides the basic formulation of the problem. In Section 4, the neighborhood structures that are used in our VNS approach are analytically described. The proposed VNS metaheuristic algorithm is described in Section 3, where we also discuss the necessary adaptation of the VNS framework in order to fit the specific problem requirements. Section 5 exposes the obtained results, and the paper concludes with Section 6.

2. Model formulation

The problem discussed in this paper is to satisfy the demand of items in each period at the lowest possible total cost. The demand is given for a finite planning horizon and is assumed to be not stationary. It can be satisfied by both manufactured new items and remanufactured returned ones (both known as "serviceables"). Also, the number of returns is known for all periods and assumed to be not stationary. The returns can be completely remanufactured and sold as the new ones. The problem studied in our paper consists of the dynamic lot sizing model with both remanufacturing and manufacturing setup costs, as it was introduced by [36] and studied by [30]. The lot sizing problem under separate manufacturing and remanufacturing set up

costs is suitable for situations where there are separate production lines, one for manufacturing and one for remanufacturing. The aim is to determine the number of remanufactured and manufactured items per period in order to minimize the sum of set up costs of the manufacturing and remanufacturing processes and holding costs for returns and serviceables under various operational constraints. Before presenting the formulation of the ELSR problem, we first introduce some notations:

t: time period, $t = 1, 2, \ldots, T$.
$D(t)$: demand for time period t.
$R(t)$: number of returned items in period t that can be completely remanufactured and sold as new.
h_M: holding cost for the serviceable items per unit time.
h_R: holding cost for the recoverable items per unit time.
$z_M(t)$: binary decision variable denoting the initiation of a manufacturing lot in period t.
$z_R(t)$: binary decision variable denoting the initiation of a remanufacturing lot in period t.
$x_M(t)$: number of manufactured items in period t.
$x_R(t)$: number of items that are eventually remanufactured in period t.
k_M: manufacturing setup cost.
k_R: remanufacturing setup cost.
$y_M(t)$: inventory level of serviceable items in period t.
$y_R(t)$: inventory level of items that can be remanufactured in period t.

The ELSR problem can be modeled as a MILP problem as follows [30]:

$$\min C = \sum_{t=1}^{T} \left(k_R z_R(t) + k_M z_M(t) + h_R y_R(t) + h_M y_M(t) \right), \tag{1}$$

where:

$$z_R(t) = \begin{cases} 1, & \text{if } x_R(t) > 0, \\ 0, & \text{otherwise,} \end{cases} \qquad z_M(t) = \begin{cases} 1, & \text{if } x_M(t) > 0, \\ 0, & \text{otherwise,} \end{cases} \tag{2}$$

are binary decision variables denoting the initiation of a remanufacturing or manufacturing lot, respectively. Naturally, the model is accompanied by a number of constraints [30]:

$$
\begin{aligned}
&y_R(t) = y_R(t-1) + R(t) - x_R(t),\\
&y_M(t) = y_M(t-1) + x_R(t) + x_M(t) - D(t),\\
&\forall\, t = 1, 2, \ldots, T.
\end{aligned} \tag{3}
$$

$$
\begin{aligned}
&x_R(t) \leq M z_R(t),\\
&x_M(t) \leq M z_M(t),\\
&\forall\, t = 1, 2, \ldots, T.
\end{aligned} \tag{4}
$$

$$
\begin{aligned}
&y_R(0) = y_M(0) = 0,\\
&z_R(t), z_M(t) \in \{0, 1\},\\
&y_R(t), y_M(t), x_R(t), x_M(t) \geq 0,\\
&\forall\, t = 1, 2, \ldots, T.
\end{aligned} \tag{5}
$$

The constraints defined in Eq. (3) are the inventory balance equations which compute the inventory of returns and serviceables, respectively. Equation (4) ensures that a fixed setup cost is incurred when remanufacturing or manufacturing takes place, respectively. The parameter M is a sufficiently large number; [30] suggests the use of the total demand during the planning horizon. Finally, Eq. (5) ensures that the inventories are initially empty, sets the indicator variables and prevents negative (re)manufacturing or inventory.

Some interesting properties of the considered model have been identified in [36] such as that there is a possibility of attaining optimal solutions that do not adhere to the zero–inventory property. Since the ELSR problem is NP-hard, the need for efficient metaheuristic algorithms is visible.

3. Novel VNS schemes for solving ELSRP

VNS is a metaheuristic based on a systematic change of the neighborhood structures within a search introduced by Mladenović and Hansen [20]. The main idea of this trajectory-based method is to improve the incumbent solution by examining solutions belonging to different neighborhoods (intensification or local search part), and sometimes letting, temporarily, the objective function to deteriorate in order to escape from locally optimal solutions (diversification or shaking part). VNS has already been successfully used for solving various combinatorial and global optimization problems [9, 15, 34, 39].

In this paper we suggest two VNS variants with new strategies for the local search and also for the shaking phase, and describe their corresponding reasoning. The first one, since the cardinality of the proposed neighborhoods

is the same, is based on computation of the total number of local search improvements per neighborhood. This way, the most frequently used "moves" / neighborhoods are considered first. Thus, we suggest ordering neighborhoods based on the success of local search improvements per neighborhood, computed by using extensive preliminary testing, and use it in local search phase of VNS. Such a local search and VNS obtained, we call Ordered Variable Neighborhood Descent (OVND) and *Ordered General VNS (OGVNS)*, respectively. Furthermore, we also suggest a new strategy that is based on random choice of all the neighborhoods in shaking phase of OGVNS. The second one is based on random choice of neighborhoods in VND phase and shaking phase of General VNS. Thus, the second VNS variant uses Randomised VND as a local search and new Shaking and will be called *Randomised GVNS (RGVNS)*. More precisely, RGVNS takes only five randomly chosen neighborhoods in each local search step and shaking phase.

Similarly to the majority of the metaheuristic implementations in the literature, both schemes of our VNS approach for the solution of ELSRP consist of several parts. Firstly, a heuristic initialization method is applied in order to provide us with a starting solution. Afterwards, the VND heuristic is applied to improve our starting solution. The last part of our VNS approach is the shaking or perturbation phase.

3.1. Constructive Heuristic

Li et al. [18] in order to find an initial solution to the ELSRP for their block-chain based tabu search algorithm used the commercial CPLEX solver to solve a Linear Problem (LP). However, in cases where the solution of that LP did not result in integer-valued variables, they again applied CPLEX to solve an Integer Problem (IP). In this work, we have implemented a quite simple constructive heuristic where the total demand is fulfilled by a single lot (without remanufacturing units) in the first period. Although this proposed initialization method cannot guarantee a quality starting solution, it is very simple, runs in $O(T)$, and moreover it does not depend on other solvers or optimization software packages [32]. Discussion on the quality of the starting solutions over all test instances, produced by the proposed heuristic initialization method is made later in Section 5.6. Also, Baki et al. [4] recently proposed a heuristic construction method based on dynamic programming. In contrast to such sophisticated initialization methods, we implemented a quite simple constructive heuristic where the total demand is fulfilled by a

single lot (without remanufacturing units) in the first period. This initialization method is quite similar to the starting solution method proposed by Piñeyro and Viera [24] of zero-remanufacturing.

Algorithm 1 Heuristic initialization method

1: **procedure** HEURISTIC_START($T, R, D, x_R, x_M, y_R, y_M, z_R, z_M$)
2: $x_R,\ z_R,\ x_M,\ z_M,\ y_R,\ y_M \leftarrow 0$
3: $y_R(1) \leftarrow R(1)$
4: **for** $i = 2, T$ **do**
5: $y_R(i) \leftarrow y_R(i-1) + R(i)$
6: $y_M(T+1-i) \leftarrow y_M(T+2-i) + D(T+2-i)$
7: **end for**
8: $x_M(1) \leftarrow y_M(1) + D(1)$
9: $z_M(1) \leftarrow 1$
10: **end procedure**

In the pseudocode (`Heuristic_Start`) of the proposed heuristic initialization method, please note that, the expressions in line 2 use whole array operations as in Fortran. Thus, by $x_R = 0$, we denote that $x_R(t) = 0$, $\forall t \in 1 \ldots T$.

3.2. VND algorithm

The number of different neighborhoods, and their corresponding order in the Variable Neighborhood Descent (VND) and shaking procedures are very important factors for the efficiency of any VNS-based methodology, and thus they consist well-studied issues of VNS. For example, Li et al. [18] report four different neighborhoods using one-shift, two-shift, exchange, and cross two-shift operators in their paper [18]. Following their notation, a neighborhood $N(\sigma)$ is a set of neighboring solutions in the solution space that is formally defined as $N(\sigma) = \{$solution σ' obtained by applying one / or more change(s) to $\sigma\}$. VND constitutes a deterministic algorithm aiming to intensify the local search in the neighborhoods described in Section 4, via a systematic neighborhood change. As previously mentioned, the two proposed VNS schemes have quite different strategies for local search. Regarding the OGVNS variant, based on extensive testing of different combinations (described later in Subsection 5.6), the following order of neighborhoods was decided to be used in our experiments: [N_4, N_{15}, N_1, N_2, N_{10}, N_{19}, N_{12}, N_8, N_5, N_{18}, N_{16}, N_7, N_{17}, N_{14}, N_{13}, N_3, N_6, N_{11}, N_9]. Thus, the steepest descent heuristic is used for $k_{max} = |N| = 19$ neighborhood structures (N_k, where $k = 1, \ldots, k_{max}$).

Algorithm 2 VND

```
1: procedure VND(T, R, D, h_R, h_M, k_R, k_M, σ, k_max)
2:     repeat
3:         improvement ← 0
4:         for k ← 1, k_max do
5:             for t ← 1, T do
6:                 Find the best neighbor σ' of σ (σ' ∈ N_k(σ))
7:                 if the obtained solution σ' is better than σ then
8:                     Set σ ← σ'
9:                     Set improvement ← 1
10:                end if
11:            end for
12:        end for
13:    until improvement == 0
14: end procedure
```

On the other hand, the RGVNS variant takes $k_{max} = 5$ neighborhoods in random order out of the 19 in each VND step. The VND algorithm sequentially searches each neighbor for each period, with only a few exceptions (e.g., N_1 and N_2 cannot be applied for $t = 1$). The pseudo-code of the proposed VND algorithm (`VND`) is as follows:

3.3. Randomized shaking phase of VNS

Each metaheuristic method has an intensification and a diversification method. In both OGVNS and RGVNS variants, the VND algorithm and the shaking phase constitute the intensification and diversification methods, respectively. Once the VND is not able to further improve the current best (incumbent) solution by local search, the randomized shaking phase starts in order to explore larger neighborhoods (OGVNS and RGVNS consider $k_{max} = 19$ and 5 neighborhoods in the shaking, respectively). The randomized shaking phase is based on a shuffle to the order of neighborhoods, in order to get a new complete random solution. Roughly speaking, the aim of the shaking phase is to escape from locally optimal solutions, and thus it allows us to evaluate unexplored areas of the feasible region. More specifically, the randomized shaking phase contributes in an uncontrolled expanding of the search to unexplored regions in the solution space. Both variants of the proposed VNS approach explore valleys surrounding a local optimum, until a stopping condition is satisfied (i.e., a maximum computing time in our case).

We have implemented a shaking phase which can be described as large neighborhood search. The fact that we shake k consecutive neighborhoods

one-after-the-other from the beginning to the end, permits us to explore far apart valleys containing near-optimal solutions. The pseudo-code of the proposed VNS algorithm (`VNS`) is as follows:

Algorithm 3 VNS

1: **procedure** VNS($T, R, D, h_R, h_M, k_R, k_M, \sigma, k_{max}$)
2: Apply the `Heuristic_Start` method
3: **while** time $<$ max_time **do**
4: Apply the `VND` method
5: Shake k consecutive N_k ($k \in 1 \dots k_{max}$) one-after-the-other
6: Apply the `Knuth_Shuffle` method and generate a random N_k order
7: **end while**
8: **end procedure**

In order to generate a random permutation of the initial order of neighborhoods (i.e., in line 6 of the `VNS` pseudo-code), we apply the `Knuth_Shuffle` method. The latter method for generating a random permutation of a finite set was originally published by Fisher & Yates in [12], later designed for computer use by Durstenfeld in [10], and analytically described in the well-known book by Knuth [17]. The time complexity of the Knuth shuffle method (or Fisher-Yates shuffle method) is $O(T)$ (for the ELSRP with T periods). Since the Knuth shuffle method requires a random number generator, we have used a portable random generator based on the book by Press et al. [25]. This way the proposed solver is independent of the compiler used each time, (i.e., the pseudo-random generator implemented by Intel Fortran, gfortran, etc.).

4. Neighborhood structures for solving ELSRP

4.1. Neighborhood structures

The neighborhood structures used are not nested, (i.e., each one does not necessarily contain the previous) as happens usually in the case of VNS implementations for other optimization problems (i.e., 2-opt, 3-opt for the Traveling Salesman Problem (TSP)). For some optimization problems many non-nested neighborhood structures may be designed, and then the question of their order is essential for the final success of the method. Roughly speaking, one way is to make their order in non-decreasing order of their cardinality, another is to make random order and the third one is in using memory. Puchinger & Raidl reported 10-20 MILP-based neighborhoods for

the multidimensional knapsack problem [26], and also proposed to consider the faster to search (or smaller) neighborhoods first.

Although several authors in the literature suggest that a small number of neighborhoods (i.e., $|N| = 3$ or 4) is sufficient, we have followed a different approach, which helped us to develop an efficient VND. In this paper, 19 different neighborhoods have been applied for adapting the VNS methodology to the ELSRP. The reason for having so many different neighborhoods is due to the fact that each neighborhood corresponds to a combination of different "moves". Such moves are either increment or reduction in the values of the various decision variables (i.e., x_M, x_R, y_M, y_R). These different moves are appropriately combined in order to balance any change in the Eqs. 3. This way, we are able to visit neighboring solutions without any feasibility violations. A description of the neighborhoods follows:

$N_1(\sigma) = \{\sigma' \mid \sigma'$ is obtained by shifting the set-up for manufacturing $z_M(t)$ in one period from 0 to 1, by reducing only some y_M variables and a previous $x_M(i)$ variable, for some $t = 1, 2, \ldots, T$, and $i < t\}$,

$N_2(\sigma) = \{\sigma' \mid \sigma'$ is obtained by shifting the set-up for manufacturing $z_M(t)$ in one period from 1 to 0, by increasing only some y_M variables and a previous $x_M(i)$ variable, for some $t = 1, 2, \ldots, T$, and $i < t\}$,

$N_3(\sigma) = \{\sigma' \mid \sigma'$ is obtained by shifting the set-up for remanufacturing $z_R(t)$ in one period from 0 to 1, by strictly reducing some y_R variables and increasing some y_M variables, for some $t = 1, 2, \ldots, T\}$,

$N_4(\sigma) = \{\sigma' \mid \sigma'$ is obtained by shifting the set-up for remanufacturing $z_R(t)$ in one period from 0 to 1, by strictly reducing some y_R, y_M, and x_M variables, for some $t = 1, 2, \ldots, T\}$,

$N_5(\sigma) = \{\sigma' \mid \sigma'$ is obtained in case that when both the set-up for manufacturing $z_M(t)$ and remanufacturing $z_R(t)$ in one period exists, we shift the set-up for remanufacturing $z_R(t)$ in that period from 1 to 0 by increasing some y_R variables, for some $t = 1, 2, \ldots, T\}$,

$N_6(\sigma) = \{\sigma' \mid \sigma'$ is obtained in case where both the set-up for manufacturing $z_M(t)$ and remanufacturing $z_R(t)$ in one period exists, and we shift the set-up for manufacturing $z_M(t)$ in that period from 1 to 0 by increasing some y_R variables, for some $t = 1, 2, \ldots, T\}$,

$N_7(\sigma) = \{\sigma' \mid \sigma'$ is obtained by shifting the set-up for manufacturing $z_M(t)$ in one period from 1 to 0 (or reduce only that $x_M(t)$ variable) and shift the set-up for remanufacturing $z_R(t)$ in that period from 0 to 1 (or increase only that $x_R(t)$ variable), by strictly reducing some y_R variables and may also shift the set-up for manufacturing $z_M(t)$ in another period from 0 to 1, for some $t = 1, 2, \ldots, T\}$,

$N_8(\sigma) = \{\sigma' \mid \sigma'$ is obtained by shifting the set-up for remanufacturing $z_R(t)$ in one period from 1 to 0, and shifting the set-up for manufacturing $z_M(t)$ in that period from 0 to 1 (or increase only that $x_M(t)$ variable), by strictly increasing some y_R variables, for some $t = 1, 2, \ldots, T\}$,

$N_9(\sigma) = \{\sigma' \mid \sigma'$ is obtained by shifting the set-up for remanufacturing $z_R(t)$ in one period from 1 to 0 and shifting the set-up for manufacturing $z_M(t)$ in that period from 0 to 1 (or increase only that $x_M(t)$ variable), by strictly reducing some y_R variables and increasing some y_M variables, for some $t = 1, 2, \ldots, T\}$,

$N_{10}(\sigma) = \{\sigma' \mid \sigma'$ is obtained in case where between two periods with z_M values equal to 1, we reduce as much as possible some y_M variables (and perhaps shift the set-up for manufacturing $z_M(t)$ in one period from 1 to 0) and increase some x_M variables, for some $t = 1, 2, \ldots, T\}$,

$N_{11}(\sigma) = \{\sigma' \mid \sigma'$ is obtained in case where the set-up for manufacturing $z_M(t)$ and remanufacturing $z_R(t)$ in one period exists, then between two periods with z_R values equal to 1, we reduce as much as possible some y_R variables (and perhaps shift the set-up for manufacturing $z_M(t)$ in another period from 0 to 1), for some $t = 1, 2, \ldots, T\}$,

$N_{12}(\sigma) = \{\sigma' \mid \sigma'$ is obtained by shifting the set-up for remanufacturing $z_R(t)$ in one period from 1 to 0, by strictly increasing some y_M, y_R variables, and one x_M variable, for some $t = 1, 2, \ldots, T\}$,

$N_{13}(\sigma) = \{\sigma' \mid \sigma'$ is obtained by shifting the set-up for remanufacturing $z_R(t)$ in one period from 1 to 0 (or reduce only that x_R variable) and shifting the set-up for manufacturing $z_M(t)$ in that period from 0 to 1 (or increase only that x_M variable), by strictly reducing some y_M and y_R variables, for some $t = 1, 2, \ldots, T\}$,

$N_{14}(\sigma) = \{\sigma' \mid \sigma'$ is obtained by shifting the set-up for remanufacturing $z_R(t)$ in one period from 1 to 0, by strictly reducing some y_M and y_R variables, for some $t = 1, 2, \ldots, T\}$,

$N_{15}(\sigma) = \{\sigma' \mid \sigma'$ is obtained by shifting the set-up for remanufacturing $z_R(t)$ in one period from 1 to 0 (or reduce only that x_R variable), by strictly increasing some y_R variables and reducing some y_M variables, for some $t = 1, 2, \ldots, T\}$,

$N_{16}(\sigma) = \{\sigma' \mid \sigma'$ is obtained by shifting the set-up for manufacturing $z_M(t)$ in one period from 1 to 0 (or reduce only that x_M variable) and shifting the set-up for manufacturing $z_M(t)$ in another period from 0 to 1 (or increase only that x_M variable), by strictly reducing some y_M variables, for some $t = 1, 2, \ldots, T$ $\}$,

$N_{17}(\sigma) = \{\sigma' \mid \sigma'$ is obtained by reducing some y_M variables (and perhaps shifting the set-up for manufacturing $z_M(t)$ in one period from 1 to 0) and shifting the set-up for remanufacturing $z_R(t)$ in one period from 0 to 1 (or increase that x_R variable), by reducing some y_R variables, for some $t = 1, 2, \ldots, T\}$,

$N_{18}(\sigma) = \{\sigma' \mid \sigma'$ is obtained by reducing some y_R variables, by shifting the set-up for remanufacturing $z_R(t)$ in one period from 0 to 1 and increasing some y_M variables, for some $t = 1, 2, \ldots, T\}$,

$N_{19}(\sigma) = \{\sigma' \mid \sigma'$ is obtained in case where between two periods with z_R values equal to 1, we reduce as much as possible some y_R variables (and perhaps shift the set-up for remanufacturing $z_R(t)$ in one of these two period from 1 to 0) and increase one x_M variable$\}$.

Maintaining feasibility. The routines that implement these neighborhoods have been designed to return only feasible solutions (σ'), in case it is possible. Otherwise, no change is made in the current solution (σ) for that specific period t. However, in order to verify that our VNS implementation is always working with feasible solutions, we have also developed an auxiliary routine `check_feasibility` that checks the feasibility in $O(T)$ time. This way, all the intermediate solutions so as the final solution, are guaranteed to be feasible.

4.2. Illustrative example

Assume the following small example with $T = 3$ periods, in Table 1.

Table 1: Small example ($T = 3$) data.

$T = 3$	$hR = 0.2$	$kR = 200$
	$hM = 1$	$kM = 2000$
t	D	R
1	130	63
2	160	70
3	200	40

By solving this small example with Gurobi, an optimal solution equal to 2453.2 can be found. The initial solution computed by the constructive heuristic of Subsection 3.1 follows in the upper part of Table 2.

Table 2: Incumbent & optimal solutions found by the constructive heuristic and a neighboring solution in N_4, respectively.

	xR	zR	xM	zM	yR	yM
Incumbent solution	0	0	490	1	63	360
	0	0	0	0	133	200
	0	0	0	0	173	0
Optimal solution	0	0	317	1	63	187
	0	0	0	0	133	27
	173	1	0	0	0	0

The above incumbent solution has an objective value equal to 2633.8 with a mean absolute percentage error equal to 7.36%. By applying the proposed VND algorithm, an optimal solution can be reached if we search for $t = 3$ in the neighborhood N_4, (for $t = 1$ or 2 although we find feasible neighboring solutions, their corresponding objective values are worse than the current incumbent solution). Thus, the computation of the neighboring solution in N_4 for $t = 3$, is as follows: i) First, the value of the $xR(3)$ variable is increased from 0 to 173 (equal to $yR(3)$) and the corresponding binary decision variable $zR(3)$ variable is shifted from 0 to 1. ii) Second, in order to balance the equations 3, we decrease the $yR(3)$ variable from 173 to 0, and iii) The variables $yM(2)$, $yM(1)$, and $xM(1)$ are reduced from 200, 360, and

490 to 27, 187, and 317 respectively. Thus, Equations 3 are balanced again. The optimal solution found by searching for $t = 3$ in N_4 is presented in the lower part of Table 2.

5. Numerical testing

This Section presents comparative computational results of both proposed VNS variants regarding different benchmark sets. Subsection 5.4 shows the results of an extensive computational study of the VNS method, using the well-known set of benchmark problems (6480 instances with $T = 12$ periods) proposed by Schulz in [30], and compared with state-of-the-art heuristic methods. Subsection 5.5 describes a new benchmark set (with $T = 52$ periods), and demonstrate the robustness of the VNS approach, using large-scale instances compared with Gurobi optimizer. It is noteworthy that, currently, no other benchmark set for the ELSRP includes such a large number of 108 instances with this problem dimension (52 periods) in the literature.

The results of these two computational studies have shown that the first variant (OGVNS algorithm) performs better than the second variant (RGVNS algorithm) in the benchmark set by Schulz [30]. It appears that the OGVNS variant computed approximately 2.5 times lower mean absolute percentage error (0.283%) compared to the RGVNS variant (0.712%) with $k_{max} = 5$ neighborhoods. Also, the RGVNS variant with $k_{max} = 6$ neighborhoods achieved an error equal to (0.769%). This difference can be attributed to the fact that the local search step of the OGVNS variant is more efficient for small instances since it is based on $k_{max} = 19$ instead of $k_{max} = 5$ neighborhoods. However, this situation changes in the computational study regarding the new large-scale instances. More specifically, it appears that the RGVNS variant performs slightly better than the OGVNS variant. This difference can be attributed to the fact that the instances are more than four times larger, and the randomized local search step with $k_{max} = 5$ performs quicker and better than the extensive ordered local search step with $k_{max} = 19$ neighborhoods. Due to these reasons, the following Subsections 5.4 and 5.5 describe the efficiency and effectiveness of the OGVNS and the RGVNS variant, respectively.

5.1. Computing environment

We ran the experiments on a computer running Ubuntu Linux 14.04 64Bit with an Intel Core i7 4770K CPU at 3.5 GHz with 8 MB L3 cache and

32 GB DDR3 1600MHz main memory. Both VNS implementations were implemented in Fortran and compiled, using the Intel Fortran 64 compiler XE v.14.0.1.106 with the –fast option (on Linux it is equivalent to the collection of options: –ipo, –O3, –no-prec-div, –static, and –xHost).

5.2. Stopping condition

Two stopping conditions were used in the proposed VNS approach. First, our VNS metaheuristic algorithm stops if an (known) optimal solution has been computed. Second, we stop our VNS metaheuristic algorithm in 30 seconds. This time limit was carefully selected due to two reasons. First, this is a common time limit in the metaheuristics literature and acceptable from a computational point of view. Second, the paper by Schulz [30] did not report analytic computation times whereas the paper by Moustaki et al. [21] and Li et al. [18] reported high worst-case time limits equal to 180 seconds and 430.98 seconds, respectively. However, all the instances of the benchmark set of Schulz [30] were solved by our OGVNS algorithm within average only 8.3 seconds.

5.3. Test instances by Schulz [30]

Our OGVNS algorithm was applied on exactly the same set of benchmark problems proposed by Schulz in [30], which is an extended version of the set of benchmark problems suggested in [36]. This set of test instances has been solved by CPLEX 11 in order to find their optimal solutions. Specifically, it consists of a full factorial study of several ELSRP instances with a common planning horizon of $T = 12$ time periods. Each of the setup and the holding costs k_R, k_M, and h_R, h_M, respectively, takes three different values. The returns and demands are drawn from normal distributions with both small and large deviations. The mean of the returns' distribution assumes also three different values (return ratios). The exact configuration of the test problems is reported in Table 3.

For each specific combination of parameter values, 20 different problem instances were produced in [30]. The specific test suite was selected in our study because it contained a large number of 6480 different problem instances. Also, it facilitated comparisons with the results reported in [30] for the adapted SM algorithm and its enhanced versions, and the results reported in [21] for the PSO implementation adapted to the ELSRP.

The obtained statistics were compared with the corresponding values reported in the thorough analysis of Schulz [30] for four versions of the adapted

Table 3: Parameter values for the test problems [30].

Parameter Description	Value(s)
Setup costs	$k_M,\ k_R\ \in \{\ 200, 500, 2000\ \}$
Holding cost for serviceable products	$h_M = 1$
Holding cost for recoverable products	$h_R\ \in \{\ 0.2, 0.5, 0.8\ \}$
Demand for time period t	$D(t) \sim \mathcal{N}\left(\mu_D, \sigma_D^2\right),\ \mu_D = 100,$ $\sigma_D^2 =\ 10\%,\ 20\%$ of μ_D (10% small variance, 20% large variance)
Returns for time period t	$R(t) \sim \mathcal{N}\left(\mu_R, \sigma_R^2\right),\ \mu_R = 30\%, 50\%, 70\%$ of $\mu_D,$ $\sigma_R^2 = 10\%, 20\%$ of μ_R (10% small variance, 20% large variance)

SM heuristic. The first version refers to SM with the options of (a) manufacture only or (b) remanufacture (and manufacture if necessary), henceforth denoted as SM_2. The second version refers to SM_2 with the additional options of (c) manufacturing first and remanufacture later, or (d) remanufacture first and manufacture later, henceforth denoted as SM_4. Moreover, comparisons included the enhanced versions SM_2^+ and SM_4^+ that are derived from the previous ones by additionally checking if their solutions admit one of the following improvements: (i) two consecutive time windows can be combined or (ii) a remanufacturing lot can be increased [30]. All numerical results are reported in Tables 4-7. Specifically, Table 4 reports the average, standard deviation, and maximum values of the percentage errors for the aforementioned variants of the SM algorithm as well as for PSO and OGVNS. The first block of the Table refers to all problem instances, followed by three blocks that refer to the special cases of small and large variance for demand and returns, as well as to different return ratios. Tables 5-7 complement the results for different values of the manufacturing and remanufacturing setup cost, as well as for different holding costs of the recoverable items. The error (or optimality gap) of each heuristic algorithm is computed as the difference between the objective value of the incumbent solution and the objective value of the (known) optimal solution.

5.4. Experimental results on Schulz [30] benchmarks (T = 12)

As it is shown in Tables 4-7, the proposed OGVNS approach outperforms the state-of-the-art heuristic methods (i.e., SM_2, SM_4, SM_2^+, SM_+^4, and PSO) from the literature and was able to find 75.8% of the optimal solutions and achieve an average cost error equal to 0.283% in a few seconds.

Table 4: Percentage cost error for all instances as well as for different variance of demand, returns, and return ratios.

		Algorithm	Avg. (%)	Std. (%)	Max. (%)
All Instances		SM_2	7.5	7.9	49.2
		SM_4	6.1	7.6	47.3
		SM_2^+	6.9	7.9	49.2
		SM_+^4	2.2	2.9	24.3
		PSO	4.3	4.5	49.8
		OGVNS	0.3	0.8	8.9
Demand	Small	SM_2	7.2	7.9	43.6
	Variance	SM_4	6.0	7.6	47.3
		SM_2^+	6.6	7.9	43.5
		SM_+^4	2.1	2.8	18.9
		PSO	4.4	4.6	49.8
		OGVNS	0.3	0.7	8.9
	Large	SM_2	7.8	8.0	49.2
	Variance	SM_4	6.1	7.5	43.9
		SM_2^+	7.2	8.0	49.2
		SM_+^4	2.4	3.0	24.3
		PSO	4.1	4.5	48.3
		OGVNS	0.3	0.8	6.2
Returns	Small	SM_2	7.3	7.8	47.2
	Variance	SM_4	6.1	7.6	47.3
		SM_2^+	6.8	7.8	47.2
		SM_+^4	2.2	2.9	21.1
		PSO	4.3	4.6	46.7
		OGVNS	0.3	0.8	6.2
	Large	SM_2	7.7	8.0	49.2
	Variance	SM_4	6.1	7.5	46.3
		SM_2^+	7.1	8.0	49.2
		SM_+^4	2.3	2.9	24.3
		PSO	4.2	4.5	49.8
		OGVNS	0.3	0.7	8.9
Return Ratio	30%	SM_2	5.5	5.5	31.3
		SM_4	3.7	4.5	28.5
		SM_2^+	4.9	5.4	31.3
		SM_+^4	1.2	1.8	12.1

Table 4: Percentage cost error for all instances as well as for different variance of demand, returns, and return ratios.

	Algorithm	Avg. (%)	Std. (%)	Max. (%)
	PSO	3.5	3.1	45.5
	OGVNS	0.2	0.6	5.6
50%	SM_2	8.5	9.4	40.1
	SM_4	7.3	8.2	41.8
	SM_2^+	8.0	9.3	39.8
	SM_+^4	2.3	2.7	16.2
	PSO	4.1	4.0	34.0
	OGVNS	0.3	0.7	6.0
70%	SM_2	8.4	8.0	49.2
	SM_4	7.2	8.7	47.3
	SM_2^+	8.0	8.0	49.2
	SM_+^4	3.3	3.5	24.3
	PSO	5.1	5.9	49.8
	OGVNS	0.5	1.1	7.9

Table 5: Percentage cost error for different levels of manufacturing setup cost.

	Algorithm	Avg. (%)	Std. (%)	Max. (%)
$K^M = 200$	SM_2	4.3	4.5	20.2
	SM_4	3.4	3.6	17.6
	SM_2^+	3.5	4.0	20.2
	SM_+^4	2.3	2.6	13.5
	PSO	4.0	3.1	45.5
	OGVNS	0.4	0.8	6.0
$K^M = 500$	SM_2	5.4	5.2	25.1
	SM_4	3.9	3.9	19.3
	SM_2^+	4.8	4.9	23.7
	SM_+^4	2.1	2.5	12.8
	PSO	4.5	4.1	27.5
	OGVNS	0.3	0.8	8.9
$K^M = 2000$	SM_2	12.8	9.9	49.2
	SM_4	10.9	10.4	47.3
	SM_2^+	12.6	9.9	49.2
	SM_+^4	2.3	3.4	24.3
	PSO	4.4	5.9	49.8
	OGVNS	0.1	0.4	5.2

Table 6: Percentage cost error for different levels of remanufacturing setup cost.

	Algorithm	Avg. (%)	Std. (%)	Max. (%)
$K^R = 200$	SM_2	10.9	9.1	49.2
	SM_4	6.6	7.8	40.2
	SM_2^+	10.0	9.4	49.2
	SM_+^4	1.9	2.1	11.8
	PSO	5.7	5.5	49.8
	OGVNS	0.5	1.0	8.9
$K^R = 500$	SM_2	7.9	6.6	34.7
	SM_4	8.1	8.2	47.3
	SM_2^+	7.3	6.6	34.7
	SM_+^4	3.4	3.2	19.1
	PSO	3.8	4.1	37.4
	OGVNS	0.3	0.7	6.0
$K^R = 2000$	SM_2	3.7	6.0	29.4
	SM_4	3.5	5.7	25.7
	SM_2^+	3.6	5.9	29.4
	SM_+^4	1.4	2.9	24.3
	PSO	3.3	3.5	45.5
	OGVNS	0.1	0.3	3.1

Our findings in Tables 5 and 6 show that the mean absolute percentage error of the solutions was inversely proportional to the manufacturing & remanufacturing cost values. The same findings have also been verified in the recent work by Li et al. in [18], regarding their block-chain based tabu search algorithm for the ELSRP. One possible justification of this pattern is the fact that the manufacturing / re-manufacturing decision seems appealing as the respective setup costs become smaller; and thus more interesting combinations occur that needs to be examined. In other case, if the manufacturing / re-manufacturing cost values become higher, then it is prohibitive either to manufacture or re-manufacture and, therefore, keeping a stock of products in the inventory is the only viable way (that leads to fewer interesting combinations to check).

Table 7: Percentage cost error for different levels of holding cost.

	Algorithm	Avg. (%)	Std. (%)	Maximum (%)
$h^R = 0.2$	SM_2	5.9	8.0	42.9
	SM_4	5.3	8.0	47.3
	SM_2^+	5.8	8.0	42.9
	SM_+^4	1.7	2.5	21.1

Table 7: Percentage cost error for different levels of holding cost.

	Algorithm	Avg. (%)	Std. (%)	Maximum (%)
	PSO	4.5	5.2	49.8
	OGVNS	0.2	0.7	6.2
$h^R = 0.5$	SM_2	7.5	7.7	49.2
	SM_4	6.5	7.6	42.4
	SM_2^+	7.0	7.7	49.2
	SM_+^4	2.3	3.0	24.3
	PSO	4.3	4.5	45.5
	OGVNS	0.3	0.7	5.9
$h^R = 0.8$	SM_2	9.1	7.7	44.4
	SM_4	6.3	7.0	40.3
	SM_2^+	8.1	7.8	44.4
	SM_+^4	2.8	3.0	20.6
	PSO	4.0	3.9	42.9
	OGVNS	0.3	0.8	8.9

The performance of each aforementioned method is measured by the percentage error, which is defined as the percentage gap between the optimal solution (z_{opt}) and the best solution found by our OGVNS algorithm (z_{best}). The mean absolute percentage error per each problem instance is computed as $100 \times (z_{best} - z_{opt})/z_{opt}$. The same way of computing the percentage error was used in the studies by Schulz [30] and Moustaki et al. [21]. Therefore, a fair comparison between these five approaches (i.e., SM_2, SM_4, SM_2^+, SM_+^4, and PSO) is guaranteed. However, it would not be correct to compare the proposed OGVNS algorithm with the recent work by Li et al. [18], since the authors used a different way of computing the mean absolute percentage error (i.e., $100 \times (z_{best} - z_{opt})/z_{best}$), as it is reported in their paper. Regardless of this difference, Li et al. [18] report in their paper that their block-chain tabu search algorithm achieved an average cost error equal to 0.00082% (using this different way of computing the error) on the same set of benchmark problems. On the other hand, our current implementation achieves an average cost error equal to 0.283% on the same set of benchmark problems. However, it should be noted that our proposed implementation has two distinct benefits. First, it does not depend on any other, commercial or not, integer programming solver (such for example as CPLEX which is required in their implementation). For example, in case of larger problems (e.g., $T = 52$ periods), even the CPLEX solver might require a large amount of computational time to provide only a starting solution. Second, our proposed OGVNS approach is considerably faster. The average and maximum

computational time for each test instance is only 8.3 and 30 seconds respectively, whereas the block-chain based tabu search algorithm presented in [18] require 53.4 and 490.4 seconds on average and maximum case, respectively, as it is depicted in Table 8. In case that the same time-limit of 490.4 seconds was also used by our solver instead of 30 seconds then, even lower average optimality gap would have been found. However, the special focus was to provide not only optimal or near-optimal solutions for the ELSRP but also to achieve low computational times, too.

Table 8: Computational times of TS-DLRR [18] and OGVNS.

			Computational times	
		Algorithm	Avg. (sec.)	Max. (sec.)
All instances		TS-DLRR	53.68	490.38
		OGVNS	8.30	30.00
Demand	Small variance	TS-DLRR	58.68	490.38
		OGVNS	8.00	30.00
	Large variance	TS-DLRR	48.69	484.33
		OGVNS	7.60	30.00
Returns	Small variance	TS-DLRR	54.48	490.38
		OGVNS	8.30	30.00
	Large variance	TS-DLRR	52.80	484.33
		OGVNS	8.00	30.00
Return ratio	30%	TS-DLRR	56.51	490.38
		OGVNS	7.50	30.00
	50%	TS-DLRR	51.75	475.93
		OGVNS	8.00	30.00
	70%	TS-DLRR	52.80	484.33
		OGVNS	9.10	30.00
k_M	200	TS-DLRR	54.79	484.33
		OGVNS	11.90	30.00
	500	TS-DLRR	65.12	490.38
		OGVNS	8.80	30.00
	2000	TS-DLRR	41.14	413.68
		OGVNS	3.90	30.00
k_R	200	TS-DLRR	65.38	478.91
		OGVNS	12.20	30.00
	500	TS-DLRR	61.65	490.38
		OGVNS	09.20	30.00
	2000	TS-DLRR	34.02	484.33
		OGVNS	3.30	30.00
h_R	0.2	TS-DLRR	49.50	475.93
		OGVNS	7.20	30.00
	0.5	TS-DLRR	55.18	484.33

Table 8: Computational times of TS-DLRR [18] and OGVNS.

		Computational times	
	Algorithm	Avg. (sec.)	Max. (sec.)
	OGVNS	8.50	30.00
0.8	TS-DLRR	56.37	490.38
	OGVNS	8.90	30.00

5.5. Experimental results on larger instances ($T = 52$)

Although the proposed VNS approach outperforms other metaheuristic approaches in the well-known benchmark set by Schulz [30], we demonstrate the robustness by additional computational experiments with even larger instances. Thus, we have developed a new full factorial study with a common planning horizon T of 52 periods for each instance. This new proposed benchmark was designed with the same parameter values as in Schulz [30], but it features significantly more difficult instances that are more than four times larger. In this new set of benchmark problems, both setup cost parameters k_R and k_M can be equal to 200, 500, and 2000. Additionally, the holding cost h_M for the serviceable items per unit time is set to one, and the holding cost h_R for the recoverable items per unit time can be equal to 0.2, 0.5, and 0.8. Moreover, the customer demands D follow a normal distribution with a mean of 100 units per period, and the amounts of returned products R follow a normal distribution with a mean of 30, 50, and 70 units per period. Furthermore, the coefficient of variation in the normal distributions was set to 10% (small variance) and 20% (large variance).

For each setting of the return and demand values, four instances were randomly drawn. Thus, totally $3^4 \times 2^2 \times 4 = 108$ different instances were created. Efforts were made to solve each one of them using the latest version of the state-of-the-art Gurobi optimizer v5.6.2 within a reasonable amount of time set to 1 hour and tolerance set to 10^{-4}. However, due to the increased computational difficulty, the Gurobi optimizer solved only 54 instances to optimality. Regarding the remaining instances, the Gurobi optimizer has only computed an upper bound of the optimal objective value. Therefore, the proposed RGVNS implementation is compared using all the instances with either optimal solutions or only suboptimal. Our RGVNS variant is compared against Gurobi in order to present the mean optimality error as in the previous Subsection 5.4 and the differences in CPU time.

It is noteworthy that this new benchmark set is more difficult and larger than those that have ever been used in the literature, for the ELSR problem.

Also, it is the largest one with a realistic planning horizon of 52 weeks per year. Apart from the Schulz data set, other smaller data sets include the data set by Li et al. [18] with only 10 instances with dimension $T = 50$, solved to optimality. This new benchmark set is publicly available from the authors web site: `http://users.uom.gr/~sifalera/benchmarks.html`. Interested readers may find all the 108 instances, with either the optimal solutions found by Gurobi in the 54 instances or the currently best known solutions for the remaining 54 instances. Regular updates will be made, once new optimal solutions or new best feasible solutions are found, in the future.

As it is shown in Figure 1, the RGVNS variant achieves an average cost error equal to 2.44% (note that, the optimal values are denoted with bold font). However, this performance was achieved using only 30 seconds in average, compared to 2450.55 seconds that was required by Gurobi. Totally, the Gurobi solver required 81.69 times more computational time for reaching these (either optimal or suboptimal) solutions. Nevertheless, the proposed RGVNS approach computed a feasible solution for all the remaining 54 instances, whereas it was not possible for the Gurobi optimizer to compute the optimal solution.

No	Gurobi 5.6.2		Randomized VNS		
	z_{Best}	CPU (secs)	z_{VNS}	CPU (secs)	Error (%)
1	8698.8	3600.00	8895.2	30	2.26
2	8781.8	3600.00	9185.4	30	4.60
3	**8541.6**	3187.88	8793.8	30	2.95
4	8943.8	3600.00	9391.2	30	5.00
5	9717.0	3600.00	9853.0	30	1.40
6	9962.5	3600.00	10240.5	30	2.79
7	9598.0	3600.00	9955.0	30	3.72
8	9803.5	3600.00	10327.0	30	5.34
9	**10266.2**	1095.08	10573.8	30	3.00
10	10812.8	3600.00	11184.8	30	3.44
11	10290.8	3600.00	10445.4	30	1.50
12	10745.6	3600.00	11027.0	30	2.62
13	13201.0	3600.00	13568.0	30	2.78
14	12131.4	3600.00	12387.4	30	2.11
15	13018.4	3600.00	13403.6	30	2.96
16	11853.2	3600.00	12230.6	30	3.18
17	14236.5	3600.00	14429.5	30	1.36
18	13151.0	3600.00	13660.0	30	3.87
19	13902.0	3600.00	14278.5	30	2.71
20	13495.5	3600.00	13725.0	30	1.70
21	14842.4	3600.00	15063.6	30	1.49
22	14122.2	3600.00	14595.4	30	3.35
23	14561.2	3600.00	14854.4	30	2.01
24	13865.9	3600.00	14428.2	30	4.06
25	**25657.2**	1254.05	26068.2	30	1.60
26	**21247.4**	3327.17	22016.8	30	3.62
27	**24364.4**	403.35	24988.2	30	2.56
28	**20329.0**	2009.03	20479.4	30	0.74
29	**26561.0**	574.85	26888.0	30	1.23
30	**22332.5**	840.33	22851.5	30	2.32
31	**26625.5**	685.63	27326.0	30	2.63
32	**23229.5**	508.37	23945.5	30	3.08
33	**27872.6**	724.77	28482.2	30	2.19
34	**24116.8**	335.65	25426.0	30	5.43
35	**26762.4**	293.72	27823.6	30	3.97
36	**24065.2**	334.73	24785.8	30	2.99
37	**10622.2**	2181.19	10937.4	30	2.97
38	**12011.0**	3550.75	12266.8	30	2.13
39	**10652.2**	199.03	10867.0	30	2.02
40	11741.6	3600.00	12088.6	30	2.96
41	12249.5	3600.00	12585.0	30	2.74
42	13845.0	3600.00	13998.5	30	1.11
43	**12309.0**	569.93	12616.0	30	2.49
44	13627.0	3600.00	13895.0	30	1.97
45	13348.0	3600.00	13584.4	30	1.77
46	**15030.8**	3422.24	15543.4	30	3.41
47	**13635.6**	688.26	13979.0	30	2.52
48	15051.8	3600.00	15788.2	30	4.89
49	15625.4	3600.00	15982.8	30	2.29
50	15447.8	3600.00	15984.8	30	3.48
51	14997.8	3600.00	15409.6	30	2.75
52	15176.6	3600.00	15494.0	30	2.09
53	16782.0	3600.00	17172.0	30	2.32
54	17102.5	3600.00	17757.0	30	3.83

No	Gurobi 5.6.2		Randomized VNS		
	z_{Best}	CPU (secs)	z_{VNS}	CPU (secs)	Error (%)
55	16591.4	3600.00	17149.5	30	3.36
56	17217.9	3600.00	17544.0	30	1.89
57	18047.5	3600.00	18215.6	30	0.93
58	18780.2	3600.00	19158.4	30	2.01
59	17742.8	3600.00	18337.2	30	3.35
60	18646.5	3600.00	19212.6	30	3.04
61	**27623.8**	3200.40	28428.2	30	2.91
62	**24892.6**	2689.20	25562.4	30	2.69
63	**26587.4**	1496.08	28083.0	30	5.63
64	**24252.4**	936.57	24596.2	30	1.42
65	**29328.5**	2200.10	30033.0	30	2.40
66	26961.5	3600.00	27544.0	30	2.16
67	**28484.0**	713.44	29139.0	30	2.30
68	**27019.0**	2115.69	28116.0	30	4.06
69	**30515.4**	1149.58	31117.4	30	1.97
70	28758.4	3600.00	30014.6	30	4.37
71	**29864.6**	658.30	30497.4	30	2.12
72	28195.1	3600.00	30267.6	30	7.35
73	**14443.4**	3.47	14835.4	30	2.71
74	**18364.0**	145.99	18517.4	30	0.84
75	**14954.0**	7.75	15007.0	30	0.35
76	**17857.8**	29.37	17979.2	30	0.68
77	**18546.0**	10.78	18903.5	30	1.93
78	**23069.5**	119.99	23266.5	30	0.85
79	**18657.5**	10.29	18821.0	30	0.88
80	**23329.0**	33.39	23449.5	30	0.52
81	**20999.8**	18.80	21431.8	30	2.06
82	**26519.6**	46.33	27048.6	30	1.99
83	**21114.4**	9.71	21649.4	30	2.53
84	**26162.8**	12.50	27142.6	30	3.75
85	**19646.0**	1423.74	19834.8	30	0.96
86	**22567.6**	1134.49	23128.6	30	2.49
87	**19880.4**	2107.85	19963.6	30	0.42
88	**22483.8**	427.87	22646.6	30	0.72
89	23013.5	3600.00	23190.0	30	0.77
90	27076.0	3600.00	27632.0	30	2.05
91	**22706.5**	775.33	23090.5	30	1.69
92	26754.0	3600.00	26793.0	30	0.15
93	25890.8	3600.00	26118.8	30	0.88
94	30229.2	3600.00	30669.6	30	1.46
95	**26188.8**	2839.44	26935.8	30	2.85
96	**29504.4**	2795.49	30231.2	30	2.46
97	**32952.4**	2250.36	33512.0	30	1.70
98	33332.2	3600.00	33940.0	30	1.82
99	**33072.4**	2309.81	33586.2	30	1.55
100	**33115.0**	2936.20	33539.2	30	1.28
101	36285.9	3600.00	36581.0	30	0.81
102	37205.5	3600.00	38265.5	30	2.85
103	**36173.5**	3128.83	37194.0	30	2.82
104	**36817.0**	3253.05	37861.5	30	2.84
105	38728.0	3600.00	39020.2	30	0.75
106	40310.4	3600.00	41261.8	30	2.36
107	38611.2	3600.00	39281.0	30	1.73
108	**39826.1**	3083.32	40932.4	30	2.78
Avg.		2450.55		30	2.44

Figure 1: Results on larger instances ($T = 52$).

5.6. Discussion

In Table 11, we report the evolution of the optimality gap, (i.e., percentaged minimum error, average error, and maximum error) after each phase of the OGVNS algorithm regarding the set of benchmarks by Schulz [30]. As we can see, the heuristic start that was employed reached an average error of 135.9%. Although this initialization method did not contribute a quality starting solution, it was rapid and thus it permitted us to vastly improve it during the next phase. The VND algorithm is afterward applied to that starting solution, reducing the average error to 8.4%. Last, using the shaking procedure based on a complete random order of neighborhoods, an average error of 0.283% is reached.

Table 11: Percentage cost error after each OGVNS phase.

	Min. (%)	Avg. (%)	Max. (%)
Heuristic start	13.7	135.9	385.3
VND	0.0	8.4	50.9
Shaking phase	0.0	0.3	8.9

Although a single significant improvement is sometimes more effective than several minor improvements, we decided to count the number of improvements caused by each neighborhood structure instead of the magnitude of the improvements per neighborhood structure. This way, the most often used neighborhood structures were placed first in the order of the VND algorithm, in order to accelerate the search. As it was shown, neighborhoods N_{15}, N_4, N_2, and N_1 contributed the most in the VND method with 29.5%, 26.0%, 12.1%, and 11.1%, respectively. This information resulted after an extensive preliminary testing of the proposed OGVNS algorithm on all instances. Afterward, the neighborhoods were sorted according to the number of their local search improvements. This order was finally used by the OVND algorithm on all instances. An equally interesting option would be to test a different order based on the magnitude of the improvements per neighborhood structure instead of the numbers of improvements.

6. Conclusions and future work

In this paper we have addressed the economic lot sizing problem with product returns and recovery in reverse logistics. We have proposed two novel

VNS metaheuristic algorithms that employ new strategies for both the local search step and the shaking process. Our VNS approach tackles the NP-hard ELSR problem efficiently and outperforms the state-of-the-art heuristic methods from the literature (SM variants and PSO). We presented several new neighborhoods for this combinatorial optimization problem and an efficient local search method for exploring them. Finally, we also described a new simple heuristic initialization method for this problem. Based on extensive numerical testing, using a recent large set of benchmark problems with 6480 instances, our approach was able to achieve an average optimality gap equal to 0.283% within average 8.3 seconds. Finally, a new benchmark set with the currently largest instances (52 periods) in the literature has been developed and made publicly available. The results got by using this latter set showed that the proposed VNS approach is quite efficient in solving large problems with a small optimality gap.

A subject for a future work is the adaptation of OGVNS to multi-product dynamic lot sizing problems in closed-loop supply chain [33], and other combined problems of inventory and network optimization [31]. Furthermore, it would be interesting to develop a parallel implementation of our proposed metaheuristic OGVNS algorithm (e.g., [8, 11]).

Acknowledgement

The authors would like to express their deep appreciation to Dr. T. Schulz for providing the complete test set that was used in the experimental part of the paper. Also, the work by the third author was conducted at National Research University Higher School of Economics and supported by RS grant 14-41-00039. Finally, the authors gratefully acknowledge the helpful suggestions of two anonymous reviewers.

References

[1] E. Akcali, S. Cetinkaya, Quantitative models for inventory and production planning in closed-loop supply chains, International Journal of Production Research 49 (2011) 2373–2407.

[2] B. Almada-Lobo, R.J. James, Neighbourhood search meta-heuristics for capacitated lot-sizing with sequence-dependent setups, International Journal of Production Research 48 (2010) 861–878.

[3] B. Almada-Lobo, J.F. Oliveira, M.A. Carravilla, Production planning and scheduling in the glass container industry: A VNS approach, International Journal of Production Economics 114 (2008) 363–375.

[4] M.F. Baki, B.A. Chaouch, W. Abdul-Kader, A heuristic solution procedure for the dynamic lot sizing problem with remanufacturing and product recovery, Computers & Operations Research 43 (2014) 225–236.

[5] J. Beltran, D. Krass, Dynamic lots sizing with returning items and disposals, IIE Transactions 34 (2002) 437–448.

[6] N. Brahimi, S. Dauzere-Peres, Single item lot sizing problems, European Journal of Operational Research 168 (2006) 1–16.

[7] L. Buschkühl, F. Sahling, S. Helber, H. Tempelmeier, Dynamic capacitated lot-sizing problems: a classification and review of solution approaches, OR Spectrum 32 (2010) 231–261.

[8] T. Davidović, T.G. Crainic, MPI parallelization of variable neighborhood search, Electronic Notes in Discrete Mathematics 39 (2012) 241–248.

[9] A. Divsalar, P. Vansteenwegen, D. Cattrysse, A variable neighborhood search method for the orienteering problem with hotel selection, International Journal of Production Economics 145 (2013) 150–160.

[10] R. Durstenfeld, Algorithm 235: Random permutation, Communications of the ACM 7 (1964) 420.

[11] M. Eskandarpour, S.H. Zegordi, E. Nikbakhsh, A parallel variable neighborhood search for the multi-objective sustainable post-sales network design problem, International Journal of Production Economics 145 (2013) 117–131.

[12] R. Fisher, F. Yates, Example 12, Statistical Tables, London (1938).

[13] R. Geyer, L.N. Van Wassenhove, A. Atasu, The economics of remanufacturing under limited component durability and finite product life cycles, Management Science 53 (2007) 88–100.

[14] B. Golany, J. Yang, G. Yu, Economic lot–sizing with remanufacturing options, IIE Transactions 33 (2001) 995–1003.

[15] S. Hanafi, J. Lazić, N. Mladenović, C. Wilbaut, I. Crévits, New variable neighbourhood search based 0-1 mip heuristics, Yugoslav Journal of Operations Research - (2014) –. (to appear).

[16] W.L. Ijomah, A model-based definition of the generic remanufacturing business process (2002).

[17] D.E. Knuth, The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, volume 2, Addison-Wesley, 1969.

[18] X. Li, F. Baki, P. Tian, B.A. Chaouch, A robust block-chain based tabu search algorithm for the dynamic lot sizing problem with product returns and remanufacturing, Omega 42 (2014) 75–87.

[19] R.T. Lund, Remanufacturing, Technology Review 87 (1984) 19–23.

[20] N. Mladenović, P. Hansen, Variable neighborhood search, Computers & Operations Research 24 (1997) 1097–1100.

[21] E. Moustaki, K.E. Parsopoulos, I. Konstantaras, K. Skouri, I. Ganas, A first study of particle swarm optimization on the dynamic lot sizing problem with product returns, in: Proceedings of the XI Balkan Conference on Operational Research (BALCOR 2013), Belgrade-Zlatibor, Serbia, 2013, pp. 348–356.

[22] G. Nenes, S. Panagiotidou, R. Dekker, Inventory control policies for inspection and remanufacturing of returns: A case study, International Journal of Production Economics 125 (2010) 300–312.

[23] P. Piñeyro, O. Viera, Inventory policies for the economic lot–sizing problem with remanufacturing and final disposal options, Journal of Industrial and Management Optimization 5 (2009) 217–238.

[24] P. Piñeyro, O. Viera, The economic lot-sizing problem with remanufacturing and one-way substitution, International Journal of Production Economics 124 (2010) 482–488.

[25] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical recipes in Fortran 77: The art of scientific computing, volume 1, 2 ed., Cambridge University Press, 1992.

[26] J. Puchinger, G. Raidl, Bringing order into the neighborhoods: relaxation guided variable neighborhood search, Journal of Heuristics 14 (2008) 457–472.

[27] M.J. Retel Helmrich, R. Jans, W. van den Heuvel, A.P. Wagelmans, Economic lot-sizing with remanufacturing: complexity and efficient formulations, IIE Transactions 46 (2014) 67–86.

[28] K. Richter, M. Sombrutzki, Remanufacturing planning for the reverse wagner/whitin models, European Journal of Operational Research 121 (2000) 304–315.

[29] K. Richter, J. Weber, The reverse wagner/whitin model with variable manufacturing and remanufacturing cost, International Journal of Production Economics 71 (2001) 447–456.

[30] T. Schulz, A new silver–meal based heuristic for the single–item dynamic lot sizing problem with returns and remanufacturing, International Journal of Production Research 49 (2011) 2519–2533.

[31] A. Sifaleras, Minimum cost network flows: Problems, algorithms, and software, Yugoslav Journal of Operations Research 23 (2013) 3–17.

[32] A. Sifaleras, Classification of network optimization software packages, in: M. Khosrow-Pour (Ed.), Encyclopedia of Information Science and Technology, 3rd ed., IGI Global, Hershey, PA, 2015, pp. 7054–7062.

[33] A. Sifaleras, I. Konstantaras, General variable neighborhood search for the multi-product dynamic lot sizing problem in closed-loop supply chain, in: 3rd International Conference on Variable Neighborhood Search (VNS'14), Djerba, Tunisia, 2014.

[34] A. Sifaleras, D. Urošević, N. Mladenović, Preface: EURO Mini Conference (MEC XXVIII) on Variable Neighborhood Search, Electronic Notes in Discrete Mathematics 39 (2012) 1–4.

[35] O. Tang, R.H. Teunter, Economic lot scheduling problem with returns, Production and Operations Management 15 (2006) 488–497.

[36] R.H. Teunter, Z.P. Bayindir, W. Van den Heuvel, Dynamic lot sizing with product returns and remanufacturing, International Journal of Production Research 44 (2006) 4377–4400.

[37] R.H. Teunter, K. Kaparis, O. Tang, Multi–product economic lot scheduling problem with separate production lines for manufacturing and remanufacturing, European Journal of Operational Research 191 (2008) 1241–1253.

[38] R.H. Teunter, O. Tang, K. Kaparis, Heuristics for the economic lot scheduling problem with returns, International Journal of Production Economics 118 (2009) 323–330.

[39] D. Urošević, Variable neighborhood search for maximum diverse grouping problem, Yugoslav Journal of Operations Research 24 (2014) 21–33.

[40] H.M. Wagner, T.M. Whitin, Dynamic version of the economic lot size model, Management Science 5 (1958) 88–96.

[41] Y. Xiao, I. Kaku, Q. Zhao, R. Zhang, A reduced variable neighborhood search algorithm for uncapacitated multilevel lot-sizing problems, European Journal of Operational Research 214 (2011) 223–231.

[42] Y. Xiao, I. Kaku, Q. Zhao, R. Zhang, A variable neighborhood search based approach for uncapacitated multilevel lot-sizing problems, Computers and Industrial Engineering 60 (2011) 218–227.

[43] Y. Xiao, R. Zhang, Q. Zhao, I. Kaku, Y. Xu, A variable neighborhood search with an effective local search for uncapacitated multilevel lot-sizing problems, European Journal of Operational Research 235 (2014) 102–114.

[44] J. Yang, B. Golany, G. Yu, A concave–cost production planning problem with remanufacturing options, Naval Research Logistics 52 (2005) 443–458.

[45] S. Zanoni, A. Segerstedt, O. Tang, Multiproduct economic lot scheduling problem with manufacturing and remanufacturing using a basic period policy, Computers & Industrial Engineering 62 (2012) 1025–1033.