Flexible Enterprise Optimization with Constraint Programming

Sytze P. E. Andringa([✉]) and Neil Yorke-Smith

Algorithmics Group, Delft University of Technology, Delft, The Netherlands
{s.p.e.andringa,n.yorke-smith}@tudelft.nl

Abstract. Simulation–optimization is often used in enterprise decision-making processes, both operational and tactical. This paper shows how an intuitive mapping from descriptive problem to optimization model can be realized with Constraint Programming (CP). It shows how a CP model can be constructed given a simulation model and a set of business goals. The approach is to train a neural network (NN) on simulation model inputs and outputs, and embed the NN into the CP model together with a set of soft constraints that represent business goals. We study this novel simulation–optimization approach through a set of experiments, finding that it is flexible to changing multiple objectives simultaneously, allows an intuitive mapping from business goals expressed in natural language to a formal model suitable for state-of-the-art optimization solvers, and is realizable for diverse managerial problems.

Keywords: Enterprise simulation · Constraint Programming · Deep learning · Simulation–optimization

1 Introduction

Simulation is widely used both to evaluate enterprises and in enterprise planning [16,29]. By running a simulation based on an enterprise model under various sets of parameters, one can get insight into how an enterprise might behave in complex or future scenarios. Various managerial interventions can be analysed for the likely outcomes. Hence, decision makers (DM) putting these models into practice are often interested in finding optimal inputs with respect to an observed 'problem' [18]. A problem describes an undesired property of the system that can be tackled by taking action. For the DM there is uncertainty about what course of action is best to take [6]; simulation can provide insights.

However, finding these inputs by trying out them all through simulation can be applied with only limited success on complex simulation processes, as the number of possible inputs grows exponentially with respect to the number of input parameters [18]. On the other hand, using a pure optimization model – i.e., omitting simulation – can be incapable of capturing all complexities and dynamics of a system [18]. Hence *simulation–optimization* (SO) seeks to combine the benefits of both by using simulation to represent the actual system and optimization to find optimal simulation inputs [28].

Misinterpretation between DM and optimization model in a SO approach can be disastrous, then, as it can result in the wrong problem being solved. As such, a valid translation from problem description to formal model is crucial [1]. This translation is non-trivial. From a modeller's perspective, existing enterprise modelling (EM) methods offer only tenuous concepts of 'problems' [6]. From a computational perspective, optimization models can be difficult for a DM to understand, especially when the model is full of formal mathematics. Indeed, Grossman [12] points out that the four common decomposition techniques used in enterprise-wide optimization all have a mathematical basis, in the sense that they aim to reduce the solution space or number of constraints. The emphasis in the literature is on model performance rather than easing the DM's process. A practical example can be found in [32], where the optimization model for a clothing line is modelled by several pages full of mathematical formulae.

This paper puts forward a novel use of Constraint Programming (CP), a declarative paradigm for defining combinatorial optimization problems. CP is seen as the closest approach to "the user states a problem, the computer solves it" [8]. It allows one to describe diverse real-world problems through constraints, i.e., statements which pose some relation among the problem's variables [25]. Notably, the expressiveness of CP means its formal models can be more human-like than, for instance, mixed integer programming models. Given a CP model, algorithms called 'solvers' assign values to variables of the CP model such that every constraint is satisfied. A constraint program thus only needs to express *what* we want to solve, not *how*.

Many leading EM frameworks are descriptive in nature [29]. Since CP is declarative, we argue CP can form an effective way to model problems in an more understandable but computer-parsable format. This requires keeping the CP model simple, which can be done by modelling complex properties of the system by a simulation model and have this incorporated into the CP model according to an automatic process.

Specifically, in this paper the simulation model is represented by a "model of a model", i.e., *meta-model* [15]. Meta-modelling techniques range from descriptive representations of ontological concepts to algorithms to make faster approximations of complex computer code [3,15]. We focus on the latter, by representing the simulation model by neural networks (NN). NN are capable of learning behaviour of complex systems and require little engineering by hand, making them applicable in many domains [4,19,28]. In this paper, a NN is trained on simulation data and *automatically embedded* into the CP model. We argue this adds additional flexibility during the problem solving process, since a change of objective can be evaluated without making additional simulation calls.

This flexibility can be desirable in various situations, for example when: 1) The DM does not know all specifications of the problem it wants to solve (for example, information by a third party is required) but does know how the system currently behaves, and would like to make preparations such that problems can be solved on-the-go without making computationally expensive simulation calls. 2) The DM has a large collection of problems that needs to be solved, making setting up a separate feedback loop for every problem infeasible. 3) The DM is not able to do simulation during the problem-solving phase.

Summarising, the contributions of this paper to the literature are: 1) Showing how an easy-to-understand CP program can be constructed which is interpretable for solver software, starting from a descriptive problem description. 2) Showing how NNs can be embedded in a CP model by means of *empirical model learning*, such that no additional simulation calls are necessary when a different problem needs to be tackled for the same enterprise. 3) Providing experiments together with supplementary code for a novel SO approach that embeds a NN into a CP model. 4) Showing how a pareto front can be approximated through soft constraints, such that a DM can get proper insight in the scope of possible actions and their impact on the system.

2 Background and Related Work

2.1 Constraint Programming

CP is an expressive yet practical approach to optimization, used in a wide variety of applications [25,30]. It solves constraint satisfaction problems (CSP), which consist of a set of variables, each with a domain of permitted values, and a set of constraints specified in a logical formalism. A solution to a CSP is an assignment of a value to every variable from its domain, such that all the constraints are satisfied. *Soft CSPs* permit the constraints to be satisfied to a degree, rather than binary satisfaction. Constraint Optimization Problems (COP) include an objective function. For both CSP and COPs, the resulting model is passed to a solving algorithm. A wide variety of such solvers, complete and incomplete, are available, such as `or-tools` from Google.

Although CP and Mathematical Programming (MP) share a similar model-and-solve paradigm – there is a set of decision variables, an objective function to maximize or minimize and a set of constraints – CP is a more expressive formalism. It can be thought of as a generalisation of mixed integer programming to non-linear and non-arithmetic constraints [30]. Table 1 provides an overview of the differences between CP and MP. Notably for the purpose of this paper, the expressive nature of CP allows for models which are closer to human-level expression of problems [9].

2.2 Closely-Related Approaches

The approach of this paper, elaborated in Sect. 3, has two main characteristics. First, we use simple CP to model the problem. Second, we represent the simulation model automatically in an optimization model by representing it as a NN.

Using CP in modelling enterprises and in business analytics is not a new concept. Several studies couple Business Process Management with CP. These

Table 1. Mathematical programming vs. constraint programming [13].

Mathematical programming	Constraint programming
Typically restricted to linear and quadratic problems	Typically discrete but also continuous problems
Proves optimality with techniques such as a lower-bound proof provided by cuts and linear relaxation	Proves optimality by showing that no better solution than the current one can be found
Algebra as theoretical basis	Logic as theoretical basis
Requires that the model falls in a well-defined mathematical category	Does not make assumptions on the mathematical properties of the solution space
Is specific to a class of problems whose formulation satisfies certain mathematical properties	Has no limitation on the arithmetic constraints that can be set on decision variables

Table 2. Black box optimization vs. empirical model learning [20].

Black box optimization	Empirical model learning
Designed for problems without a complex combinatorial structure (discrete variables and non-trivial constraints)	Tends to provide best results for problems with a complex combinatorial structure
Relies on performing simulation during the search process	Simulation time has no direct impact on the solver performance
Function that describes the system is a black box	No black box assumption, allowing exploitation of its structure during the search process

typically focus on (complex) planning and scheduling problems and require the DM to describe the flow of the system in the CP model [14,31]. In contrast, this paper studies CP model that merely describe the problem to solve and do not require the DM to describe how the system operates. Differently stated, the DM can treat the simulation model as a black-box. As a result, the DM should solely focus on describing what relation between simulation in- and outputs she would like to be satisfied and does not have to consider details of internal processes, as these are modelled by means of a simulation model. We believe this makes it applicable to a wider variety of problems in the sense that the only requirement is access to a simulation model, and easier to understand as the DM can describe its problem in a more direct fashion.

The literature on SO tend to favour evolutionary search to find optimal simulation inputs [18,28,33]. This is convenient when fast evaluation of the simulation model is possible. However, if simulation is expensive – frequently the case with simulators – advanced techniques are necessary to limit the number of simulation calls [20]. Table 2 summarizes the differences between empirical model learning and classical black box optimization.

3 Methodology

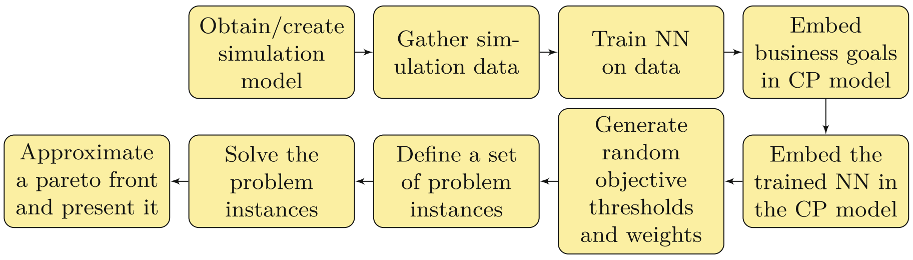
Figure 1 provides an overview of our approach. The simulation model is assumed to represent an enterprise; it can for example be derived from an enterprise model [29]. How to design a qualitative simulation model is not in the scope for this paper: the reader is refered to Kampik and Najjir [16], Laguna and Marklund [18]. This section details the two main components of our approach, namely the Neural Network and the CP model. Then, some specifications about the solving procedure are discussed.

Table 3. Concepts from the meta-model presented by Bock et al. [6] and their CP equivalents. A factual aspect describes something regarded as true, i.e., a constraint. A goal is a metric to improve upon that is expressed by other metrics, i.e., an objective function. Multiple stakeholders can be modelled by soft constraints, which provides foundation to model individual preferences [27]. An action describes something that can be undertaken, hence corresponds with a variable that can be decided upon.

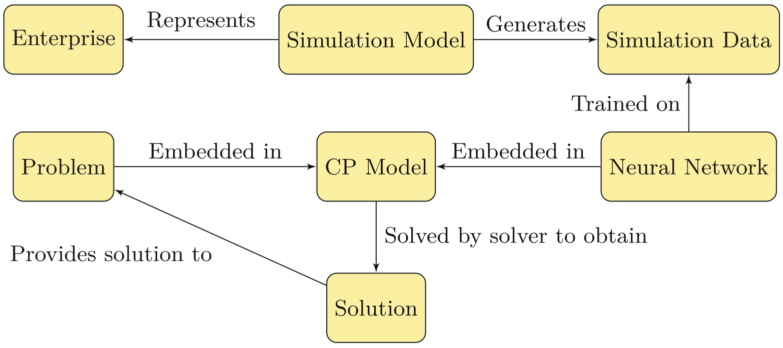
Problem conceptualization meta-model	CP equivalent
Factual aspect	Constraint
Goal	Objective function
Value	Variable
Stakeholder preference	Soft-constraint
Possible action	Decision variable

3.1 The Neural Network

The first main component is a meta-model, derived from a simulation model representing the behaviour of a system. This allows the DM to consult the meta-model instead of the simulation, such that simulation calls do not have to be made during the solving procedure. Deriving such a meta-model poses a trade-off. On one hand it should reflect the simulation model properly. On the other hand it should be convenient enough to allow optimization. This dilemma corresponds with a fundamental computational reality, namely the trade-off between expressiveness and tractability. A model should be detailed enough such that it makes sense – it is expressive – but not too detailed because otherwise computations can not be made feasibly – it lacks tractability [7]. This paper proposes using NNs as meta-model for several reasons:


Obtain/create simulation model
Gather simulation data
Train NN on data
Embed business goals in CP model
Embed the trained NN in the CP model
Generate random objective thresholds and weights
Define a set of problem instances
Solve the problem instances
Approximate a pareto front and present it

(a) Workflow: from simulation model to pareto front.



(b) Overview of the various components and how they interact.

Fig. 1. Approach summarized. A simple NN is trained on simulation data. Next, business goals and the trained NN are embedded in the CP model. By utilizing soft constraints and various weight and threshold parameters, an approximation of the pareto front for the various objectives is formed.

- NNs are able to learn behaviour of opaque or very complex systems, without requiring detailed knowledge of their components and interactions [4,19]. They are capable in dealing with both the non-linearity and uncertainty of the underlying system [28].
- NN embeddings in optimization models have shown good performance in comparison to other combinations of optimization methods with machine learning techniques [20].
- Compared to other machine learning techniques, NN are very capable in identifying what features are interesting and excel at handling high-dimensional input data. The result is that NN require very little engineering by hand and are applicable in many domains [19].

3.2 The Constraint Program

The second main component is the CP model. It intends to represent a problem to be solved. There are three main aspects to be identified in problems, namely

1) an as-is scenario which is considered to be non-optimal, 2) uncertainty about what decision would lead to the preferred situation, and 3) a preferred situation to achieve [6]. We construct the CP model accordingly:

1. The current situation is *modelled* by a simulation model and *represented* by a trained NN. A NN can be embedded into a CP model, which allows expressing constraints over the NN output [4].
2. Uncertainty is also incorporated by the NN. Training an NN creates a predictive model. In other words, the NN is used to tackle the uncertainty involved in the problem by giving insight in the correlation between variables and parameters in the decision model.
3. The preferred situation is incorporated by the objective function. In our approach, the objective function is expressed by soft-weighted constraints. The main idea is that this helps in finding an approximation of the set of dominating solutions, also referred to as the pareto front. This provides good insight into the solution space when dealing with multiple objectives.

It is reasonable to believe most – if not all – problems can be described by elements also found in CP. This is based on the theoretical conceptualization of a problem by a meta-model presented in Bock et al. [6], where the concept of a problem is decomposed. The interconnection between this decomposition and CP is shown in Table 3.

```
% pseudocode for the restaurant case as a CP model

% NN_4layer represents the neural network and maps the correlation between buyingStrategy and spoilage, success
include NN

% parameters of the model
float[] buying_constraints = [c1,...,cn]

% decision variables
var[] buyingStrategy = [b1,...,bn]    % strategy to propose
var spoilage          % spoilage percentage of resources
var success          % success percentage of customer orders

% buying strategy should be positive and is limited by buying_constraints
constraint forall (i in 1..n) (0 <= bi <= ci)

% solve two weighted soft constraints, with some (randomly) assigned weight and threshold
solve (
weighted-soft-constraint(spoilage <= spoilage_threshold, spoilage_weight),
weighted-soft-constraint(success >= success_threshold, success_weight)
)

```

Fig. 2. Pseudocode for the restaurant CP model. It closely matches a MiniZinc implementation.

As noted in Sect. 2, constraints expressed in natural language can be formalised in CP at a higher level compared to other optimization methods. The result is that constraints expressed by the DM describing what solution it aims to seek can be conveniently expressed in the CP model. For example, CP makes it convenient to state some relationship between variables should always be satisfied, such as 'machine A should never produce more than machine B' or 'department C should always have more employees than department D'; so-called *global* *constraints* are particularly useful [30]. Figure 2 shows an example CP model that highlights its readability.

Our approach embeds a NN into a CP model by expressing the value of a node as a function of values of nodes in a previous layer. This is based on the concept of *neuron constraints*, which allow one to encode complex networks using a limited number of basic components [4].

3.3 Soft Constrained Multi-Objective Solving

Most methods on multi-objective decision making in business analytics are *posteriori*, in the sense they obtain preference information – how much each objective is preferred over the others – from the DM *after* computing solutions [33]. These methods are useful when the DM is interested in the scope of actions she can take – particularly useful when there is no single dominating solution – as multiple solutions are provided instead of a single one. In CP, preference information *is* the input, since the CP model is asked what inputs are necessary to satisfy certain objectives. As such, the approach can be made posteriori by evaluating a set of problems covering the variety of possible preferences.

Our approach does not require preference information beforehand as it constructs a type of soft CSP called a *possibilistic CSP*, in which a Weighted Soft Constraint (WSC) is defined for each objective [27]. A WSC has a weight, indicating the importance of it being satisfied, and a threshold. For a maximization objective, the WSC is satisfied if the objective value exceeds the threshold, and for a minimization objective if it does not. Then, random weights and thresholds are generated in order to create a set of problem instances. Next, this set of problems is solved. Their dominating solutions form the output.

3.4 Computational Solving

As just explained, our approach produces a set of soft CP problem instances. These instances are solved and their solutions brought to bear upon the objectives of interest to the DM (Fig. 1). This solving is done by existing solver software.

However, a DM might encounter the solving procedure is slow depending on the complexity of the enterprise and managerial problem being studied, raising interest to speed it up. There are several ways this can be achieved. One way is to reduce the complexity of the problem by using less variables, putting more constraints on the problem to solve or discretizing continuous variables. The trade-off here is that it limits the system in what solutions it is able to present. Another method would be to reduce the complexity of the NN, by reducing the amount of nodes and layers, which has a trade-off against accuracy. Alternatively, the DM could experiment with different search strategies or solvers, and might find one that works particularly well on its problem. An analysis of the various options was out of the scope for this paper. For more details on different solvers and search strategies, we refer to Wallace [30].

4 Experimental Validation

This section assess the approach from Sect. 3 by means of two experiments. Three further experiments, one based on a simulation model derived from a DEMO model, one based on a simulation study performed by other work and one based on an agent-based simulation model, are reported in Andringa [2]. The goal of the experiments is to examine the proposed approach for applicability, generalizability, flexibility and ease of use.

In order to asses consistency, the same simple NN architecture was used for these experiments, as shown in Fig. 4. Training was performed in batches of 64 samples using the AdamW optimizer [21], with a learning rate set at 10^{-5}. The simulations were implemented in Python and NetLogo, the NN in PyTorch [24] and the CP model in MiniZinc [22], supported by MiniBrass [26] to implement soft-constraints. The JaCoP [17] WCSP solver was used. Source code is available under MIT licence at https://doi.org/10.4121/17060642.v1.

4.1 Experiment 1: Restaurant

The first experiment studies the case of a restaurant. It shows how to apply the approach to a simple problem. The simulation was programmed in Python. The NN was trained ±2 h on ±350000 simulation calls based on random inputs.

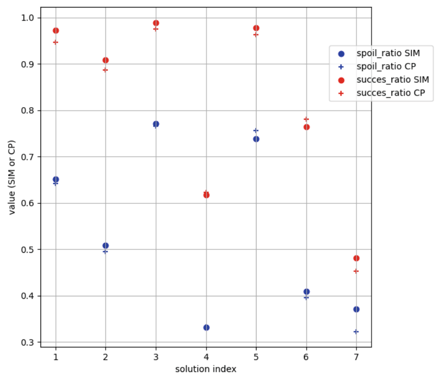
Problem Definition. The restaurant buys ingredients periodically according to some buying strategy and processes these into various dishes. Its buying strategy is represented by an integer per resource that indicates the quantity being bought each period. Buying strategies have limitations, for example due to seasonal ingredients. Resources can spoil if stored for too long. The restaurant has two objectives that characterize a trade-off: it should not buy too many resources in order to minimize spoilage but also should not buy to little in order to maximize the number of successful orders. The restaurant is interested in how its buying strategy affects its spoilage and success ratio.

Results. The corresponding CP model can be found in Fig. 2 and the solving times in Table 4. The results in Fig. 3 show the CP model is able to recognise how various courses of action have different impact on the objective outcomes. It also shows it was able to make accurate predictions.

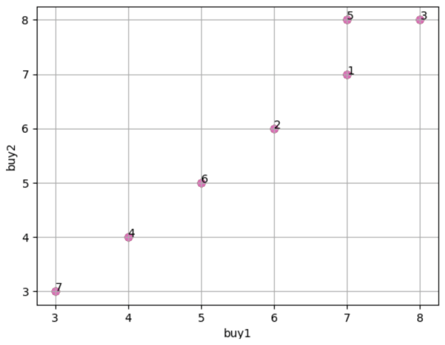
4.2 Experiment 2: Supply Chain

Supply chain models describe how various manufacturing units interact with each other by passing products and materials to each other towards some resulting product. A freely-available NetLogo model, based on a supply chain was consulted for this experiment [11]. The NN was trained on ±5 h and ±14000 random simulation calls. The purpose of the experiment is show the approach is applicable on more complex simulation models. The experiment conducted matches the

Restaurant

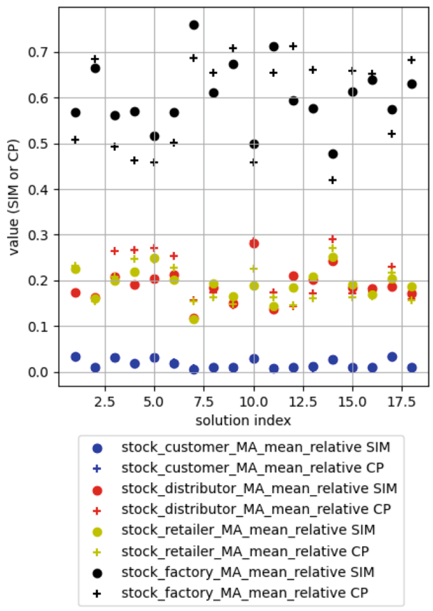


(a) Spoil and succes ratio per proposed solution.

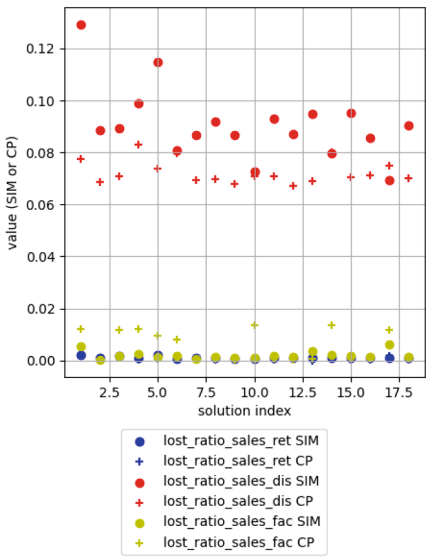


(b) Scatter plot of the proposed actions. Number annotations correspond to the index in Figure a.

Supply Chain

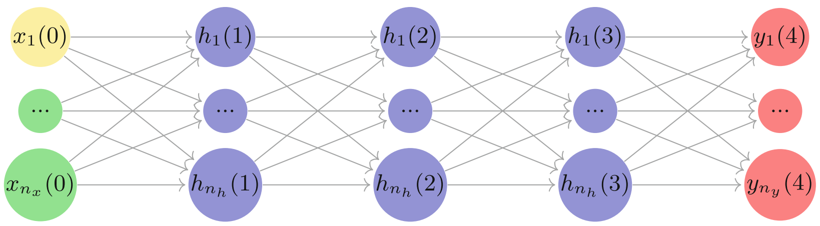


(c) Relative inventory ratio per proposed solution.



(d) Lost sales ratio per proposed solution.

Fig. 3. Experimental results. a, b and c show CP-estimations (+) and simulation estimations (•) for the objective metrics. Accuracy of the CP model can be measured by the difference between the CP and simulation estimations.



(a) General NN architecture used. Three hidden layers of width $n_h = \sqrt{{n_x}^2 + {n_y}^2} + a$, where n_x, n_y are respectively the amount of in and outputs. a is an added parameter to deal with datasets that have low n_x, n_y and was set to 4 for our experiments. Further optimization of a is out of the scope for this paper.

$$h_i(j) = \text{ReLu}\left(w_i(j)\sum_{i=0}^{n_h} w_i(j-1) + b_i(j)\right) \quad y_i(j) = \phi\left(w_i(j)\sum_{i=0}^{n_h} w_i(j-1) + b_i(j)\right)$$

$\phi(x)$	**Condition**
ReLu(x)	$\forall_y y \geq 0$
Tanh(x)	$\forall_y 1 \geq y \geq -1$
Sigmoid(x)	$\forall_y 1 \geq y \geq 0$
Softmax(x)	$\sum(y_i) = 1 \wedge \forall_y 1 \geq y \geq 0$
x	else

(b) Equations for hidden and output neurons. A neuron consists of a weight $w_i(j)$, a bias $b_i(j)$ parameter and an activation function. Neuron values depends on neuron values of the previous layer. The hidden layers used ReLu as activation function. The activation function applied to the output layers depended on the upper and lower bounds of the simulation output according to the table.

Fig. 4. Description of the NN architecture used for the conducted experiments.

famous Beer Game that represents a supply chain with a non-coordinated process [23]. Problems arise due to lack of information sharing, causing a bull-whip effect – an increase of variance of orders placed by each stage when we move from downstream stage to upstream stages – resulting in inefficient inventory management [10].

Problem Definition. The supply chain of a product consists of factory, distributor, retailer and client. They make individual marketing decisions based upon a strategy picked by the DM. Furthermore, the DM decides upon the total number of factories, retailers and distributors. There are two issues observed regarding the current state of the supply chain. First, too many sales are lost. Second, too many products are stored in the factories instead of the distributors and retailers. The number of clients and their demand is given. The DM is

Table 4. Mean and standard deviation runtimes for solving a single problem for both experiments. The number of NN parameters gives an indication about the complexity differences of the CP models.

Experiment	Solving duration (s)	Number of NN parameters
Supply chain	14.23 ± 0.79	1255
Restaurant	0.544 ± 0.025	116

interested in finding the optimal number of factories, retailers and distributors, as well as the optimal inventory policy and costumers strategy, such that little sales get lost and fewer products are stored in factories.

Results. The CP model used for this experiment has a similar structure as the model in Fig. 2, only with more in- and outputs. Solving runtimes are given in Table 4. The results in Fig. 3 show how the CP-estimations are clearly correlated with the simulation values but were sometimes off target.

From the proposed actions, the DM is able to make two main observations about the process. First, less factories with respect to the distributors and retailers resulted in less products being stored at the factories but also in more unsuccessful orders as demand could not be kept up. Second, random inventory and buying strategies performed best for our problem. To investigate this observation further, a variant of this experiment where random strategies were excluded was also performed. The outcome was that less complex strategies – strategies where market participants do not take too many factors in account to adjust their buying behaviour – were proposed. This shows a trend. The more a market participant aims to maximize its own profit by incorporating complex strategies, the more unpredictable its behaviour gets for other participants, making the market prone to the bullwhip effect. This is in line with how literature tends to tackle this problem – by regulation – as that limits market participants to put complex strategies into practice [10].

4.3 Discussion

The experiments indicate that the approach has some desirable properties:

- *Applicability* The approach can be applied to both simple and complex simulation models. The first experiment is considered to be simple, as the simulation model only had a few in- and outputs. The second experiment was performed on a complex agent-based simulation, which is used widely in real-world business problems and is capable to model complex systems properties, indicating the approach can be used for practical application [16].
- *Generizability* The same architecture as described in Fig. 4 was used for both experiments. This indicates the potential in using already existing designs, allowing a DM unfamiliar with deep learning to apply the approach on a

custom problem. This observation supports the claim that deep learning tends to require little extra engineering [19].

- *Flexibility* The solution evaluation process happens within reasonable time (see Table 4). For the supply chain experiment, a single simulation call took around the same time as a single CP evaluation and thus a significant speedup is put into place. Furthermore, solving time can be tweaked accordingly (see Sect. 3.4). As such, a change of problem can be quickly evaluated, making the approach flexible to a change of problem.
- *Ease of use* The experiments showed that a simple, easy-to-understand CP model (see Fig. 2) is sufficient to allow solving of it when following the presented approach. We believe this makes it doable for a DM with little technical knowledge to construct a CP model for its problem.

5 Conclusion and Future Work

This paper considered enterprise simulation–optimization and addressed how to find optimal simulation inputs more effectively. The proposed approach adopts a meta-model in the form of a NN to capture simulation behaviour, and embeds the NN automatically into a soft CP model.

The demonstrated proof of concept has multiple advantages: 1) DMs with little technical expertise are expected to be more comfortable with designing a CP model that a MP model, due to CP being more expressive. 2) DMs need not have expertise about deep learning to put this approach into practice, since already existing NN architectures can be used. 3) The approach is applicable to already-existing simulation models. 4) Complex properties are kept intact during the decision making process, as both NN and simulation models are able to model them. 5) A change of problem can be quickly evaluated since a meta-model is used to represent the simulation model. 6) It is not required for the DM to input preference information over the importance of objectives since a set of non-dominating solutions is presented instead of a single solution. 7) There is room for many extensions on this approach due to the general concept of using a machine learning based meta-model.

The paper demonstrated the potential of the approach. Although Fig. 3 shows good accuracy for the Restaurant experiment, for the Supply Chain experiment this can still be improved upon. As such, there is room for development. First, the simple NN architecture can be revisited. This paper used a generic and somewhat arbitrary architecture for generality and applicability reasons. What architecture suits best for this application is left for future work. An interesting direction for a better architecture is transfer learning [34]. Second, more advanced techniques regarding data extraction from simulations can be investigated for example by having what parameters to send to the simulation model depend upon what data is gathered so far. Third, in this paper we assumed the simulation model to be a black box. Not doing so allows the use of formal model checking and verification methods to create more accurate meta-models as deeper properties of the simulation model can be taken into account.

Furthermore, there is potential in improving the solving procedure. The thresholds and weights of the soft constraints are randomly generated. More effective strategies can be put into place such that the generated problems more evenly spread the objective space. This paper did only minor experiments with trying various solvers and search strategies. Solving time can possible be reduced by experimenting with these, as mentioned in Sect. 3.4.

Building on the approach of this paper, it can be interesting to consider constraint acquisition [5]. Here, the CP model is allowed to make simulation calls to obtain more knowledge when it considers it possesses too less. This gives up a form of flexibility, in that solving becomes dependent on the simulation runtime. However, it is expected to be more accurate than our approach as it can consult the simulation model in case of doubt. Constraint acquisition is different from black box optimization, and allows exploitation of the model structure (see Table 2). An interesting follow-up study would be a performance comparison between the proposed methodology, constraint acquisition and other multi-criteria optimization methods such as evolutionary search.

Acknowledgements. We thank the EEWC'21 reviewers for their suggestions. This research was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under grant number 952215.

References

1. Abushark, Y., Thangarajah, J., Miller, T., Winikoff, M., Harland, J.: Requirements specification in the prometheus methodology via activity diagrams. In: Proceedings of the International Conference on Autonomous Agents & Multiagent Systems (AAMAS), pp. 1247–1248. ACM (2016)
2. Andringa, S.: Applying Constraint Programming to Enterprise Modelling. Master's thesis, Delft University of Technology (2021)
3. Atkinson, C., Kuhne, T.: Model-driven development: a metamodeling foundation. IEEE Softw. **20**(5), 36–41 (2003)
4. Bartolini, A., Lombardi, M., Milano, M., Benini, L.: Neuron constraints to model complex real-world problems. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 115–129. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23786-7_11
5. Bessiere, C., Koriche, F., Lazaar, N., O'Sullivan, B.: Constraint acquisition. Artif. Intell. **244**, 315–342 (2017)
6. Bock, A., Kudryavtsev, D., Kubelskiy, M.: Towards more expressive problem structuring: A theoretical conceptualization of 'problem' in the context of enterprise modeling. In: Proceedings of the 20th Conference on Business Informatics (CBI), IEEE (2018)
7. Brachman, R.J., Levesque, H.J.: The tradeoff between expressiveness and tractability. In: Knowledge Representation and Reasoning, pp. 327–348. Elsevier (2004)
8. Brouard, C., de Givry, S., Schiex, T.: Pushing data into CP models using graphical model learning and solving. In: Simonis, H. (ed.) CP 2020. LNCS, vol. 12333, pp. 811–827. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58475-7_47
9. Buscemi, M.G., Montanari, U.: A survey of constraint-based programming paradigms. Comput. Sci. Rev. **2**(3), 137–141 (2008)

10. Chinna Pamulety, T., Madhusudanan Pillai, V.: Performance analysis of supply chains under customer demand information sharing using role play game. Int. J. Ind. Eng. Comput. **3**(3), 337–346 (2012)
11. Gil, A.: Netlogo model: Artificial supply chain (2012), École Polytechnique Montréal
12. Grossmann, I.E.: Challenges in the application of mathematical programming in the enterprise-wide optimization of process industries. Theor. Found. Chem. Eng. **48**(5), 555–573 (2014). https://doi.org/10.1134/S0040579514050182
13. IBM: Mathematical programming versus constraint programming, DOcplex v2.22 documentation (2021)
14. Jimenez-Ramirez, A., Barba, I., Del Valle, C., Weber, B.: Generating multi-objective optimized configurable business process models. In: 6th International Conference on Research Challenges in Information Science (RCIS), IEEE (2012)
15. Jin, R., Chen, W., Simpson, T.: Comparative studies of metamodelling techniques under multiple modelling criteria. Struct. Multi. Optim. **23**(1), 1–13 (2001)
16. Kampik, T., Najjar, A.: Integrating multi-agent simulations into enterprise application landscapes. In: De La Prieta, F., et al. (eds.) PAAMS 2019. CCIS, vol. 1047, pp. 100–111. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24299-2_9
17. Kuchcinski, K., Szymanek, R.: JaCoP - java constraint programming solver. In: Abstract from CP Solvers: Modeling, Applications, Integration, and Standardization, co-located with the 19th International Conference on Principles and Practice of Constraint Programming. Uppsala, Sweden (2013)
18. Laguna, M., Marklund, J.: Optimizing business process performance. In: Business Process Modeling, Simulation and Design, 2nd Edn, pp. 439–472. Chapman and Hall CRC (2013)
19. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
20. Lombardi, M., Milano, M., Bartolini, A.: Empirical decision model learning. Artif. Intell. **244**, 343–367 (2017)
21. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: Proceedings of 7th International Conference on Learning Representations (ICLR). OpenReview.net (2019)
22. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: towards a standard CP modelling language. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 529–543. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74970-7_38
23. O'donnell, T., Maguire, L., McIvor, R., Humphreys, P.: Minimizing the bullwhip effect in a supply chain using genetic algorithms. Int. J. Prod. Res. **44**(8), 1523–1543 (2006)
24. Paszke, A.e.: PyTorch: an imperative style, high-performance deep learning library. In: Proceedings of the 33th Conference on Neural Information Processing Systems (NeurIPS), vol. 32, pp. 8024–8035 (2019)
25. Rossi, F.: Constraint (Logic) programming: a survey on research and applications. In: Apt, K.R., Monfroy, E., Kakas, A.C., Rossi, F. (eds.) WC 1999. LNCS (LNAI), vol. 1865, pp. 40–74. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44654-0_3
26. Schiendorfer, A., Knapp, A., Anders, G., Reif, W.: MiniBrass: soft constraints for MiniZinc. Constraints **23**(4), 403–450 (2018). https://doi.org/10.1007/s10601-018-9289-2

27. Schiex, T.: Possibilistic constraint satisfaction problems or "how to handle soft constraints?". In: Uncertainty in Artificial Intelligence, pp. 268–275. Elsevier (1992)
28. Teerasoponpong, S., Sopadang, A.: A simulation-optimization approach for adaptive manufacturing capacity planning in small and medium-sized enterprises. Exp. Syst. Appl. **168**, 114451 (2021)
29. Vernadat, F.: Enterprise modelling: research review and outlook. Comput. Ind. **122**, 103265 (2020)
30. Wallace, M.: Building Decision Support Systems. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-41732-1
31. Wiśniewski, P., Kluza, K., Jemioło, P., Ligęza, A., Suchenia, A.: Business process recomposition as a way to redesign workflows effectively. In: Proceedings of the 16th Conference on Computer Science and Intelligence Systems, IEEE (2021)
32. Yaghin, R., Sarlak, P., Ghareaghaji, A.: Robust master planning of a socially responsible supply chain under fuzzy-stochastic uncertainty (a case study of clothing industry). Eng. Appl. Artif. Intell. **94**, 103715 (2020)
33. Yalcin, A., Kilic, H., Delen, D.: The use of multi-criteria decision-making methods in business analytics: a comprehensive literature review. Technol. Forecast. Soc. Change **174**, 121193 (2022)
34. Zhuang, F., et al.: A comprehensive survey on transfer learning. Proc. IEEE **109**(1), 43–76 (2021)