Kärcher, Jens; Meyr, Herbert

A machine learning approach for predicting the best heuristic for a large scaled Capacitated Lotsizing Problem

OR Spectrum

OR Spectrum (2025) 47:889–931
https://doi.org/10.1007/s00291-024-00804-9

ORIGINAL ARTICLE

A machine learning approach for predicting the best heuristic for a large scaled Capacitated Lotsizing Problem

Jens Kärcher[1] · **Herbert Meyr**[1]

Received: 12 November 2023 / Accepted: 9 December 2024 / Published online: 4 February 2025

Abstract

For some NP-hard lotsizing problems, many different heuristics exist, but they have different solution qualities and computation times depending on the characteristics of the instance. The computation times of the individual heuristics increase significantly with the problem size, so that testing all available heuristics for large instances requires extensive time. Therefore, it is necessary to develop a method that allows a prediction of the best heuristic for the respective instance without testing all available heuristics. The Capacitated Lotsizing Problem (CLSP) is chosen as the problem to be solved, since it is a fundamental model in the field of lotsizing, well researched and several different heuristics exist for it. The CLSP addresses the problem of determining lotsizes on a production line given limited capacity, product-dependent setup costs, and deterministic, dynamic demand for multiple products. The objective is to minimize setup and inventory holding costs. Two different forecasting methods are presented. One of them is a two-layer neural network called CLSP-Net. It is trained on small CLSP instances, which can be solved very fast with the considered heuristics. Due to the use of a fixed number of wisely chosen features that are relative, relevant, and computationally efficient, and which leverage problem-specific knowledge, CLSP-Net is also capable of predicting the most suitable heuristic for large instances.

Keywords Algorithm selection problem · Machine learning · Neural network · Heuristics · Capacitated lotsizing problem · Mixed-integer linear programming

✉ Jens Kärcher
Jens.Kaercher@uni-hohenheim.de

Herbert Meyr
H.Meyr@uni-hohenheim.de

[1] Department of Supply Chain Management, University of Hohenheim, 70593 Stuttgart, Germany

1 Introduction

In recent times Machine Learning (ML) and Operations Research (OR) get more and more integrated. Examples include the approximation of a computationally intensive fitness function (see e.g. Lucas et al. 2020), the identification of good initial solutions (see e.g. Bengio et al. 2020), the control of the entire search process (see e.g. Arnold and Sörensen 2019) or the setting of reasonable parameters depending on the algorithm used (see e.g. Zennaki and Ech-Cherif 2010). The goal is always to improve the algorithms with respect to their solution quality, convergence rate and robustness. A recent literature review can be found in Karimi-Mamaghan et al. (2022).

Another example of the integration of ML methods represents the automatic selection of a most suitable algorithm for the solution of an instance of an optimization problem. If exact solution methods are not able to solve larger instances in a given time limit, faster heuristics are required. However, when multiple heuristics are available, it is often unclear which one will yield the highest solution quality for a particular instance. Ideally, one would like to test all the heuristics to select the one that produces the best solution. As shown in Fig. 1, the computation times of the heuristics 1–3 increase strongly with the problem size, which is why a computation of all available heuristics would require a huge amount of time, especially for larger instances. In application areas where an optimization task has to be solved very fast (e.g. real-time-scheduling), usually not all heuristics can be tested. Therefore, a prediction of the best heuristic for the considered instance would be very helpful. This problem is referred to in the literature as Algorithm Selection Problem

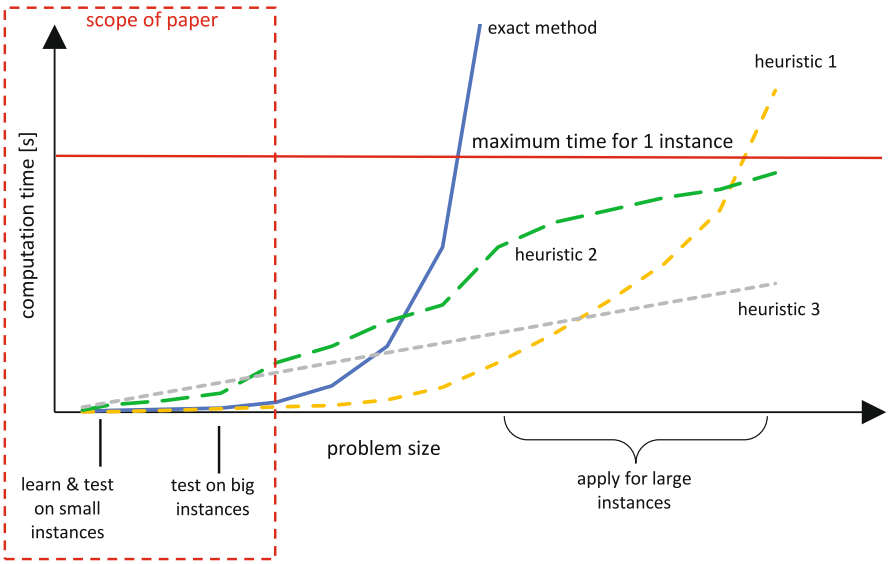


Fig. 1 Basis idea - Learn on small instances to predict the best heuristic for a large instance if exact methods are too time-consuming

(see Karimi-Mamaghan et al. 2022), Automated Algorithm Selection (see Kerschke et al. 2019; Dantas and Pozo 2018) or Adaptive Recommendation Model (see Chu et al. 2019).

According to Rice (1976), the Algorithm Selection Problem (ASP) can be divided into four elements. For a given optimization problem, there are a number of different instances (problem space). Each instance can be described by quantitative features (feature space). For solving the instances, different solution methods are available (algorithm space), whose performance can be evaluated for each instance using key figures (performance space).

In the following we concentrate on the ASP and apply it to a problem, that is very well known in OR: the lotsizing problem. Lotsizing problems are crucial and continue to be studied today in various forms. A fundamental model in this field is the Capacitated Lotsizing Problem (CLSP), as it serves as a basis for numerous extensions and variations. The CLSP addresses the challenge of determining optimal lotsizes on a production line under constraints such as limited capacity, product-dependent setup costs, and deterministic, dynamic demand for multiple products. The primary objective is to minimize both setup and inventory holding costs. We consider the CLSP fundamental for several reasons. First, multi-level CLSP models (see e.g. Buschkühl et al. 2010), which have evolved from the basic CLSP, are crucial for job shop production, where scheduling is typically addressed subsequent to lotsizing. Second, the development of simultaneous lotsizing and scheduling problems (see e.g. Copil et al. 2017) has also originated from the CLSP framework. In these models, microperiods for sequencing are embedded within the macroperiod structure of the CLSP, allowing for more precise and efficient scheduling. Furthermore, the CLSP is already very well researched, which is why there are many heuristics, some of which have very short computation times for small instances (see e.g. Karimi et al. 2003; Jans and Degraeve 2007, 2008; Buschkühl et al. 2010).

Given a set of heuristics for the CLSP, the question is how to learn which type of CLSP instance works best with which heuristic. Our approach uses a data set that consists of small CLSP instances, but still has different characteristics. For this training set, all considered heuristics can be tested quickly due to the short computation times, so that it is known which CLSP instance works best with which heuristic (input–output pairs). If in the next step patterns can be derived from the input–output pairs, these insights can be used to predict the best heuristic for a very large CLSP instance for which no best heuristic is yet known. This approach can be regarded as a classification task and belongs to the domain of supervised learning within the field of ML. One particularly successful ML technique for addressing classification tasks is the utilisation of an artificial neural network (ANN) (see Goodfellow et al. 2016, p. 105 f.). To be able to check whether the forecasting method to be developed can also be applied to very large CLSP instances, it must be tested using CLSP instances that are sufficiently bigger than the small instances of the training set (see Fig. 1, dashed red rectangle).

A forecasting method is to be developed that ensures instances can be classified regardless of their size without the need to adapt the forecasting method. We show that and how knowledge gained from small instances can be effectively applied to larger instances. For this purpose, suitable features must be derived to describe

the CLSP instances, what, to the best of the authors' knowledge, has not yet been done in the literature for a lotsizing problem. These features, which can be computed from the input data of a CLSP instance, must be quickly computable, able to quantitatively capture and express essential aspects of a CLSP instance in a single number (relevance), and independent of the problem instance's dimensionality (e.g., in terms of the number of products or periods considered) (relativity). In order to train the forecasting methods, a huge set of CLSP instances is required, covering a wide range of possible solutions. By applying the considered heuristics to the data set, a side effect is a comprehensive comparison of the selected heuristics, which to the best of the authors' knowledge has not yet been done in this way in the literature. Furthermore, insights can be gained under which conditions the selected heuristics of the CLSP perform better.

This paper is structured as follows. First, in Sect. 2, a brief literature review on the use of neural networks to solve OR problems is given. Section 3 describes the methodology of the research. Subsequently, in Sect. 4, the CLSP is introduced and a selection of heuristics for the CLSP are explained. Section 5 deals with the generation of CLSP instances and the training set. The following Sect. 6 is dedicated to the design, configuration and training of two different forecasting methods that can be applied to select the best heuristic. Finally, in Sect. 7 the presented forecasting methods are applied for testing purposes to different data sets.

2 Literature review

In the following, it is shown to what extent artificial neural networks are already applied in the literature for the selection of solution methods. Subsequently, publications are presented that deal with solving lotsizing problems using an ANN.

2.1 Selection of solution methods using an ANN

In the following, only publications are presented whose ASP-approach is based on an ANN.

Smith-Miles (2008) presents an ASP for the Quadratic Assignment Problem (QAP) using three metaheuristics. The instances are described using the features proposed by Stützle (2006). The author develops three ANNs. Two of the three nets pursue the goal of predicting the performance of the respective metaheuristic in terms of the percentage deviation from the best known objective function value. The third network, on the other hand, solves a classification task with the goal of determining the best metaheuristic depending on the instance.

In the work of Smith-Miles et al. (2010), an ASP for the Travelling Salesman Problem (TSP) is presented. In order to describe the instances, 12 features are derived from the two-dimensional location information of the cities. One of the two proposed ANNs predicts the search effort required by each of the two available algorithms to find the best solution. The other ANN assigns to each algorithm the

probability that it finds the best solution for the considered instance with the least search effort.

In contrast, the goal of Kanda et al. (2011) is to predict a performance ranking of four metaheuristics depending on the TSP instance. The authors propose three different ANNs. In the work of Kanda et al. (2012), the authors revisit the approach of determining a ranking of solution algorithms for the TSP. The authors present two ANNs. The first ANN predicts the ranking position using five output nodes, each representing one of five available metaheuristics. The second network performs multiple classification of metaheuristics depending on the instance.

Kanda et al. (2016) compare three different ML methods for ranking five metaheuristics to solve a TSP instance. In addition to the ANN presented in Kanda et al. (2012), a k-Nearest Neighbor formulation from Brazdil et al. (2009) and an adapted version of the predictive clustering tree from Vens et al. (2008) are used.

In the work of Miranda et al. (2018), an ASP for the MaxSAT problem is modeled as a classification task. There are three metaheuristics to choose from for solving the MaxSAT problem, each of which is applied with four different parameter settings. The focus of the work is on the derivation and evaluation of appropriate features to describe the instances. Also Sadeg et al. (2021) establish an ASP for the MaxSAT problem. However, the features proposed in Nikolić et al. (2013) are used to describe the instances, since they can be computed very fast.

In addition to the QAP, TSP, and MaxSAT problems, an ASP for the Vehicle Routing Problem with Time Windows (VRPTW) is also established in the literature. Gutierrez-Rodríguez et al. (2019) present an ANN that assigns instances to one of four possible metaheuristics depending on their characteristics. The authors develop and test two different feature sets.

2.2 ANN in lotsizing

In the literature there are several papers on lotsizing problems where artificial neural networks are used as part of the solution process.

Zwietering et al. (1991) investigate the applicability of an ANN to solve a dynamic uncapacitated single-product lotsizing problem with a rolling planning horizon. In the work of Aarts et al. (2000), a decomposition approach for a dynamic real-time single-product lotsizing problem with overtime is presented. The authors use an ANN to predict regeneration points within a schedule. Since the schedule between two regeneration points is independent of the rest of the schedule, a rescheduling can subsequently be performed taking into account updated conditions.

Gaafar and Choueiki (2000) create two different networks for two different planning horizons of a dynamic single-product lotsizing problem. In the input layer, the first two nodes include the fixed costs and the holding cost rate. The remaining input nodes contain the demand data. The number of output nodes corresponds to the length of the planning horizon. Each node takes either the value zero or one, resulting in a setup or order pattern. In contrast, Radzi et al. (2006) generate one ANN each for three heuristics of the dynamic single-product lotsizing problem. The structure of the nets is similar to that of Gaafar and

Choueiki (2000). However, the networks aim to approximate the way the considered heuristics work. Further, Megala and Jawahar (2006) propose a genetic algorithm and a Hopfield neural network to solve a dynamic single-product lotsizing problem with a capacity constraint and a discount price structure.

Wong et al. (2012) present a two-stage approach for a stochastic dynamic single-product lotsizing problem, which is based on an ANN and an adapted Ant Colony algorithm. The ANN is used to learn the relationship between two decision variables and the expected total cost. Senyiğit et al. (2013) also consider a stochastic dynamic single-product lotsizing problem and compare four neural networks which are trained with different learning approaches and are used to predict the total cost of the chosen order policy.

In the work of Senyiğit and Atici (2013), 40 different configurations of an ANN are studied for solving a dynamic single-product lotsizing problem without capacity constraints. Eight different activation functions are combined with five different learning rules. The networks include 15 input nodes, a hidden layer with 30 nodes, and 12 output nodes. The input nodes receive as input the demand of 12 periods, the holding cost rate, the ordering or setup cost rate, and the variable cost rate of the considered product. Each output node specifies the probability that an order is to be placed.

Lee et al. (2001) address a lotsizing and scheduling problem with multiple machines. Two separate ANNs are developed. The first ANN is constructed in a way that the order of the lots can be derived from the neural weight matrix of the ANN. In contrast, the second ANN determines a lotsize parameter (between 0 and 1), which can be used to calculate the lotsize in the next step.

Rohde (2004) decomposes the planning task of simultaneous lotsizing and scheduling over the total planning horizon into two levels. Using an ANN, lotsize stocks and setup times are anticipated at the lower short-term planning level in order to incorporate them as a reaction at the upper planning level within the Master Production Schedule.

Moreover, several publications explore lotsizing problems using Deep Reinforcement Learning (DRL). Here, a policy or a value function is learned by a neural network (see e.g. Paternina-Arboleda and Das 2005). Park and Jang (2022) develop a novel scheduling method for the discrete lotsizing problem (DLSP) with multiple products on a single machine using reinforcement learning. They formalize the DLSP as a sequential decision-making problem using the Markov decision process and propose a scheduling method based on a deep Q-network. Another DRL algorithm is applied by Rummukainen and Nurminen (2019) to the stochastic economic lot scheduling problem. They use proximal policy optimization (PPO) to identify optimal parameter values. Van Hezewijk et al. (2023) further explore the scalability of PPO for large instances of the stochastic capacitated lotsizing problem.

Overall, up to the authors' knowledge, ANNs have not yet been applied to solve an ASP for lotsizing problems, and particularly not for the CLSP.

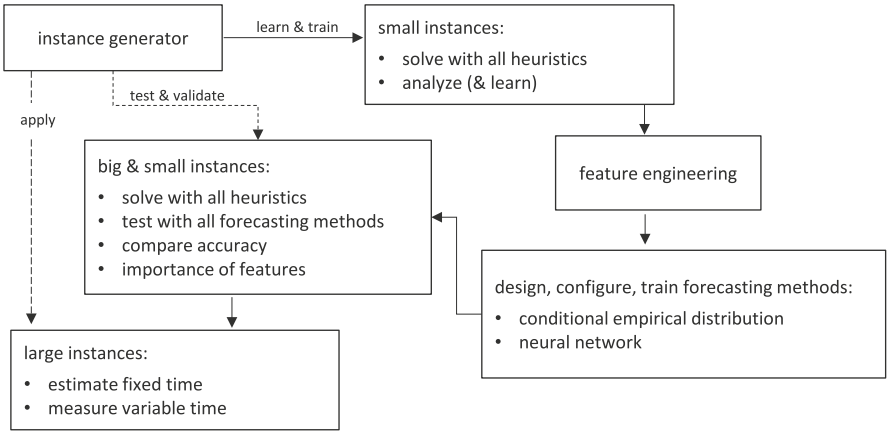


Fig. 2 Research Methodology

3 Research methodology

To address an algorithm selection problem for the CLSP, a comprehensive data set is required for configuring, training, and testing the forecasting methods. Due to the limited availability of CLSP instances in the literature, the first step involves generating CLSP instances using an instance generator. It is essential that the resulting data set includes a wide range of potential scenarios. In addition, these instances should have problem sizes small enough to ensure that they can be solved quickly using the selected heuristics. An ASP only occurs if none of the heuristics dominates the others. Thus we concentrate on some exemplary heuristics whose performance for different types of problem instances is probably varying. Once the small CLSP instances are solved using the selected heuristics, initial insights into the heuristics' performance can be gained, and the best heuristic for each instance can be identified. To build the training set, appropriate features that describe the CLSP instances must be derived and calculated (feature engineering).

In the next step, two forecasting methods are developed and trained. The first approach is based on empirical frequency distributions and provides some insight into the importance of features. The second approach is an artificial neural network called CLSP-Net. Both methods try to identify as many patterns as possible in the training set and take them into account when selecting the heuristic.

Subsequently the forecasting methods are applied for testing purposes to a data set with small instances as well as to a data set with big instances. It is investigated which of the forecasting methods achieves the highest accuracy. Furthermore, it is verified whether the forecasting methods (especially CLSP-Net) are

able to successfully classify big and previously unknown CLSP instances without any further training phase.

To compare the computation times of the different approaches, additional CLSP instances are generated using the instance generator. These instances are significantly larger in problem size compared to the previously considered CLSP instances. The research methodology is depicted in Fig. 2. As a final step, the test bed is enriched with CLSP instances from the literature to gain further insights into the performance of the forecasting methods, especially CLSP-Net.

4 The capacitated lotsizing problem (CLSP)

The following section is addressed to the CLSP. Section 4.1 contains the mathematical formulation of the model. In Sect. 4.2 a selection of heuristics for the CLSP, which are employed in the numerical study, is presented and differences between the heuristics are shown.

4.1 Model formulation

The Capacitated Lotsizing Problem (CLSP) represents a dynamic, single-stage, multi-product problem with limited production capacity. In each period t ($t = 1, \ldots, T$), production lots for multiple products j ($j = 1, \ldots, J$) can be placed. For this purpose, a production line is available, which has a limited time capacity K_t in each period t, which must not be exceeded. The production of one unit of a product j requires a_j time units.

The demand of a product j in period t is defined by d_{jt} and must be exactly satisfied in each period. Late delivery is not possible. If the demand of a product j in period t cannot be produced in period t due to the limited capacity of the production line, it is possible to produce it in previous periods. In this case, there is a holding cost of hc_j per period for holding one unit of product j. In order to produce product j in period t, the production line must be set up, which incurs setup costs of sc_j. For each period t and each product j, the continuous non-negative variables X_{jt} and I_{jt} and the binary setup variable Y_{jt} are introduced. X_{jt} represents the production quantity of product j in period t. In contrast, I_{jt} denotes the inventory level of product j at the end of period t.

Conservation of the setup-state across periods is not possible. Moreover, the sequence of lots within a period is not determined. The CLSP presents an NP-hard problem (see Bitran and Yanasse 1982; Chen and Thizy 1990).

The mathematical formulation of the model is as follows (see Günther 1987):

Model CLSP

Objective function:

$$Min \quad \sum_{j=1}^{J} \sum_{t=1}^{T} hc_j \cdot I_{jt} + \sum_{j=1}^{J} \sum_{t=1}^{T} sc_j \cdot Y_{jt} \tag{1}$$

Constraints:

$$I_{j,t-1} + X_{jt} - d_{jt} = I_{jt} \qquad \forall \; j,t \tag{2}$$

$$\sum_{j=1}^{J} X_{jt} \cdot a_j \leq K_t \qquad \forall \; t \tag{3}$$

$$X_{jt} \leq Y_{jt} \cdot \sum_{p=t}^{T} d_{jp} \qquad \forall \; j,t \tag{4}$$

The objective (1) is to minimize the sum of holding and setup costs. In the inventory balance equation (2), it is ensured that the demand for each product j must be fulfilled mandatorily in each period t. In contrast, restriction (3) ensures for each period t that the capacity of the production line is not exceeded. Due to restriction (4), production of product j in period t is only possible if the production line is set up for product j in period t. In this basic CLSP version, setup times are not considered.

4.2 CLSP heuristics

Over the years, many heuristics for the CLSP have been developed. Structured overviews of solution methods for single-stage, capacitated lotsizing problems can be found, for example, in Maes and Van Wassenhove (1988), Karimi et al. (2003), Jans and Degraeve (2007), Buschkühl et al. (2010) and Copil et al. (2017).

Buschkühl et al. (2010) classify the solution approaches to the multi-level CLSP into the five categories »mathematical programming heuristics«, Lagrangean heuristics »decomposition and aggregation heuristics«, »metaheuristics«, and »problem-specific greedy heuristics«. In order to classify greedy heuristics, they propose the following set of criteria.

First, greedy heuristics — also referred to as common-sense heuristics — can basically be divided into two types. *Constructive heuristics* start with an empty solution matrix whose size is determined by the number of periods and products of the considered lotsizing problem. Then, the solution matrix is completed period by period until a feasible solution is obtained.

Table 1 Comparison of the considered heuristics (see Buschkühl et al. 2010)

Heuristic	Type	Direction	Feasibility check	Cost criterion
Eisenhut (1975)*	C	F	–	PPB
Lambrecht and Vanderveken (1979)	C	F	FB	SM
Dixon and Silver (1981)	C	F	LA	SM
Dogramaci et al. (1981)	I (L4L)	A	SP	LTC
Günther (1987)	I (L4L)	F	LA	CPP (G)

Notation:

C = Constructive, I = Improving, L4L = Lot-for-Lot

F = Forward, A = Considering the complete Schedule at each Step

FB = Feedback, LA = Look-Ahead, SP = Smoothing Procedure

CPP = Cost per Setup and Production Time, G = Groff, LTC = Least Total Cost, PPB = Part Period Balancing, SM = Silver-Meal

* Not used in numerical study

In contrast, *improving heuristics* are based on an initial solution, which can be generated, for example, by applying the lot-for-lot rule.[1] Since the initial solution for the considered lotsizing problem is usually infeasible or has a poor solution quality, it is adjusted by using shifting strategies until a feasible and better solution is reached.

Another criterion is the planning direction. This can be *forward* or *backward*, whereby only the adjacent periods are considered when determining the individual lotsizes. Another possibility is that in each step of the heuristic, *all planning periods* of the current production plan are considered, so that a specific planning direction is not predetermined. This type of planning can only occur in combination with an improving heuristic.

Furthermore, differences can be observed between the heuristics with respect to the procedure for checking the feasibility of a solution. Heuristics that follow a *look-ahead* strategy perform a feasibility check in each iteration. Here, the lotsizes are adjusted, if necessary, by a corresponding minimum stock quantity, which is required to avoid a capacity violation in later periods. In contrast, heuristics with a *feedback* strategy apply the feasibility check at the end of the process to the possibly infeasible production schedule. In the case of a capacity bottleneck, selected production lots or partial quantities are shifted forward to earlier periods until the resulting production schedule is feasible. A third way to ensure the feasibility of a solution can be recognized in the *smoothing* strategy, by which the existing infeasible solution is successively transformed into a feasible solution using several subprocedures.

Last but not least, the heuristics can also be distinguished on the basis of the applied cost criterion, which is used in the context of lot shifts to select the corresponding lot. The best-known criteria include *Silver Meal* (see Silver and Meal

[1] In the case of a lot-for-lot rule, the actual demand for each period is scheduled as the lotsize for each product.

1969, 1973), *Groff* (see Groff 1979), *Part Period Balancing* (see DeMatteis 1968), *Cost per Setup and Production Time* (see van Nunen and Wessels 1978) and *Least Total Cost* (see Dogramaci et al. 1981).

In the following section, the heuristics of Lambrecht and Vanderveken (1979), Dixon and Silver (1981), Dogramaci et al. (1981) and Günther (1987) are presented and their differences are briefly explained. These heuristics are used in the numerical study. The classification based on the presented criteria can be found in Table 1.

As previously mentioned, there are many other types of heuristics that are specifically designed to solve the CLSP. Examples of such approaches include decomposition-based heuristics (see e.g. Kirca and Kökten 1994), Lagrangian heuristics and metaheuristics (see e.g. Jans and Degraeve 2007). However, we have not selected all possible heuristics for inclusion in our study. The intention of this research is not to provide a comprehensive comparison of all CLSP variants and heuristics. Instead, we concentrate on the most relevant heuristics within the category of greedy heuristics as categorised by Buschkühl et al. (2010). In the selection process, we ensured a balanced representation of both constructive and improvement heuristics of a similar age. Those additionally needed to be wide-spread enough in their approach (see Table 1), well enough documented and without too many parameter choices being necessary. Consequently, we have chosen these four heuristics because they are sufficiently distinct from one another, while ensuring that no individual heuristic dominates the others.

The method of *Lambrecht and Vanderveken* (1979) is based on the heuristic of Eisenhut (1975), in which a lotsizing schedule is determined period by period in ascending order. For this purpose, Eisenhut (1975) uses a demand matrix which is reduced by the last considered period after each iteration. In each period, at least the current demand is initially scheduled. If capacity is still available, a check is made to see whether costs can be reduced by shifting future demands to earlier periods. A cost criterion based on the part-period balancing criterion is used to prioritize the products (see Eecken et al. 1975).

The problem with the Eisenhut (1975) method is that it does not check whether the defined lotsizes will cause capacity bottlenecks in later periods. Furthermore, since a subsequent adjustment of individual lots in the procedure is not possible, future bottlenecks cannot be eliminated. This is why the heuristic of Eisenhut (1975) may abort, although a feasible solution would be possible (see Lambrecht and Vanderveken 1979, p. 322).

Lambrecht and Vanderveken (1979) try to fix this problem. In their heuristic, the second step checks whether the capacity of the current planning period is sufficient to completely meet the period demand. If there is a capacity shortage, a feedback procedure is triggered in which the lotsizes of previous periods are changed until the bottleneck is eliminated. Another significant difference to the heuristic of Eisenhut (1975) is the use of a different cost criterion, which Lambrecht and Vanderveken (1979) derive from the Silver-Meal criterion.

The method of *Dixon and Silver* (1981) represents a forward constructive method, in which in each period first the still outstanding current period demand is fulfilled. If capacity remains in a period, production lots of individual products are increased by whole period requirements of subsequent periods to save additional setup costs.

Furthermore, it is checked whether the current plan will result in bottlenecks in later periods. If this is the case, lots of the current planning period are also increased

by partial quantities of a future period demand, which is generally accompanied by an increase in costs. When selecting the lots to be shifted, products are prioritized according to the Silver-Meal criterion. The Silver-Meal criterion is based on the fact that for the optimal lotsize in the case of a single-product lotsizing problem, the mean cost per time unit is minimal (see Silver and Meal 1969, 1973).

The four-step algorithm of *Dogramaci et al.* (1981) is a heuristic that differs significantly from the previously explained forward methods. Instead of scheduling production quantities period by period, at the beginning of the heuristic all production quantities are set to the level of the respective demand without regard to possible capacity violations (lot-for-lot solution). After that, it is tried to reduce costs by merging lots. Here, preference is given to those lots that provide the highest overall cost saving. The next step aims to eliminate capacity violations and to achieve a valid solution. Therefore lots are again shifted to earlier periods. Afterwards, lots may be moved to earlier periods one more time to further reduce costs. This is done on the condition that the resulting solution is also feasible. Last but not least, a fine-tuning is performed to investigate possibilities of shifting lots to later periods with regard to additional cost reductions. A major difference from the methods presented so far is that when selecting the lot to be shifted, all lots from all periods are considered simultaneously. This significantly increases the computational effort.

The heuristic of *Günther* (1987) is very similar to the procedure of Dixon and Silver (1981). However, in contrast to Dixon and Silver (1981), a lot-for-lot policy is implemented for all periods and products at the beginning of the heuristic. Starting from this usually infeasible initial solution, period by period, beginning in the first period, on the one hand, improvements are sought by merging lots that have already been scheduled and, on the other hand, it is checked whether infeasibilities will occur in future periods due to the lots that have been scheduled so far.

If this is the case, a part of the future demand of selected products must already be shifted forward into the current period. As in the heuristic of Dixon and Silver (1981), as a first step it is tried to eliminate the infeasibilities by shifting those lots which would reduce the costs anyway. The cost criterion »Cost per Setup and Production Time« used here is based on the criterion of the heuristic of Groff (1979) and, in contrast to Dixon and Silver (1981), only approximately describes the reduction of the average period costs per unit of capacity.

If infeasibilities still exist, the next step is to add partial quantities of future demand to the already scheduled lots, which leads to an increase in cost. In contrast to Dixon and Silver (1981), the cost criterion takes into account the additional setup costs that may be incurred due to the shifting of partial quantities.

5 Instance generation and evaluation

In order to ensure that the forecasting method to be developed is not only trained for a specific type of CLSP instance, but can be applied to any scenario, the following section aims to generate a data set that is as realistic and wide-ranging as possible. In Sect. 5.1 the creation of the data set J72T12G24 is described, before in Sect. 5.2 the results of the heuristics are presented and analyzed.

5.1 Data set J72T12G24

One goal is that the forecasting method to be developed has a high generalizability. For this purpose, the forecasting method must be trained with a broad and extensive data set representing a large part of all CLSP scenarios. The data set is called J72T12G24 because the instances include $J = 72$ products and have a planning horizon of $T = 12$ periods. It is assumed that $G = 24$ product families ($g = 1, ..., G$) can be derived. The characteristics of the instances can be adjusted or changed according to the parameters used in the model (production coefficient, holding cost rate, setup cost rate, capacity). Table 13 in Appendix A shows the implementation of the value assignment of the individual model parameters.

Since the level of the setup cost rate and the holding cost rate in conjunction with the respective demand quantity of a product is decisive for whether costs can be reduced by merging lots (backward shifting of complete lots), three different scenarios are assumed with regard to the setup and holding cost rate. In the base scenario (SC2), the ratio of setup and holding costs is balanced, so that merging lots makes sense in about half of all cases. In scenario SC1, on the other hand, the setup cost rates are relatively high, which is why costs can almost always be saved by merging lots. In contrast, the setup cost rates in scenario SC3 are relatively low, so that costs can only be reduced by merging lots in very few cases. The ratio of average setup costs and average holding costs in the case of a lot shift of the size of an average period demand by one period is denoted with sc^{ratio} and can be found in Appendix A for the respective scenarios.

Regarding the demand, six different demand scenarios D1,..., D6 are generated, which lead to very different characteristics of the instances (fluctuating, increasing, decreasing, steady, seasonal, mix). The generation of the different demand scenarios can be found in Appendix A. It should be noted that a scenario is applied to all products of an instance. For example, if scenario D2 is applied, all products of an instance show an increasing demand over time. A combination of the different demand patterns is realized in scenario D6. A seasonal demand pattern can be found in scenario D5. Figure 3 shows examples of these demand scenarios for 10 selected products.

The instances contained in the data set J72T12G24 also have different average cumulative capacity utilization rates (U1, U2). It should be noted that the demand in utilization scenario U2 is on average 20% higher than in utilization scenario U1.

When the demand scenarios are combined with the utilization scenarios and the setup cost scenarios, the data set J72T12G24 can be divided into 36 ($= 6 \cdot 2 \cdot 3$) different categories. From each category, 200 instances are generated, so that the data set comprises a total of 7,200 instances.

5.2 Results of heuristics

In the following, each instance of the data set J72T12G24 is solved using the heuristics of Lambrecht and Vanderveken (1979), Dixon and Silver (1981), Günther (1987), and Dogramaci et al. (1981) in order to identify the heuristic with the best

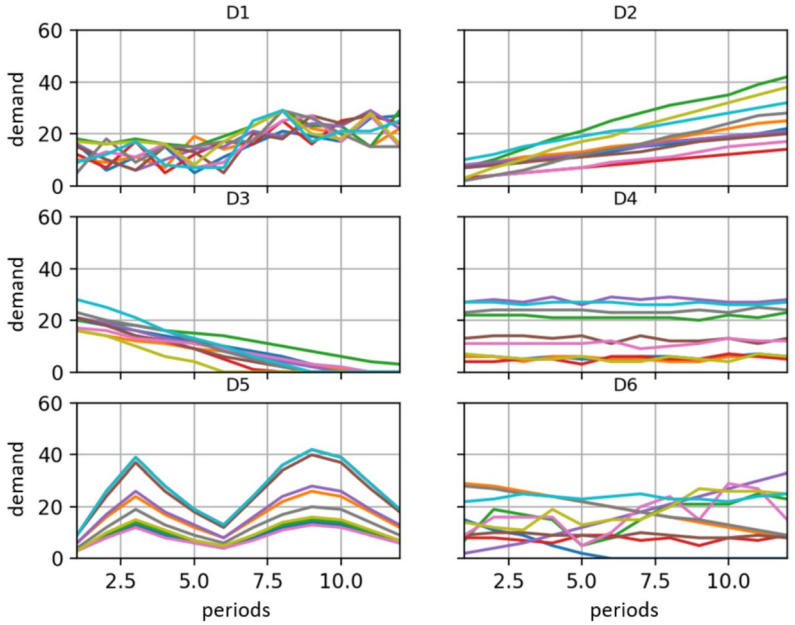


Fig. 3 Exemplary demand scenarios D1,..., D6 of 10 selected products

solution quality. All tests are performed with a laptop that has the following technical specifications: Intel Core i5–8265U 1.6 GHz CPU, 16 GB RAM, Win 10 Pro 64-bit. In order to ensure comparable results in terms of runtime, the heuristics were implemented in Python 3.7 and executed on the same machine, given that TensorFlow is also developed in Python. In addition, the Gurobi Optimizer is used to solve the CLSP to optimality.

Table 2 shows the results of the heuristics depending on the utilization scenario and the demand scenario. Here, *obj.* indicates the average deviation from the optimal objective function value in percent and *cpu* indicates the average computation time in seconds. Furthermore, *best* indicates the number [no.] or the fraction [%] of all instances which is best solved with the considered heuristic. If more than one heuristic determines the best objective function value, the heuristic that has the shorter computation time is selected. Note that the evaluation does not additionally distinguish between the different setup cost scenarios (SC).[2] Each combination of U and D can therefore be assigned 600 instances. The best heuristic with respect to the features *obj.*, *cpu* and *best [%]* is highlighted in bold for each combination of U and D.

It can be seen that the average deviation from the optimal objective function value (obj.) is significantly higher for all heuristics in utilization scenario U2 than in utilization scenario U1. Furthermore, it should be noted that all considered heuristics rarely find a very good solution for the instances of the categories

[2] The results depending on U and SC can be found in Table 16 in Appendix B.

Table 2 Computational results depending on U and D

		L&V (1979)		D&S (1981)		DPA (1981)		G (1987)	
		obj. [%]	cpu [s]	obj. [%]	cpu [s]	obj. [%]	cpu [s]	obj. [%]	cpu [s]
		best [no.]	best [%]	best [no.]	best [%]	best [no.]	best [%]	best [no.]	best [%]
U1	D1	4.22	**0.19**	3.50	1.39	**1.78**	34.46	3.28	0.61
		4	0.7	87	14.5	376	**62.7**	133	22.2
	D2	11.08	**0.28**	**3.33**	1.45	7.57	31.46	7.34	0.76
		30	5.0	287	**47.8**	8	1.3	275	45.8
	D3	8.93	**0.08**	0.76	1.32	**0.38**	15.91	1.19	0.40
		0	0.0	173	28.8	415	**69.2**	12	2.0
	D4	**0.95**	**0.20**	1.13	1.32	1.06	41.20	1.33	0.54
		26	4.3	298	**49.7**	39	6.5	237	39.5
	D5	5.81	**0.23**	4.43	1.17	**4.16**	25.96	5.18	0.52
		7	1.2	205	34.2	321	**53.5**	67	11.2
	D6	9.23	**0.11**	1.32	1.35	**0.38**	35.97	1.30	0.54
		2	0.3	56	9.3	428	**71.3**	114	19.0
U2	D1	11.57	**0.31**	8.04	1.46	**6.78**	23.66	10.45	0.79
		2	0.3	173	28.8	219	**36.5**	206	34.3
	D2	22.97	**0.39**	**6.63**	1.68	16.71	26.08	14.24	1.16
		3	0.5	295	49.2	6	1.0	296	**49.3**
	D3	8.84	**0.09**	1.04	1.34	**0.67**	21.15	1.49	0.43
		0	0.0	194	32.3	396	**66.0**	10	1.7
	D4	**1.95**	**0.18**	2.19	1.17	2.20	32.59	2.55	0.46
		25	4.2	344	**57.3**	25	4.2	206	34.3
	D5	11.20	**0.30**	**6.84**	1.12	16.53	17.13	9.28	0.60
		5	0.8	298	**49.7**	135	22.5	162	27.0
	D6	11.07	**0.11**	2.04	1.30	**0.99**	32.19	2.17	0.53
		2	0.3	143	23.8	345	**57.5**	110	18.3
All		8.99	**0.21**	**3.44**	1.34	4.93	28.15	4.98	0.61
		106	1.5	2553	35.5	2713	**37.7**	1828	25.4

L&V: Lambrecht & Vanderveken, D&S: Dixon & Silver, DPA: Dogramaci et al., G: Günther

The best heuristic with respect to the features obj., cpu and best [%] is highlighted in bold

U2/D1, U2/D2 and U2/D5 (obj. > 5%). The demand scenario D2 (increasing demand) seems to be particularly difficult, since even in the utilization scenario U1 an obj. > 5% is determined for all heuristics except Dixon & Silver.

Compared to the other three heuristics, the heuristic of Lambrecht and Vanderveken finds the best solution for only 106 of the total 7,200 instances (1.47%), but has the shortest average computation time among all heuristics (< 0.5 s).

The heuristic of Dixon & Silver performs best in the categories U1/D2, U1/D4, U2/D2, U2/D4, and U2/D5 (best [%] > 47%). All in all, over 35% of the instances are best solved with the heuristic of Dixon & Silver. The computation times are

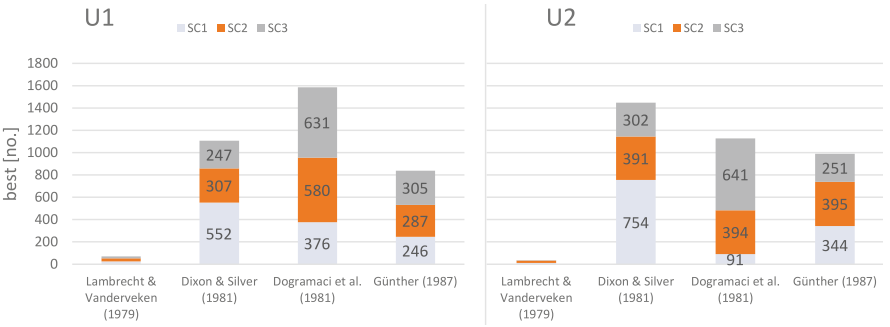


Fig. 4 Number of best solved instances

on average less than 2 s in all categories, but are the second highest compared to the rest of the heuristics.

The heuristic of Dogramaci et al. dominates the categories U1/D1, U1/D3, U1/D5, U1/D6, U2/D1, U2/D3 and U2/D6 (best [%] > 36%). More than 37% of all instances are best solved by the heuristic of Dogramaci et al. However, the average computation times are much higher (between 15–41 s) compared to the rest of the heuristics.

Furthermore, it can be seen that especially the instances of the categories U1/D2 and U2/D2 are best solved by the heuristics of Günther and Dixon & Silver (best [%] > 45%). About 25% of all instances can be best solved by Günther's heuristic, with average computation times of less than 1 s. In the category U1/D4, all heuristics show a very good solution quality (obj. < 2%).

It can be summarized that depending on the scenarios, different heuristics show the best solution quality and thus none of the considered heuristics dominates in all scenarios. Consequently, a selection problem can be identified. If there is no prior knowledge or experience about the performance of individual heuristics, it is impossible to assess which heuristic is better for a specific CLSP instance. Hence, the logical choice is to randomly select from all heuristics with equal probability. However, if initial performance insights are available, these can inform the selection process. According to Table 2, 1.5% of all CLSP instances are best solved with heuristic of Lambrecht and Vanderveken (1979), 35.5% with Dixon and Silver (1981), 37.7% Dogramaci et al. (1981), and 25.4% of all instances with the heuristic of Günther (1987). Thus, heuristics can be selected according to this empirical distribution.

Figure 4 shows for each heuristic the number of best solved instances for both utilization scenarios and the three setup scenarios. In addition to the findings from Table 2, it can be seen that the heuristic of Dixon & Silver performs particularly well for high setup costs (SC1) and consequently rather many lot shifts. In contrast, the heuristic of Dogramaci et al. performs particularly well for lower setup costs (SC3) and consequently rather few lot shifts. Regarding the setup cost scenario SC2, in U1 the heuristic of Dogramaci et al. can also best solve most

instances. In contrast, in U2 the heuristics of Dixon & Silver, Günther and Dogramaci et al. can best solve about the same number of instances.

6 Forecasting methods

In the following section, two different forecasting methods for selecting the best heuristic depending on the characteristics of a CLSP instance will be developed, configured and trained. For this purpose, Sect. 6.1 first presents some features, which can be used to characterize the instances. Subsequently, two different forecasting methods are explained, which are classified into conditional empirical distribution based (Sect. 6.2) and neural network based (Sect. 6.3) approaches. The forecasting methods must first be trained and configured before they can be used or tested. For this reason, the data set J72T12G24 (7,200 instances) is divided into a training set (3,840 instances), configuration/validation set (960 instances) and test set (2,400 instances).

6.1 Feature engineering

When dealing with a new, not yet considered instance of the CLSP, we want to be able to forecast the one of the four heuristics that performs best for this specific instance. In ASPs, so-called »features« are searched for, which are able to describe a certain instance in a quantitative manner, i.e., by using a set of meaningful numbers. These numbers serve as key indicators to characterize an instance. The process of finding such indicators is called »feature engineering«. Note that features are only related to the instance itself, i.e., to the input of an optimization problem before it is solved. However, if a simple and intuitive benchmark algorithm was available, which could be used to find some (feasible or even infeasible) solution of the optimization problem quickly, also indicators could be calculated which evaluate the output/results of the optimization process and thus the performance of this benchmark algorithm. In the following, we will — at least partly — use the lot-for-lot policy as such a benchmark algorithm. This allows us (and hopefully also a forecasting method) to judge the »difficulty« of the CLSP instance in terms of how simple or hard it may be to find feasible solutions.

Features should be both relative and relevant. Because lessons learned on small instances shall be transferred to larger instances, the features must not depend on the problem size. Their total number and their interpretation should be identical for any problem size. Thus features consisting of absolute numbers are not helpful. Instead, *relative* numbers have to be searched for. For example, the total capacity $\bar{K} := \sum_t K_t$ remains meaningless until it is set in relation to the total demand $\bar{D} := \sum_{j,t} d_{tj} \cdot a_j$. If the resulting average utilization $\bar{u} := \frac{\bar{D}}{\bar{K}}$ exceeds 100%, it is impossible to find a feasible solution. This remains true for CLSP instances of any size.

As this example shows, features should also be *relevant*. They should express important properties of an instance like such an impossibility — or more general the grade of difficulty — to find feasible solutions. Thus, on the one hand, a relevant feature can express the level of difficulty in identifying a feasible solution. On the

other hand, a feature can also describe the difficulty of an instance in terms of solution quality. Of course, irrelevant features should be avoided because they unnecessarily consume computation time and increase complexity (for example, of the input layer of a neural network). But it is hard to judge ex ante whether a feature will prove relevant for forecasting an instance's best algorithm or not. This is the reason why in the following also some features will be proposed which cannot be directly associated with feasibility or solution quality.

6.1.1 Features related to utilization

Based on a lot-for-lot solution, two types of lot shifts can be distinguished for the CLSP. The first is the case where demand-driven production is not possible in selected periods due to the capacity constraints of the production line and therefore a lot-for-lot solution is not a feasible solution. Consequently, production quantities must be shifted to earlier periods which have a remaining capacity. If only partial lots are shifted, these shifts are generally accompanied by an increase in costs.

The second case is that, despite a feasible lot-for-lot solution, lot shifts can be useful in order to achieve a better solution quality. This is the case if setup costs can be saved by merging lots and this saving exceeds the additional resulting holding costs. These shifts are primarily executed to improve the solution quality, but at the same time they can also contribute to reducing capacity bottlenecks.

If a given CLSP instance has sufficient capacity in each period, a lot-for-lot solution already represents a feasible solution, so that for this instance the level of difficulty in terms of feasibility can be classified as very easy. If the instance also has a very low ratio of setup and holding costs, lot shifts can seldom save costs, so that the level of difficulty in terms of solution quality can also be rated as simple.

If, on the other hand, an CLSP instance does not have sufficient capacity for a lot-for-lot solution, shifts have to be performed due to the capacity violations. If, in addition, the ratio of setup and holding costs of the instance is very high, many lot shifts are also necessary to generate a good solution. In this case, the instance can be classified as very hard.

In the following, features for the CLSP are presented, which try to express the level of difficulty of an CLSP instance in terms of feasibility or solution quality. Since in the CLSP different products compete for the capacity of a single production line, we propose to capture the machine utilization in the case of a lot-for-lot policy. As shown in Table 3, feature 1 represents the average utilization of a period, feature 2 represents the maximum utilization, feature 3 represents the minimum utilization and feature 4 represents the standard deviation of the utilization. The four features inform a forecasting procedure about the range of capacity utilizations of a lot-for-lot solution. If, for example, an instance has a maximum capacity utilization $> 100\%$ in some periods, it can be concluded that lot shifts to earlier periods are absolutely necessary to avoid capacity violations. However, a statement about how these lot shifts should look like cannot be derived.

Thus, obviously, features 1–4 hardly allow any conclusions about the general solvability of the CLSP instance. If there is sufficient remaining capacity in earlier periods to eliminate an overload due to an advance production, the instance is still

Table 3 Features related to utilization

No.	Feature description	Formula
1	Average utilization	$\bar{u} = \frac{\sum_{j,t} d_{jt} \cdot a_j}{\sum_t K_t}$
2	Maximum utilization	$u^{max} = \max_{0<t\leq T} \left[\frac{\sum_j d_{jt} \cdot a_j}{K_t} \right]$
3	Minimum utilization	$u^{min} = \min_{0<t\leq T} \left[\frac{\sum_j d_{jt} \cdot a_j}{K_t} \right]$
4	Standard deviation of utilization	$u^{std} = \sqrt{\frac{\sum_t (u_t - \bar{u})^2}{T}}$ with $u_t = \frac{\sum_j d_{jt} \cdot a_j}{K_t}$
5	Average cumulative utilization	$\bar{uc} = \sum_{t=1}^{T} \left[\frac{\sum_j \sum_{h=1}^{t} d_{jh} \cdot a_j}{\sum_{h=1}^{t} K_h} \right] / T$
6	Maximum cumulative utilization	$uc^{max} = \max_{0<t\leq T} \left[\frac{\sum_j \sum_{h=1}^{t} d_{jh} \cdot a_j}{\sum_{h=1}^{t} K_h} \right]$
7	Minimum cumulative utilization	$uc^{min} = \min_{0<t\leq T} \left[\frac{\sum_j \sum_{h=1}^{t} d_{jh} \cdot a_j}{\sum_{h=1}^{t} K_h} \right]$
8	Standard deviation of cumulative utilization	$uc^{std} = \sqrt{\frac{\sum_{t=1}^{T} (uc_t - \bar{uc})^2}{T}}$ with $uc_t = \frac{\sum_j \sum_{h=1}^{t} d_{jh} \cdot a_j}{\sum_{h=1}^{t} K_h}$

solvable. However, it is well known that the solvability of an CLSP instance can be easily verified by checking whether in each period the accumulated capacity is greater than or equal to the accumulated demand. For this reason, it is proposed to consider cumulative capacity utilization ratios as further information in addition to simple capacity utilization ratios. Thus Table 3 also compromises features 5–8, representing the average, minimum and maximum cumulative utilization as well as its standard deviation. As soon as feature 6 has a value greater than one, no feasible solution exists for the considered instance. Unfortunately, it cannot be concluded that a value close to one automatically implies that it is hard to find a feasible solution. For instances with constant high period utilizations (see U2/D4), both the average and the maximum cumulative utilization have a high value. Nevertheless, for this kind of CLSP instance already a lot-for-lot solution can be a feasible solution.

Table 4 Features related to demand

No.	Feature description	Formula
9	Share of total demand for the 1st third of time periods	$\Delta d_1^{ges} = \frac{\sum_j \sum_{t=1}^{\lfloor T/3 \rfloor} d_{jt} \cdot a_j}{\sum_j \sum_{t=1}^{T} d_{jt} \cdot a_j}$
10	Share of total demand for the 2nd third of time periods	$\Delta d_2^{ges} = \frac{\sum_j \sum_{t=\lceil T/3 \rceil}^{\lfloor 2T/3 \rfloor} d_{jt} \cdot a_j}{\sum_j \sum_{t=1}^{T} d_{jt} \cdot a_j}$
11	Share of total demand for the 3rd third of time periods	$\Delta d_3^{ges} = \frac{\sum_j \sum_{t=\lceil 2T/3 \rceil}^{T} d_{jt} \cdot a_j}{\sum_j \sum_{t=1}^{T} d_{jt} \cdot a_j}$
12	Slope of the regression line	$b = \frac{\sum_t (t - \frac{T+1}{2}) \cdot (d_t - \bar{d}_t)}{\sum_t (t - \frac{T+1}{2})^2}$ with $d_t = \sum_j d_{jt} \cdot a_j \quad \bar{d}_t = \frac{\sum_t d_t}{T}$

6.1.2 Features related to demand

Section 5.2 has shown that the individual heuristics perform better in certain demand scenarios than in others. Thus forecasting procedure should be informed how the total demand changes over time. Therefore we try to express the course of the demand curve using some further features. Two possibilities are proposed.

The first option is to divide the planning horizon into a certain number of sections. If the share of total demand is then determined for each of these sections, conclusions can be drawn about the demand curve. In the following, the planning horizon is divided into three sections, which is why features 9, 10 and 11 (see Table 4) each represent a share of total demand. It should be noted that in calculating the shares, demand is weighted by the corresponding production coefficient. Increasing the number of sections up to T might help to better catch seasonal effects, but also entails the danger of overfitting.

If seasonal effects are less important, it may be helpful to catch a linear trend in demand. This can be done by regression analysis. Thus, as a further option, a linear regression approximates the slope of the demand curve. This is done by feature 12 where the production coefficients a_j again serve to express total demand in units of time. Note that if capacity varies over time, additional features would be necessary to describe the curve of capacity by analogy.

6.1.3 Features related to lot shifting and product properties

Furthermore, features might be useful, which only describe certain properties of the CLSP instance and do not directly refer to the difficulty of an instance in terms of infeasibility or solution quality. In order to capture the heterogeneity of the products of an instance with respect to production time, holding and setup cost rate, the coefficient of variation for the respective parameter is calculated in features 13–15 of Table 5. The more homogeneous the products are, the less influence the correct

Table 5 Features related to lot shifting and product properties

No.	Feature description	Formula
13	Coefficient of variation of the production coefficient	$v^{a} = \frac{a^{std}}{\overline{a}} = \frac{\sqrt{J \cdot \sum_j \left(a_j - \frac{\sum_j a_j}{J}\right)^2}}{\sum_j a_j}$
14	Coefficient of variation of the holding cost rate	$v^{hc} = \frac{hc^{std}}{\overline{hc}} = \frac{\sqrt{J \cdot \sum_j \left(hc_j - \frac{\sum_j hc_j}{J}\right)^2}}{\sum_j hc_j}$
15	Coefficient of variation of the setup cost rate	$v^{sc} = \frac{sc^{std}}{\overline{sc}} = \frac{\sqrt{J \cdot \sum_j \left(sc_j - \frac{\sum_j sc_j}{J}\right)^2}}{\sum_j sc_j}$
16	Coefficient of variation of the relative holding cost rate	$v^{rhc} = \frac{rhc^{std}}{\overline{rhc}}$ with $rhc^{std} = \sqrt{\frac{\sum_j (rhc_j - \overline{rhc})^2}{J}}$ $\overline{rhc} = \frac{\sum_j rhc_j}{J} \quad rhc_j = \frac{hc_j}{a_j}$
17	Average ratio of setup and holding costs	$\Delta sc = \frac{\sum_j sc_j / \left(\overline{d_j} \cdot hc_j\right)}{J}$ with $\overline{d_j} = \frac{\sum_t d_{jt}}{T}$

selection of the lot to be shifted has on the solution quality. If the entire demand of all products in a considered period cannot be produced in the same period, individual lots must be shifted to earlier periods. Since the holding cost rate per released unit of capacity.[3] (hc_j/a_j) affects the selection of the lot to be shifted, the spread of the relative inventory cost rate is included in feature 16 as a coefficient of variation (see Table 5). A very low coefficient of variation means that the products of the considered instance have very similar relative holding cost rates and the selection of the lot to be moved has rather little influence on the solution quality. Even if capacity is sufficient, setup costs may be saved by merging lots and thus the solution quality can be improved. For this reason, feature 17 contains the average ratio of setup and holding costs for a lot shift of the size of an average period demand. The smaller the ratio of setup and holding costs, the fewer lot shifts are necessary to achieve a good solution quality.

6.2 Conditional empirical distribution

The first approach for predicting the best heuristic could be to use conditional probabilities, which can be calculated from the training set using the features derived in Sect. 6.1. In the following, the approach is briefly explained using feature 12 »slope of the regression line« as an example.

[3] In the following referred to as »relative inventory cost rate«

In the first step, the slope of the regression line must be determined for all CLSP instances of the training and validation set, and a z-score standardization[4] must be performed so that an expected value of 0 and a standard deviation of 1 result. If the considered feature is feature 9, 10 or 11 (share of demand 1st, 2nd, 3rd third in total demand), the standardization is performed together, otherwise the information about the demand pattern would be lost.

In the next step, the feature »slope of the regression line« is transformed into a categorical feature for the training set. For this purpose, both the number of classes and the intervals have to be defined. In order to avoid a manual determination of the intervals, they are set in a way that each class contains the same number of observed values (CLSP instances). Thus, if the number of classes is fixed at three, each class contains 1,280 ($= 3,840/3$) instances. Since it is known for each instance of the training set which heuristic leads to the best solution quality, a frequency distribution of the heuristics can be determined and stored for each class.

To predict the best heuristic for a previously unknown CLSP instance, the slope of the regression line must first be calculated and standardized using the learned parameters of the training and validation set (mean and standard deviation of the population). Afterwards, the matching class can be determined and a random selection of the heuristic can be performed according to the stored probability distribution. Since a frequency distribution is recorded separately for each class and only serves as a probability distribution for the respective class, it can be considered as conditional probabilities, which is why the forecasting approach is called »conditional empirical distribution«.

The performance of this approach can be significantly improved by combining several features. Figure 5 shows an example where the features »slope of the regression line« and »average ratio of setup and holding costs« are combined. For clarity, only 500 of the total 3,840 instances in the training set are shown. For each feature, the number of classes is three, which is why the combination of features results in a total of nine categories.[5] These are separated from each other by red, dashed lines in Fig. 5.

It should be noted that the intervals are determined successively when more than one feature is used. In the present example, the intervals for feature »slope of the regression line« are determined first, so that there are initially 166 or 167 ($= 500/3$) instances in each class. Afterwards, each class is subdivided into three further classes depending on feature »average ratio of setup and holding costs«, whereby the intervals can be set individually for each original class. This procedure ensures that each of the nine categories has the same number of instances ($500/9 = 55.56$). At the same time, it can be noted that the sequence of the selected features has an impact on the intervals of the resulting categories and thus on the performance of the forecasting approach. For the example considered in Fig. 5, the following statement holds.

[4] To standardize an observation from a population, the mean of the population is subtracted from the respective observation and the result is divided by the standard deviation of the population.

[5] When classes of several features are combined, the term categories is used in the following.

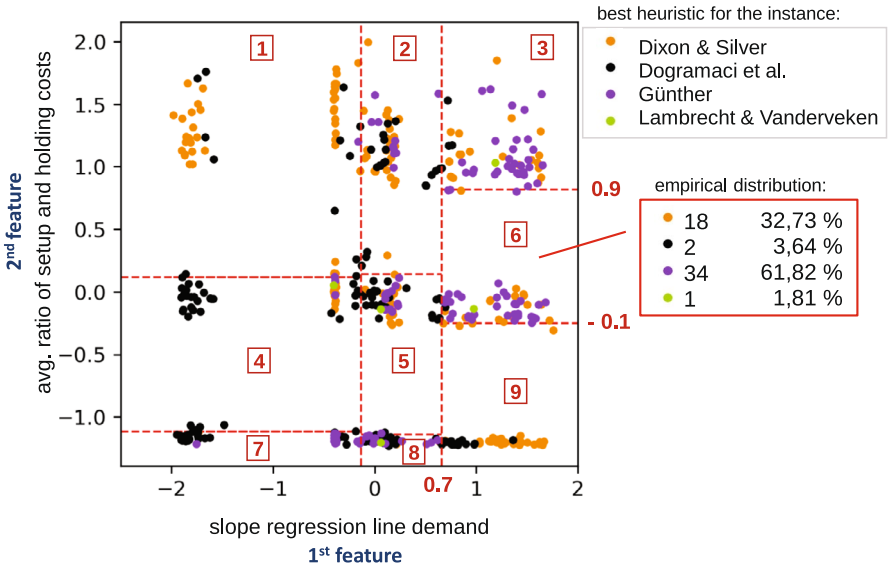


Fig. 5 Conditional probabilities - Ex. with 2 features, 3 classes per feature

If a CLSP instance r is assigned to category 6 with probability $P(cat.6) = P(feature_1(r) > 0.7) \cap P(-0.1 \leq feature_2(r) \leq 0.9)$, then with probability $P(D\&S \mid cat.6) = 32.73\%$ Dixon and Silver (1981), with probability $P(DPA \mid cat.6) = 3.64\%$ Dogramaci et al. (1981), with probability $P(G \mid cat.6) = 61.82\%$ Günther (1987) and with probability $P(L\&V \mid cat.6) = 1.81\%$ Lambrecht and Vanderveken (1979) is selected as the best heuristic.

The described method shows considerable similarities with decision trees. For example, Fig. 5 can be interpreted as the result of a pruned decision tree, where the pruning takes place after consideration of the second feature. However, there are notable differences from classical decision trees. In traditional decision tree models, class intervals are typically established using specific criteria, such as Gini impurity or information gain. In addition, predictions are usually based on the most common observation within a category. Structured overviews of decision trees can be found, for example, in Costa and Pedreira (2023) and Kotsiantis (2013).

In the following, an appropriate configuration of the described forecasting approach is to be found. The question is how many and, in particular, which features should be combined with each other, which sequence should be applied here and how many classes should be generated per feature. It should be noted that the number of resulting categories increases exponentially depending on the number of features and the number of classes per feature. While the combination of two features and a number of classes of three results in a total of nine ($= 3^2$) categories, the combination of three features with an identical number of classes already leads to 27 ($= 3^3$) categories. The more categories are introduced, the fewer instances each category contains. If more categories are used than there are instances in the training set, some categories will remain empty, and therefore no frequency distribution can be

captured. Therefore, in this case, the prediction draws randomly from all heuristics according to the distribution contained in the training data set. Moreover, it should be noted that the possible feature and sequence combinations also strongly increase with the number of features to be selected. While 272 possible combinations have to be checked when selecting 2 out of 17 features, there are already 4,080 possible combinations when selecting 3 out of 17 features.

To answer the questions raised before, for the combination of 1–4 features in conjunction with three, five, six and nine classes per feature, all possible feature and sequence combinations are tested. The setting of the intervals and the determination of the empirical distribution for each category is based on the training set. Then, based on the validation set, an accuracy is measured for each feature and sequence combination. To get rid of random noise and to achieve representative validation results, each accuracy is calculated from the average of 50 runs. Table 6 shows the validation results. Here, *no. cat.* defines the number of resulting categories, *inst./cat.* the number of instances per category, and *acc. top 10* the range of average accuracies of the top ten feature combinations.

It can be seen that with an increasing number of features as well as with an increasing number of classes per feature better accuracies can be achieved. However, a higher number of classes does not lead to better performance without limitation. It can be seen that in the case of one feature and two features, a number of classes of nine performs slightly worse than a number of classes of six.

In the case of the combination of three features with nine classes each, an accuracy of up to 69.4% can be achieved. This results from the combination of the features »average ratio of setup and holding costs« (1st feature), »slope of the regression line« (2nd feature) and »standard deviation of cumulative utilization« (3rd feature).

Regarding the combination of four features, it should be noted that to determine the best feature combination, 57,120 possibilities would theoretically have to be checked, which would require a very high computational effort. However, the experiments with three features show that among the ten best feature combinations of an experiment, the feature »average ratio of setup and holding costs« is applied as the first feature in 58% of the cases and as at least one of the first three features in 100% of the cases. This feature seems to be a significant one. For this reason, to reduce the computational effort, the »average ratio of setup and holding costs« is fixed as the first feature in all experiments with four features. The highest accuracy with 73.0% can be achieved with the combination of the features »average ratio of setup and holding costs« (1st feature), »slope of the regression line« (2nd feature), »standard deviation of utilization« (3rd feature), »minimum cumulative utilization« (4th feature) and six classes per feature.

Table 7 shows which and how often features are included in the best and worst ten feature combinations. Considered are the feature combinations of the experiments with four features in combination with three, five and six classes per feature.[6] While feature 4 (standard deviation of utilization) is included in 60.0% of the ten best

[6] Since in these experiments feature 17 (average ratio of setup and holding costs) was fixed as first feature, it is not included in the listing.

Table 6 Conditional empirical distribution - results training & validation

Classes	3		5		6		9	
Features	no. cat.	inst./cat.	no. cat.	inst./cat.	no. cat.	inst./cat.	no. cat.	inst./cat.
	acc. top 10 [%]		acc. top 10 [%]		acc. top 10 [%]		acc. top 10 [%]	
1	3	1280.0	5	768.0	6	640.0	9	426.7
	35.4 – 39.0		37.9 – 40.1		**38.6 – 43.6**		38.5 – 42.6	
2	9	426.7	25	153.6	36	106.7	81	47.4
	47.1 – 49.2		51.0 – 54.4		**54.8 – 61.1**		55.2 – 59.9	
3	27	142.2	125	30.7	216	17.8	729	5.3
	60.4 – 61.0		64.0 – 64.9		67.8 – 68.8		**68.7 – 69.4**	
4*	81	47.4	625	6.1	1296	3.0	6561	0.6
	67.0 – 67.4		69.6 – 70.1		**72.1 – 73.0**		63.1 – 63.6	

* Ratio of setup costs and holding costs fixed as first feature

The best feature combinations, depending on the number of features, are highlighted in bold

Table 7 Importance of features [4 features, classes = 3,5,6]

Top 10			Last 10		
No.	Feature	Share [%]	No.	Feature	Share [%]
4	Std. utilization	60.0	16	Coefficient of variation rhc_j	93.3
12	Slope of the regression line	43.3	13	Coefficient of variation a_j	83.3
1	∅ utilization	43.3	14	Coefficient of variation hc_j	83.3
3	Min. utilization	33.3	6	Max. cumulative utilization	36.7
11	Share of demand 3/3	33.3	15	Coefficient of variation sc_j	3.3
8	Std. cumulative utilization	26.7			
7	Min. cumulative utilization	23.3			
9	Share of demand 1/3	23.3			
5	∅ cumulative utilization	10.0			
2	Max. utilization	3.3			

feature combinations, feature 2 (maximum utilization) is used in only 3.3% of the ten best feature combinations. It can therefore be concluded that feature 4 is more important than feature 2. Furthermore, it can be seen that in the best feature combinations mainly features are applied, which in some way describe the utilization in a lot-for-lot solution or the level of difficulty of an instance. The right-hand side of Table 7 shows the features that are included in the ten worst feature combinations. It seems that features 16, 13 and 14 are of limited relevance. It's interesting that these features only describe certain properties of the instance and do not directly refer to the level of difficulty of an instance in terms of feasibility or solution quality.

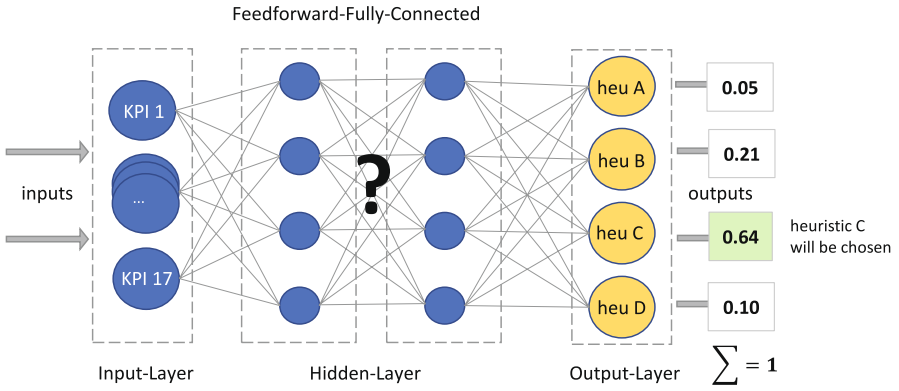


Fig. 6 Structure of CLSP-Net

6.3 Neural network - CLSP-Net

In the following section the selection of the best heuristic shall be done by an ANN, which is named CLSP-Net. Since the best heuristic is already known for the instances contained in the data set J72T12G24, the selected learning approach corresponds to a classification task, which can be assigned to the subfield of supervised learning (see Goodfellow et al. 2016, p. 105). The theoretical basics of ML are assumed to be known.[7]

Section 6.3.1 first introduces the basic structure of CLSP-Net. This is followed by a description of the basic procedure for developing and training CLSP-Net in Sect. 6.3.2. In Sect. 6.3.3, the hyperparameters of CLSP-Net will be tuned and the results of the validation phase will be explained.

6.3.1 Basic structure of CLSP-Net

CLSP-Net is a Feedforward-Fully-Connected neural network. The *input layer* comprises 17 nodes. In order to apply CLSP-Net independent of the size of the CLSP instance, features are passed to the input layer, which have to be calculated for each instance beforehand (see Fig. 6). In the *hidden layers*, a ReLU function[8] is implemented as activation function. The exact number of hidden layers as well as the number of nodes per hidden layer are determined in the course of Sect. 6.3.3. The *output layer* consists of four nodes. Each node represents a heuristic. The output nodes take values in the range [0, 1] (activation function: softmax). Finally, for the considered CLSP instance the heuristic is selected whose output node has the highest value. To evaluate the performance of CLSP-Net, the rate of correctly classified instances (accuracy) is used as a

[7] For further literature see Goodfellow et al. (2016), Murphy (2012) or Bishop (2006).

[8] An overview of currently used activation functions in the field of ML can be found in Nwankpa et al. (2021).

performance criterion. As a loss function, the categorical cross-entropy is used. This measures the difference between the probability distribution output by CLSP-Net and the real distribution of class labels (here: heuristics) (see Goodfellow et al. 2016, p. 132). Furthermore, the RMSProp algorithm of Hinton (2012) is used as an optimizer.

6.3.2 Workflow CLSP-Net

The workflow for training and configuring CLSP-Net can be described as follows. First, for each instance of the data set J72T12G24, the features presented in Sect. 6.1 have to be calculated. At the same time, the best heuristic is added to each instance as a label.

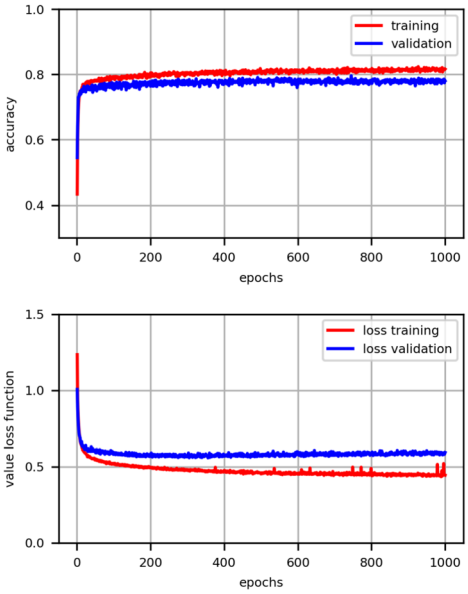
After that, the training set has to be standardized. Here, an independent z-score standardization is performed for each feature, resulting in an expected value of 0 and a standard deviation of 1. Then, the test set can be standardized based on the learned parameters of the training set (mean and standard deviation of the population). In the next step, the labels (heuristics) have to be coded by using one-hot encoding.

Afterwards, the CLSP-Net can be trained and the hyperparameters can be tuned based on the validation set. For example, the number of hidden layers, the number of nodes per hidden layer, the number of training epochs or a possible regularization have to be defined. As soon as CLSP-Net is configured, its performance is evaluated on the basis of the test set, which is completely unknown for CLSP-Net (see Sect. 7.1).

6.3.3 Hyperparameter tuning

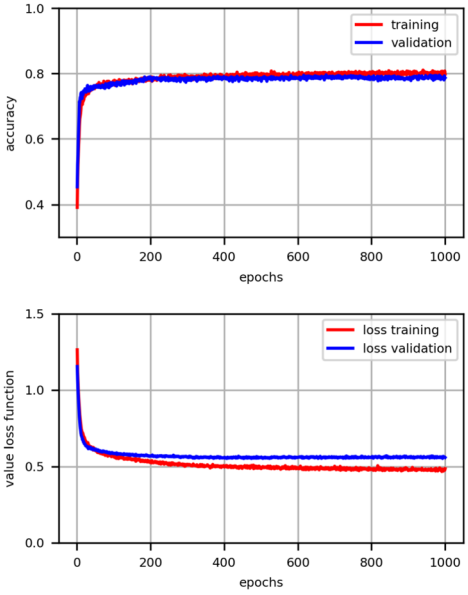
CLSP-Net is implemented using Python 3.7 in conjunction with the ML library Keras. TensorFlow is used as the backend engine. As part of the tuning of the hyperparameters, the network is successively enlarged until overfitting occurs.

It is shown that a very small network with one hidden layer of four nodes achieves a correct classification rate of over 60% (validation) after only a few training epochs. Based on this initial finding, additional nodes are systematically added to the hidden layer of the single-layer network. Beginning with 64 nodes in the hidden layer, an overfitting of the network to the training data can be observed. This can be reduced by using a dropout regularization. Here, during signal forwarding from the hidden layer to the output layer, a proportion of the values (dropout rate) is randomly deleted (see Srivastava et al. 2014). The highest detected correct classification rate of a network with one hidden layer is 79.58% and is achieved with 128 nodes in combination with a dropout regularization of 10% (see Fig. 7). The rest of the configuration can be seen in Fig. 7. The upper left graph shows the accuracy as a function of training epochs for both the training set and the validation set. Training is performed over 1,000 epochs. The lower left graph shows the value of the loss function (categorical cross-entropy), which is to be minimized during the training



configuration ANN:	
input nodes	17
hidden layers	1
hidden nodes per layer	128
activation function	ReLU
output nodes	4
learning rate	0.002
batch size	512
dropout regularization	0.1
best performance training [%]	82.37
best epoch training	927
runtime training [s]	31.3
best performance validation [%]	79.58
best epoch validation	745

Fig. 7 Performance CLSP-Net (1 hidden layer, 128 nodes)



configuration ANN:	
input nodes	17
hidden layers	2
hidden nodes per layer	16
activation function	ReLU
output nodes	4
learning rate	0.002
batch size	512
dropout regularization	0.05
best performance training [%]	81.04
best epoch training	940
runtime training [s]	28.3
best performance validation [%]	79.58
best epoch validation	677

Fig. 8 Performance CLSP-Net (2 hidden layers, 16 nodes)

Next, a network with two hidden layers is created. Analogously to the procedure for the single-layer network, the number of nodes is first limited to four nodes per hidden layer and then increased step by step in powers of 2. Here, it can be seen that overfitting sets in at 16 nodes per hidden layer. Figure 8 shows the performance of the best configuration. This comprises two hidden layers with 16 nodes each and uses a dropout regularization of 5% during training. The highest accuracy is achieved in epoch 677 and amounts to 79.58%. This means that the same good result can be achieved as with the single-layer network with 128 nodes. However, it can be seen that the accuracy for the two-layer network fluctuates significantly less over the course of training and is on average slightly higher. In addition, the two-layer network has significantly fewer parameters to be trained (network size) and is therefore preferable to the single-layer network.

7 Numerical study

In this section, the previously presented forecasting approaches are tested and evaluated in a numerical study. Section 7.1 shows the results for small test instances of the data set J72T12G24. In Sect. 7.2 it will be verified whether the forecasting methods (especially CLSP-Net) can be applied to big instances without a new training phase. Section 7.3 shows how much time is needed to create CLSP-Net and to use it for predicting the best heuristic. In 7.4 Section the test bed is enriched with two data sets from the literature.

7.1 Test results small instances

In the following, the presented forecasting methods are evaluated using the test set of the data set J72T12G24. Additionally as a benchmark a prediction is made using a uniform distribution and the empirical distribution (see Sect. 5.2). To get rid of random noise and to achieve representative test results, 50 test runs are executed. Then, besides the mean also standard deviation and maxima and minima of these test samples are reported. The test set includes 2,400 CLSP instances and was not used to train or validate the forecasting methods. The numerical results can be found in Table 8.

A random selection according to a uniform distribution achieves an average accuracy of 25.05% after 50 test runs, with the highest rate observed at 26.42% and the lowest rate at 22.33%. In contrast, if the heuristic is selected using the empirical distribution, the average accuracy improves by nearly 11% points to 33.13%. Regarding the forecasting method »conditional empirical distribution«, the combination of the features »average ratio of setup and holding costs« (1st feature), »slope of the regression line« (2nd feature), »standard deviation of utilization« (3rd feature), and »minimum cumulative utilization« (4th feature) in conjunction with six classes achieves the highest accuracy. Applied to the test set, this combination results in an average rate of 70.63%, which is about 2.5% points lower than the rate of the validation phase. The highest accuracy can be reached using CLSP-Net. The ANN

Table 8 Test results - small instances

Accuracy [%]	Uniform distribution*	Empirical distribution*	Conditional emp. dist.*	CLSP-Net
Mean	25.05	33.13	70.63	78.04
Std.	0.96	0.78	0.55	–
Max.	26.42	34.83	72.00	–
Min.	22.33	30.58	69.58	–

* Results of 50 test runs

achieves a rate of 78.04% for the test set. This is only 1.5% points below the accuracy observed during the validation process.

It can be noted that the performance of the first three methods is (in some cases drastically) below the performance of CLSP-Net. It is questionable whether the performance of the forecasting method »conditional empirical distribution« could be improved to above 78.04% (CLSP-Net) by combining five or more features.

A main challenge of the forecasting method »conditional empirical distribution« is the high computational effort caused by testing all feature combinations. Moreover, the more categories and thus frequency distributions have to be generated, the higher the computational effort is per feature combination. It should be noted, that the computation time for the experiment with four features and six classes per feature exceeds five hours despite fixing the first feature.

Another disadvantage can be seen when looking at experiment »4 features / 9 classes«, where in the validation phase in the best case only an accuracy of 63.6% is achieved (see Table 6). The reason for this is that in this experiment more categories are derived than there are instances in the training set. This means that not in all categories a frequency distribution can be captured. More training data were needed to address this problem. Consequently, it can be stated that the performance of the forecasting method »conditional empirical distribution« strictly relates to the size of the training data set so that computation times soon are prohibitive.

7.2 Performance for big instances

In the following, it is checked whether the best forecasting method CLSP-Net as well as the remaining methods are able to classify significantly larger instances without a new training phase. Therefore, a new data set J108T18G36 is created whose instances contain 108 products, 18 periods and 36 product families. Consequently, the new CLSP instances are more than twice as large (factor: 2.25) in terms of problem size when compared to the instances of the data set J72T12G24 used for training. The instances themselves are generated along the same criteria as the instances of the data set J72T12G24 (see Sect. 5.1). For each problem category (U, D, SC) 10 instances are created, so that the data set contains 360 CLSP instances.[9]

[9] The exact values for generating the model parameters can be found in Appendix A.

Table 9 Test results - big instances

Accuracy [%]	Uniform distribution*	Empirical distribution*	Conditional emp. dist.*	CLSP-Net
Mean	25.08	33.80	68.70	78.61
Std.	2.02	2.37	1.66	–
Max.	29.72	38.61	72.50	–
Min.	19.72	28.89	64.72	–

* Results of 50 test runs

Table 9 contains the test results for the data set J108T18G36. At this point, it should be noted again that the data set J108T18G36 is not used for training or validation and thus it is completely unknown for each of the forecasting methods. The accuracy of the methods »uniform distribution« and »empirical distribution« is almost identical compared to the smaller instances. The forecasting method »conditional empirical distribution« decreases by about 2% points to 68.70%. CLSP-Net still achieves the highest accuracy with 78.61%. This is marginally higher than the accuracy of 78.04% achieved in the data set J72T12G24. When looking at this example it can be stated that the features derived in Sect. 6.1 allow to transfer experiences learned on small instances to unknown instances of substantially larger size.

7.3 Computation times of large instances

The purpose of this section is to show whether computation time can indeed be saved by predicting the best heuristic using CLSP-Net compared to testing all available heuristics and choosing the best one. The creation of the training data set and the configuration and training of the ANN causes a one-time expenditure of around 61–62 h. It should be noted that the training time required for each configuration to be tested is approximately 30 s (see Fig. 8).[10] In contrast, the computation time for predicting the best heuristic for an unknown CLSP instance which is about 16 times larger than the training instances using CLSP-Net is 1.67 s.

According to this, it can be stated that the use of CLSP-Net to select a heuristic increases the solution time for a very large instance by only a few seconds (< 2 s). On the other hand, it is also obvious that the creation of CLSP-Net (especially the generation of the training, validation and test set) takes a lot of time.

Figure 9 shows the computation times for the considered heuristics as well as for the exact solver Gurobi (mipgap = 0.1%) depending on the problem size. To determine the computation times, 10 instances of the scenario combination U2-SC2-D2 were solved and their computation times averaged for each problem size. The instances of the data set J72T12G24 serve as a reference value for the problem size which is then doubled, tripled, etc..

[10] In comparison, the training phase of the forecasting method »conditional empirical distribution« for the combination of four features and six classes per feature requires around 5–6 h.

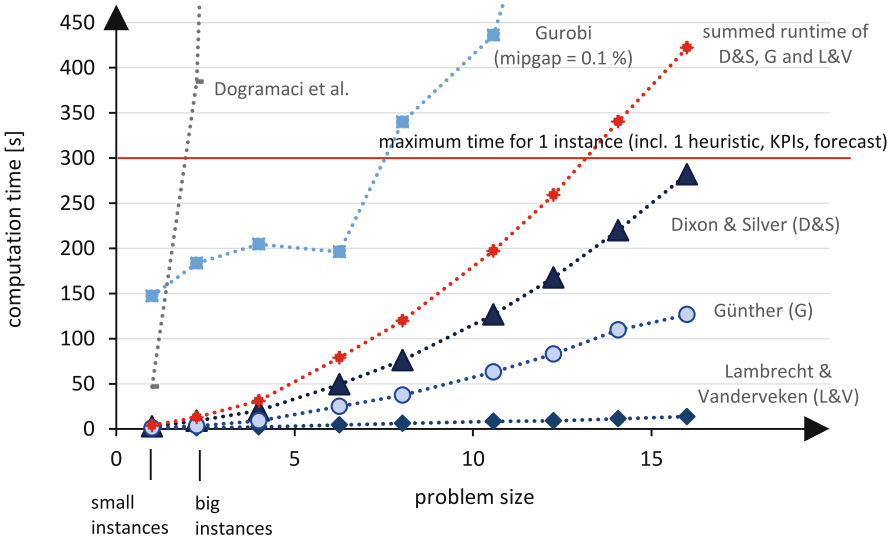


Fig. 9 Computation times depending on the problem size (U2-SC2-D2)

In the following, it is assumed that the total solution time for an instance of the scenario combination U2-SC2-D2 must not exceed 300 s. Since the heuristic of Dogramaci et al. already requires an average computation time of more than 350 s for a problem size of 2, it is excluded as a heuristic for very large instances of the type U2-SC2-D2. Gurobi, on the other hand, can meet the time requirement up to a problem size of 7 on average, although the computation time varies greatly depending on the instance being solved. The remaining heuristics have an average computation time of less than 300 s even for a problem size of 16. The red dashed line represents the summed computation time of the heuristics of Dixon & Silver, Günther and Lambrecht & Vanderveken.

It can be seen that up to a problem size of 12, theoretically all three heuristics can be compared without violating the time constraint in order to select the best heuristic. Starting with a problem size of 13, it is no longer possible to test all three heuristics while meeting the time constraint. At this point, CLSP-Net had to be used to select the best heuristic with an accuracy of about 78%. In addition, CLSP-Net might also be beneficial for much smaller instances if computation times are crucial, since prediction with CLSP-Net takes only a few seconds (problem size 16: < 2 s) and computation time can be saved when running two heuristics less.

7.4 Performance on data sets from the literature

In this section the test bed is enriched with two data sets from the literature. The first one (ABC set) is introduced by Maes and Van Wassenhove (1986) and constists of

720 CLSP instances. The second one (TTM set)[11] is originating from the data set used by Trigeiro et al. (1989) and constists of 751 CLSP instances. Since our CLSP model does not consider setup times, the TTM set was adapted (setup time for all products was set to 0). The CLSP instances vary in size, with the smallest instance comprising four products and 15 periods and the largest instance comprising 24 products and 30 periods. The number of products within an instance ranges from four to 30, while the number of periods within an instance ranges from 12 to 30.

Table 10 shows the computational results of the heuristics. In both data sets, DPA has the smallest average deviation from the optimal objective function value (obj. [%]). At the same time, DPA can best solve most instances of both data sets (best) and requires the highest average runtime (cpu [s]) compared to the other heuristics. It is noticeable that all heuristics except D&S perform worse in data set TTM than in data set ABC. This observation suggests that the instances in the TTM set are more challenging. One difference to data set J72T12G24 is that L&V can best solve a significantly higher proportion of instances in both external data sets. In data set J72T12G24 L&V can only solve 1.5% of the instances best and thus does not play a significant role.

The following section will examine the performance of the proposed methodology using the extended data set. For this purpose, a new neural network, designated as CLSP-Net V1.1, will be trained. Following the computation and standardisation of the features and the identification of the best heuristic, instances from the ABC set and TTM set will be divided into training, validation, and test sets in accordance with the procedures outlined in Sect. 6. The configuration of the new network is identical to that of CLSP-Net, but the training process now incorporates the training sets from J72T12G24, ABC, and TTM.

Table 11 presents the test results of CLSP-Net V1.1, aggregated by the various data sets and heuristics. Here, *corr.* denotes the number of correctly classified instances, while *labels* indicates the total number of labels included in the test set. Additionally, *acc. [%]* represents the accuracy as a percentage.

As can be observed, CLSP-Net V1.1 achieves an overall accuracy of 75.43% for the entire test set (All). This performance is slightly lower compared to the test results presented in Sect. 7.1. Analyzing the individual data sets, it is evident that CLSP-Net V1.1 attains a high accuracy of over 78% for the J72T12G24 data set. The decline in performance is due to the external data sets, where CLSP-Net V1.1 only achieves an accuracy of 59.17% (ABC) and 63.60% (TTM).[12]

Upon analysing the individual heuristics, it becomes evident that the D&S instances and DPA instances are recognised with particularly high accuracy (> 76%). However, there are significant differences between the data sets. CLSP-Net

[11] As provided at https://suerie.de/testsets.html.

[12] For comparison, we also trained a network with an identical configuration using only the training sets from ABC and TTM. After applying stronger regularization to mitigate overfitting, this network attained an accuracy of 58.33% (ABC) and 61.43% (TTM) on the test sets. The primary reason for the inferior performance is thus attributed to the limited number of training instances.

Table 10 Computational results for data set ABC and TTM

	L&V		D&S		DPA		G	
	obj. [%] best [no.]	cpu [s] best [%]	obj. [%] best [no.]	cpu [s] best [%]	obj. [%] best [no.]	cpu [s] best [%]	obj. [%] best [no.]	cpu [s] best [%]
ABC	7.69	**0.17**	5.01	0.82	**2.73**	53.54	3.64	0.34
	146	20.3	73	10.1	394	**54.7**	107	14.9
TTM	21.66	**0.09**	4.68	0.48	**3.34**	15.08	4.75	0.22
	87	11.6	81	10.8	448	**59.7**	135	18.0

L&V: Lambrecht & Vanderveken, D&S: Dixon & Silver, DPA: Dogramaci et al., G: Günther
The best heuristic with respect to the features obj., cpu and best [%] is highlighted in bold

Table 11 Accuracy of CLSP-Net V1.1 on enriched test bed

	L&V		D&S		DPA		G		All	
	corr.	labels	corr.	labels	corr.	labels	corr.	labels	corr.	labels
Test sets	acc. [%]		acc. [%]		acc. [%]		acc. [%]		acc. [%]	
J72T12G24	1	2	698	876	781	950	399	572	1879	2400
	50.00		79.68		**82.21**		69.76		78.29	
ABC	8	11	4	15	125	202	5	12	142	240
	72.73		26.67		61.88		41.67		59.17	
TTM	1	4	10	14	136	210	12	22	159	250
	25.00		**71.43**		64.76		54.55		63.60	
All	10	17	712	905	1042	1362	416	606	2180	2890
	58.82		**78.67**		76.51		68.65		75.43	

L&V: Lambrecht & Vanderveken, D&S: Dixon & Silver, DPA: Dogramaci et al., G: Günther
The best accuracy of CLSP-Net V1.1 in dependence of heuristic and test set is highlighted in bold

V1.1 shows the poorest performance in recognising L&V instances, which can be attributed to the relatively small number of L&V instances within the training set.

In Table 12, we compare the various approaches based on the average deviation from the optimal objective function value. In addition to CLSP-Net V1.1, the considered CLSP heuristics are also employed for comparative analysis. It is assumed that each respective heuristic is used for all instances in the test set. ASopt always selects the best heuristic and can be interpreted as a baseline of what could be achievable by having an algorithm selector that always selects the best heuristic.

CLSP-Net V1.1 achieves an overall deviation of 2.40%, which is very close to the best achievable performance of 2.08% (ASopt). Moreover, CLSP-Net V1.1 outperforms all heuristics. In the ABC set, the DPA heuristic achieves a nearly identical optimality gap. It is very intresting that the lower accuracy of CLSP-Net V1.1 in the ABC and TTM sets results in a deviation from ASopt by approximately 0.6% points.

Table 12 Optimality gap performance [%] on test sets

Test sets	L&V	D&S	DPA	G	CLSP-Net V1.1	ASopt
J72T12G24	8.78	3.64	4.94	5.18	2.26	2.01
ABC	7.89	4.98	2.68	3.69	2.62	2.03
TTM	23.27	4.87	3.71	5.22	3.46	2.81
All	9.96	3.86	4.63	5.06	2.40	2.08

In contrast, the optimality gap for J72T12G24 deviates from ASopt by approximately 0.2% points.

8 Summary and conclusions

Two different forecasting methods have been presented for predicting the best heuristic depending on the characteristics of the considered instance for the Capacitated Lotsizing Problem (CLSP). As input, the forecasting methods use features, which are to be computed beforehand for each CLSP instance. We propose that features should be selected that are both relative and relevant. The use of relative features ensures that the forecasting method can be applied for different problem sizes, since the number of features to be determined and their interpretation are identical for any problem size and therefore no adjustments are necessary. We have identified 17 features for the CLSP.

For training and validation of the forecasting methods, an extensive data set (J72T12G24) is created, which considers the lotsizing of fictitious tire manufacturers. The data set J72T12G24 considers six different demand scenarios, two scenarios regarding utilization, and three different scenarios regarding the ratio of setup and holding costs. The data set includes a total of 7,200 »small« CLSP instances, which have very different characteristics due to the combination of scenarios. All 7,200 instances are solved using the heuristics of Lambrecht and Vanderveken (1979), Dixon and Silver (1981), Dogramaci et al. (1981) and Günther (1987). The results show that the selected heuristics have different solution qualities depending on the characteristics of each instance, and none of the heuristics dominates the rest of the heuristics.

The best recognized approach to select the best heuristic depending on the characteristics of the considered CLSP instance is based on an artificial neural network (ANN) with two hidden layer called CLSP-Net. As input, CLSP-Net is given 17 features, and as output, CLSP-Net determines a probability for each of the four heuristics, whereby the heuristic with the highest value has to be selected. The numerical tests show that the trained CLSP-Net achieves an accuracy of 78% for the test set of data set J72T12G24. An alternative forecasting method uses conditional probabilities and reaches an accuracy of 70%.

With the data set J108T18G36, another data set is created that contains »big« CLSP instances 125% larger than the data set J72T12G24. It should be mentioned

that none of the forecasting methods was trained on the basis of such big instances. The numerical tests show that CLSP-Net achieves a comparable accuracy of 78% without a new training phase. Therefore, it can be stated that it is possible to train an ANN on the basis of small instances in order to subsequently apply it to big instances without significant loss of performance. However, this finding is based on the assumption that the big instances have similar patterns like the small instances of the training set.

Furthermore, the use of a set of »large« CLSP instances serves to demonstrate that the execution of all four heuristics and the selection of the best one would take up to 450 s. Using CLSP-Net, we can predict the best heuristic with a probability of about 78% in less than 2 s. Following that, it requires an average of about 112 s ($= 450/4$) to execute the predicted heuristic. This indicates that computation time can be saved compared to testing all heuristics regardless of the problem size. But it also can be seen that the creation of the training set in particular is very time-consuming. The time required for the configuration of CLSP-Net is of course dependent on the user, whereby the pure training time per configuration is only 30 s. For our experiments, in total, a one-time time expenditure of more than 61 h has been measured. Therefore, the variable time of a single forecast stands in contrast to the fixed time to generate and solve the test bed and to design and train the ANN. Such a procedure is only useful if there is sufficient time for preparing the planning problem, but only short time when a certain instance substantiates. Promising examples of such situations are online-/real-time-planning or iterative heuristics like local search or decomposition methods, which have to select appropriate instance-specific rules in every iteration.

With CLSP-Net V1.1, a second neural network was trained. The configuration is identical to that of CLSP-Net, but the training process involved, in addition to J72T12G24, two data sets sourced from the literature. Due to the limited number of instances in the external data sets, CLSP-Net V1.1 achieves an accuracy of 75%, which is slightly lower than that of CLSP-Net. With regard to the average deviation from the optimal objective value, CLSP-Net V1.1 achieves an optimality gap of 2.40%, which differs by only 0.6% points from the best achievable gap.

Further research should examine in more detail which of the proposed features are important and which are less important. During the validation phase of the forecasting method »conditional empirical distribution« (see Sect. 6.2), initial insights into the suitability of the proposed features could already be obtained (for example, features 17, 4 and 12 appear to be highly relevant). Further insights could be achieved, for example, with the help of a decision tree algorithm. This is done by Visentin et al. (2024).

A. Numerical study – input data

In Table 13 the mathematical implementation of the instance generator is shown. In the following tables the input data for the generation of the data sets J72T12G24 (Table 14) and J108T18G36 (Table 15) can be found.

Table 13 Mathematical implementation of the instance generator

Model parameters	Value assignment		with.	
Production coefficient	$a_j \sim \mathcal{N}(\mu_g, \sigma_g^2)$	$\forall j \in FP_g$	$\mu_g \sim \mathcal{G}(a + cv_g, b + cv_g)$ $\sigma_g^2 \sim \mathcal{G}(a, b)$	$\forall g$ $\forall g$
Holding cost rate	$hc_j \sim \mathcal{N}(\mu_g, \sigma_g^2)$	$\forall j \in FP_g$	$\mu_g \sim \mathcal{G}(a + cv_g, b + cv_g)$ $\sigma_g^2 \sim \mathcal{G}(a, b)$	$\forall g$ $\forall g$
Setup cost rate	$sc_j \sim \mathcal{N}(\mu_g, \sigma_g^2)$	$\forall j \in FP_g$	$\mu_g = \bar{d}_g \cdot \bar{hc} \cdot sc^{ratio}$ $\sigma_g^2 \sim \mathcal{G}(a, b)$	$\forall g$ $\forall g$
Demand-Scenario	Description	Value assignment	with.	
D1	Fluctuating demand, increase in $t = 7$	$d_{jt} \sim \mathcal{G}(a, b)$ $d_{jt} \sim \mathcal{G}(a + x, b + x)$	$\forall j, t = 1, ..., 6$ $\forall j, t = 7, ..., T$ $x > 0$	
D2	Increasing demand	$d_{j1} = S_j \forall j$ $d_{jt} = d_{jt-1} + E_j \cdot (1 + c_{jt})$ $\forall j, t = 2, ..., T$	$S_j \sim \mathcal{G}(a, b)$ $E_j \sim \mathcal{G}(a, b)$ $c_{jt} \sim \mathcal{G}(a, b)$	$\forall j$ $\forall j$ $\forall j, t$
D3	Decreasing demand	$d_{j1} = S_j \forall j$ $d_{jt} = d_{jt-1} - R_j \cdot (1 + c_{jt})$ $\forall j, t = 2, ..., T$	$S_j \sim \mathcal{G}(a, b)$ $R_j \sim \mathcal{G}(a, b)$ $c_{jt} \sim \mathcal{G}(a, b)$	$\forall j$ $\forall j$ $\forall j, t$
D4	Steady demand	$d_{jt} \sim \mathcal{N}(\mu_j, \sigma_j^2) \forall j, t$	$\mu_j \sim \mathcal{G}(a, b)$ $\sigma_j^2 \sim \mathcal{G}(a, b)$	$\forall j$ $\forall j$
D5	Seasonal demand	$d_{jt} = w_j \cdot cs_t \forall j, t$	$w_j \sim \mathcal{G}(a, b)$	$\forall j$
D6	Mix	Random drawing D1 - D5		

Notation:

j = index products
FP_g = set of products in product family g
g = index product family
sc^{ratio} = ratio of setup and holding costs
$\bar{hc}$ = ø holding costs
$\bar{d}_g$ = ø demand of g
$\mathcal{N}$ = normal distribution
μ = expected value
σ^2 = variance
$\mathcal{G}$ = uniform distribution
a = lower bound
b = upper bound
S = initial value
E = absolute increase
c = fluctuation coeff.
R = absolute reduction
cs = seasonal coefficient
w = base demand
cv_g = value depending on g by which the interval $[a, b]$ is shifted

Table 14 Input data - DC J72T12G24

Model parameters	Input data	$seed = 1$
Production coeff. a_j	$\mu_g \sim \mathcal{G}(0.05 + cv_g, 0.8 + cv_g)$	$\forall\, g$
	$\sigma_g^2 \sim \mathcal{G}(0.01, 0.05)$	$\forall\, g$
	with .	
	$cv_g = 0$	$\forall g = [1, ..., 8]$
	$cv_g = 0.2$	$\forall g = [9, ..., 16]$
	$cv_g = 0.4$	$\forall g = [17, ..., 24]$
Holding cost rate hc_j	$\mu_g \sim \mathcal{G}(2 + cv_g, 15 + cv_g)$	$\forall\, g$
	$\sigma_g^2 \sim \mathcal{G}(0.2, 0.8)$	$\forall\, g$
	with .	
	$cv_g = 0$	$\forall g = [1, ..., 8]$
	$cv_g = 2$	$\forall g = [9, ..., 16]$
	$cv_g = 4$	$\forall g = [17, ..., 24]$
Setup cost rate sc_j	$\sigma_g^2 \sim \mathcal{G}(5, 20)$	$\forall\, g$
Capacity K_t	$K_t = 1000$	$\forall\, t$
Demand-Scenario	Input data	
D1	$d_{jt} \sim \mathcal{G}(5, 20)$	$\forall j, t = [1, 2, 3, 4, 5, 6]$
	$d_{jt} \sim \mathcal{G}(15, 30)$	$\forall j, t = [7, 8, 9, 10, 11, 12]$
D2	$S_j \sim \mathcal{G}(2, 12)$	$\forall\, j$
	$E_j \sim \mathcal{G}(0.5, 2.5)$	$\forall\, j$
	$c_{jt} \sim \mathcal{G}(-0.5, 0.5)$	$\forall\, j, t$
D3	$S_j \sim \mathcal{G}(15, 25)$	$\forall\, j$
	$R_j \sim \mathcal{G}(0.5, 3.1)$	$\forall\, j$
	$c_{jt} \sim \mathcal{G}(-0.5, 0.5)$	$\forall\, j, t$
D4	$\mu_j \sim \mathcal{G}(5, 30)$	$\forall\, j$
	$\sigma_j^2 \sim \mathcal{G}(0.5, 1.0)$	$\forall\, j$
D5	$w_j \sim \mathcal{G}(10, 32)$	$\forall\, j$
	$cs_t = [0.3, 0.8, 1.2, 0.85, 0.6, 0.4, 0.75, 1.1, 1.3, 1.2, 0.9, 0.6]$	
SC-Scenario	Description	Input data
SC1	Relatively high setup costs	$sc^{ratio} = 2$
SC2	Base scenario	$sc^{ratio} = 1$
SC3	Relatively low setup costs	$sc^{ratio} = 0.1$

Table 15 Input data - DC J108T18G36

Model parameters	Input data	$seed = 1$
Production coeff. a_j	$\mu_g \sim \mathcal{G}(0.05 + cv_g, 0.8 + cv_g)$	$\forall\, g$
	$\sigma_g^2 \sim \mathcal{G}(0.01, 0.05)$	$\forall\, g$
	with .	
	$cv_g = 0$	$\forall g = [1, ..., 12]$
	$cv_g = 0.2$	$\forall g = [13, ..., 24]$
	$cv_g = 0.4$	$\forall g = [25, ..., 36]$
Holding cost rate hc_j	$\mu_g \sim \mathcal{G}(2 + cv_g, 15 + cv_g)$	$\forall\, g$
	$\sigma_g^2 \sim \mathcal{G}(0.2, 0.8)$	$\forall\, g$
	with .	
	$cv_g = 0$	$\forall g = [1, ..., 12]$
	$cv_g = 2$	$\forall g = [13, ..., 24]$
	$cv_g = 4$	$\forall g = [25, ..., 36]$
Setup cost rate sc_j	$\sigma_g^2 \sim \mathcal{G}(5, 20)$	$\forall\, g$
Capacity K_t	$K_t = 1500$	$\forall\, t$
Demand-Scenario	Input data	
D1	$d_{jt} \sim \mathcal{G}(5, 20)$	$\forall j, t = [1, ..., 9]$
	$d_{jt} \sim \mathcal{G}(15, 30)$	$\forall j, t = [10, ..., 18]$
D2	$S_j \sim \mathcal{G}(2, 12)$	$\forall\, j$
	$E_j \sim \mathcal{G}(0.25, 1.25)$	$\forall\, j$
	$c_{jt} \sim \mathcal{G}(-0.5, 0.5)$	$\forall\, j, t$
D3	$S_j \sim \mathcal{G}(15, 25)$	$\forall\, j$
	$R_j \sim \mathcal{G}(0.25, 1.55)$	$\forall\, j$
	$c_{jt} \sim \mathcal{G}(-0.5, 0.5)$	$\forall\, j, t$
D4	$\mu_j \sim \mathcal{G}(5, 30)$	$\forall\, j$
	$\sigma_j^2 \sim \mathcal{G}(0.5, 1.0)$	$\forall\, j$
D5	$w_j \sim \mathcal{G}(10, 32)$	$\forall\, j$
	$cs_t = [0.3, 0.5, 0.8, 1.2, 1.05, 0.85, 0.6, 0.45, 0.4, 0.75, 0.95, 1.1, 1.3, 1.25, 1.2, 0.9, 0.85, 0.6]$	

SC-Scenario	Description	Input data
SC1	Rlatively high setup costs	$sc^{ratio} = 2$
SC2	Base scenario	$sc^{ratio} = 1$
SC3	Relatively low setup costs	$sc^{ratio} = 0.1$

B. Numerical study – results

Table 16 shows the results of the heuristics for data set J72T12G24 depending on the utilization and setup cost scenario.

Table 16 Computational results in dependence of U and SC

		L&V (1979)		D&S (1981)		DPA (1981)		G (1987)	
		obj. [%]	cpu [s]	obj. [%]	cpu [s]	obj. [%]	cpu [s]	obj. [%]	cpu [s]
		best [no.]	best [%]	best [no.]	best [%]	best [no.]	best [%]	best [no.]	best [%]
U1	SC1	10.05	**0.25**	**2.51**	2.22	3.44	50.86	2.68	0.91
		26	2.2	552	**46.0**	376	31.3	246	20.5
	SC2	6.49	**0.19**	1.45	1.51	2.87	39.18	**1.39**	0.64
		26	2.2	307	25.6	580	**48.3**	287	23.9
	SC3	3.57	**0.10**	3.27	0.26	**1.35**	2.44	5.74	0.14
		17	1.4	247	20.6	631	**52.6**	305	25.4
U2	SC1	17.41	**0.28**	**4.50**	2.14	10.17	41.37	4.62	0.89
		11	0.9	754	**62.8**	91	7.6	344	28.7
	SC2	10.48	**0.24**	3.70	1.51	8.96	32.77	**3.24**	0.71
		20	1.7	391	32.6	394	32.8	395	**32.9**
	SC3	5.92	**0.17**	5.18	0.39	**2.81**	2.26	12.23	0.38
		6	0.5	302	25.2	641	**53.4**	251	20.9
All		8.99	**0.21**	**3.44**	1.34	4.93	28.15	4.98	0.61
		106	1.5	2553	35.5	2713	**37.7**	1828	25.4

L&V: Lambrecht & Vanderveken, D&S: Dixon & Silver, DPA: Dogramaci et al., G: Günther

obj.: average deviation from the optimal objective function value, cpu: average computation time, best: number [no.] or fraction [%] of all instances which is best solved with the considered heuristic

The best heuristic with respect to the features obj., cpu and best [%] is highlighted in bold

Funding Open Access funding enabled and organized by Projekt DEAL.

References

Aarts E, Reijnhoudt M, Stehouwer H, Wessels J (2000) A novel decomposition approach for on-line lot-sizing. Eur J Oper Res 122(2):339–353. https://doi.org/10.1016/S0377-2217(99)00237-4

Arnold F, Sörensen K (2019) What makes a VRP solution good? The generation of problem-specific knowledge for heuristics. Comput Oper Res 106:280–288. https://doi.org/10.1016/j.cor.2018.02.007

Bengio Y, Frejinger E, Lodi A, Patel R, Sankaranarayanan S (2020) A learning-based algorithm to quickly compute good primal solutions for stochastic integer programs. In: Hebrard E, Musliu N (Eds.), Integration of Constraint Programming, Artificial Intelligence, and Operations Research.

CPAIOR 2020. Lecture Notes in Computer Science 12296. Springer, Cham, pp 99–111. https://doi.org/10.1007/978-3-030-58942-4_7
Bishop CM (2006) Pattern recognition and machine learning. Springer, New York
Bitran GR, Yanasse HH (1982) Computational complexity of the capacitated lot size problem. Manage Sci 28(10):1174–1186. https://doi.org/10.1287/mnsc.28.10.1174
Brazdil P, Giraud-Carrier C, Soares C, Vilalta R (2009) Metalearning: applications to data mining. Springer, Berlin, Heidelberg
Buschkühl L, Sahling F, Helber S, Tempelmeier H (2010) Dynamic capacitated lot-sizing problems: a classification and review of solution approaches. OR Spectrum 32(2):231–261. https://doi.org/10.1007/s00291-008-0150-7
Chen W-H, Thizy J-M (1990) Analysis of relaxations for the multi-item capacitated lot-sizing problem. Annals Oper Res 26(1–4):29–72. https://doi.org/10.1007/BF02248584
Chu X, Cai F, Cui C, Hu M, Li L, Qin Q (2019) Adaptive recommendation model using meta-learning for population-based algorithms. Inf Sci 476:192–210. https://doi.org/10.1016/j.ins.2018.10.013
Copil K, Wörbelauer M, Meyr H, Tempelmeier H (2017) Simultaneous lotsizing and scheduling problems: a classification and review of models. OR Spectrum 39(1):1–64. https://doi.org/10.1007/s00291-015-0429-4
Costa VG, Pedreira CE (2023) Recent advances in decision trees: an updated survey. Artif Intell Rev 56(5):4765–4800. https://doi.org/10.1007/s10462-022-10275-5
Dantas AL, Pozo ATR (2018) A meta-learning algorithm selection approach for the quadratic assignment problem. In: IEEE Congress on Evolutionary Computation. CEC 2018. IEEE, Washington, DC, USA, pp 1–8. https://doi.org/10.1109/CEC.2018.8477989
DeMatteis JJ (1968) An economic lot-sizing technique, I: the part-period algorithm. IBM Syst J 7(1):30–38. https://doi.org/10.1147/sj.71.0030
Dixon PS, Silver EA (1981) A heuristic solution procedure for the multi-item, single-level, limited capacity, lot-sizing problem. J Oper Manag 2(1):23–39. https://doi.org/10.1016/0272-6963(81)90033-4
Dogramaci A, Panayiotopoulos JC, Adam NR (1981) The dynamic lot-sizing problem for multiple items under limited capacity. AIIE Trans 13(4):294–303. https://doi.org/10.1080/05695558108974565
Eisenhut PS (1975) A dynamic lot sizing algorithm with capacity constraints. AIIE Trans 7(2):170–176. https://doi.org/10.1080/05695557508974999
Gaafar LK, Choueiki M (2000) A neural network model for solving the lot-sizing problem. Omega 28(2):175–184. https://doi.org/10.1016/S0305-0483(99)00035-3
Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge MA
Groff GK (1979) A lot sizing rule for time-phased component demand. Prod Invent Manag 20(1):47–53
Günther H-O (1987) Planning lot sizes and capacity requirements in a single stage production system. Eur J Oper Res 31(2):223–231. https://doi.org/10.1016/0377-2217(87)90026-9
Gutierrez-Rodríguez AE, Conant-Pablos SE, Ortiz-Bayliss JC, Terashima-Marín H (2019) Selecting meta-heuristics for solving vehicle routing problems with time windows via meta-learning. Expert Syst Appl 118:470–481. https://doi.org/10.1016/j.eswa.2018.10.036
Hinton G (2012) Lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude. Slides of Lecture Neural Networks for Machine Learning. URL: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (visited on 05/16/2023)
Jans R, Degraeve Z (2007) Meta-heuristics for dynamic lot sizing: a review and comparison of solution approaches. Eur J Oper Res 177(3):1855–1875. https://doi.org/10.1016/j.ejor.2005.12.008
Jans R, Degraeve Z (2008) Modeling industrial lot sizing problems: a review. Int J Prod Res 46(6):1619–1643. https://doi.org/10.1080/00207540600902262
Kanda J, de Carvalho A, Hruschka E, Soares C, Brazdil P (2016) Meta-learning to select the best meta-heuristic for the traveling salesman problem: a comparison of meta-features. Neurocomputing 205:393–406. https://doi.org/10.1016/j.neucom.2016.04.027
Kanda J, de Carvalho A, Hruschka E, Soares C (2011) "Using meta-learning to recommend meta-heuristics for the traveling salesman problem". In: 10th International Conference on Machine Learning and Applications. ICMLA 2011. IEEE, Washington, DC, USA, pp. 346–351. https://doi.org/10.1109/ICMLA.2011.153
Kanda J, Soares C, Hruschka E, de Carvalho A (2012) A meta-learning approach to select meta-heuristics for the traveling salesman problem using mlp-based label ranking. In: Huang T, Zeng Z, Li C, Leung CS (Eds.), Neural Information Processing. ICONIP 2012. Lecture Notes in Computer Science 7665. Springer, Berlin, Heidelberg, pp 488–495. https://doi.org/10.1007/978-3-642-34487-9_59

Karimi B, Fatemi Ghomi S, Wilson J (2003) The capacitated lot sizing problem: a review of models and algorithms. Omega 31(5):365–378. https://doi.org/10.1016/S0305-0483(03)00059-8

Karimi-Mamaghan M, Mohammadi M, Meyer P, Karimi-Mamaghan AM, Talbi E-G (2022) Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: a state-of-the-art. Eur J Oper Res 296(2):393–422. https://doi.org/10.1016/j.ejor.2021.04.032

Kerschke P, Hoos HH, Neumann F, Trautmann H (2019) Automated algorithm selection: survey and perspectives. Evol Comput 27(1):3–45. https://doi.org/10.1162/evco_a_00242

Kirca Ö, Kökten M (1994) A new heuristic approach for the multi-item dynamic lot sizing problem. Eur J Oper Res Lotsizing Models Prod Plan 75(2):332–341. https://doi.org/10.1016/0377-2217(94)90078-7

Kotsiantis SB (2013) Decision trees: a recent overview. Artif Intell Rev 39(4):261–283. https://doi.org/10.1007/s10462-011-9272-4

Lambrecht MR, Vanderveken H (1979) Heuristic procedures for the single operation, multi-item loading problem. AIIE Trans 11(4):319–326. https://doi.org/10.1080/05695557908974478

Lee I, Gupta JND, Amar AD (2001) A multi-neural-network learning for lot sizing and sequencing on a flowshop. In: G. Lamont (Ed.), Proceedings of the 2001 ACM Symposium on Applied Computing. SAC 2001. ACM Press, New York, NY, USA, pp 36–40. https://doi.org/10.1145/372202.372210

Lucas F, Billot R, Sevaux M, Sörensen K (2020) Reducing space search in combinatorial optimization using machine learning tools. In: Kotsireas IS, Pardalos PM (Eds.), Learning and Intelligent Optimization. LION 2020. Lecture Notes in Computer Science 12096. Springer, Cha, pp 143–150. https://doi.org/10.1007/978-3-030-53552-0_15

Maes J, Van Wassenhove LN (1986) A simple heuristic for the multi item single level capacitated lotsizing problem. Oper Res Lett 4(6):265–273. https://doi.org/10.1016/0167-6377(86)90027-1

Maes J, Van Wassenhove LN (1988) Multi-item single-level capacitated dynamic lot-sizing heuristics: a general review. J Oper Res Soc 39(11):991–1004. https://doi.org/10.1057/jors.1988.169

Megala N, Jawahar N (2006) Genetic algorithm and Hopfield neural network for a dynamic lot sizing problem. Int J Adv Manuf Technol 27(11):1178–1191. https://doi.org/10.1007/s00170-004-2306-1

Miranda ES, Fabris F, Nascimento CGM, Freitas AA, Oliveira ACM (2018) Meta-learning for recommending metaheuristics for the MaxSAT problem. In: 7th Brazilian Conference on Intelligent Systems. BRACIS 2018. IEEE, Washington, DC, USA, pp 169–174. https://doi.org/10.1109/BRACIS.2018.00037

Murphy KP (2012) Machine learning: a probabilistic perspective. Adaptive computation and machine learning series. MIT Press, Cambridge, MA

Nikolić M, Marić F, Janičić P (2013) Simple algorithm portfolio for SAT. Artif Intell Rev 40(4):457–465. https://doi.org/10.1007/s10462-011-9290-2

Nwankpa CE, Gachagan A, Marshall S (2021) Activation functions: comparison of trends in practice and research for deep learning. In: 2nd International Conference on Computational Sciences and Technology. INCCST 2020. Jamshoro, Pakistan. url: https://strathprints.strath.ac.uk/id/eprint/75897

Park TJ, Jang YJ (2022) Discrete lot-sizing problem of single machine based on reinforcement learning approach. In: Proceedings of the 2022 International Symposium on Semiconductor Manufacturing Intelligence. ISMI 2022

Paternina-Arboleda CD, Das TK (2005) A multi-agent reinforcement learning approach to obtaining dynamic control policies for stochastic lot scheduling problem. Simul Model Pract Theory 13(5):389–406. https://doi.org/10.1016/j.simpat.2004.12.003

Radzi, NHM, Haron H, Johari TIT (2006) Lot sizing using neural network approach. In: Proceedings of the 2nd IMT-GT Regional Conference on Mathematics, Statistics and Applications. Penang, Malaysia

Rice JR (1976) The algorithm selection problem. In: Rubinoff M, Yovits MC (Eds.), Advances in computers, vol 15. pp 65–118. https://doi.org/10.1016/S0065-2458(08)60520-3

Rohde J (2004) Hierarchical supply chain planning using artificial neural networks to anticipate base-level outcomes. OR Spectrum 26(4):471–492. https://doi.org/10.1007/s00291-004-0170-x

Rummukainen H, Nurminen JK (2019) Practical reinforcement learning-experiences in lot scheduling application. IFAC-PapersOnLine 52(13):1415–1420. https://doi.org/10.1016/j.ifacol.2019.11.397

Sadeg S, Hamdad L, Kada O, Benatchba K, Habbas Z (2021) Meta-learning to Select the best metaheuristic for the MaxSAT problem. In: Chikhi S, Amine A, Chaoui A, Saidouni DE, Kholladi MK (Eds.), Modelling and Implementation of Complex Systems. MISC 2020. Lecture Notes in Networks and Systems 156. Springer, Cham, pp. 122–135. https://doi.org/10.1007/978-3-030-58861-8_9

Senyiğit E, Atici U, (2013) Artificial neural network models for lot-sizing problem: a case study. Neural Comput Appl 22(6):1039–1047. https://doi.org/10.1007/s00521-012-0863-z

Senyiğit E, Düğenci M, Aydin ME, Zeydan M, (2013) Heuristic-based neural networks for stochastic dynamic lot sizing problem. Appl Soft Comput 13(3):1332–1339. https://doi.org/10.1016/j.asoc.2012.02.026

Silver EA, Meal HC (1969) A simple modification of the EOQ for the case of a varying demand rate. Prod Invent Manag 10(4):51–55

Silver EA, Meal HC (1973) A heuristic for selecting lot size quantities for the case of a deterministic time-varying demand rate and discrete opportunities for replenishment. Prod Invent Manag 14(2):64–74

Smith-Miles KA (2008) Towards insightful algorithm selection for optimisation using meta-learning concepts. In: IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence). IJCNN 2008. IEEE, Washington, DC, USA, pp 4118–4124. https://doi.org/10.1109/IJCNN.2008.4634391

Smith-Miles KA, van Hemert J, Lim XY (2010) Understanding TSP difficulty by learning from evolved instances'. In: Blum C, Battiti R (Eds.), Learning and Intelligent Optimization. LION 2010. Lecture Notes in Computer Science 6073. Springer, Berlin, Heidelberg, pp 266–280. https://doi.org/10.1007/978-3-642-13800-3_29

Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(56):1929–1958

Stützle T (2006) Iterated local search for the quadratic assignment problem. Eur J Oper Res 174(3):1519–1539. https://doi.org/10.1016/j.ejor.2005.01.066

Trigeiro WW, Thomas LJ, McClain JO (1989) Capacitated lot sizing with setup times. Manage Sci 35(3):353–366

van Hezewijk L, Dellaert N, Van Woensel T, Gademann N (2023) Using the proximal policy optimisation algorithm for solving the stochastic capacitated lot sizing problem. Int J Prod Res 61(6):1955–1978. https://doi.org/10.1080/00207543.2022.2056540

van Nunen J, Wessels J (1978) Multi-item lot size determination and scheduling under capacity constraints. Eur J Oper Res 2(1):36–41. https://doi.org/10.1016/0377-2217(78)90121-2

Vander Eecken J, Lambrecht MR, Lindebrings JP (1975) Selection of lot sizing procedures for the case of a deterministic time varying demand rate over a finite horizon. Bedrijfseconomische Verhandeling, D.T.E.W., No. 7501

Vens C, Struyf J, Schietgat L, Džeroski S, Blockeel H (2008) Decision trees for hierarchical multi-label classification. Mach Learn 73(2):185–214. https://doi.org/10.1007/s10994-008-5077-3

Visentin A, Gallchóir A, Kärcher J, Meyr H (2024) Explainable algorithm selection for the capacitated lot sizing problem. In: Dilkina B (Ed.), Integration of Constraint Programming, Artificial Intelligence, and Operations Research. CPAIOR 2024. Lecture Notes in Computer Science 14743. Cham: Springer, pp. 243–252. https://doi.org/10.1007/978-3-031-60599-4_16

Wong J-T, Su C-T, Wang C-H (2012) Stochastic dynamic lot-sizing problem using bi-level programming base on artificial intelligence techniques. Appl Math Model 36(5):2003–2016. https://doi.org/10.1016/j.apm.2011.08.017

Zennaki M, Ech-Cherif A (2010) A new machine learning based approach for tuning metaheuristics for the solution of hard combinatorial optimization problems. J Appl Sci 10(18):1991–2000. https://doi.org/10.3923/jas.2010.1991.2000

Zwietering PJ, van Kraaij MJAL, Aarts EHL, Wessels J (1991) Neural networks and production planning. Memorandum COSOR 9115. Technische Universiteit Eindhoven