# From Data to Model Structure: A Generative Algorithm to develop System Dynamics Models

Guido A. Veldhuis[1*], Freek P.J. van Wermeskerken[2], Olaf H. Visker[3], Patrick Steinmann[4], Sebastiaan Deuten[5], Rob P.M. van Waas[6]

[1*,2,3,4,5,6] TNO - The Netherlands Organization for Applied Scientific Research, Oude Waalsdorperweg 63, 2597 AK The Hague, The Netherlands
[1] Institute for Management Research (IMR) Radboud University, Heyendaalseweg 141, 6525 AJ Nijmegen, The Netherlands
[4] Delft University of Technology, Faculty of Technology, Policy, and Management, Jaffalaan 5, Delft, 2628 BX, The Netherlands

[1*]Guido.veldhuis@tno.nl, [2]Freek.vanwermeskerken@tno.nl, [3]Olaf.Visker@tno.nl, [4]P.Steinmann@tudelft.nl, [5]Sebastiaan.deuten@tno.nl, [6]Rob.vanwaas@tno.nl

## Abstract

Developing simulation models, including System Dynamics (SD) models, remains largely a manual process that involves specifying equations to capture the dynamic behaviour of the system under study. While artificial intelligence (AI) applications are increasingly used for model calibration and analysis, their role in model development remains unexplored. We introduce a generative algorithm designed to derive system structures from observed behaviours. The algorithm constructs net flow equations based on time series data of stocks in the system. We describe the algorithm's operations and discuss its potential as a model development support tool within the SD modelling process. To evaluate its efficacy, we apply it to three test cases of increasing complexity: a simple goal-seeking system represented by a cooling teacup, an oscillatory predator-prey model, and a complex model involving interactions between workforce, production, and inventory management. The algorithm successfully identifies structures in simpler cases but achieves only partial success in the complex case. This exploration shows both the potential and limitations of using generative algorithms to support SD model development. We concludes with a discussion of the challenges encountered and reflects on the future of modeller-algorithm collaboration, setting the stage for further research on 'human-in-the-loop' AI support tools.

## Keywords

- System Dynamics
- Artificial intelligence
- Machine learning
- Model building methodology
- Operational Research

## Introduction

Artificial intelligence (AI) is increasingly becoming a part of activities traditionally seen as crafts, such as programming and arts. The application of AI in Operational Research (OR) has been a topic of interest for decades and has recently received renewed attention, although the number of publications remains below the surge seen in other fields (Ormerond, 2020; de Bock et al. 2023). AI applications in OR have traditionally focused on tasks such as design, scheduling, process planning, and control (Kobbacy et al., 2007). However, AI has yet to be applied in defining the operations of simulation models, such as System Dynamics (SD) models, which remains largely a craft. An algorithm capable of suggesting sets of equations that replicate observed behaviour could significantly improve the speed and validity with which models are developed.

SD is a modelling approach used in OR to model, understand, and analyse the behavior of complex systems over time, particularly as a result of feedback loops, time delays, and accumulations within systems. This approach is used to understand problem behaviour and identify leverage points for interventions, aiding in policy analysis and decision-making (Forrester, 1961). It operates in two primary modes: a descriptive mode that aligns with a soft OR perspective similar to problem-structuring methods, and a predictive/prescriptive mode more similar to hard OR, addressing problems through forecasting and optimization (Kunc, 2017). The SD modelling process is a thoroughly described and debated process in which the model builder captures a dynamic hypothesis in a set of equations (Sterman, 2000; Martinez-Moyano & Richardson, 2013). Currently, algorithms facilitate only certain steps of the modelling process, such as calibration, structural analysis, and output analysis (Rahmandad, Oliva, & Osgood, 2015; Bengio et al., 2021; Schoenenberger et al., 2021; Steinmann et al., 2020; Yücel & Barlas, 2011). The question arises: Can AI aid in the development of the model structure itself, specifically the equations? Recent advancements in natural language models and foundation models have led to the development of tools that extract causal information from text to support the model development process (Kolides et al., 2023; Veldhuis et al. 2024, Hosseinichimeh et al., 2024). However, at present, defining the structure of a

simulation model remains solely a task for humans, with no computer-aided tools available to assist in delineating the causal structure of the model or in specifying its equations.

The use of algorithms to derive system structure from numerical data has received some attention from SD scholars. Early attempts have relied on using neural networks to replicate Causal Loop or Stock and Flow models (Chen et al., 2011; Drobek et al., 2015; Abdelbari & Shafi, 2017; 2018). Results have been mixed. Zhao (2019) employed an interactive workflow, reminiscent of human modelling processes, to generate and validate new hypotheses of structure, utilising both numerical and non-numerical information, and demonstrating the effectiveness of this method for two single stock models. Abdelbari & Shafi (2019) combined hybrid genetic programming and simulated annealing to formulate and calibrate differential equations, starting from either a user defined adjacency matrix or a synthetically generated and evolving matrix. Their approach is partially successful in identifying the structure of two relatively simple SIR models. Further afield, researchers have explored the idea of model induction in other domains including agent-based modelling (Epstein et al., 2023, Rand, 2019), game theory (DeDeo et al., 2010), and hydrology (Remmers et al., 2020). For instance, Vu et al. (2019, 2023) use a genetic programming approach to evolve a set of human defined equations that specify the behaviour of agents. Brunton et al. (2016) introduced the Sparse Identification of Nonlinear Dynamical Systems (SINDy) method, which uses sparse regression and machine learning to discover governing equations for nonlinear dynamical systems from data without assuming any prior knowledge.

As Ormerod (2021) has argued that the time for the OR community to get involved with the development of AI is now. Sterman, (2018, p. 40) highlights harnessing data to *'develop, test, communicate and implement rigorous, reliable and effective insights in the dynamics of complex systems'* as the *'next frontier' for system dynamics*. In this paper we venture into this frontier by presenting a prototype algorithm capable of inferring system structure from observed behaviour over time. The system structure is expressed as a SD model: a combination of stock (integrals), flow equations and information feedback. The only input for the algorithm is a set of time series data of the stocks in a system, referred to as the system behaviour. Subsequently, the algorithm derives the stock and net flow equation one stock at a time and calibrates the associated parameters.

We will begin by describing the operations of the algorithm, which include tree generation, node optimisation, error evaluation, and the tree search strategy. Following this, we will discuss the algorithm's purpose and its role within the System Dynamics (SD) modelling process. The performance of the algorithm will then be evaluated through the reconstruction of three existing SD models. In the discussion, we will examine our results and their implications, compare them with

existing research, and explore future research directions to enhance the integration of generative algorithms with System Dynamics modelling.

# Method: a generative algorithm to construct stock and flow models

## Introduction to the approach

The goal of the algorithm is to find a set of differential equations that, when simulated from a known initial value, fit the available data of the stocks. To simplify and scope this approach, we establish the differential equation for each stock separately. After the differential equations for each stock are found, or iteratively during the process, a human modeler can separate in- and outflows from the net flow equation, match variables in formulas and then combine the results into one model.

This approach relies on generating differential equations and progressively adding complexity during the search while calibrating each solution to the available data. It begins by fitting simple, short, polynomial differential equations to the data. Following this, a promising intermediate solution is selected. This solution is then iteratively increased in complexity and fitted again.

The inputs to the approach are the time series data of all the stocks, assuming that the system's behaviour is known but its structure is unknown. An overview of the algorithm is provided in Figure 1, which includes:

- *Stock data: this is the input data,*
- *Tree generation: a framework (a 'Tree' network) where parametrised differential equations of increasing complexity are generated,*
- *Tree search strategy: a strategy to determine which parametrised differential equation should be solved or increased in complexity,*
- *Solution: the search should lead to a differential equation that fits the stock data,*
- *Node optimisation: finding the best-fitting parameters for a parametrised differential equation,*
- *Integration of differential: integration of a differential equation to derive its outcome, and*
- *Error evaluation: evaluation of the fit between the stock data and the outcome.*

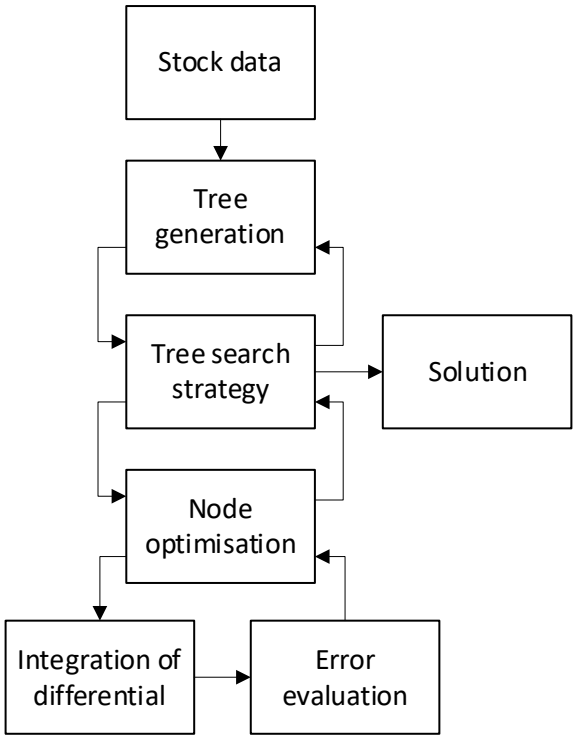


*Figure 1: An overview of the different parts of the algorithm and their interactions.*

Using a tree-based approach to generate different models (nodes) offers some notable advantages, particularly in terms of transparency and controllability, distinguishing it from other methods like deep learning or genetic algorithms, which are often more black-box in nature. Tree structures offer a transparent and structured representation of the model development process, facilitating the tracing of hypothesis origins and understanding the logic behind each structural decision. This clarity greatly improves the user's ability to interpret, validate, and refine the models.

However, the systematic exploration and evaluation of models (nodes) place substantial demands on computational resources, with computational complexity growing exponentially. Consequently, while tree-based models offer the advantages of transparency and control, their computational requirements necessitate careful resource allocation and optimisation strategies to ensure efficiency and practical usability in real-world applications.

## Tree generation

The algorithm employs a tree structure to make clear and concise decisions in its search for the best-fitting differential equation. A tree consists of nodes, each containing a parametrised (by a number of real-valued constants) differential equation. Therefore, the terms 'node' and 'parametrised differential equation' refer to the same concept. When referring to a differential equation, we specifically mean a non-parametrised equation: it contains no variables, only stocks and values.

The goal is to iteratively increase the complexity of the parametrized differential equation in order to find a good fit with the available data. For example, the parametrized differential equation (a node) for a stock (named *money*) could be $\frac{d\ money}{d\ t} = C * money$. This node expresses the well-known exponential growth or decay system. Essentially, in our approach the nodes are functions composed of distinct combinations of mathematical operators and relationships.

A significant part of our approach overlaps with the field of symbolic regression, which aims to find the simplest mathematical expression that best fits the data. In this field, expression trees are employed to manipulate and administer mathematical expressions (equations). In our prototype approach, we utilise an expression tree exclusively comprising unary operators. The general symbolic regression problem, featuring binary operators, is acknowledged as NP-hard, indicating that searching for solutions within the domain spanned by the binary expression tree is time-consuming (Virgolin & Pissis, 2022). The strategy employed utilises a concatenation of unary operators, the span of which encompasses most differential equations used in system dynamics. Naturally, this implies that systems where the differential equation is not readily represented by a unary expression tree pose a greater challenge to solve.

We define an operator as a small mathematical transformation of the parametrized equation into another (more complex) parametrized equation. The root node of the tree consists of the (parametrized) equation 'zero'. We can then (iteratively) keep adding operators to (well-fitting) parametrized equations to increase the complexity. The **operators**[1] we used for the algorithm are (C stands for a real-valued variable, X stands for a specific stock, F stands for a parametrized equation):

- F → F + C
- F → F + C * X
- F → F + C / X
- F → F * C * X
- F → F * C / X
- F → F + F * C * X
- F → C * non_linear_transformation(F)[2]

By starting at the root node, the algorithm generates the children nodes of the root node by applying all operators for all stocks, resulting in 14 children nodes when there are two stocks. This is illustrated in the top tree in Figure 2. As it iteratively generates new nodes, the number of nodes grows exponentially with the depth of the tree. To mitigate this growth, the algorithm checks whether the parametrised differential equation of the newly generated node is equivalent to that of

[1] Note that more operators could be included, such as: F → F + F * C / X, or MAX/MIN.
[2] We added a non-linear lookup function to experiment with fitting systems depending on lookup function, see the third use-case in the Results section.

another node already in the tree. Two parametrised differential equations are considered equal if there is a one-to-one correspondence (algebraically equivalent) between them. For example, the parametrised equations 0 + C simplifies to C and similarly, 0 + C * X * C / X simplifies to C * C. This is illustrated in the bottom tree in Figure 2. Both processes demonstrate how seemingly different expressions can represent the same relationship, thereby being considered equivalent. In order to determine whether two parametrized differential equations are equal, the Python library SymPy is used to transform string representations of the formulas into a standard form (Meurer et al. 2017). To abstract from specific numerical values, constants within these formulas are replaced by the number one. Subsequently, all coefficients in a formula are normalized or removed, facilitating a comparison based on the structural composition of the equations rather than their numerical specifics.

When a node is not added to the tree (pruned), it proves efficient to store this information. Therefore, we establish parent-child relationships between the parent node and the more complex, readily available node. This process introduces cycles in the parent-child relations, transforming the structure from a tree to a directed network. Despite this, we continue to refer to the directed network as a 'tree' because complex equations are generally positioned further from the root node than simpler ones.

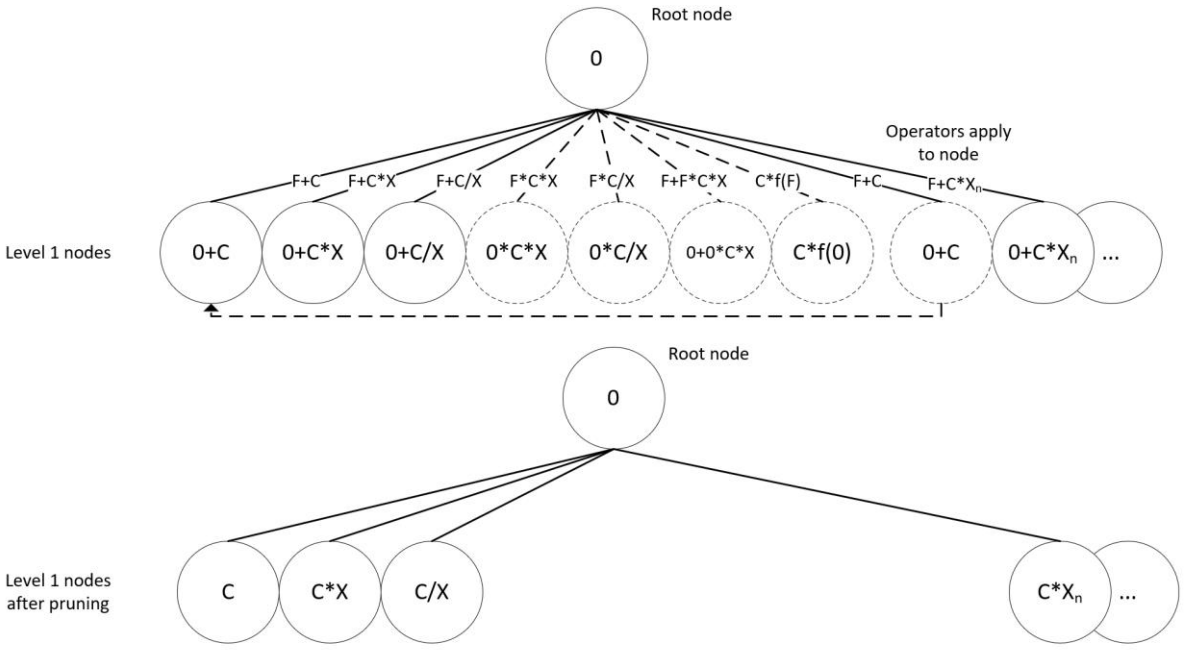


*Figure 2: illustration of tree generation and pruning. The upper tree shows operators being applied for stock X to $X_n$. The dotted lines indicate equivalent formulas (0+C = 0+C and 0 = 0*C*X = 0*C/X etc.). The bottom tree is the resulting pruned tree.*

## Node optimisation

The explanation of the tree search strategy is best understood last, hence we proceed with detailing the node optimisation phase. During this phase, the algorithm seeks to identify the parameters in the parametrised differential equation that most accurately fit the data. This entails determining values for all constants in the equation, integrating the differential, and evaluating the error to assess how well the integrated result matches the data.

In node optimisation, we employ the Nelder-Mead algorithm from the Python SciPy library (Virtanen et al., 2020). This algorithm offers the advantage of not requiring derivatives, which can be complex and time-consuming to calculate, especially since the error value is dependent on time series data of stock values. The Nelder-Mead algorithm recognized for achieving significant error reduction with relatively few evaluations. Selecting this algorithm is pragmatic for our prototype, though other optimisation algorithms might enhance performance or efficiency.

When the degrees of freedom in the parametrised differential equation are fewer than the number of constants in the formula, the efficiency of the optimisation process significantly decreases. For instance, the node with the formula 0 + C * X * C * X = C*C * X*X has one degree of freedom (the value of ‘C*C’) but two constants, resulting in an infinite number of optimal solutions. The algorithm identifies such cases using the SymPy library (Meurer et al. 2017) and removes a constant from the formula, a step taken during the generation of new nodes, which reduces the dimension of the problem and improves the optimisation process.

The Nelder-Mead algorithm is employed with default stopping criteria and does not impose bounds on the constants, acknowledging that the problem is unbounded. The initial values of all constants significantly affect the likelihood of finding the global minimum, especially in complex problems. Generally, large absolute values result in the integration producing large absolute stock time series, whereas values near zero lead to nearly constant stock time series. However, choosing fixed initial values for the constants can cause the Nelder-Mead algorithm to terminate prematurely, such as when it reaches the bottom of a Rosenbrock-like optimisation valley. This presents a challenge, as we assume no prior knowledge of appropriate constant values, and there might be orders of magnitude differences between constant values, even within the same model. We address this challenge by determining appropriate initial values for optimisation with the Nelder-Mead algorithm in three consecutive steps.

First, the (two-sided) derivative of the stock is calculated. This pointwise derivative is used to estimate the order and sign of the initial values. Then, for each time step available in the stock data, , the algorithm optimises the constants to yield the desired derivative of the stock. Taking the mean of

the resulting values provides a preliminary estimate of the initial parameter value. Second, using the results of the first step as a starting point, we perform a similar optimisation, focusing on the derivative of the stock data. However, in this step, the constants are fixed over time. Despite this additional step, the resulting initial values can sometimes be too far removed from suitable starting values for the Nelder-Mead algorithm to converge on a global optimum. In the third step, using the results from the second step, we apply two iterations of the computationally intensive Powell optimisation strategy on the entire node optimisation problem. This algorithm is known for its ability to take larger steps within the optimisation domain. The result is then used as the initial values for the Nelder-Mead algorithm, which ultimately determines the final parameter values. This step-wise approach remains more efficient than running computationally expensive optimisation algorithms, such as Powell optimisation, on the node optimisation problem alone. Efficiency is crucial, as many nodes require optimisation during the search.

## Integration of differential

With the constants proposed during node optimisation, the algorithm solves the resulting initial value problem (with initial values specified in the data) using the solve_ivp algorithm from the SciPy library (Virtanen et al., 2020). This process requires the values of stocks, which serve as inputs to the equation, for all time steps. When stock values are unavailable for certain time steps, the data is linearly interpolated to fill these gaps. This interpolation introduces a minor error, which has been found to be negligible provided the data adequately represents the phenomenon in question.

The algorithm employs the explicit 3rd order Runge-Kutta method. Other methods resulted in more integration failures or decreased efficiency. This step, occurring during the iterative process, involves integrating generated differential equations with parameter values that may significantly deviate from their true values, being either excessively small or large. Consequently, integration can often be slow, as the (absolute) value of the derivative might become large.

To enhance integration efficiency additional stop criteria are defined. The algorithm identifies when the absolute value of the solution under evaluation exceeds the original stocks significantly or when integration excessively concentrates iterations within a small domain (relative to the time domain and interval of the stock data). This situation often arises when the derivative's absolute value approaches infinity within the bounded domain, complicating the determination of error values crucial for guiding the optimisation. As the optimisation process necessitates an error value, continuing integration with ever-increasing or constant error values fails to effectively direct the optimisation. Thus, to mitigate this issue, the derivative's returned values are maintained at the level detected at the moment of identification.

## Error evaluation

To assess the fit between the stock data and the integration, selecting an efficient loss function is crucial, especially given the frequent calculations involved. Additionally, the loss function should promote a broad domain of attraction and yield smooth error values. Based on these criteria we have selected the Symmetric Mean Absolute Percentage Error (SMAPE). It offers rapid evaluation, a smooth error landscape, and a wider and clearer domain of attraction compared to other loss functions, thanks to its symmetrical attribute. An added advantage of SMAPE is that it measures percentage error, making the absolute error value a more intuitive indicator of fit that is less influenced by the magnitude of the stock data. Other loss functions, such as the Dynamic Time Warping (DTW) loss function, proved too time-consuming, or had a smaller domain of attraction, like the Mean Squared Error (MSE) and Mean Absolute Percentage Error (MAPE). These limitations made them less suitable for our purposes compared to SMAPE, which balances computational efficiency and the breadth of the domain of attraction effectively.

## Tree search strategy

The tree search strategy in principle has two options to search for an optimal solution:

- Expand a node in the tree, adding more (non-optimised) nodes with more complex dynamics to the tree.
- Optimise a node, which returns a specific time series of the stock and an error value of the fit with the actual data.

The search always starts at the root node. We decided to optimise all nodes that were added to the tree whenever a node is expanded. This leaves us with one choice: which node to expand next. The algorithm is terminated, yielding a satisfactory fit, when a node's optimisation returns an exceedingly small error value. Figure 3 illustrates the search algorithm for a two stock problem.

It's important to note that each operator introduces a single degree of freedom (the constant C) and for each operator there is a value for C that simplifies the equation back to its preceding parametrised form. This property implies that if the algorithm effectively identifies the (global) optimal values for the constants that minimise the error, then the error for any given node should be less than or equal to that of its parent node. This only holds when the node, in comparison to the parent, adds complexity.

In the tree search strategy we encounter the exploitation versus exploration dilemma. Using exploitation (smallest error) for the decision on which node to expand results in the algorithm to do a depth first search, since errors are not increasing. However, this strategy often fails, because the solution node is often not on the depth-first search path. In general, the path from the root node of the tree to the solution node is non-convex, meaning the child node leading to the solution node is

not necessarily the node with minimal error. A strategy that effectively navigates this dilemma requires backtracking.

An effective approach to explore the solution space is to pursue diverse behaviour, meaning a solution is significantly different from the previous result. Therefore, the algorithm tracks the changes in error values. We found that the strategy to expand the node with the largest error improvement works well, as it ensures that significantly different behaviour is tested. Also, it still correlates strongly (initially) with a good absolute error value, since increasing complexity (of an equation) implies a non-increasing error, as explained above.

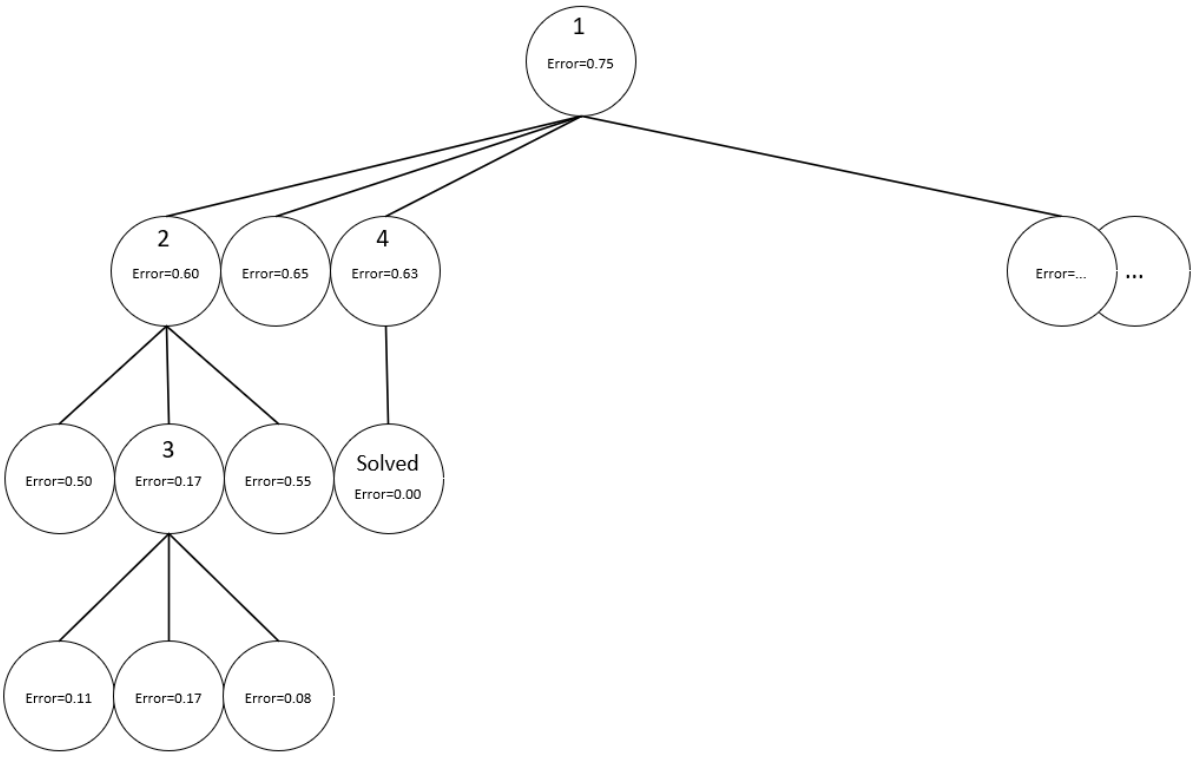


*Figure 3: illustration of tree search strategy. The numbers indicate the order in which nodes are expanded. The search proceeds down the tree until node 3 after which the algorithm backtracks to node 4. Since node 4 has a greater improvement in error values compared to the children of node 3.*

## A place for generative algorithms in the System Dynamics modelling process

Ormerod (2021) has discussed several ways in which AI can be positioned in OR. The algorithm presented in this paper is designed as a support tool for a human modeller. Following this, we describe the role and added value of the generative algorithm in the modelling process. To provide context, a brief description of the SD modelling process is included.

The SD modelling process starts with identifying a dynamic complex problem. A dynamic hypothesis is then formulated about the causal feedback structure underlying the problem. This is followed by the formalisation of the hypothesis into a model to test it. In the context of this paper, only stock and

flow simulation models are considered. To determine the model's soundness and usefulness, a series of validation tests are performed. The model can then be utilised to identify leverage points for interventions and develop, test and compare different policy options, supporting their implementation (see Figure 4). For more information on the process, the reader is referred to Forrester (1961) and Sterman (2000).

The interaction between the generative algorithm workflow and the System Dynamics (SD) modelling process begins at step (ii), the formulation of a dynamic hypothesis, and can continue into step (iii), the formulation of a stock and flow model (Figure 4). During the formulation of a dynamic hypothesis, the algorithm assists in identifying plausible model structures, including potentially multiple alternative structures. In the model formulation phase, the algorithm's output can be transferred to an SD modelling environment and incorporated within an SD model. In the examples presented in the next section, we rely entirely on the algorithm to generate the model structure. In real-world applications, the output of the algorithm can be adjusted by a modeller or integrated with model structures already developed by the modeller.

The user must specify at least one reference mode (time series data) for a stock (step 1 in the right part of Figure 4). The algorithm can then search for an equation and parameters that fit the provided data (step 2). The user thus needs a preliminary understanding of the system to select reference modes for inclusion in the search. Additionally, the user may choose to include numerous other reference modes for stock variables or constants (which are essentially stock values that remain unchanged during the model's time horizon).

The user will then need to evaluate the output of the algorithm (step 3). This involves reviewing the error scores, comparing the behaviour over time of the solution to the reference mode data, and ensuring that the generated equation and parameters are conceptually explainable. Once the results are deemed useful, the equation and parameters can be transferred to the SD modelling environment, often utilising one of the several commercially available software packages (step 4). However, the prototype algorithm has no contextual knowledge of the system or understanding of the model purpose beyond an adequate fit with the reference mode data. Therefore, the user is required to define the flows, auxiliary variables, and their units. Since the algorithm currently outputs one net-flow equation, this may also involve splitting the equation into multiple separate equations for conceptually distinct inflows and outflows of the stock. An additional integration step might be necessary in cases where the algorithm is used to find equations for multiple stocks (step 5). Through the iterative nature of both processes, the algorithm can assist the user in formulating a valid dynamic hypothesis (step ii) and stock and flow model (step iii).

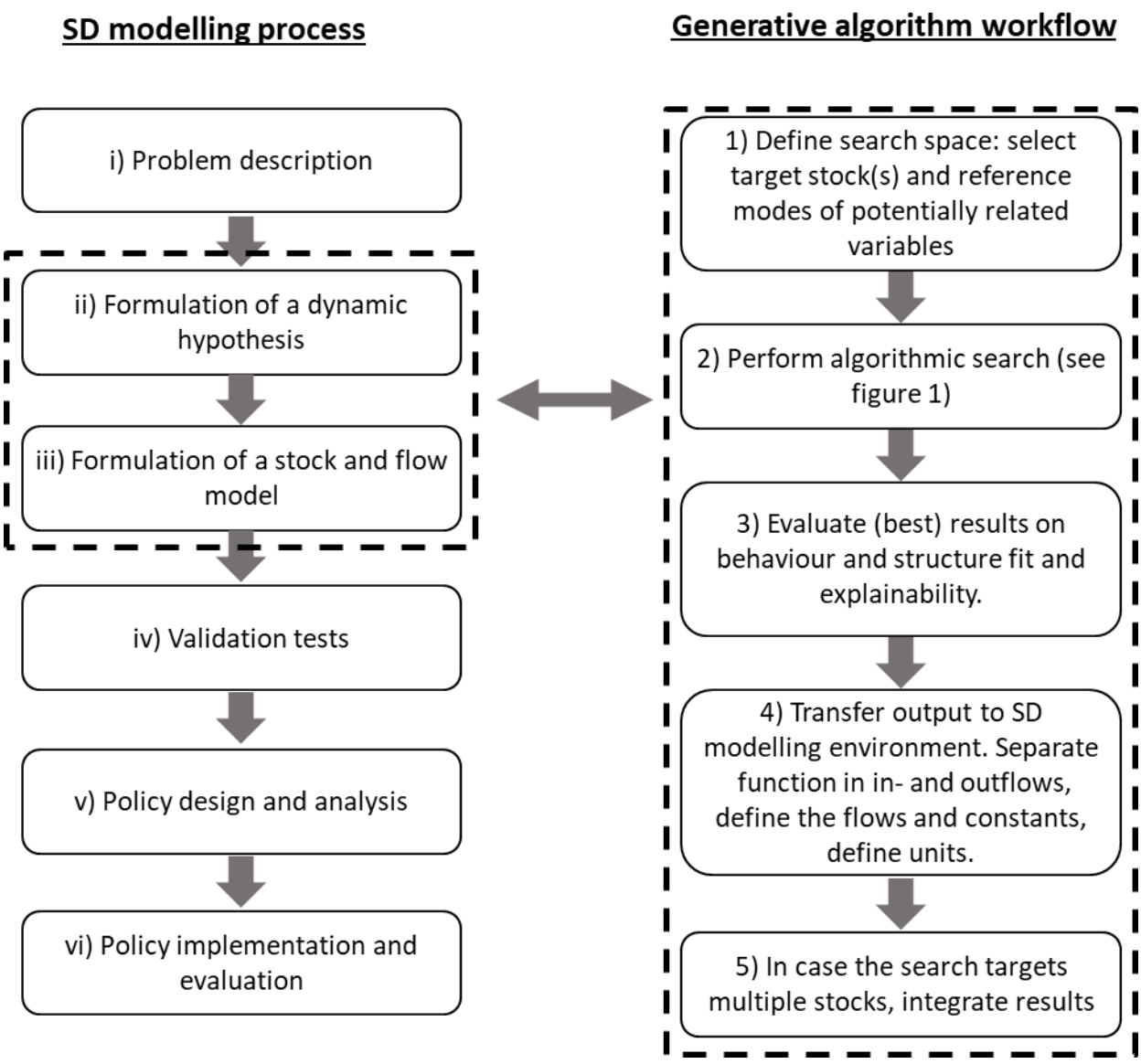


*Figure 4: Positioning of the use of a generative algorithm (right) within the System Dynamics (SD) modelling process (left). The algorithm is used to generate system structure and is therefore utilized in both the dynamic hypothesis formulation step and the stock and flow model development step. Although both processes are depicted as linear, in practice, many iterations may occur within and between steps.*

## Materials: three test cases

We will evaluate the algorithm's capability to replicate the system structure (stock and flow equations) using time series data of the target behaviour and other stocks in the model, if present. Our test set consists of existing models which serve as a ground truth against which the algorithm's output can be compared.

To assess the algorithm's performance across various conditions, we have chosen three test cases of increasing complexity, approximating complexity levels used in System Dynamics (SD) education (Schaffernicht et al., 2016). The models in the three test cases exhibit increasing complexity due to a rising number of stocks and feedback loops.

- Case 1 explores the dynamics of a cooling tea cup, illustrating simple goal-seeking behaviour through a local balancing feedback loop. This system demonstrates how temperature decreases over time towards a stable equilibrium. The model is based on Newtons Law of Cooling $\frac{dT}{dt} = k(T - T_{ambient})$ and was used in earlier work on model structure generation by Zhao (2019).
- Case 2 describes the interaction between a population of deers and wolves and is an example of the classic predator-prey model, capable of producing oscillatory behaviour (Volterra, 1928). Its behaviour is driven by one major balancing feedback loop and several local feedback loops, both goal seeking and reinforcing.
- Case 3 is a more elaborate model that describes the dynamics between workforce, production and inventory management, adapted from Sterman (2000). With the specific initial values and constants selected for this test case, the model displays dampening oscillatory behaviour. It features five stocks and multiple major and local feedback loops.

Certain functions used in some SD models have not yet been integrated in the algorithm, such as higher order delay functions or logical functions (if, min, max). For this reason the test set does not include cases that require such operations. Complex functions such as lookups are especially interesting in our context because they often represent entire submodels are -systems that are not explicitly modelled, and can introduce substantial behavioural variance into the system depending on their specification (Eker et al., 2014). For testing purposes, we have included one non-linear relation in the three search which is used in the third case mode.

## Results

In this section we present the results of applying the generative algorithm on the three defined cases: teacup, predator-prey, and inventory workforce. For the teacup and predator-prey the algorithm successfully generates a structure identical to the original model. However, for the inventory workforce model, the algorithm achieves partial success, which will be discussed in more detail.

For the first case, the teacup case, the algorithm successfully identifies the structure (equations) of the system together with accurate constants. Table 1 shows the original equation, the found equation and the SMAPE error value.

*Table 1: original (O) and generated (F) differential equations and SMAPE for stocks in case one: teacup*

| Stock | O/F | Equation | SMAPE |
|---|---|---|---|
| Teacup Temperature ($S_1$) | O | $\frac{dT}{dt} = \frac{-(S_1 - 22)}{10} = 2.2 - 0.1 * S_1$ | NA |

| | | | |
|---|---|---|---|
| | $F_1$ | $\frac{dT}{dt} = 2.19 - 0.100 * S_1$ | ~0.001 |

For the second case, the predator-prey case, the algorithm also successfully identifies the structure (equations) together with accurate constants (Table 2).

*Table 2: original (O) and generated (F) differential equations and SMAPE for stocks in case two: predator-prey*

| Stock | O/F | Equation | SMAPE |
|---|---|---|---|
| Deer ($S_2$) | O | $\frac{dT}{dt} = 0.8 * S_2 - 0.04 * S_3$ | NA |
| | $F_2$ | $\frac{dT}{dt} = -0.040 * S_2 * S_3 + 0.804 * S_3$ | ~0.002 |
| Wolves ($S_3$) | O | $\frac{dT}{dt} = 0.02 * S_2 * S_3 - 1.6 * S_3$ | NA |
| | $F_3$ | $\frac{dT}{dt} = 0.02 * S_2 * S_3 - 1.606 * S_3$ | ~0.003 |

In the third case, the inventory-workforce model, the algorithm struggles to generate an appropriate model structure (equations) and constants for some aspects of the system. Figure 5 show the structure of the more complex model with five different stocks.

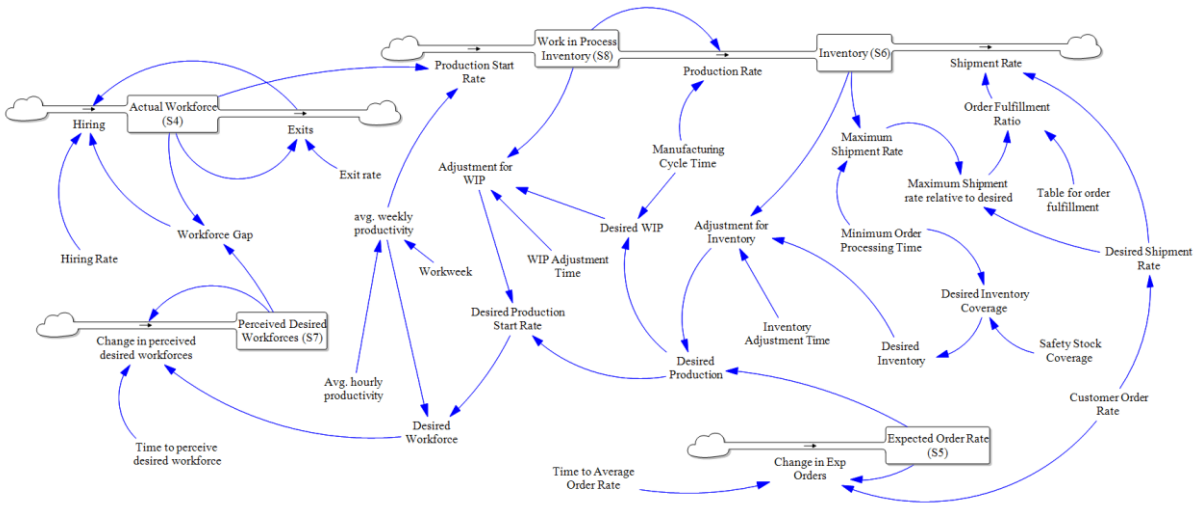


*Figure 5: inventory-workforce model*

Table 3 shows the original equation, the generated equation and the SMAPE values for the best solution generated. These correspond to the model shown in Figure 5. The equations have been algebraically simplified to a minimal amount of operations or variables for the purpose of comparison. Although the SMAPE values are relatively low for all variables, indicating a good fit to the data, the correspondence in system structure is limited for variables: $S_4$, $S_7$ and $S_8$). Different reasons for the varying success of the algorithm exist which are discussed below.

*Table 3 original (O) and generated (F) differential equations and SMAPE for stocks in case three: inventory-workforce*

| Stock | O/F | Equation | SMAPE |
|---|---|---|---|
| | O | $\frac{dT}{dt} = (0.05 * S_7) - (0.05 * S_4)$ | NA |

| | | | |
|---|---|---|---|
| Actual workforce ($S_4$) | $F_{4a}$ | $\frac{dT}{dt} = (0.0450 * S_7) - (0.000483 * S_8) - \left(\frac{2.47 * 10^5}{S_6}\right)$ | ~0.006 |
| | $F_{4b}$ | $\frac{dT}{dt} = (0.0500 * S_7) - (0.0500 * S_4)$ | ~0.001 |
| Expected Order Rate ($S_5$) | O | $\frac{dT}{dt} = (1.25 * 10^3) - (1.25 * 10^{-1} * S_5)$ | NA |
| | $F_5$ | $\frac{dT}{dt} = \left(\frac{2.43 * 10^7}{S_6}\right) - \left(2.43 * 10^3 * \frac{S_5}{S_6}\right)$ | ~0.001 |
| Inventory ($S_6$) | O | $\frac{dT}{dt} = (0.125 * S_8) - (Lookup[S_6])$ | NA |
| | $F_6$ | $\frac{dT}{dt} = (0.0650 * S_6) + (1.10 * S_5) - (0.0001.64 * S_6 * S_7) - \left(\frac{7.09 * 10^6}{S_4}\right)$ | ~0.008 |
| Perceived Desired Workforce ($S_7$) | O | $\frac{dT}{dt} = (0.000778 * S_5) - (0.00486 * S_6) - (0.00417 * S_8) - (0.25 * S_7)$ | NA |
| | $F_7$ | $\frac{dT}{dt} = -(0.00215 * S_5) + \left(5.09 * 10^3 \frac{S_5}{S_8}\right) - (4.29 * 10^{-7} * S_7 * S_8) + (0.0338 * S_7) - \left(\frac{3.69 * 10^7}{S_8}\right) + 15.6$ | ~0.023 |
| Work in process inventory ($S_8$) | O | $\frac{dT}{dt} = (10 * S_4) - (0.125 * S_8)$ | NA |
| | $F_{8a}$ | $\frac{dT}{dt} = -(\frac{9.52 * 10^6}{S_4}) + (\frac{7.63 * 10^8}{S_8})$ | ~0.005 |
| | $F_{8b}$ | $\frac{dT}{dt} = (9.97 * S_4) - (0.125 * S_8)$ | ~0.001 |

For the stock *Actual workforce ($S_4$)* two generated solutions are shown. $F_{4a}$ is the solution where the SMAPE value triggers the algorithm to terminate. Disregarding the termination condition allows the algorithm to successfully identify the structure and constants, as show in $F_{4b}$. The algorithm resolves the $F_{4a}$ node before reaching the $F_{4b}$ node in the tree optimising a solution with a good fit using stocks *Perceived Desired Workforce ($S_7$), Inventory ($S_6$)* & *Work in process inventory ($S_8$)* instead of *Perceived Desired Workforce ($S_7$)* & *Actual workforce ($S_4$)*. This example illustrates that a low error value is not a guarantee for finding an appropriate structural fit, as in some cases low error values can be achieved based on structural erroneous solutions. A closer inspection by a human modeller can reveal such errors. Figure 6 compares the original stock values and the estimated values. Notice that the erroneous solution $F_{4a}$ shows a different mode of behaviour in the initial time units of the simulation, unlike the correct solution, $F_{4b}$, which does not show this divergence.

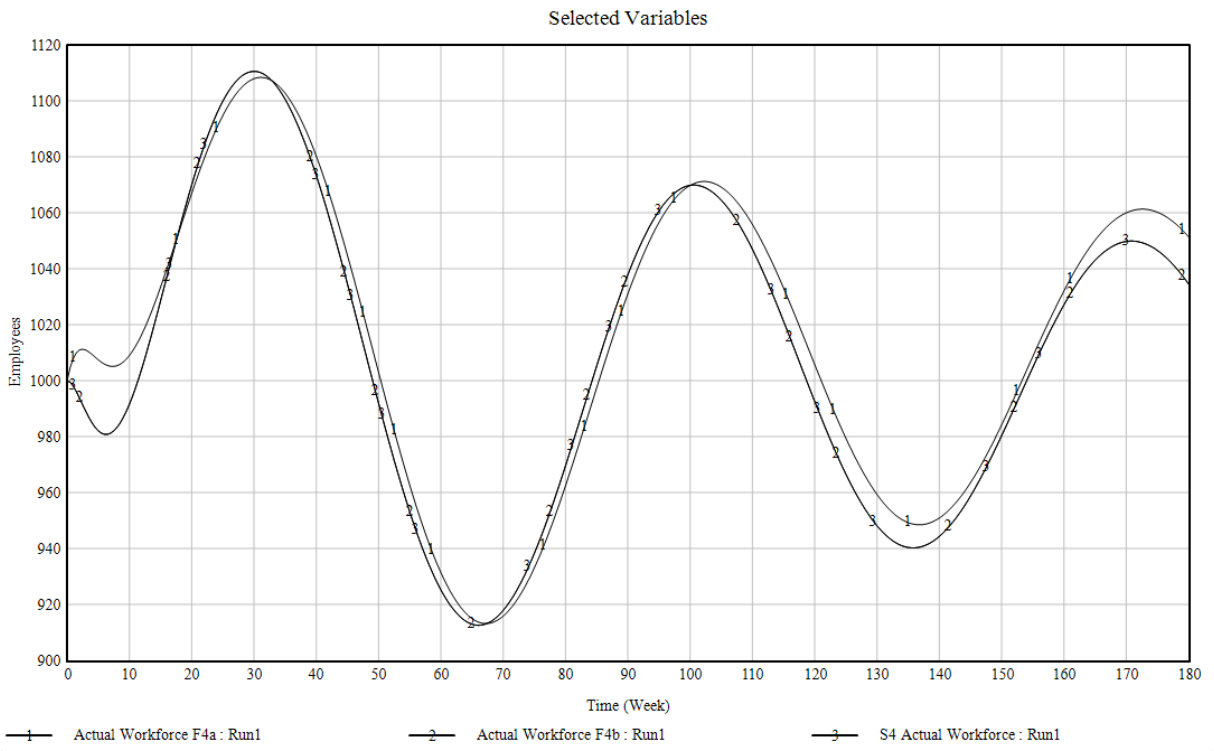


*Figure 6: graph of original values of stock Actual Workforce and the estimate of solution $F_{4a}$*

For the stock *Expected Order Rate ($S_5$)* the algorithm finds a solution with a very small error value but partly fails to replicate the target model structure. The algorithm does generate the node containing the correct model structure but fails in successfully optimising that node, thus failing to identify the correct parameters to reach a low error value. The optimisation algorithm embedded within the algorithm has difficulty as a result of the constants $1250$ and $0.125$ being on different orders of magnitude. Consequently, the tree search algorithm proceeds deeper into the tree until it finds a node giving nearly identical behaviour to the target behaviour, but erroneously adding a stock to the equation in the process (see Figure 7). It uses the stocks *Expected Order Rate ($S_5$)* & *Inventory ($S_6$)* instead of only relying on *Expected Order Rate ($S_5$)*.

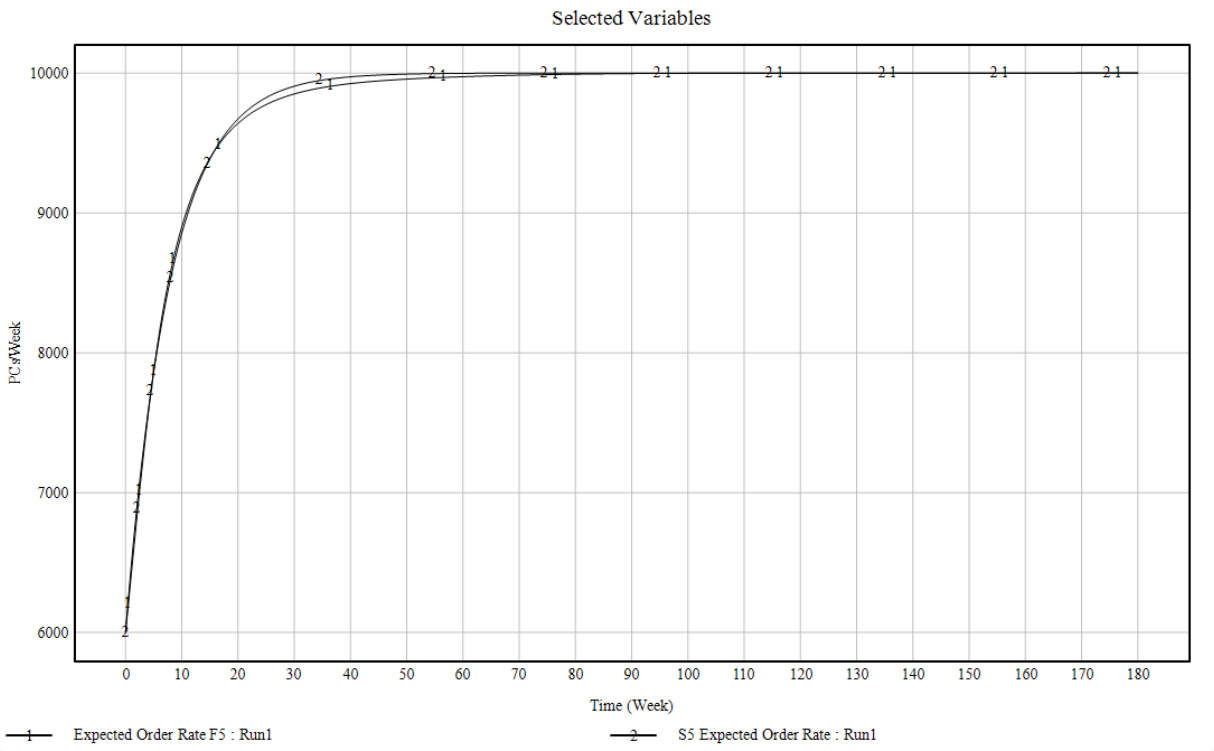


*Figure 7: graph of original values of stock Expected Order Rate and the estimate of solution $F_5$*

For the stock *Inventory ($S_6$)* the algorithm finds a solution with a low SMAPE error value, however not the original solution (see Figure 8). In terms of structure the relation to *Inventory ($S_6$)* is identified, but *Work in process inventory ($S_8$)* is excluded while *Expected Order Rate ($S_5$)* is erroneously included. The latter error may have resulted from the challenging conditions set for the algorithm. Since no constants were input, only stocks, the similarity between the *Customer Order Rate* and *Expected Order Rate ($S_5$)* likely contributed to this deviation.

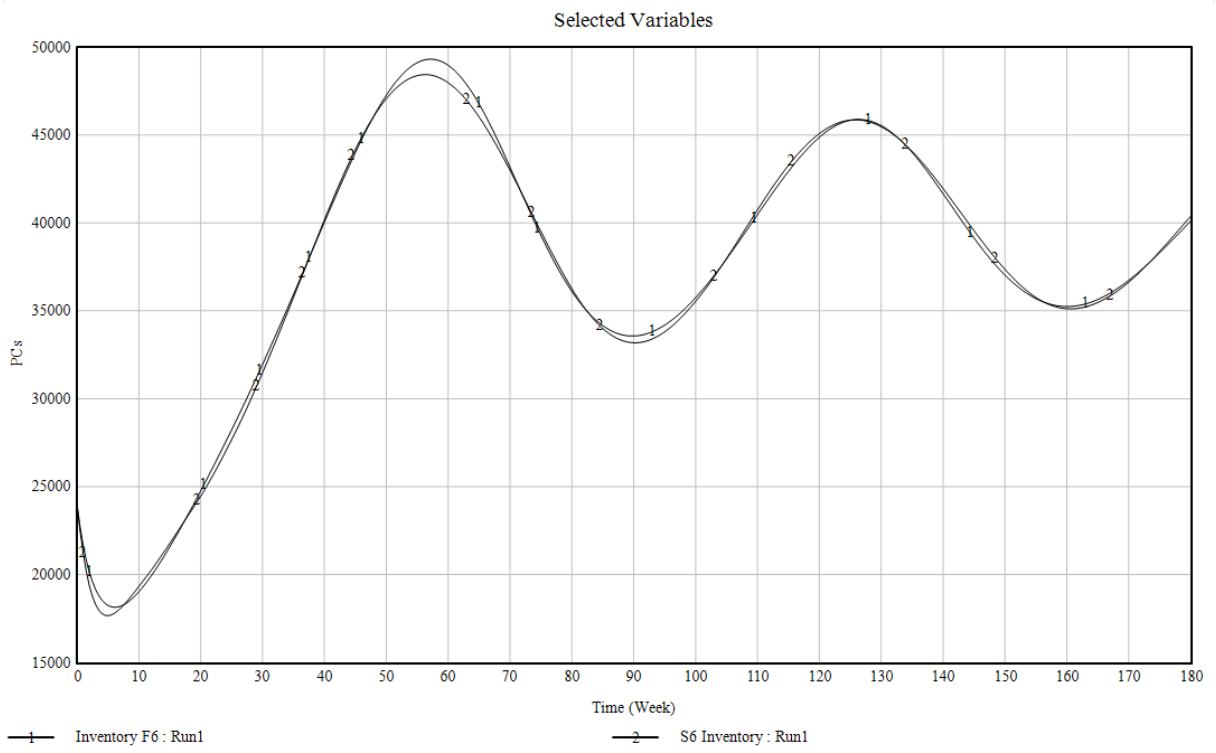


*Figure 8: graph of original values of stock Inventory and the estimate of solution $F_6$*

For the stock *Perceived Desired Workforce ($S_7$)*, the algorithm generates the correct node but fails to optimise the solution node, reaching a local optimum at $\frac{dT}{dt} = (1.98 * S_5) - (0.193 * S_6) - (0.0103 * S_8) - (11.0 * S_7)$ with a SMAPE value of 0.035 (see Figure 9). This value is not low enough to trigger termination and would therefore not be brought to the attention of the analyst. At a lower level in the tree, it identifies different formula ($F_7$) with a SMAPE value of 0.023. The relation with *Expected Order Rate ($S_5$), Perceived Desired Workforce ($S_7$) & Work in process inventory ($S_8$)* are included, but the dependency on the stock *Inventory ($S_6$)* is omitted. A possible explanation is that the effect of Inventory on Perceived Desired Workforce, calculated through the desired production (start rate), is relatively small and unstable compared to the large and stable impact of the Expected Order Rate. Thus, a strong fit can be more easily achieved by the algorithm when ignoring Inventory.

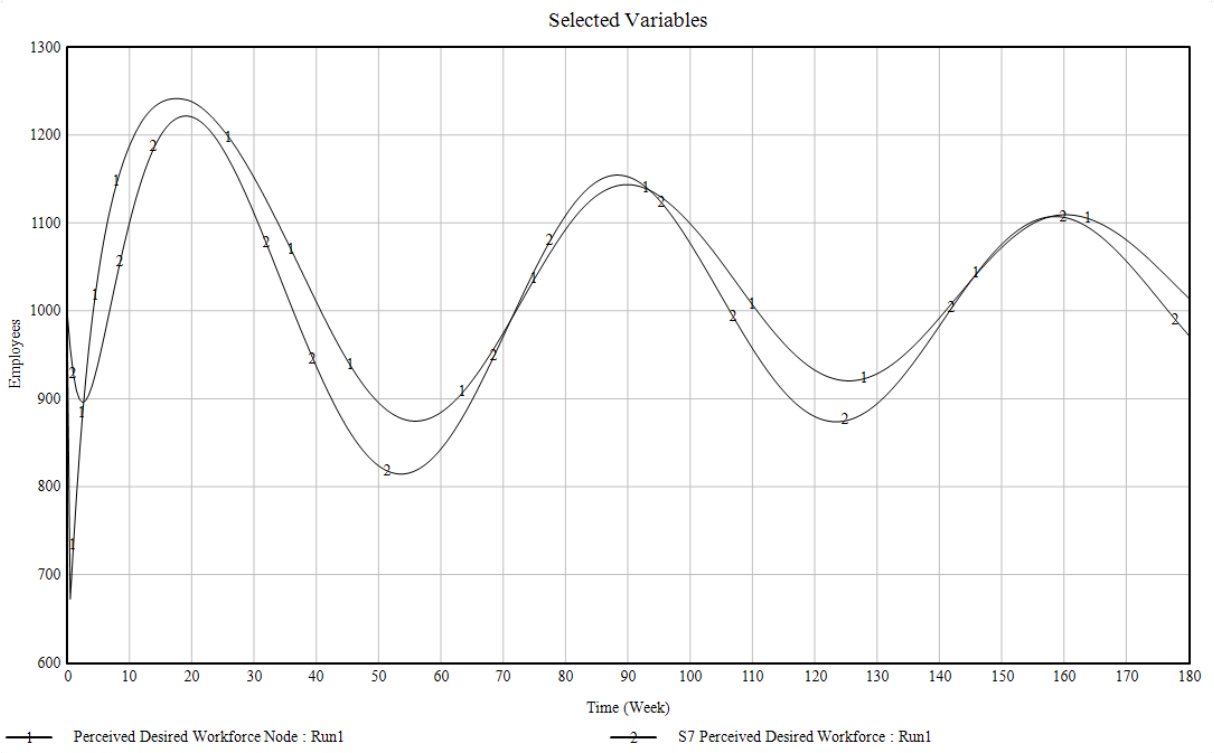


*Figure 9: graph of original values of stock* Perceived Desired Workforce ($S_7$) *and the result of solution* $F_{node}$

For the stock *Work in process inventory ($S_8$)* the algorithm terminates at a node different from the solution node due to SMAPE value lower than the termination condition, similar to *Actual workforce ($S_4$).* In this equation($F_{8a}$) the corresponding dependencies are identified, however, using a different equation than the original. If the algorithm is not terminated at this node, it continues searching and does successfully identify the correct solution ($F_{8b}$). Both $F_{8a}$ and $F_{8b}$ have similar behaviour, which also closely resembles the behaviour of the original equation, as can be seen in Figure 10.

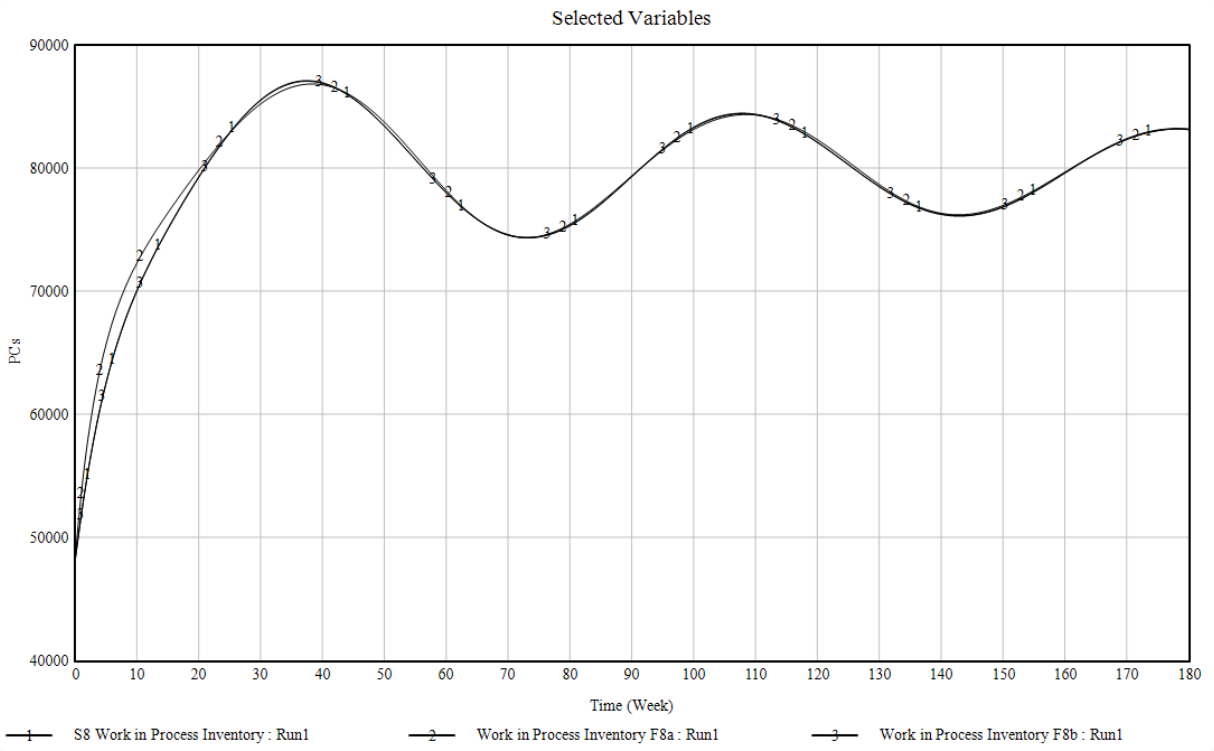


*Figure 10: graph of original values of stock Work in process inventory ($S_8$) and the result of solution $F_{8a}$ and $F_{8b}$*

Figure 11 shows the dependencies between stocks and flows which were found (green arrows), missed (red dotted arrows) and erroneously added (purple dashed arrows). Note that the algorithm did not find that the outflow of *Work in process inventory ($S_8$)* equals the inflow for *Inventory ($S_6$)*, as both equations are generated independently of each other.

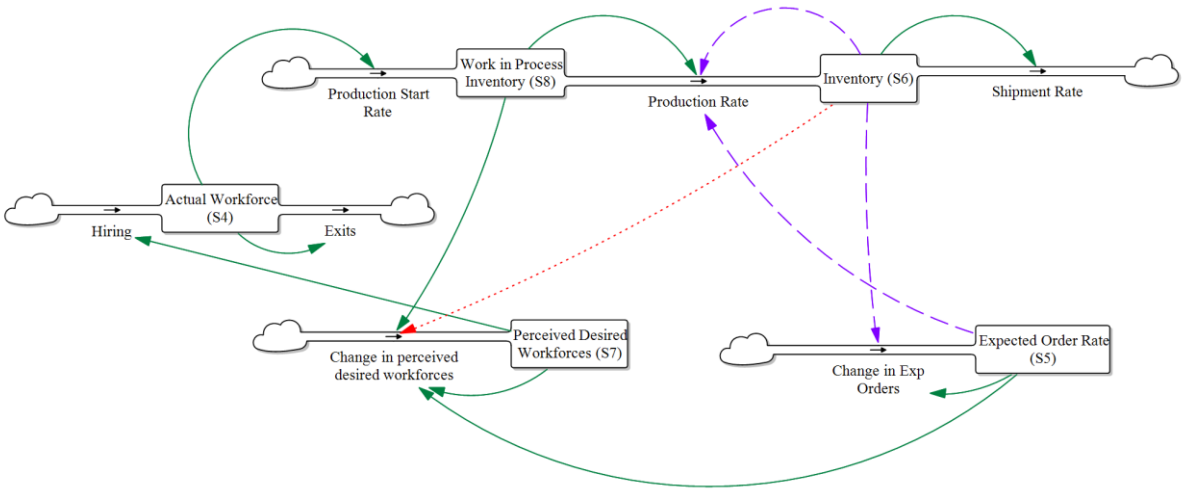


*Figure 11: The inventory-workforce model as generated by the algorithm. A green arrow indicates identified dependencies, a red dotted arrow missed dependencies and a purple dashed arrow indicates dependencies erroneously added by the algorithm.*

The generative algorithm replicated the original models for the teacup and predator-prey cases. However, the algorithm faced challenges with the more complex inventory-workforce model. While

it achieved low SMAPE values for all variables, it struggled to accurately replicate the system structure for all stocks. It missed dependencies and added erroneous relations. The main difficulties arose to in dealing with constants on different orders of magnitude and the algorithm's sensitivity to termination conditions. These findings highlight the need for further refinement of the algorithm to better handle complex models and ensure structural accuracy alongside low error values.

## Discussion

We will discuss noteworthy observations and implications of our results, compare them with the work of other researchers on similar topics, and explore potential future research directions building on these insights.

We applied a generative algorithm to infer the stock-flow structure of three different case studies from the System Dynamics canon using time-series data from stock variables as the only source of information. The overall strategy is to start proposing simple SD systems and to increase complexity in steps. By focusing on improvements of the (data) fit, the algorithm is able to backtrack to simpler systems or increase the complexity even further on (promising) already complex systems. For the simple Teacup model and slightly more complex Predator-Prey model, the algorithm identified the exact system structure and parameters. However, the more complex inventory-workforce model proved to be more challenging. While the algorithm identified several dependencies between stocks, it was unable to generate the correct net-flow equations for several of the stocks in the model. These results illustrate the current capabilities of the algorithm and highlight its limitations, which we will discuss further below.

The convergence of the field of artificial intelligence and SD has thus far been limited (Armenia et al. 2024). Recently, progress was made in the area of Natural Language Processing to support model development by identifying causal text and using Large Language Models to support the modelling process (Akhavan & Jalali, 2023; Hosseinichimeh et al., 2024; Veldhuis et al. 2024). Our work represents additional progress in the quest for using AI in the SD modelling process, by directly inferring system structure from time-series data. Several other attempts to infer system structure from data have relied on the use of neural networks (Chen et al., 2011; Drobek et al., 2015; Abdelbari & Shafi, 2017; 2018). Neural networks can be difficult, if not impossible, to translate into stock and flow structure. This is problematic as results will be difficult to review by a modeler, integrate in a larger model and used to transfer insights and recommendations to an audience. Transparency of the model structure, whether made by a human or an algorithm, is key in OR and SD modelling (de Bock et al., 2023). To achieve this, the output of the approach presented here consists of a set of a net flow equation which can be integrated into a standard SD workflow as described. The output of the

presented algorithm is similar to that of the approach by Abdelbari & Shafi (2019), who tested their genetic programming approach to generate two SEIR models. Similar to our results with the more complex inventory-workforce model, their findings on the simpler SEIR models were mixed, successfully identifying some equations while also uncovering spurious causal relations and omitting existing ones. Remmers et al. (2020, p. 9) observed that for hydrological models it may be "challenging and in most cases impossible to infer model structure from model output". We observe that, at least for the small and medium-sized stock-flow models, progress has been made, notably with regards to the correct generation of a two stock predator-prey model and the mixed results on the far more complex inventory-workforce model.

In the third case study, the algorithm made numerous errors, including suggesting spurious causal effects that defy logic. Our results, as well as those of others, indicate that model structures generated by algorithms require careful validation by a modeller. Vu et al. (2023) similarly noticed a conflict between theory and identified model structure when applying their own model inference algorithm to data of alcohol consumption behaviour patterns. Interestingly, Greig et al. (2019) discussed how finding structures "unlikely to be designed by a human modeler" might actually be an added value of generative tools, as they challenge preconceived notions of system operation. This tension between expected and surprising structure, which generative algorithms can reveal, deserves future research attention.

Nonetheless, scrutiny of the results is required and significant improvements are still needed. Several errors in our current results stemmed from failing to successfully optimise the parameters of the correct solution node, causing the algorithm to continue its search. This issue partly arose from our desire to include a highly efficient optimisation approach, given the large number of nodes in the tree that require optimisation. Improving both the optimisation method and error measurement could significantly improve the performance of the algorithm. Many alternative approaches are available, some of which are already used in combination with generative algorithms (see Abdelbari & Shafi, 2019). In addition, the current algorithm does not consider units and unit consistency, which could also serve as markers for the quality of the solution. Exploring methods to search simultaneously or iteratively for sets of equations, rather than individual stocks as we did, could further improve results. Lastly, a more intelligent search strategy could incorporate general SD modelling knowledge, such as recognizing and prioritising simple archetypal system structures commonly used in SD (see Hines, 2005). The aforementioned areas for improvement are of a technical nature, we will now turn to a second important path forward: collaboration between the modeller and algorithm.

Generative algorithms require supervision and interpretation by experienced SD modellers. Our current work demonstrates several reasons for this. In the inventory-workforce model, the algorithm generated incorrect equations that showed somewhat plausible behaviour with moderate error values. Blindly using these equations could lead to incorrect assumptions about the system structure. Furthermore, the output of the current algorithm consists of net flow equations for each stock. Disaggregating the net flow equations in relevant inflows, outflows and auxiliary variables is common in SD as it aids the communication of the model structure. The current approach could be improved by splicing the equations based on combinations of parameters, stocks and operators. Additionally, separate net flow equations must be integrated into a model, possibly merging them with existing model structures. However, deciding what structure to include into a model and at wat level of aggregation requires an understanding of the model's purpose and context. The current algorithm has no such understanding. But even for future algorithms such understanding might be problematic to acquire as the purpose and context of a model might not be fully articulated and evolve during the modelling process (Veldhuis et al. 2024). Generative algorithms can create better understanding of a system and its behaviour not by replacing the human modeller, but by fostering a more structured and efficient dialogue between models and data, which authors including Box et al. (1978) and Homer (1996) have highlighted as the key for effective systems analysis.

Rather than considering the current algorithm, or indeed any generative algorithm, as an autonomous tool, viewing it as an instrument to be used by a modeller invites discussion on the implementation of a "human in the loop" approach and its benefits. A primary benefit of this approach is the ability to infuse human domain expertise, along with an understanding of the model's purpose and context, into the generative process. This can be accomplished by pre-specifying likely and unlikely relationships, valid value ranges for constants and other goals and constraints. For example, stock values may have physical or logical limitations, known dependencies, or dependencies that can be excluded. This type of information can be gathered through common SD practises, such as consulting with problem stakeholders. Adding information restricts the search space of the algorithm, thereby improving its efficiency and validity, which is crucial for overcoming the challenge of scaling this and similar approaches to larger systems (Brunton et al., 2016). Furthermore, problems within dynamic complex systems are prone to local optima which results in non-convex search spaces which are difficult and time-consuming for an algorithm to escape. Human understanding could guide the algorithm through the search space by prioritising certain performance criteria or suggesting directions to explore or exclude. As such, the use of generative algorithms in modelling becomes not a replacement for manual modelling, "but rather part of a

reflexive and iterative process of scientific inquiry to better understand a phenomenon" (Vu et al. 2023 p. 20).

The search space for inferring a causal model from data is large and expands rapidly as more variables are introduced. If the potential operations of the relations are unknown and remain unrestricted, the search space essentially becomes infinite (Remmers et al. 2020; Vu et al. 2023). Until now, both we and other researchers have primarily focused on simple textbook cases. As efforts shift towards more realistic data, it will inevitably lead to a scenario where multiple, if not many, imperfect solutions emerge, each representing various levels of scope, aggregation and complexity. Determining the 'best' model could itself be problematic if the system's underlying dynamics are poorly understood (e.g., Auping, Pruyt & Kwakkel, 2014) or significant uncertainties are present (Kwakkel & Pruyt, 2015). We contend that in such cases, generative algorithms can be particularly useful, not by aiming to find a single model but by facilitating the development of a variety of models, thereby supporting an ensemble approach to modelling. (Bankes, 1993). Here, too, the "human in the loop" process proposed earlier is beneficial, a generative algorithm identifies a number of plausible structures, and a human analyst then selects one or more of these candidate solutions for further analysis and use in an ensemble. This would also externalize the trade-off between empirical fit and theoretical interpretability identified by Vu et al. (2019) and quantified by Abdelbari & Shafi (2018). Some structures may fit the data better but are more complex and harder to understand, while simpler and more intuitive structures might fit the data poorly. Choosing between these alternatives should remain a human task, heavily influenced by the model's purpose and the problem's context.

Although significant progress is still needed before generative algorithms can become effective model-building assistants, real advancements are currently being made in this direction. Our findings emphasize the essential role of human oversight in the use of generative algorithms, and we advocate for viewing these algorithms as tools to deepen rather than supplement our own investigation of problems. Future research should focus on enhancing the capacity of generative algorithms to handle more complex and realistic problems by improving the search and optimisation processes, and perhaps most crucially, integrating context and problem-specific knowledge into the generative process. These advancements can lead to generative algorithms becoming powerful aids for modellers, facilitating a more efficient and insightful exploration of complex problems through the effective combination of human understanding and algorithmic efficacy.

## Disclosure of interest

We have no conflicts of interest to disclose.

## Declaration of funding

No funding was received.

## References

Abdelbari, H., & Shafi, K. (2017). A computational Intelligence-based Method to 'Learn' Causal Loop Diagram-like Structures from Observed Data. *System Dynamics Review*, *33*(1), 3-33.

Abdelbari, H., & Shafi, K. (2018). Learning structures of conceptual models from observed dynamics using evolutionary echo state networks. *Journal of Artificial Intelligence and Soft Computing Research*, *8*(2), 133-154.

Abdelbari, H., & Shafi, K. (2019). A System Dynamics Modeling Support System Based on Computational Intelligence. *Systems*, *7*(4), 47.

Akhavan, A., & Jalali, M. S. (2023). Generative AI and simulation modeling: how should you (not) use large language models like ChatGPT. System Dynamics Review.

Armenia, S., Franco, E., Iandolo, F., Maielli, G., & Vito, P. (2024). Zooming in and out the landscape: Artificial intelligence and system dynamics in business and management. *Technological Forecasting and Social Change*, *200*, 123131.

Auping, W. L., Pruyt, E., & Kwakkel, J. H. (2014). Dealing with multiple models in system dynamics: Perspectives on the future of copper. International Journal of System Dynamics Applications (IJSDA), 3(4), 17-35.

Bankes, S. (1993). Exploratory modeling for policy analysis. Operations research, 41(3), 435-449.

Bengio, Y., Lodi, A., & Prouvost, A. (2021). Machine learning for combinatorial optimisation: a methodological tour d'horizon. *European Journal of Operational Research*, *290*(2), 405-421.

Box, G. E., Hunter, W. H., & Hunter, S. (1978). Statistics for experimenters (Vol. 664). New York: John Wiley and sons.

Brunton, S. L., Proctor, J. L., & Kutz, J. N. (2016). Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, *113*(15), 3932-3937.

Chen, Y. T., Tu, Y. M., & Jeng, B. (2011). A machine learning approach to policy optimisation in system dynamics models. *Systems Research and Behavioral Science*, *28*(4), 369-390.

De Bock, K. W., Coussement, K., De Caigny, A., Slowiński, R., Baesens, B., Boute, R. N., ... & Weber, R. (2023). Explainable AI for operational research: A defining framework, methods, applications, and a research agenda. *European Journal of Operational Research*.

DeDeo, S., Krakauer, D. C., & Flack, J. C. (2010). Inductive game theory and the dynamics of animal conflict. PLoS computational biology, 6(5), e1000782.

Drobek, M., Gilani, W., Molka, T., & Soban, D. (2015, June). Automated equation formulation for causal loop diagrams. In *International Conference on Business Information Systems* (pp. 38-49). Springer, Cham.

Eker, S., Slinger, J., van Daalen, E., & Yücel, G. (2014). Sensitivity analysis of graphical functions. System Dynamics Review, 30(3), 186-205.

Epstein, J. M., Garibay, I., Hatna, E., Koehler, M., & Rand, W. (2023). Special Section on" Inverse Generative Social Science": Guest Editors' Statement. Journal of Artificial Societies and Social Simulation, 26(2).

Forrester J.W. (1961). Industrial dynamics. Cambridge, Mass., M.I.T. Press

Hines, J. (2005) Molecules of Structure: Building Blocks for System Dynamics Models. Version 2.02. Retrieved from: https://www.vensim.com/documentation/Models/Molecules/molecule.pdf

Hosseinichimeh, N., Majumdar, A., Williams, R., & Ghaffarzadegan, N. (2024). From text to map: a system dynamics bot for constructing causal loop diagrams. System Dynamics Review, e1782.

Homer, J. B. (1996). Why we iterate: scientific modeling in theory and practice. System Dynamics Review: the Journal of the System Dynamics Society, 12(1), 1-19.

Kobbacy, K. A., Vadera, S., & Rasmy, M. H. (2007). AI and OR in management of operations: history and trends. *Journal of the Operational Research Society*, *58*(1), 10-28.

Kolides, A., Nawaz, A., Rathor, A., Beeman, D., Hashmi, M., Fatima, S., ... & Jararweh, Y. (2023). Artificial intelligence foundation and pre-trained models: Fundamentals, applications, opportunities, and social impacts. Simulation Modelling Practice and Theory, 126, 102754.

Kunc, M. (Ed.). (2017). *System dynamics: soft and hard operational research*. Springer.

Kwakkel, J. H., & Pruyt, E. (2015). Using system dynamics for grand challenges: the ESDMA approach. Systems Research and Behavioral Science, 32(3), 358-375.

Li, J., Ye, Z., & Zhang, C. (2022). Study on the interaction between big data and artificial intelligence. Systems Research and Behavioral Science, 39(3), 641-648.

Oliva, R. & Sterman, J.D. (2010). Death Spirals and Virtuous Cycles: Human Resource Dynamics in Knowledge-Based Services. Handbook of Service Science.P. P. Maglio, C. A. Kieliszewski and J. C. Spohrer (eds). New York, Springer:321-358.

Ormerod, R. (2021). The fitness and survival of the OR profession in the age of artificial intelligence. *Journal of the Operational Research Society*, *72*(1), 4-22.

Martinez-Moyano, I. J., & Richardson, G. P. (2013). Best practices in system dynamics modeling. *System Dynamics Review*, *29*(2), 102-123.

Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., ... & Scopatz, A. (2017). SymPy: symbolic computing in Python. PeerJ Computer Science, 3, e103.

Rahmandad, H., Oliva, R., & Osgood, N. D. (2015). Analytical Methods for Dynamic Modelers. Cambridge, Mass., M.I.T. Press

Rand, W. (2019). Theory-interpretable, data-driven agent-based modeling. Social-behavioural modeling for complex systems, 337-357.

Remmers, J. O., Teuling, A. J., & Melsen, L. A. (2020). Can model structure families be inferred from model output? Environmental Modelling & Software, 133, 104817.

Rouwette, E. A., Vennix, J. A., & Mullekom, T. V. (2002). Group model building effectiveness: a review of assessment studies. *System Dynamics Review: The Journal of the System Dynamics Society*, *18*(1), 5-45.

Schaffernicht, M. F., & Groesser, S. N. (2016). A competence development framework for learning and teaching system dynamics. *System Dynamics Review*, *32*(1), 52-81.

Schoenenberger, L., Schmid, A., Tanase, R., Beck, M., & Schwaninger, M. (2021). Structural analysis of system dynamics models. *Simulation Modelling Practice and Theory*, *110*, 102333.

Sterman, J. D. (1984). Appropriate summary statistics for evaluating the historical fit of system dynamics models. *Dynamica*, *10*(2), 51-66.

Sterman, J.D. (2000). *Business dynamics*. McGraw-Hill, Inc.

Sterman, J.D. (2018). System dynamics at sixty: the path forward. System Dynamics Review, 34(1-2), 5-47.

Steinmann, P., Auping, W. L., & Kwakkel, J. H. (2020). Behavior-based scenario discovery using time series clustering. *Technological Forecasting and Social Change*, *156*, 120052.

Veldhuis, G. A., Blok, D., de Boer, M. H., Kalkman, G. J., Bakker, R. M., & van Waas, R. P. (2024). From text to model: Leveraging natural language processing for system dynamics model development. System Dynamics Review, e1780.

Virgolin, M., & Pissis, S. P. (2022). Symbolic regression is np-hard. arXiv preprint arXiv:2207.01018.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., ... & Van Mulbregt, P. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. Nature methods, 17(3), 261-272.

Vu, T. M., Probst, C., Epstein, J. M., Brennan, A., Strong, M., & Purshouse, R. C. (2019). Toward inverse generative social science using multi-objective genetic programming. In Proceedings of the genetic and evolutionary computation conference (pp. 1356-1363).

Vu, T. M., Buckley, C., Duro, J. A., Brennan, A., Epstein, J. M., & Purshouse, R. C. (2023). Can social norms explain long-term trends in alcohol use? Insights from inverse generative social science. *Journal of artificial societies and social simulation: JASSS*, *26*(2).

Volterra, V. (1928, 04). Variations and Fluctuations of the Number of Individuals in Animal Species living together**.** *ICES Journal of Marine Science***,** 3(1), 3-51. doi:10.1093/icesjms/3.1.3

Xu, L. D. (2022). Systems research on artificial intelligence. Systems Research and Behavioral Science, 39(3), 359-360.

Yücel, G., & Barlas, Y. (2011). Automated parameter specification in dynamic feedback models based on behaviour pattern features. *System Dynamics Review*, *27*(2), 195-215.

Zhao, W. (2019). *Automated Model Conceptualization and Interactive Modeling Environment: A Software Prototype* (Master's thesis, The University of Bergen).