Economic Greenhouse Decision Support

Embedding a Long Short-Term Memory Network in a Constraint Programming Decision Support System

Dirk van Bokkem

Economic Greenhouse Decision Support

Embedding a Long Short-Term Memory Network in a Constraint Programming Decision Support System

by

Dirk van Bokkem

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday May 17, 2022 at 13:45.

Student number:	4454057	
Project duration:	September 1, 2021 – May 17, 2022	
Thesis committee:	Dr. N. Yorke-Smith,	TU Delft, supervisor
	Dr. S. Dumančić,	TU Delft, daily supervisor
	Dr. G. Iosifidis,	TU Delft
	M.D. van den Hemel, MSc.	Delphy B.V.

TU Delft

Abstract

The increasing global food demand, accompanied by the decreasing number of expert growers, brings the need for more sustainable and efficient solutions in horticulture. Consultancy company Delphy aims to face this challenge by taking a more data-driven approach, by means of autonomous growing inside the greenhouse. The controlled environment of greenhouses enable data collection and precise control. Delphy's current solutions focus on the needs of the crop, but a grower also needs to consider the economic aspect of taking certain decisions on the greenhouse climate. A potential method for solving this complex problem is Constraint Programming (CP). In this work, the applicability of CP for the greenhouse economic optimal control problem will be studied. The contributions of this work are threefold; First, the greenhouse climate is modelled with Long Short-Term Memory (LSTM) and Temporal Convolutional Network (TCN) machine learning models. Secondly, an LSTM model is embedded into a CP model. Lastly, the profit of the grower is optimised through this CP decision support system (DSS). The performed experiments show that both types of time-based machine learning models can model greenhouse temperature and humidity deficit with reasonable accuracy, while light and CO_2 are harder to predict. The correctness of the LSTM-in-CP embedding is validated. The implemented DSS is not yet practically applicable, as the search space is too large to come to reasonable results for realistic instances. For small instances however, the DSS is able to improve the decisions of the grower, demonstrating the potential of using CP for economic greenhouse decision making.

Preface

The thesis that lies before you is the result of months of studying literature, learning about crops and greenhouses, unravelling complex models, and implementation efforts, and simultaneously forms the conclusion of my time as a Computer Science student at TU Delft. This thesis would not have been possible without the help of many people.

Firstly, I would like to thank my supervisor Dr. Neil Yorke-Smith for his overall guidance and helping me set up a thesis project at Delphy, and my daily supervisor Dr. Sebastijan Dumančić, whose feedback and directions helped me greatly and reassured me I was going the right way forward.

Furthermore, I want to thank everyone from team Delphy Digital for the discussions, introducing me to the interesting world of horticulture, and making me feel welcome. A special thanks to my internship supervisor Max van den Hemel, our weekly meetings were key to the successful completion of my thesis.

Lastly, I am very thankful to all my friends and family who have supported me throughout my studies. Thank you Elisabeth Hengeveld, for helping me get through my stressful moments, I could not have finished this thesis without you. Shanti Gajadin, thank you for always pushing me beyond my limits. Thanks to Naqib Zarin for being my thesis sparring partner. And finally I am forever grateful to my parents, thank you for supporting me in all my decisions.

Dirk van Bokkem
Delft, May 2022

Contents

1 Introduction

With the rapidly increasing world population, effects of climate change, and decrease of expert grower knowledge, sustainable solutions in agri- and horticulture are needed to continue meeting the global food demands in the future. Data-driven greenhouses can play a big part in this solution, as the controlled environment allows for higher productivity, prolonged cultivation periods, leading to better crop yields [1].

Consultancy company Delphy takes on this task by doing research in data-driven cultivation for food and flowers. Delphy's goal is to contribute in global health, food safety, sustainability, and the well-being of humans on earth, by providing knowledge and expertise to growers around the globe to optimise production[1]. While human expert knowledge is still a big part of Delphy's success, the decreasing number of experts and growing food demand require an additional approach. Therefore Delphy is working on an autonomous cultivation system.

Many user adjustable settings exist in modern day greenhouses, but there is little knowledge on the long-term economic consequences of these short-term decisions [1]. Therefore the aim of this research is to understand what effect daily greenhouse actions have on the eventual yield and profit. The first step will be creating a model that captures the greenhouse climate and crop growth, followed by the implementation of a decision support system (DSS) that aids the grower in making economic decisions within the greenhouse. The contributions of this study will thus be an economic decision making framework, modelling the greenhouse and crop, and leveraging these models within the decision making process.

1.1. Motivation

Greenhouses are energy-intensive, but do provide a controlled environment for crops to grow closer to their location of consumption [2]. Data-driven cultivation in greenhouses can help both in meeting the increasing global food demands, as well as lowering the impact of said greenhouses on the climate.

While the potential benefits of data-driven cultivation seem obvious, this approach is not yet widely adopted by growers, who rely largely on their own experience in the field. Many decisions made by the grower are based on economic objectives, which is why an economic data-driven DSS could aid in the adoption of such techniques. What is missing for growers, is an extension of their own experience; a tool that can help them understand long-term economic effects of short-term decisions [3]. Especially in times of uncertainty regarding energy and gas prices, growers are in need of economic decision support as experience alone might not suffice in making the best decisions.

An underexposed but promising method in developing such systems in a horticultural setting is Constraint Programming (CP). Also, modelling the greenhouse and crop is predominantly done in mathematical models, while the complexity of the greenhouse and crop provides a good playing field for machine learning models, utilising sensor data. The interplay between such machine learning models and CP is interesting in the complex greenhouse environment as well as outside the horticultural domain.

[1]https://delphy.nl/en/about-delphy/

1.2. Delphy

This thesis was carried out through an internship at Delphy. Delphy is a consultancy company in the global food and flower business, as is entailed in their slogan: "Worldwide Expertise in Food & Flowers". With a total of over 250 employees, of which 180 in the Netherlands, Delphy works on knowledge development & knowledge implementation globally.

Data-driven cultivation is more and more becoming a topic of interest for Delphy, as they see the number of expert growers declining while the demand for food and flowers is increasing. Team Delphy Digital is specifically working on this digitalisation. Step by step, Delphy Digital is working towards an autonomous greenhouse system in which sensor data is used to aid in the decision making process. Their current system can be divided into three parts:

1. *Crop strategy*
   In the Quality Management System (QMS), the crop's characteristics are modelled, and a cultivation strategy is set up to reach a certain yield goal.
2. *Cultivation profilers*
   The cultivation profilers will use the crop strategy and sensor data to create various daily targets in the greenhouse, such as lighting, heating and CO_2 in the Climate Profiler.
3. *Operational controller*
   To reach the daily targets, the profiler outputs need to be converted to certain setpoints (e.g. opening windows) in the climate computer. This is done by the Operational Controller.

Previous work on the autonomous cultivation system was done by another student, by adding the crop profiler [4]. In this work, the aim was to reach an optimal climate by looking at the conversion of light to plant growth. Currently, this model focuses on the needs of the plant. While this leads to good crop yields, the ultimate goal for growers is not necessarily getting the largest possible yield, but making the most profit. As mentioned in the recommendations of [4], what is missing is an economic model that incorporates costs and benefits of greenhouse actions to get an optimal profit.

Part of the improvement of this work was done by another student [5], by improving the photosynthesis model and incorporating the costs of greenhouse lighting into the model. There is still room for improvement, as a more generic economic model including other climate variables could help in making a better economic decision.

The objective for Delphy is thus to develop a generic, intelligent, and economic DSS, that supports growers in their daily decision making process. The costs and resources used in the greenhouse should be optimally distributed to reach long-term cultivation targets, leading to a better profit.

1.3. Problem statement

Modern day greenhouses provide growers with an extensive but complicated set of tools to create a controlled environment that is tailored to the crop's needs. The developments in sensors and controls has improved the efficiency in favor of the crop, resulting in better yield production. Although this improvement helps the overall profitability of the grower, the focus is not on the economics but rather on the crop. While crop growth seems an important goal for a grower, the real driving force to adopt certain strategies lies in the profitability. Since each of the available controls that stimulates crop growth has a certain cost for its operation, an optimal cultivation in the grower's perspective would entail maximising the yield while minimising the costs.

The greenhouse environment is a complex, non-linear, multi-input-multi-output (MIMO) system. Multiple variables determine the inside climate, which in turn determines the state of the crop. The variables that determine the inside climate can be roughly divided into three categories; the current state of the inside climate, the outside climate, and the actions in the greenhouse such as opening the windows and injecting CO_2. An overview of the greenhouse and crop states, and their relations can be seen in Figure 1.1. The states are not defined by these dynamic relations only, but also depend on some static properties of the greenhouse and crop, such as the transmissivity of the greenhouse glass and the size of the greenhouse. However, the necessity of these constant properties within a model or system depends on how the problem is modelled.

The main difficulty in modelling the greenhouse and crop and making decisions, lies in the varying timescales. Opening the windows can cause a shift in the inside temperature within the hour, while the full lifespan of a tomato from flower to fruit ranges between one to two months [6]. To determine the best actions in the greenhouse, one has to understand the impact of short-term decision on the long-term yield.

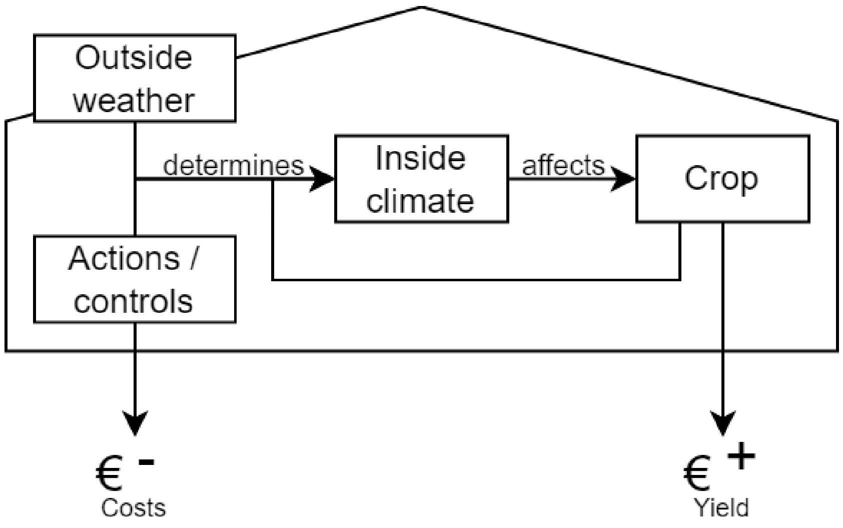


Figure 1.1: Greenhouse and crop states, with their relations and economic effect.

Another challenge, is understanding the inter-dependencies between all these variables. Some decisions affect multiple state variables. For example, opening the windows has an effect on the temperature, humidity, and CO_2. But there also exist cyclical dependencies. The inside climate changes the state of the crop, while the crop in turn has an effect on the climate e.g. through transpiration [6].

The greenhouse and thus the decision making process are also influenced by the weather and its uncertainty. In the near future, the weather can be forecasted with relative accuracy, but this becomes an issue when trying to make predictions and decisions for the long-term.

As mentioned, CP is an interesting and useful method for solving such a complex problem, in which a declarative model is created with imposed constraints on so-called decision variables, where state-of-the-art solvers can find feasible solutions and additionally solve for optimality [7]. A sub-goal of this research is studying the applicability of CP in an economic greenhouse DSS.

1.4. Research questions

Following the problem statement and the objective from Delphy, a research question can be formulated. Two important aspects of this research question are the effect of short-term decisions on the long-term profit, and studying the applicability of CP in economic greenhouse decision making. The main research question to be answered in this research is:

> How can short-term decisions on the climate in a greenhouse with certain costs be made such that it leads to a better long-term profit, using Constraint Programming?

To be able to make decisions in the greenhouse, both short- and long-term effects of these decisions on the greenhouse and crop need to be modelled. Understanding and modelling these effects will be the first stage of this research. These models will be used in a CP approach, which entails the second stage of this research. The following sub-questions help in guiding both stages and answering the main question:

1. How can short-term decisions and their long-term profit in a greenhouse be modelled?
2. How can this model be used in combination with Constraint Programming?
3. How does this Constraint Programming model aid growers in the economic decision making process?

1.5. Scope

In this project, the focus will be on tomato, mainly because this is one of the most studied crops and data of tomato cultivations is available at Delphy. Still, the approach will be as general as possible to enable an easy rollout to other crops in the future. While a crop's health and production efficiency is reliant on multiple aspects such as irrigation, nutrients, use of pesticides, and labour like leaf pruning; only the climate will be taken into account in this thesis, which consists of light, temperature, humidity, and CO_2. The greenhouse climate has a large impact on the crop and the energy needed to realise this climate is a major cost item [8]. An assumption on the magnitude of these costs of operating the greenhouse will be made, as well as tomato prices that determine the revenue. Reasonable assumptions will be made, but the prediction of future energy costs and tomato auction prices are out of the scope of this project. Lastly, the tomato yield is determined by both the quality and quantity of the tomatoes. However, in this research only the quantity will be taken into account.

1.6. Contributions

In this work, multiple components contributed to the realisation of the DSS. Firstly, I implemented two types of time-based neural networks to predict the inside climate of the greenhouse; a Long-Short-Term Memory (LSTM) and a Temporal Concolutional Network (TCN). Secondly, I embedded an LSTM in a CP model, by reducing it to its mathematical form so that it can handle decision variables. Thirdly, I implemented a complete CP model that incorporates a greenhouse and crop model and aims to maximise the grower's profit. Finally, as requested by Delphy, I built a framework including a clear separation of these three components to easily alter and extend this DSS.

1.7. Outline

The structure of this thesis will be as follows. First, the greenhouse and crop, relevant models, and other approaches will be explained in more detail in chapter 2, including related work. Next, an overview of the developed framework of the DSS will be presented in chapter 3. Then the implementation details of the framework will be laid out in chapter 4, starting with the implementation of the machine learning models based on time series data, followed by the embedding of such a model into a CP model, and concluded by the complete CP DSS. Experiments that demonstrate and validate the implemented models and DSS will be laid out in chapter 5. In chapter 6, a conclusion with answers to the research questions will be given, along with the limitations of this study with directions for future work.

2 Background & Related Work

Various approaches to the control of greenhouses for optimal crop growth have been applied, some of them also focusing on the economic aspect. Many of these approaches rely on greenhouse and crop models. In this section, a general background of greenhouses and plant physiology is given, including a review of relevant literature related to both modelling the greenhouse and crop, as well as decision making support in greenhouses.

2.1. Greenhouse climate

Greenhouses are closed systems that provide a way for growers to cultivate crops that would normally not grow in a certain region or season [9]. The crops are protected from extreme weather conditions and their production is optimised through various techniques, like artificial LED lighting and CO_2 injection [10, 11].

Various types of greenhouses exist, but generally they consist of a sturdy framework covered with translucent material, allowing light to pass through and heat to be retained inside. The Venlo-style greenhouse (see Figure 2.1), the main type of greenhouse present in the Netherlands, has an aluminum or steel frame and a glass cover, providing a high light transmission [9]. Windows regulate the ventilation inside and multiple additional techniques provide these greenhouses with better control, such as heating pipes, artificial lighting, and CO_2 injection. Because the Venlo-style greenhouse is expensive, globally many growers use other types of greenhouses, depending on the local environment. For example, both in China and Spain, plastic covers and less sturdy frameworks suffice in environments without snowfall, reducing the costs of construction [9]. In this research, the focus will be on the Venlo-style greenhouse, as the data available from Delphy is generated in this type of greenhouse.

Looking more in-depth at the operation of greenhouses, we arrive at energy and mass balances [11]. These balances model the current state of the greenhouse climate and are influenced by both outside weather and actions inside the greenhouse. Generally, these balances consist of three elements; energy or mass coming into, leaving, and generated inside the greenhouse [11]. Due to the complexity of greenhouses and the vast amount of variables, often assumptions are made on the materials and volumes inside the greenhouse, such as homogeneity of the cover material [11].

The environmental factors that influence these balances can be divided roughly into the following three categories:

- *Outside weather*
  Outside radiation, temperature, humidity, and CO_2 levels can be utilised by the greenhouse, depending on the need of the crop. Absolute humidity is considered here, as relative humidity becomes less useful in the situation where you are dealing with both inside and outside humidity. Even at 100% relative humidity outside, moisture inside the greenhouse can be reduced by ventilation because of temperature differences [12]. Wind speed also plays an important role, as it influences the effectiveness of opening the windows. Depending on the actions taken inside the greenhouse, the effects of the outside climate can be either weakened or strengthened.

- *Actions in the greenhouse*
  The most basic and cost-efficient actions in the greenhouse include ventilation through the opening

Figure 2.1: Venlo-style greenhouse at Delphy.

and closing of windows, and heat reduction through the use of a shading screen. More expensive actions that influence the inside climate are heating pipes and artificial lighting, providing respectively more heat and radiation. Due to the positive effect of CO_2 on crop production, in some greenhouses CO_2 can be injected. To regulate humidity, fogging systems can be installed.

- *Inside climate*
  The inside climate relevant for crop production can be modelled by light, temperature, CO_2, and humidity. Wind speed within the greenhouse also influences this inside climate, but because of the closed environment of the greenhouse, this effect will be neglected and homogeneity throughout the greenhouse is assumed for each of these environmental factors. The inside climate can be seen as the current state of the greenhouse. All four inside climate variables are explained below in more detail.

  – *Light* is an important factor for crop growth, but crops do not use the full spectrum of radiation wavelengths. The wavelength range that crops can use is called Photosynthetically Active Radiation (PAR) and ranges from 400 to 700 nm. Since we are interested in the usable wavelengths, light inside the greenhouse is often expressed as PAR-light in $\mu mol/m^2/s$.

  – *Temperature* is dependent on many greenhouse elements, such as sun radiation, artificial lighting, the heating system, and ventilation. It is expressed in °C.

  – *Humidity* is the amount of water vapour in the air and can be expressed in various ways. Firstly, we have the absolute humidity (g/m^3), which as the name implies is the absolute amount of water vapour in the air. Air can only contain a certain amount of vapour, depending on the temperature. The amount of water vapour in the air, compared to how much water vapour this air can contain given the current temperature, is called relative humidity (%). Lastly, we have humidity deficit, which perhaps is the most important in greenhouses because it indicates how much transpiration can occur in the crop. Humidity deficit (g/m^3) namely tells us how much water vapour the air can still absorb.

  – *CO_2* is a building block for crop production and thus is included in the inside climate variables as well. It is measured in concentration within the air (ppm).

One additional environmental factor that is heavily interconnected with the greenhouse climate, is the crop itself. Especially the amount of CO_2 and air humidity in the greenhouse affect, but are also affected by the crop [6]. Often the climate and crop model are separated, but in [6] it is argued that for an optimal control of the greenhouse system these two should be combined. This indeed provides a more complete view on the greenhouse system, but makes it even more complex. Therefore, automating the process of finding relations between these subsystems is an interesting topic of study.

2.2. Crop

Although the focus is on the control of the greenhouse, the main topic of interest is the crop and its potential production. For the crop to function properly and deliver the fruits needed for a profitable greenhouse system, it needs a favorable climate to stimulate crop growth and crop developement. Crop growth is the

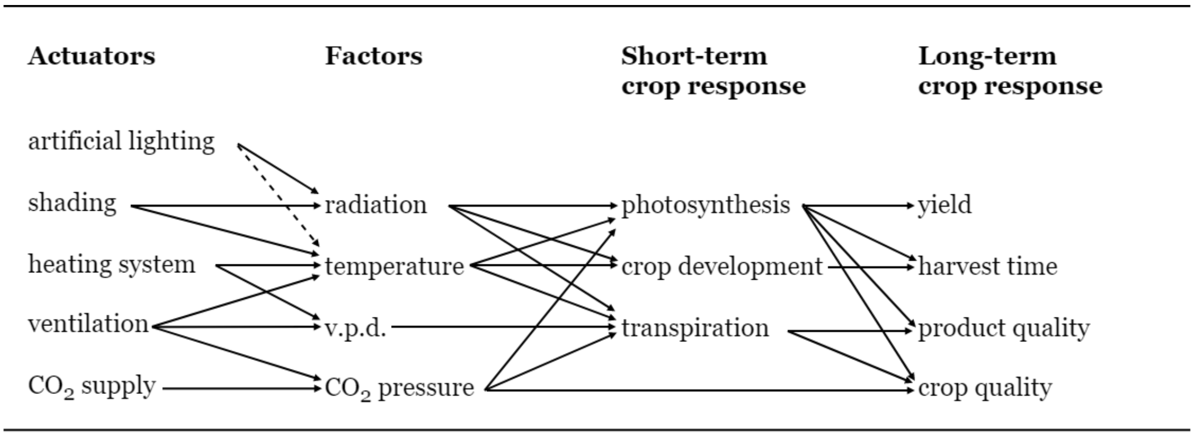


Figure 2.2: Basic overview of important relations between the inside climate, actions in the greenhouse, and crop processes. Source: [13], with the addition of the actuators "shading" and "artificial lighting".

overall increase in size of the crop and its biomass. Crop developement entails the formation and maturation of plant organs, such as flowers and fruits. The most important processes studied in the literature that together stimulate growth and development are photosynthesis and (evapo)transpiration (evaporation is the general term for converting water to vapour including the water in the soil, transpiration is evaporation explicitly occurring from the plant's leaves, evapotranspiration is the umbrella term for the two). In Figure 2.2, an overview is given that relates these two short-term crop processes with the greenhouse actuators and climate factors, which eventually lead to a long-term crop response. From this figure we see the importance of focusing on photosynthesis and evapotranspiration to get good crop production results in the long-term. These two processes and a general understanding of the crop will be discussed in the following paragraphs.

On a high level, the crop tries to maintain three balances; energy, water, and assimilates (carbohydrates or sugars used for growth and energy within the plant) [12]. The energy balance is influenced greatly by the surrounding environment of the crop. For example through radiation or greenhouse temperature. Because the crop tries to maintain the energy balance, it will release water to cool down if the greenhouse temperature is higher than the crop temperature through transpiration [12]. At the same time, this affects the water balance, which causes the crop roots to take up more water, also affecting the assimilates balance through uptake of nutrients from the soil [12]. Lastly, the production and consumption of assimilates are contained within the assimilates balance, which is an important aspect of crop development [12].

The main driver of crop development is the process of photosynthesis, in which the plant uses radiation energy to develop dry matter in the form of plant organs. Many different photosynthesis models exist, each excluding or including different variables. However, the overall narrative is the same; with energy obtained through light interception, a chemical reaction occurs in the plant that converts water and carbon dioxide (CO_2) into assimilates and oxygen. The efficiency of the plant production is an interplay between photosynthesis and transpiration.

Transpiration is the evaporation of water by a crop through its leaves, which affects the production of plant organs [14]. Generally, the crop tries to balance the transpiration rate with the water uptake, to maintain the water balance. This means that the transpiration rate will be restricted to match the available water [12]. To understand what happens in the crop, we need to zoom in on its leaves. Water enters the crop through its roots and exits from the leaves through small openings called stomata [15]. These stomata regulate the processes within the crop, by closing in stressful situations and opening when the conditions are right. Effectively, this regulates the activity of the plants. For example, when the humidity is low, these stomata will close, so that water is contained within the crop. In doing so, the crop stops its own photosynthesis process, because the nutrients intake stops and it cannot release oxygen [12]. A humidity too high is also not desired as the amount of vapour that the air can absorb is smaller, thus lowering the transpiration rate.

We can see that the importance of maintaining a desirable climate is evident, but it is not the only way to steer crop growth and development. A very important steering option for the grower is the pruning of leaves and flowers. The grower can decide on the amount of stems, or plant organs like leaves and flowers can be removed. Ultimately, these decisions regulate the amount of fruits that will be produced and their growth time spans. While these so-called labour actions have a substantial effect on crop production, they are not in the scope of this study.

To stimulate crop growth, we thus focus on the photosynthesis and (evapo)transpiration processes. At the same time, for the plant to be able to maintain its balances, all the conditions must be just right. We cannot stimulate photosynthesis indefinitely when it can lead to the plant not maintaining its balances causing it to die. Generally speaking, it can be seen that during the day it is beneficial for a crop to have high light levels in combination with CO_2, accompanied by "good" ranges of temperature and humidity such that the crop keeps its processes going. Determining these "good" ranges is a complex problem, and exactly how the crop is affected by the combination of all these environmental factors remains difficult to model. Trying to understand the crop and greenhouse is done both through mathematical models and machine learning models, which will be laid out in the following sections.

2.3. Mathematical models

The models on which DSSs rely are at least as important as the algorithms themselves. And to decide on short-term actions in a greenhouse that lead to long-term benefits for the crop, both a greenhouse and crop model are needed. While in recent years the field is growing with machine learning models, many mathematical models already exist. The large field of mathematical greenhouse and crop modelling seems to be dominated, but not limited to the following models:

Bram Vanthoor — In [16], B. Vanthoor presents a greenhouse climate model, created to aid in designing greenhouses for diverse climates around the world. The states of the temperature, vapour pressure, and CO_2 concentration in the greenhouse are described by differential equations. Four different greenhouse designs in different climate regions were used to validate the model. While the model is able to simulate the greenhouse indoor climate with an overall root mean squared error (RMSE) lower than 10%, many parameters need to be set in order for the model to work. Next to a greenhouse model, B. Vanthoor also created a tomato crop growth model. Similarly, this model is sufficiently accurate, but is complex and needs many parameters.

Cecilia Stanghellini — C. Stanghellini mainly focuses on the impact of the greenhouse climate on the transpiration of the crop [14]. The relation between greenhouse climate, crop temperature, and transpiration are described in an energy balance. Again, complicated differential equations and many parameters are used. Some interesting findings are that the transpiration of a crop is mainly influenced by the radiation, temperature and humidity in the greenhouse. Also, an interesting point is made to take a different approach in managing the greenhouse climate, namely by defining a 'transpiration set-point' instead of managing the temperature and humidity directly. An important application that follows is reducing the high costs of humidity management in greenhouses.

TOMGRO — In [17], J. W. Jones et al. present a greenhouse and crop model designed specifically for tomatoes. Again through mathematical equations, the relations between various elements of both the greenhouse and crop are modelled. In TOMGRO, the development of each of the plant's organs are modelled by these relations. Although extensive and well-performing, the impracticability is demonstrated partly by the many parameters used in the model that are specific to a certain tomato variety. This means that these parameters need to be derived for every variety in order to use the model.

2.4. Machine learning models

Mathematical models of the greenhouse and crop can be quite complex and often need many parameters to work, making their implementation impractical and difficult [18]. Machine learning models can potentially cover the complex dynamics of the greenhouse and crop without many parameter settings or calibrations. An adverse consequence of using machine learning models is that they work as a black box and do not provide useful knowledge on what happens in the greenhouse, which is especially troublesome in the case of extreme situations that occur outside of the domain on which such models were trained [19]. Also, a greenhouse always needs a training cultivation before the model can be used, or give up some of its accuracy by using a model trained on a different greenhouse. Taking the trade-off between complexity and understandability into account, machine learning models can still prove to be a suitable solution for greenhouse modelling, as they are becoming increasingly effective at prediction in complex situations. Because we want to discover hidden relations between environmental factors and the crop, neural networks (NN) are considered. Since the greenhouse environment is time dependent, NNs that can handle sequential data are especially interesting. Two of these have already been applied in the context of greenhouses and are laid out below, preceded by the general concept of NNs.

2.4.1. Neural Networks

Based on the neurons and connections in a brain, machine learning NNs learn patterns by strengthening or weakening connections between neurons. In an iterative manner, the connections are updated based on an error function. Over the years, many types of NNs have emerged that are applicable in different domains, but the most straightforward implementation of an NN is the so-called 'multilayer perceptron'. A schematic overview of such an NN can be seen in Figure 2.3, where each circle represents a neuron.

On a high level, multilayer perceptrons are structured by an input layer, one or more hidden layers, and an output layer. The input layer takes a vector of input values, such as the temperature outside, the temperature of the heating system, and the amount of ventilation in the greenhouse. These input values are forward-propagated through one or more hidden layers. Lastly, in the output layer, these propagated values result in some pre-defined output, such as the temperature inside the greenhouse. In a supervised learning approach, the ground truth output value or values following the forward-propagation of input values are known. Comparing the predicted value with the ground truth value results in an error; how far off the prediction is. Through back-propagation, this error is used to update the network to obtain a better mapping of input to output values.

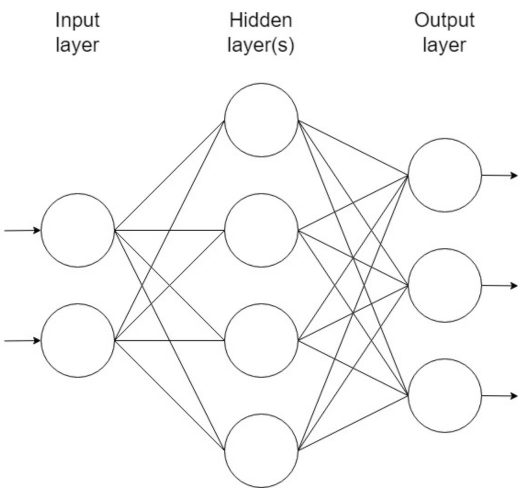


Figure 2.3: Schematic overview of a multilayer perceptron neural network, where each circle represents a neuron.

More specifically, each neuron is represented by a linear combination of variables in its previous input layer with adjustable coefficients known as weights and biases [20]. For a layer of size M and input layer of size D, the so-called activation of each neuron a_j is functionally defined as follows:

$$a_j = \sum_{i=1}^{D} w_{ji} x_i + w_{j0}, \tag{2.1}$$

where $i \in 1..D$, $j \in 1...M$, and the weights and biases are represented w_{ij} and w_{j0} respectively [20]. In the training process, these weights and biases are updated by a gradient proportional to the error of the prediction.

NNs can be used for two types of machine learning methods: classification and regression. In classification, the NN should connect the input to a certain label, i.e. classifying the input. Regression deals with predicting continuous values, which is precisely what we are trying to do when predicting the inside climate and crop growth. The loss function generally used in regression is the mean squared error (MSE), which gives us the average squared difference between the predicted- and actual values. Squaring the values results in outliers having a big influence on the error value, which has a positive effect on the accuracy of our model during training. However, the root mean squared error (RMSE) is preferred when assessing the models' accuracy. Because the RMSE is the square root of the MSE, it is in the same unit as our target value, giving a more interpretable error value.

The multilayer perceptron is the NN in its most basic form, but many adaptations and extensions on this technique exist. In the case of sequential or time-based data, often recurrent or convolutional NNs are used. Two of these time-based NNs are laid out below.

2.4.2. Long Short-Term Memory

A Long Short-Term Memory (LSTM) network is a recurrent NN (RNN), that finds its use mostly in language modelling, but is generally effective in domains with sequential data [21]. Its introduction was made in 1997, and its main purpose was solving the vanishing or exploding gradient problem, which will be explained below [22].

An RNN is an NN that takes into account the inputs of previous timesteps for the prediction of the current timestep, by adding a loop in the network. At each timestep, the input is forward-propagated through the network, with additionally keeping an internal hidden state that is used in the next timestep. The benefit of such an RNN over a classic NN, is that these networks have a memory of past states and can utilise this information. For example in language processing, such an RNN can be used to predict the next word in a sentence, by being able to memorise the previous words in that sentence.

While RNNs have successfully been applied in various domains, they suffer from short-term memory. In an RNN, each timestep can be seen as a separate layer. Because the weights and biases are updated in the back-propagation process by a gradient that depends on the error of the prediction and the gradient in the previous layer, the effect of back-propagation of the error through multiple of these layers can cause the gradients to explode or vanish. This results in the network 'forgetting' about timesteps far back in the past. The LSTM network was specifically designed to mitigate this issue, mainly by "enforcing constant error flow through internal states of special units" [22].

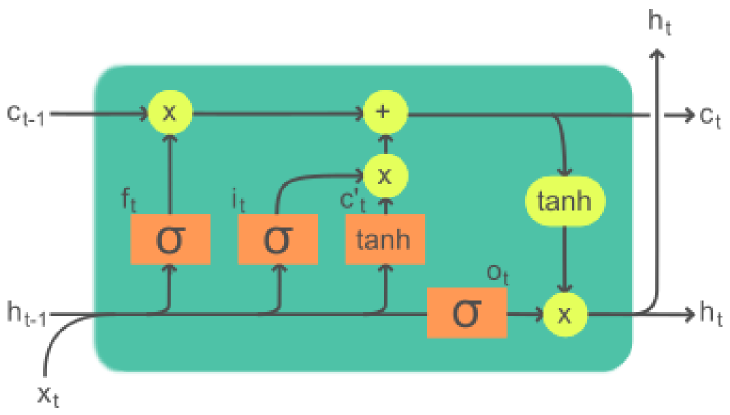


Figure 2.4: Schematic overview of an LSTM cell, showing the calculations for one timestep. From “Wikimedia Commons,” by G. Chevalier, 2018 (https://commons.wikimedia.org/wiki/File:LSTM_Cell.svg) [23], with the addition of indications for the gates f_t, i_t, c'_t, and o_t. Licensed under CC BY SA 4.0

These 'special units' are so-called LSTM cells. A schematic overview of such an LSTM cell can be seen in Figure 2.4. The propagation of values in the LSTM network goes through several gates (input, output, cell, and forget gate) and updates the internal states (cell and hidden state) of the LSTM cell. The gates control the flow of information from the input to the output, and provide a way for the cell to "forget" irrelevant information. Static weight matrices W and U for the input and hidden state respectively, along with bias vectors b are used in each of these gates, whose values represent the trained model. These weights and biases remain the same for each timestep. This results in the following calculations in an LSTM cell for one timestep t;

$$i_t = \sigma_r(W_i \cdot X_t + U_i \cdot h_{t-1} + b_i) \tag{2.2}$$
$$f_t = \sigma_r(W_f \cdot X_t + U_f \cdot h_{t-1} + b_f) \tag{2.3}$$
$$c'_t = \sigma_a(W_c \cdot X_t + U_c \cdot h_{t-1} + b_c) \tag{2.4}$$
$$o_t = \sigma_r(W_o \cdot X_t + U_o \cdot h_{t-1} + b_o) \tag{2.5}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot c'_t \tag{2.6}$$
$$h_t = o_t \odot \sigma_a(c_t), \tag{2.7}$$

with input gate i, forget gate f, output gate o, cell state c (candidate cell state c'), and hidden state h. In equations 2.2, 2.3, 2.4, and 2.5, matrix multiplications are performed. In equations 2.6 and 2.7, element-wise multiplications are performed, denoted with $\odot$. In the equations, two types of activation functions are used; activation σ_a for the cell and hidden states and recurrent activation σ_r for the gates. Generally, in an LSTM network the sigmoid function is used for σ_a and the tanh function for σ_r.

Again, during training the weights and biases are updated until they reach a reasonable prediction power. The general operation of these RNNs is thus quite similar to that of classic NNs, but they differ in being able to hold memory in their cell and hidden states.

Related Work

In [24], an LSTM is used to predict multiple greenhouse climate variables, such as temperature and relative humidity including their minimum and maximum values. Some experiments of finding the best parameters are discussed and the final predictions look promising. However, the figures suggest that there is some copying behaviour occurring. This tends to happen when the model relies too much on its previous value of the target variable, rather than also including other variables such as actions in the greenhouse.

Three time-serial deep NN models are compared in [25]. Out of an ANN, NARX model, and LSTM model, LSTM performed the best, with a standard error of the prediction (SEP) of temperature and CO_2 within 5%. However, the humidity prediction did not perform very well. An important but expected result of this research is that the accuracy of the time-based algorithm decreases as more time steps ahead are predicted. Nevertheless, it is shown that deep learning models can perform well in predicting the greenhouse climate.

Next to predicting the greenhouse climate, LSTM is also used for crop growth prediction. In [26], the Leaf Area Index (LAI) of bell peppers is predicted using LSTM. The LAI indicates the size of the leaf surface and in turn can be used to predict crop growth, but this is not explained in this paper. The model is trained on simulated data, but also validated with LAI data measured of a real crop, showing a high accuracy.

A more direct yield prediction approach is applied in [27]. Here LSTM is compared to a Support Vector Regression (SVR) implementation and an implementation of a Random Forest (RF). In all implementations, a one-step-ahead prediction was done. However, it is not clear if the full sequence shown is a multi-stage prediction (using the prediction of the current timestep to predict the next timestep), or if all predictions are simply based on the previous timestep. Of course, long-term yield predictions are more relevant. Since the environmental data is on an hourly basis and the yield data is gathered weekly, there is a mismatch which is dealt with by interpolation. While real hourly yield data is impractical to obtain, interpolating weekly data to hourly samples may not give accurate results of the effect of environmental factors on yield. Still, the results show the potential of yield prediction through an LSTM.

2.4.3. Temporal Convolutional Network

A more recently developed time series NN is the Temporal Convolutional Network (TCN), which in general works well for sequential data. It was first introduced in [28], as a different application of the very similar WaveNet model introduced in [29] which specifically was used on audio data.

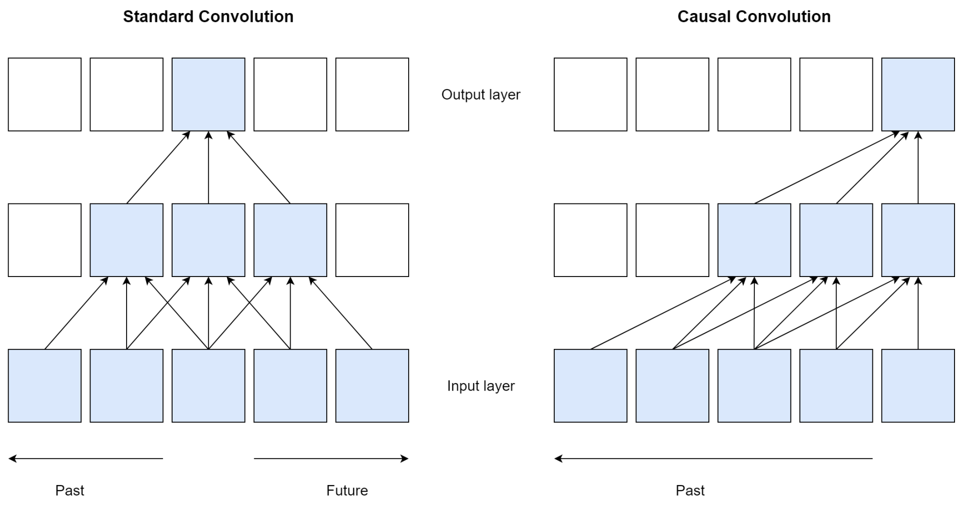


Figure 2.5: Convolutional dependencies of output layer on input layer in a CNN (left) and causal convolutional dependencies in a TCN (right).

A TCN is a special type of convolutional NN (CNN). In a CNN, from input to output in multiple layers, convolutional calculations are done on so-called kernels of input data, which in essence are dot-products that relate one output item to an expanding amount of input items from its previous layers using the corresponding connection weights. To make this more clear, see the left image in Figure 2.5, where one output item in the top layer depends on the previous layer with a kernel size of three. Each of these three items in turn depends on three items in the input layer, thus making the output item dependent on the full input.

When considering time-based data however with consecutive values in both the input- and output layers, this structure would result in an output item of a certain timestep being dependent on future values. Since this contradicts the temporal construct in a time-based prediction, a causal convolution is enforced in a TCN, i.e. items in the output layer are solely dependent on present and past values in the input layer [29]. Such a causal convolution is illustrated in the right image of Figure 2.5. Zeroes are added to the left (past) of the input items such that every output item has a full input range on which it depends.

What may stand out in the figure, is that many weights are needed for all the connections between each consecutive layer and that depending on the amount of layers and kernel size, many connections may be needed to cover the complete desired input size. Therefore in TCN, dilated convolutions are introduced that more efficiently cover the complete input sequence [29]. This is illustrated in Figure 2.6. In each layer, a different dilation factor is used, which determines how many items of the previous layer are used to compute a value.

The resulting structure allows for a long input history with relatively few weights. The range of this long input history that an output item depends on is called the receptive field, and when designing a TCN with an input size of *lag* timesteps, the receptive field should be large enough to cover the lag. The size of the receptive field R is calculated as follows: $R = 1 + \cdot(k-1) \cdot \sum_i d_i$, where k is the kernel size and d_i the dilation factor on layer i. There exist more complex structures of these networks, such as multiple stacks of convolutional layers on top of each other, again increasing the receptive field.

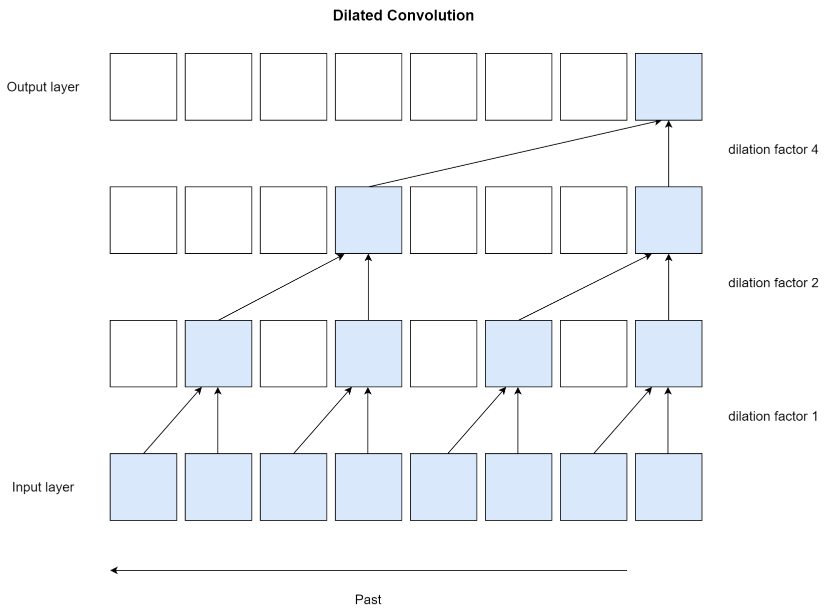


Figure 2.6: Dilated convolution in a TCN, with a kernel size of 2.

Related Work

Recently, TCNs have gained attention in potentially performing better than LSTMs in a wide range of sequential modelling tasks. Specifically in cases where you want to train on long histories of data, a TCN should be preferred because of its longer memory, according to S. Bai et al. [30].

In [31], an LSTM and TCN are combined for crop yield prediction. Here, the environmental and yield data of one week are used to predict the yield after this week. First, the LSTM is applied, followed by the TCN. In their results, they compare their model to using only an LSTM or TCN, among others. For this yield prediction, it seems that the LSTM-TCN combination works best, followed by the LSTM-only and then the TCN-only. Also in this work it is not clear if a multi-stage prediction is done.

2.5. Other approaches

The complex nature of greenhouses lends itself well for automated optimisation techniques. Various approaches have been implemented, such as Genetic Algorithms and Model Predictive Control. The goal of these algorithms, is to optimise the controls of the greenhouse for the objective of maximising either production or profit. Some of these approaches and their relation to economic decision making are laid out below.

2.5.1. Genetic Algorithms

Due to the complexity of greenhouse systems, many recent works in this field approach the problem using genetic algorithms (GA) [32–34], as the optimal decision variables are searched in a stochastic manner, making them independent of special properties of the objective function [34].

[32] evolves control setpoints that outperform "the original setpoints in two objectives: maximizing the economic value of the crop yield and minimizing the variables costs". The algorithm uses a simulation based on the model of B. Vanthoor [16]. The problem is modelled as a multi-objective optimisation problem, providing useful insights on the trade-offs for growers. However, it can be debated that the resulting profit may actually be of most interest for the grower.

In [33], a GA is implemented using engineering constraint rules, greatly improving its optimisation performance and also its practicality. By including engineering constraints, such as interactions between ventilation and CO_2, the algorithm is more tailored to the specific greenhouse conditions. Also, in the paper the effect of changing the amount of collocation points is investigated, essentially setting the size of time intervals. It is concluded that it is a trade-off between real-time performance and correctly coping with the time-delay change of the greenhouse controllers.

Lastly, [34] used and compared two evolutionary algorithms: MSCEA and ESEA. A simplified plant model is used that calculates the yield and profit, which the algorithms use to optimise. While the usefulness of evolutionary algorithms in economic greenhouse decision making is demonstrated well, the algorithms only simulate and optimise for a time interval of 4 hours, so the eventual yield and profit for the long-term is not taken into account.

2.5.2. Particle Swarm Optimisation and Model Predictive Control

Two other approaches for controlling the greenhouse are Particle Swarm Optimisation (PSO) and Model Predictive Control (MPC). Based on animal swarms and herds in nature, PSO algorithms combine "individual and herd knowledge" to reach an optimal solution. MPC is a technique that is used to follow an objective trajectory as good as possible, using a moving time window.

In [35], a PSO algorithm is applied for optimising controller outputs of an MPC, related to air temperature inside the greenhouse. While energy consumption was taken into account in the optimisation objective, the paper mainly focuses on set-point tracking, which is more about controlling the greenhouse, not so much decision-making.

Other work also combined PSO with MPC [36]. Again, while the techniques are good in using future predictions to optimise the greenhouse controls according to a certain control trajectory, they do not necessarily involve decision making to optimise crop production or profit.

2.6. Constraint Programming

A particularly useful method for solving complex real-world problems that was not found in the literature regarding the greenhouse optimal control problem, is Constraint Programming (CP). As discussed in [7], its most important features are *declarative problem modelling, propagation of the effects of decisions*, and *efficient search for feasible solutions.* Many research is already done on understanding and modelling the greenhouse and crop [2], mainly through complicated equations. CP could prove to be a useful approach in greenhouse decision making, by providing a more natural way of modelling the problem making it more understandable. The propagation of the effects of decisions in this approach can be insightful for the decision maker. Lastly, the modelled problem can be solved by the most efficient available solvers, resulting in a practical application that is useful for greenhouse decision makers. A sub-goal of this research is to find out if CP can be applied in this economic greenhouse DSS.

As the name implies, CP revolves around constraints, which are applied on so-called decision variables. In CP, a declarative high-level model of the problem at hand is composed of some constants, constraints, and these decision variables; variables whose values will be decided at runtime. Such a model is called a *Constraint Satisfaction Problem* (CSP) [37]. An important component of a CSP, is the set of values these variables can take, which is called their domain [37]. Such a CSP can be solved for feasibility or additionally for optimality. In an iterative manner, a solver updates these domains as values are propagated through the model. With the constraints and domains, the solver can remove inconsistent solutions to effectively cut down the search space [37]. How a CSP is solved, depends on the solver, but in general a search tree is created and the search can backtrack through this tree when inconsistencies occur [37]. A formal definition of CP is given in [37]:

> *Given a finite set of discrete variables $X = \{x_1, x_2, \ldots, x_n\}, 1 \leq i \leq n$, each with its value in a finite domain, $x_i \in D_i \subset \mathbb{Z}$, and a finite set of constraints $C = \{c_1, c_2, \ldots, c_m\}$ each expressed on a subset of the variables, $c_j(x_{j1}, x_{j2}, \ldots, x_{jk}) \subset \mathbb{Z}^k$, $1 \leq j \leq m$, one must find a combination of values from the domain of each variable that simultaneously satisfies every constraint.*

2.6.1. MiniZinc

MiniZinc is a medium-level CP modelling language, designed to be both simple and expressive [38], and provides a way to declaratively model CP problems. The models created in MiniZinc are translated to models in FlatZinc, "a lowel-level solver input language" [38], which provides an easy way for solvers to solve instances of the model. The MiniZinc language is thus an interface for modellers, while for solvers this is FlatZinc. The intention of this study is to model the economic greenhouse decision system in CP, so MiniZinc is used. The JaCoP solver is responsible for handling the FlatZinc models that follow.

Models in MiniZinc consist of parameters, decision variables, and constraints on these variables.

- *Parameters* define the instance of the model and can be seen as constants [39]. Within the context of CP, these can be seen as variables whose value is already decided. For example, the outside weather is simply taken from a forecast and is constant in the eyes of the model. Parameters can be of type *integer*, *floating point number*, *Boolean*, or *string*. It is also possible to create arrays or sets of these.

- *Decision variables* represent the set of decisions that can be taken and each have a set of possible values; the domain [40]. These variables can be seen as mathematical or logical variables; variables that can be assigned values that satisfy the constraints and fall within the given domain [39]. In the greenhouse DSS, the actions are the decision variables, such as the heating tubes. The possible types for decision variables are *integers*, *floating point numbers*, *Booleans*, or *sets*. Again, arrays or sets of decision variables can be created.

- *Constraints* are imposed on the decision variables, defining the set of feasible solutions. Essentially, they are Boolean expressions that involve decision variables, parameters, or both [39].

Next to these three basic components of a model, MiniZinc also provides a way to annotate expressions with *annotations*. This is useful for giving information to the solver, such as the structure of certain variable arrays or the order of traversing the search tree, to help in finding solutions more efficiently. The most basic search annotations include *int_search* and *float_search*, which take in an array of variables, an order of traversing this array, a method of assigning values to these variables, and for *float_search* additionally a precision float value that indicates when two float values are considered equal [39]. A more extensive search annotation is *seq_search*, in which a list of search annotations can be given that will be executed sequentially [39]. Some solvers also implement the *priority_search* annotation, which allows for even more complex searches [41]. With this annotation, the modeller can use (nested) variable arrays to model variable and value selection during search.

Lastly, in MiniZinc there is a possibility of creating *functions* and *predicates* to model more complex constraints. A function takes one or more inputs, does some operations on these inputs, and returns an output. A predicate can be used to impose constraints on given inputs, meaning it simply is a function with a Boolean output [39].

2.6.2. Empirical Model Learning

In the case of the greenhouse optimal control problem, a part of the CP model will hold the greenhouse and crop model, either through mathematical equations or a machine learning model. However, the tools available to build machine learning models do not directly provide a way to utilise these models in a CP approach, but the trained weights can be extracted and embedded into the CP model. In [42], the combination of learning relations between variables from data and embedding these relations into an optimisation model is called Empirical Model Learning (EML).

In an earlier study, the same authors explore this approach by embedding a trained NN into a CP algorithm, for a "temperature aware workload allocation problem", with promising results [43]. In their approach, a Neuron Constraint is added for each node in the network. The NN in this work is relatively simple, but the authors claim that the same techniques can be used for complex recurrent networks.

Another student has done an embedding of an NN in CP in his Master's thesis [44]. Here, an NN was trained on simulation data of an Enterprise Modelling optimisation problem. An important point in his work is that by automatically converting the NN into something usable by the CP model, the understandability that is the merit of the CP model can be maintained to some extent. The decision maker does not have to model this incomprehensible NN manually, but can still use the predictive power in the otherwise descriptive model.

3 Framework

In this research, I developed a DSS that leverages machine learning models. This approach can be divided into the stages of preparing data, training the models, embedding these models, and finally decision making. One of the requirements from Delphy within this study is creating the framework for this decision making system. The framework I developed is a pipeline connecting each of the stages (see Figure 3.1). Using this method provides a clear overview of the process, as well as modularity for re-use of certain elements, and a possibility for adding models that may be desired by Delphy. Throughout this pipeline, output files are stored in a single experiment folder, along with relevant metadata. The individual stages can be run independently when necessary output of previous stages is present. Each of these stages in this pipeline will be laid out in this chapter on a high level, the implementation details will follow in the subsequent chapter.

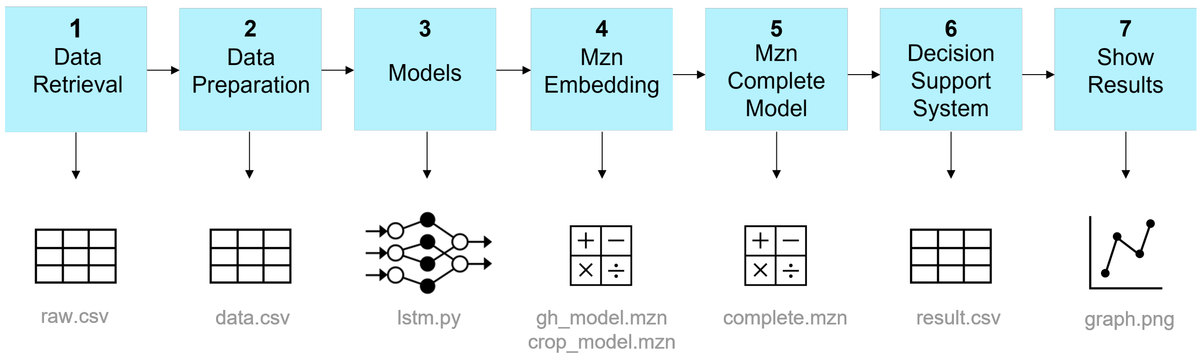


Figure 3.1: A simplified overview of the system pipeline showing the process of all stages in the DSS, along with their outputs. Each of these stages in the pipeline can be run independently, given the outputs of earlier stages. For a more in-depth overview see Appendix A.

3.1. Design choices

To build the framework, I used several third-party libraries, which will be laid out in this section. I wrote the framework in Python 3.8 (`python.org`), mainly because of the available machine learning and plotting libraries. In stage 1 of the pipeline, data is accessed through an API of *Zensie*. Zensie is a platform of 30Mhz (`30mhz.com`), a company managing the data of Delphy. Data preparation in stage 2 is done through a set of operations provided by the *Pandas* library [45]. The machine learning models in stage 3 I implemented with *Keras* [46], and *Keras-TCN* [47] for TCN additionally. For CP modelling I used the MiniZinc language [38], because of its expressiveness and simplicity, also enabling the creation of a solver-independent model. Still, I used a single solver during implementation of the framework, which is the *JaCoP* solver [48]. JaCoP was used because it supports real numbers and has implemented the exponential function, which is used in the activation functions of the machine learning models. MiniZinc comes with a Python interface that eases the use of MiniZinc code within a Python setting, which I used to run the CP model in stage 6. However, this interface is still being developed (see `minizinc.org/doc-2.5.5/en/python.html`). Therefore in stage 4 and 5 of the pipeline, MiniZinc files are created that can also be called directly from the MiniZinc IDE or the command line interface. Lastly, I used *Matplotlib* [49] to generate figures in stage 7.

3.2. Pipeline

The pipeline I constructed consists of separating the different stages and their outputs in a numbered folder structure. Each experiment using one or more of these stages stores all outputs in one folder, including a *metadata* file. Here, all relevant information needed in successive stages is stored, such as chosen features or time lag. These successive stages either retrieve the necessary information from this file, or the user fills these in manually when running a stage separately from previous stages.

1. **Data retrieval**
   The pipeline starts with the retrieval of data, as this is needed in training the models, as well as the decision making process. In the data retrieval part of the pipeline, a connection is made with Zensie to extract data from the available sensors. After choosing a timeframe, timestep frequency, and a set of sensors, a raw data file is created. Five-minute data provides the most detailed information but due to the complexity of the built system by amount of timesteps, features, and NN specific parameters; hourly data was used throughout this study, as this reduces the complexity significantly but still produces reasonable results. For now, the outside weather forecast is retrieved from historical Zensie data, but to make future predictions, actual weather forecast data should be consulted.

2. **Data preparation**
   In this raw data, missing values are guessed through a linear interpolation between preceding and succeeding known values. If the first or last values of a sensor are missing, the closest known value is used. Domain knowledge, in both the mathematical and figurative sense, is utilised through a configuration file in which the bounds for each feature are stored. Bounds are retrieved from the training data in the case of missing feature configurations or if larger (more realistic) bounds are found in the data. The bounds are used both in scaling the data for training the NNs and in the CP model. The scaling of the data is also done in this data preparation stage.

3. **Models**
   The greenhouse and crop models are created and trained in the third stage of the pipeline. Depending on the type of model, various hyper-parameters can be set such as time lag used in training the time series NNs, or the used loss function. The LSTM and TCN models share some functionalities, like preparing the data into a training and test set. The weights of the model are trained and stored to be used later in the embedding into the CP model. The implementation details of these models are given in section 4.1.

4. **Greenhouse & Crop model embedding in CP**
   After the models are trained, they are embedded into a MiniZinc model in stage four. Step-by-step, a MiniZinc file is written that represents the model. The weights and biases are loaded and written to a data file. Each model that is created in this stage, is accessible in the complete model through a single predicate. A simplification of this structure can be seen in Figure 3.2.

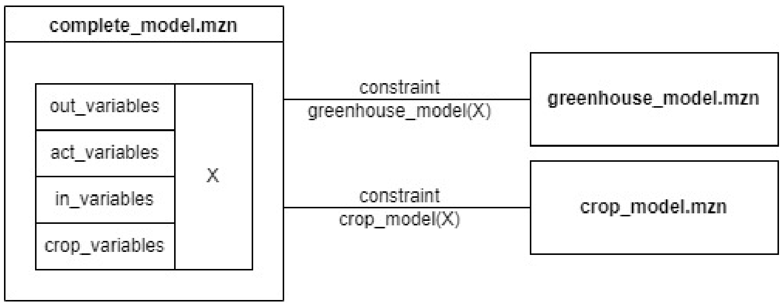


Figure 3.2: Simplification of connection between the complete MiniZinc model, and the greenhouse and crop models.

This fourth stage is not limited to machine learning models. Because the crop is harder to model and validate, a correctly functioning LSTM crop model was not realised in this study. Therefore the more simplistic Lintul-3 model [50] was used, which also works through a single predicate, simultaneously demonstrating the extendability for future model additions and improvements by Delphy. The model embedding in CP is explained in more detail in section 4.2.

5. **CP model**
   Next, the complete MiniZinc model is built, by defining the greenhouse and crop variables and connecting these to the greenhouse and crop models. In this stage, economic variables are defined and connected to the greenhouse action and crop variables. A search strategy is set up and specific constraints on the variables are added, such as some time-wise consecutive greenhouse action values being similar. Lastly, the objective of the model is set up, generally maximising the profit. This complete MiniZinc model that entails the DSS, is explained in more detail in section 4.3.

6. **Decision support system**
   The complete MiniZinc model including greenhouse and crop models, is solved for optimality in the sixth stage of the pipeline. A MiniZinc instance with the model and data files is created. A solver is selected and will start to solve the CP problem. Given the vast amount of possible solutions to trace, a timeout is given to arrive at an intermediate, sufficient solution. If the problem is satisfiable, the best solution so far will be stored, which is the result of the DSS. This result includes the retrieved weather predictions, decided actions in the greenhouse, and predicted inside climate.

7. **Showing results**
   Finally, the last stage includes functions to show the results of the DSS in a meaningful way. Interesting visualisations such as comparing decisions made by the decision support system with actual decisions made in the greenhouse in a real-world cultivation period, are available through these functions.

3.3. Data & Hardware

Throughout the pipeline in training, prediction, and decision making; data of two greenhouse departments of Delphy was used. The initial implementation was done based on data of department 3.5, but to include CO_2 injection of which the sensor is not available in department 3.5, also data of department 6 was used. While department 3.5 is missing a CO_2 injection sensor, department 6 is less realistic because various experiments in the field of energy usage are conducted here. The used sensors for outside weather were the same for both datasets. Because Delphy does research in various crops, a department can not be tied to a single strategy and crop. Therefore specific cultivation periods were taken into account in this study. The specifics of both departments' cultivation periods are laid out below.

Department 3.5 | From week 43 2020 to week 38 2021, a tomato variety Merlice cultivation was done here, with the goal of experimenting with various sensors to know which ones are most related to crop development. The sensors in department 3.5 that were used in this thesis include all those relevant for the inside climate, and for the actions in the greenhouse we have LED and SON-T lighting, under- and growtubes (heating), a shading screen, and lee- and windowside ventilation. The CO_2 injection sensor is missing here. Up to November 24th some data is missing, which is why only the period 25/11/2020 - 20/9/2021 of this cultivation was taken into account. Department 3.5 covers an area of 150m^2.

Department 6 | From week 37 2021 to week 38 2022, there is a cultivation in department 6, again with tomato variety Merlice. This cultivation is thus not finished but will be used in this study, because the CO_2 injection sensor is available here. Department 6 is about energy-saving and thus several experiments concerning energy preservation are conducted here. Sensors for the same inside climate variables as in department 3.5 are used in department 6. The possible actions in this department are under- and growtubes (heating) also, as well as lee- and windowside ventilation. Only LED lighting is used here in the form of two LED strands, in combination with two energy preserving screens. Again, some data is missing so the data of the period 26/9/2021 - 14/3/2022 is used. Department 6 covers an area of 1000m^2.

The implementation of the greenhouse and crop models, their embedding in CP, and the CP model itself, will be laid out in the following chapter, followed by the evaluation of these implemented components in the experiments chapter. For these evaluations, a laptop with i7 64-bit CPU at 2.60 GHz, 16 GB RAM, and NVIDIA GeForce RTX 2070 GPU was used.

4 Implementation

The pipeline framework was constructed mainly to facilitate three components that make up the DSS; the time series NNs, their embedding in CP, and the complete CP model. The overall structure of these components and the libraries used was explained in the previous chapter. The implementation details will be laid out in this chapter.

4.1. Time Series Neural Networks

Both the greenhouse climate and crop growth can be modelled as a time series. The states of the greenhouse and of the crop change over time, and we are interested in their future states. Multiple features in the categories of outside weather, actions in the greenhouse, inside climate, and crop growth are included in these models, meaning we are dealing with a multivariate time series problem.

As seen in the literature, many complex processes occur in the crop that are difficult to model, while these influence the environment substantially [6]. Therefore in this research, I utilised temporal NNs to cover the complex relations between greenhouse and crop. The idea here is to not explicitly model the crop processes like transpiration, but let the NN learn what happens in the greenhouse when certain actions are taken. The goal is that these crop processes are automatically encompassed in this model.

In this section, the structure of these time series problems and their features, as well as the implemented NNs will be laid out. The LSTM and TCN networks are discussed, including design decisions.

4.1.1. Time Series Data

Generally, the input to training an NN consists of multiple feature vectors, each accompanied by an output vector. Each of these feature vectors serves as one data point. For time series, an extra dimension of time is added, leading to one data point matching feature vectors of multiple timesteps to one output vector or output vectors of multiple timesteps. Dealing with time series data comes with challenges and possibilities. Deciding how many timesteps to use for training or prediction can be difficult, but historical data can provide meaningful information. Furthermore, the data points of time itself can be useful, by extracting seasonality features.

Seasonality

Seasonality in time series data can be important information to utilise in a model. Think of the daily cycle of natural light. The time of day directly relates to the amount of sunlight that can be expected within the greenhouse. Other than the day-night cycle, the yearly cycle of seasons can also be very informative, especially in the cultivation of crops. These seasonalities are encompassed in the date and time data points, which are represented by the time dimension of the dataset. However, by simply passing these linear datapoints to the machine learning model, we lose the cyclical information these datapoints hold. We can decompose these dates and times into their sine- and cosine components, enabling us to capture the cyclical relation.

To create the cyclical features for the day cycle, we want to express the time of the day. In this computation, the amount of seconds in the day is used but any precision (miliseconds, hours) can be used depending on the use case of the features. In our case, the smallest reasonable timesteps used will be minutes. For assurance, one extra precision step is taken into account, so the cyclical day features will be calculated using seconds. I calculated the two cyclical time features as follows:

$$\sin\left(2\pi\frac{m}{24\times 60\times 60}\right) \qquad \cos\left(2\pi\frac{m}{24\times 60\times 60}\right), \tag{4.1}$$

where m is the time of day in minutes. Similarly, the yearly seasons can be modelled through a sine and cosine component of the current day divided by the amount of days in a year.

I performed a small experiment to examine the usefulness of the cyclical features and see if these indeed improve the performance of the model. With data of department 3.5 of the period 1/1/2021 - 20/9/2021, I trained two LSTM models; one including and one excluding the cyclical features. All outside weather, inside climate, and action features relevant in department 3.5 were included. I did a multi-stage prediction of three days starting from 20/12/2020 for both scenarios, resulting in an averaged RMSE over 5 runs. With cyclical features, the resulting average RMSE was 0.0351, without cyclical features this was 0.0321. What is clear, is that the model does not gain useful information from the cyclical to significantly improve the prediction, and instead performs slightly worse. Also, the cyclical features add some needless additional complexity. Possibly these cyclical features would improve models that are not time-based, and could benefit from this extra seasonality information. However since we see no improvement, from this point on the cyclical features will not be included in any models.

4.1.2. Multivariate multi-stage prediction

There are multiple possibilities for prediction using time series data, depending on the exact structure of the data and the problem at hand. A multi-step ahead prediction could be implemented when the goal is to predict multiple timesteps in the future for one input-batch of data. Another approach more suitable to the problem at hand is multi-stage prediction. This involves a step-by-step prediction where the prediction of one input-batch serves as part of the input for the next batch. This is especially useful in the context of CP, as the actions involved in the prediction need to be decided for each timestep, while the inside climate prediction depends on its previous state (see Figure 4.1). Using a multi-step ahead prediction in this case, would disable the possibility to decide the actions in these steps-ahead. To conclude, in this study I thus implemented a multi-stage prediction.

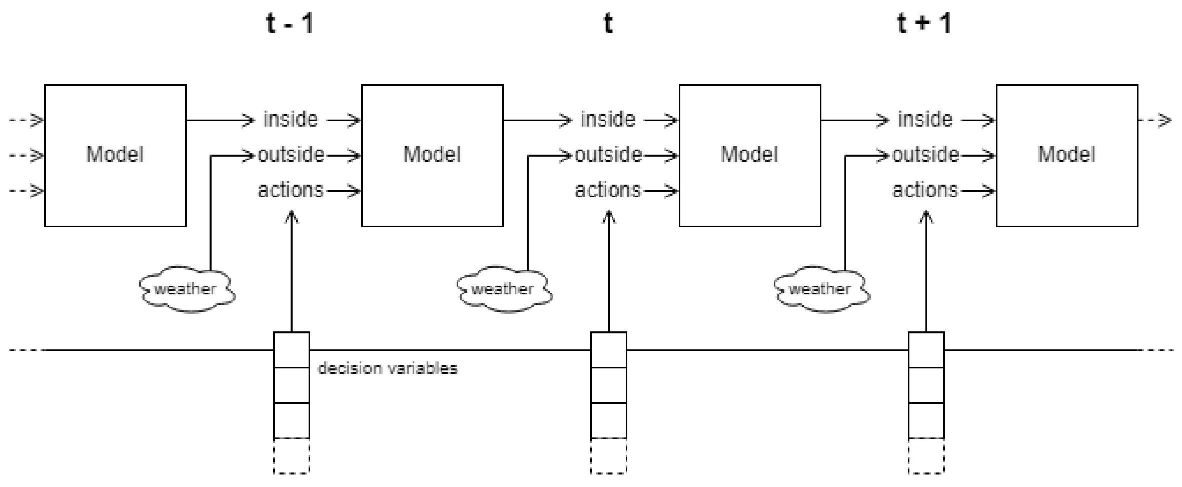


Figure 4.1: Multi-stage prediction in the context of greenhouse decision making. The input of each timestep is the inside climate, forecasted outside weather, and the decided actions of the previous timestep(s). The output is the inside climate for the next timestep.

4.1.3. LSTM and TCN

The LSTM and TCN models I implemented follow a hierarchical class structure so that these models could be used interchangeably in experiments, and for sharing functionality. Firstly, I implemented an overarching class *ML*, that forms the basic structure of the implementing classes. It includes train, predict, and multi-timestep predict functions that are expected to be overwritten in implementing classes. In addition, I implemented a save and load function that makes it possible to save models for later use.

Initially, I implemented a Deep Learning model to predict the inside climate of the next timestep with only one timestep of input data. However, this resulted in a model that simply copied the value of the climate of the previous timestep. This gave low error values for predictions where the input was known. However, when doing a multi-stage prediction where values of the previous timestep are used as input for the next timestep, the resulting prediction becomes infinitely high or low if values of the previous timestep are simply copied. Still, this initial implementation of a non time-based model, incentivised the need for an additional subclass *ML_TimeSeries* that separates time-based and non time-based models.

Both the LSTM and TCN models I implemented as subclasses of this *ML_TimeSeries* class. Both models take as input the relevant parameters for training. For the LSTM this included *epochs* (amount of forward- and backward propagation rounds), *batch_size* (amount of datapoints to use in each epoch), *lag* (amount of previous timesteps to take as input), *test_size* (amount of datapoints to return as test set), *validation_split* (amount of datapoints to use as validation set during training), *loss* function, *optimizer* function, *patience* (amount of epochs to take into account for early stopping), and finally *layers* (list that represents the amount of LSTM layers and their units). For the TCN model this also included *epochs*, *batch_size*, *lag*, *test_size*, *validation_split*, *loss*, *optimizer*, and *patience*. Additional parameters used in the TCN model are *nb_filters* (amount of filters in the convolutional layers, similar to LSTM units), *dilations* (a list indicating the dilation step-size in each layer), *kernel_size*, and *dropout_rate* (amount of datapoints to drop in each epoch to avoid overfitting).

The *ML_TimeSeries* class implements a *create_x_y* function that creates time-based datapoints usable in the LSTM and TCN models, divided into a training and test set. Depending on the chosen input lag and amount of timesteps to look ahead (in our multi-stage prediction case we always look one timestep ahead), X- and y datapoints are created. If we have three features x, y, and z and our target feature is z, we have a lag of two timesteps, and we look one timestep ahead, this results in datapoints of the following structure:

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{bmatrix} => z_3,$$

$$\begin{bmatrix} x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix} => z_4,$$

$$\begin{bmatrix} x_3 & y_3 & z_3 \\ \dots & \dots & \dots \end{bmatrix} => \dots,$$

where the amount of possible datapoints depends on the training dataset and the lag. If we have 10 timesteps of data, a lag of two, and we look one ahead, there are only 10 - 2 = 8 possible datapoints.

Once the training and test set are created, the training of the model happens inside the LSTM and TCN classes. With Keras, the layer architecture is generated and the training of the model is started. The trained model is saved as an attribute of the class and can be stored for later use once the training is done.

The multi-stage prediction is implemented in the *ML_TimeSeries* class and works by starting off with an initial prediction using the first *lag* timesteps. Next, the input time window is shifted by one timestep in the future and the just-computed prediction is incorporated into this new datapoint before the next prediction is made. In this iterative manner a multi-stage prediction is done for the full prediction period.

Features

The features I used in the greenhouse machine learning models can be divided into three categories. The greenhouse features correspond to the available sensors at Delphy, in the categories outside weather, actions in the greenhouse, and the inside climate. These features are denoted respectively with the prefixes *out_*, *act_*, and *in_*. An overview of the features used can be seen in Table 4.1.

The *out_* features together form the model's knowledge of the outside weather, which is an important influence on the inside climate. Perhaps one of the most important features is *out_rad*; the sun radiation, which has an effect on both light and temperature within the greenhouse. Next, we have outside temperature *out_temp*. Absolute humidity is contained in feature *out_humid_abs*. Wind in combination with the opening of the windows has an effect on each of the inside climate variables, and is contained in the feature *out_wind*. Lastly, no sensor for the outside CO_2 concentration is available at Delphy, but this has no big influence on the effectiveness of training the model. Gradually of course CO_2 concentrations in the air are increasing, but within the training period this concentration does not fluctuate much.

Feature	Unit	Category	Explanation
out_rad	W/m^2	outside weather	Outside radiation
out_temp	°C	outside weather	Outside temperature
out_humid_abs	g/m^3	outside weather	Outside absolute humidity
out_wind	m/s	outside weather	Outside wind speed
act_tube_under	°C	action in greenhouse	Heating from undertube
act_tube_grow	°C	action in greenhouse	Heating from growtube
act_window_lee	%	action in greenhouse	Opening window leeside
act_window_wind	%	action in greenhouse	Opening window windside
act_light_led	0/1	action in greenhouse	(dep. 3.5) Light LED on/off
act_light_sont	0/1	action in greenhouse	(dep. 3.5) Light SON-T on/off
act_light_led_1	0/1	action in greenhouse	(dep. 6) Light LED 1 on/off
act_light_led_2	0/1	action in greenhouse	(dep. 6) Light LED 2 on/off
act_screen_shading	%	action in greenhouse	(dep. 3.5) Closing shading screen
act_screen_energy	%	action in greenhouse	(dep. 6) Closing energy screen
act_screen_light	%	action in greenhouse	(dep. 6) Closing light-emission screen
act_co2	kg/ha	action in greenhouse	(dep. 6) Injecting CO_2
in_par	μmol/m^2/s	inside climate	Inside PAR light
in_temp	°C	inside climate	Inside temperature
in_hd	g/m^3	inside climate	Inside humidity deficit
in_co2	ppm	inside climate	Inside CO_2

Table 4.1: Features used in the machine learning models, including their unit and category. Some of the actions are only available in either department 3.5 or 6.

The decisions a grower can make within the greenhouse are encompassed in the *act_* features. As there are different available techniques per department, some of these features can only be used in department 3.5 and some only in department 6. Both departments have two heating tubes available; features *act_tube_under* and *act_tube_grow*, respectively the under tube and the growing tube. The under tube is a heating tube placed under the crop, while the growing tube is placed close to the crop. Quite standard in any Venlo-type greenhouse are windows, both on the wind- and leeside. The feature *act_window_wind* and *act_window_lee* represent the percentage of opening of these windows. Artificial lighting in department 3.5 includes LED lighting *act_light_led* and SON-T lighting *act_light_sont*. These are discrete features; the lights are either on or off. In department 6 only LED lighting is used, contained in discrete features *act_light_led_1* and *act_light_led_2*, which represent two strands of LED lighting. The lighting in this department can be scaled down by turning off one of these strands. In department 3.5 one shading screen *act_screen_shading* is used, which by a percentage indicates how far the screen is closed to block sunlight. Department 6 has two specialised energy-saving screens installed; an energy screen *act_screen_energy* and a light emission screen *act_screen_light*. The energy screen prevents energy in the form of heat leaving the greenhouse. The light emission screen tries to keep as much light within the greenhouse as possible, both for minimising the light disturbance outside the greenhouse and for energy saving. Lastly, department 6 has a CO_2 injection feature *act_co2*. CO_2 is injected in short peaks.

The inside climate is encompassed in four *in_* features. PAR-light is contained in feature *in_par*, and is obtained by taking the mean value of multiple available PAR-light sensors in the greenhouse. This is done to account for different light intensities throughout the greenhouse, and for moments that a part of the greenhouse framework forms a shadow directly over one of the sensors. Features *in_temp*, *in_hd*, and *in_co2* respectively represent the temperature, humidity deficit, and CO_2 concentration in the greenhouse.

Normalisation

In order for NNs to work properly, often the data is normalised. This is useful, as in many cases the different features have varying ranges. For example in our case of the greenhouse system, outside light radiation (in the Netherlands) ranges from approximately 0 to 1000 W/m^2, while the wind speed (excluding heavy storms) has a much smaller range: 0 to 30 m/s^2. Larger numbers have a larger impact on the predictions in NNs, resulting in the prediction being more dependent on light than wind speed, which may not necessarily be true. Therefore I scaled all features between 0 and 1, so that each feature is equally important in the eyes of the model.

Regularisation

In NNs, often some form of regularisation is used to prevent overfitting such that the model still performs well on unseen data. In the explicit form, this means adding an additional term to the loss function, to penalise large weights. I tried this form of regularisation using L1 and L2 regularisation functions, but this did not improve the performance of the models. I did use early stopping, which is an implicit type of regularisation. With early stopping, the training is stopped when the loss on the validation set does not change anymore. Lastly, including more training data could help prevent overfitting, but since there are such different cultivations in each of the greenhouse departments of Delphy, this was not feasible.

4.1.4. Hyper-parameter tuning

Choosing the right hyper-parameters can improve the model predictive power greatly, or reduce unnecessary training time. However, there does not exist an exact rule for choosing and it is mostly a process of trial-and-error. To find good hyper-parameters, I performed a grid search around values that were obtained by an initial manual trial-and-error process. Because I used early stopping, the amount of epochs was not taken into account in the grid search, but a maximum of 1500 epochs was used. I performed the grid search for the LSTM model with the following set of hyper-parameters:

batch_size:	256, 512, 1024
lag:	1, 2, 3, 6, 12
units:	4, 8, 16, 32

For TCN, other hyper-parameters are relevant. Those used in the grid search are shown below:

batch_size:	256, 512, 1024
lag:	1, 2, 3, 6, 12
nb_filters:	32, 64, 128
dilations:	[1, 2, 4], [1, 2, 4, 8], [1, 2, 4, 8, 16]
kernel_size:	2, 3, 4, 5

I trained both types of models with the adam optimizer and I used the MSE loss function. A validation_split of 0.1 was used, 8% was used as test_set, and a patience of 20 was used for the early stopping mechanism. I measured the performance of each model by the RMSE of the actual and predicted values. For each combination of hyper-parameters, I ran an experiment 5 times and averaged the result, to account for randomness. For department 3.5, the training period was 1/12/2020 to 10/9/2021 and the prediction period was 15/9/2021 to 17/9/2021. For department 6, the training period was 25/9/2021 to 14/2/2022 and the prediction period was 15/2/2022 to 17/2/2022.

LSTM, department 3.5 | In greenhouse department 3.5, the best performing LSTM model is trained with a batch size of 512, a lag of 6 timesteps (6 hours), and 4 LSTM units, with a resulting average RMSE of 0.0366.

LSTM, department 6 | For department 6, the hyper-parameters resulting in the best performing LSTM model were a batch size of 256, a lag of 6 timesteps (6 hours), and 4 LSTM units, with a resulting average RMSE of 0.0279.

TCN, department 3.5 | The TCN model worked best in department 3.5 with a batch size of 256, lag of 6 timesteps (6 hours), [1, 2, 4, 8] as dilations, an nb_filter of 128, and a kernel size of 5, resulting in an average RMSE of 0.0433.

TCN, department 6 | In department 6, the TCN model worked the best with the same hyper-parameters as department 3.5. So a batch size of 256, lag of 6 timesteps (6 hours), [1, 2, 4, 8] as dilations, an nb_filter of 128, and a kernel size of 5, which resulted in an average RMSE of 0.03.

4.1.5. Concluding remarks

Although both models gave promising results, only the LSTM was used in the CP embedding. There was no decisive reason to choose one over the other, but the LSTM resulted in slightly better RMSE values in the grid search and at a first glance the LSTM-embedding seemed more straightforward to implement than that of the TCN model.

4.2. Model Embedding in CP

Training an NN results in a set of weights and biases that can be used to imperatively make predictions, but CP requires a declarative approach. In other words, NNs map specified inputs to outputs, while in CP we do not know what is the input and what is the output. To leverage the trained NN in a CP model, it must be reduced to a declarative mathematical form that can handle decision variables.

Learning relations from data and embedding these into an optimisation model, i.e. Empirical Model Learning (EML), is already being applied with regular NNs with promising results [42–44]. However, it appears that no embedding of LSTMs in CP has been done, while the successes of both LSTMs and CP could prove to be a powerful combination. Still, the added complexity of LSTMs could make it hard to be useful in a CP setting. An important aspect of this study is a proof-of-concept of such an LSTM-in-CP embedding. The embedding will be laid out in this section.

4.2.1. Functions & Predicates

To make the LSTM calculations possible, I implemented various functions and predicates. At the core of the LSTM cell, matrix operations are performed. And within multiple parts of the LSTM network, activation functions are used. The implementation of this functionality, as well as the complete LSTM calculation, is laid out in this subsection through MiniZinc functions and predicates.

For each step of the LSTM computation I built a separate function. At its core are the LSTM gate and state functions. To reiterate, the gates and states are updated as follows:

$$i_t = \sigma_r(W_i \cdot X_t + U_i \cdot h_{t-1} + b_i) \tag{4.2}$$
$$f_t = \sigma_r(W_f \cdot X_t + U_f \cdot h_{t-1} + b_f) \tag{4.3}$$
$$c'_t = \sigma_a(W_c \cdot X_t + U_c \cdot h_{t-1} + b_c) \tag{4.4}$$
$$o_t = \sigma_r(W_o \cdot X_t + U_o \cdot h_{t-1} + b_o) \tag{4.5}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot c'_t \tag{4.6}$$
$$h_t = o_t \odot \sigma_a(c_t), \tag{4.7}$$

with input gate i, forget gate f, output gate o, cell state c (candidate cell state c'), hidden state h, activation σ_a for the states and recurrent activation σ_r for the gates, where $\odot$ represents an element-wise multiplication. From these equations it is clear that the CP equivalent of the elementwise-multiplication (see Listing 4.1), dot-product, and elementwise-addition of matrices is needed. I implemented these matrix operations through MiniZinc functions and validated them with small matrix instances.

```
function array[int, int] of var float: matrix_multiplication(
    array[int, int] of var float: A,
    array[int, int] of var float: B
) =
    assert(
        % check matrix shapes of A and B
        cols(A) == rows(B),

        % if needed give error message
        "Matrix shapes do not match: " ++
        "(" ++ show(rows(A)) ++ "," ++ show(cols(A)) ++ ") * " ++
        "(" ++ show(rows(B)) ++ "," ++ show(cols(B)) ++ "); " ++
        show(cols(A)) ++ " != " ++ show(rows(B)),

        % else do matrix multiplication
        array2d( index_set_1of2(A), index_set_2of2(B),
        [ sum( k in index_set_2of2(A)) (A[i,k] * B[k,j])
        | i in index_set_1of2(A), j in index_set_2of2(B) ]
        )
    );
```

Listing 4.1: Example of matrix operation in MiniZinc. An assertion on the dimensions of the matrices is done before returning a 2D array object that is the result of the elementwise-multiplication of matrix A and B.

Additionally, some utility functions were implemented to handle the scaling of features, as well as capping values between a minimum and maximum value. This capping of values is used in the linear activation function to make sure that the resulting values do not exceed the domain. In a regular LSTM computation an output value can exceed the normalised domain, but within CP this leads to the problem being unsatisfiable. The activation functions that were implemented are *sigmoid*, *tanh*, and *linear*. The implemented cap-function was also used within the sigmoid and tanh functions, by capping the input values of these functions to the ranges -10.0 to 10.0 and -5.0 to 5.0 respectively. Not capping these input values resulted in MiniZinc crashing, without a clear explanation. Since we know the limits of both the sigmoid and tanh function, and their corresponding domains, we can simplify these functions by first capping the input values. This indeed eliminated the crashes. As an example, the implementation of the sigmoid function is given in Listing 4.2.

```
function var 0.0..1.0: sigmoid(
    var float: x
) =
    1 / (1 + exp(-cap(x, -10.0, 10.0)));
```

Listing 4.2: Example of activation function in MiniZinc. The input value x is capped between -10 and 10, before applying the sigmoid function to prevent MiniZinc from crashing and because we know the limits of the sigmoid function.

Next, multiple functions were implemented that represent the LSTM cell, of which a simplified pseudocode overview can be seen in Algorithm 1. Please see Appendix B for the complete MiniZinc code of the LSTM. I implemented a general gate calculation function that handles the matrix operations and activation functions for the gates as seen in Equation 4.2, 4.3, 4.4, and 4.5. I also implemented MiniZinc functions for the computation of the cell and hidden state. These two functions thus include multiple calls to the gate function. Lastly we need to do these gate and state calculations for multiple timesteps. With a time lag l, we need to update the cell and hidden state for each timestep $t \in l$. The cell and hidden state are vectors with a size of the amount of chosen LSTM units. At $t = 0$, these vectors c_0 and h_0 are initialised with zeros. For each t in l, vectors c_t and h_t are computed by using the vectors of the previous timestep c_{t-1} and h_{h-1} in the before-mentioned LSTM functions. Important to note here is that in a CP approach, we cannot simply create vectors c and h and update these. We need to create vectors for each timestep, such that the solver can find each of the intermediate values. Just as in a regular LSTM, the hidden state vector of the last timestep represents the output of the LSTM.

Algorithm 1 Pseudocode of the LSTM calculations in MiniZinc. The cell and hidden state are initialised to zero and for each timestep in the range 1 to lag, the cell and hidden state are calculated in separate functions. The gate calculation is generalised in a function as well.

```
1:  function LSTM($X, W, U, b$)
2:      $c_0 = 0$
3:      $h_0 = 0$
4:      for t ∈ 1..l
5:          $c_t$ = CELL($X_t, h_{t-1}, c_{t-1}, W, U, b$)
6:          $h_t$ = HIDDEN($X_t, h_{t-1}, c_t, W, U, b$)
7:      return $h_l$
8:
9:  function CELL($X_t, h_{t-1}, c_{t-1}, W, U, b$)
10:     return
11:     $GATE(X_t, h_{t-1}, W_f, U_f, b_f, \sigma_r) \odot (c_{t-1}) +$
12:     $GATE(X_t, h_{t-1}, W_i, U_i, b_i, \sigma_r) \odot GATE(X_t, h_{t-1}, W_c, U_c, b_c, \sigma_a)$
13:
14: function HIDDEN($X_t, h_{t-1}, c_t, W, U, b$)
15:     return $GATE(X_t, h_{t-1}, W_o, U_o, b_o, \sigma_r) \odot \sigma_a(c_t)$
16:
17: function GATE($X_t, h_{t-1}, W_g, U_g, b_g, \sigma$)
18:     return $\sigma(W_g \cdot X_t + U_g \cdot h_{t-1} + b_g)$
```

Since we want to be able to change the unit size and thus complexitiy in the LSTM, we need an additional dense layer that maps the LSTM output to the desired dimensionality of our model output, in our specific case the four inside climate targets. This dense layer was also implemented using MiniZinc functions. The output of our LSTM layer is a vector h_l of LSTM unit size u. The computation of the dense layer is then $\sigma_d(h_l \cdot W_d + b_d)$, where σ_d is the linear capped function, W_d are the weights of the dense layer and b_d the bias.

4.2.2. Greenhouse and Crop Model Embedding

The implementation of the CP embedding is done through a model class hierarchy, visualised in Figure 4.2, to improve the readability and modularity of the code. In this way, additional greenhouse and crop models can be added more easily. At the top level, we have the abstract class *Model*, which provides shared functionalities of the *Greenhouse_Model* and *Crop_Model* abstract classes. At the bottom layer we have the specific greenhouse and crop model implementations. For example, the greenhouse climate prediction with the LSTM is encompassed in the *Greenhouse_Model_LSTM* class, which provides the MiniZinc embedding of the model.

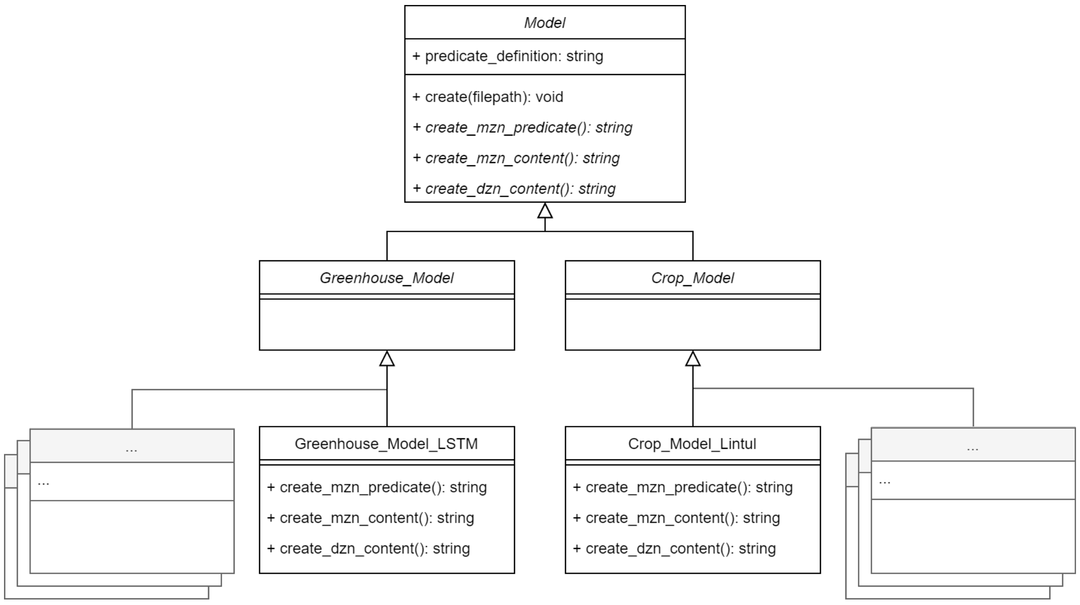

Model
+ predicate_definition: string
+ create(filepath): void
+ create_mzn_predicate(): string
+ create_mzn_content(): string
+ create_dzn_content(): string
Greenhouse_Model
Crop_Model
...
...
Greenhouse_Model_LSTM
+ create_mzn_predicate(): string
+ create_mzn_content(): string
+ create_dzn_content(): string
Crop_Model_Lintul
+ create_mzn_predicate(): string
+ create_mzn_content(): string
+ create_dzn_content(): string
...
...

Figure 4.2: Class diagram of the abstract greenhouse and crop model classes. As visualised in the diagram, the DSS is extensible with other greenhouse and crop models.

At the top level in the abstract class *Model*, the function *create(filepath)* performs the model embedding and writes the output to the given filepath. As explained in chapter 3, the embedded model is written to a separate MiniZinc file and is callable through a single predicate. This predicate is defined in both the *Greenhouse_Model* and *Crop_Model* classes, as can be seen in Listing 4.3 and Listing 4.4 respectively. The predicate definitions enforce the implementing models to use the given inputs. For example, an implementing greenhouse model must use the 2D array *greenhouse_vars* to model the greenhouse climate. This array is the combination of out_vars, act_vars, and in_vars. In *Model*, the MiniZinc model is generated by the build-up of a string with the model content (helper functions and var definitions), and the predicate definition and content.

```
predicate greenhouse_model(
    array[int,int] of var float: greenhouse_vars
) =
```

Listing 4.3: MiniZinc predicate definition of the greenhouse model.

```
predicate crop_model(
    array[int] of var float: in_temp,
    array[int] of var float: in_par,
    array[int] of var float: in_hd,
    array[int] of var float: in_co2,
    var float: crop_growth
) =
```

Listing 4.4: MiniZinc predicate definition of the crop model.

The function *create_dzn_content()* is used to create data content that will be appended to the MiniZinc data file. While the intended usage of the MiniZinc data file is to separate the model and an instantiation of this model with different parameters, the model and data generated in our framework are so intertwined and already separated through the Python instance of the model, that the MiniZinc data file is only used to not clutter the model with large chunks of data such as the weights and biases of the LSTM model.

Since the models make use of some of the earlier mentioned functions and predicates like matrix multiplications and LSTM calculations, a folder with these so-called *helper mzn files* is written to the given filepath along with the generated models.

Greenhouse LSTM Model in CP

The specific greenhouse embedding that I implemented and used in the complete CP model, is the *Greenhouse_Model_LSTM*. It takes as input an LSTM object that was generated in stage 3 of the pipeline and generates an LSTM embedding in MiniZinc. I created an extra *LSTM_to_MiniZinc* that provides a more general LSTM embedding functionality. Here the kernel weights W_i, W_f, W_c, W_o, recurrent kernel weights U_i, U_f, U_c, U_o, and biases b_i, b_f, b_c, b_o of the LSTM are loaded, as well as the weights and biases of the accompanying dense layer W_d and b_d. The relevant *helper mzn files* are referenced and the arrays holding the weights and biases are defined in an mzn string. The values of these weights and biases are appended to a dzn string. The mzn and dzn strings are returned by this *LSTM_to_MiniZinc* object.

In the *Greenhouse_Model_LSTM* class, these mzn and dzn parts are supplemented with the predicate content that actually connects the predicate input to the LSTM output. The LSTM and dense functions are called with the predicate input and above-mentioned weights and inputs. Since this is a multi-stage prediction, within this predicate the output of the LSTM for each batch of inputs is connected to the input of the next timestep. The resulting *greenhouse_model* predicate can be seen in Listing 4.5.

```
predicate greenhouse_model(
    array[int,int] of var float: greenhouse_vars
) =

  % loop over all to-be-predicted timesteps
  forall(batch in 1..rows(greenhouse_vars) - 1) (
    let {
        % perform LSTM and Dense layer calculations
        array[1..1, 1..m] of var 0.0..1.0: result =
            dense_forward(transpose(
                lstm_forward(
                    array2d(1..1, 1..n, [ greenhouse_vars[t, feature]
                            | t in batch..batch + l - 1, feature in 1..n ]
                    ),
                    Wi, Ui, bi,
                    Wf, Uf, bf,
                    Wc, Uc, bc,
                    Wo, Uo, bo,
                    n, m, u, l)
                ),
            Wd, bd, "linear_capped"
            );
    } in
            % place outputs in input array
            forall(target in 1..m_1) (greenhouse_vars[batch + 1, n - m+target] =
                custom_round(result[1, target], 2)
                ::defines_var(greenhouse_vars[batch + 1, n - m + target]))
            );
```

Listing 4.5: MiniZinc predicate of the greenhouse LSTM model.

During evaluation of the LSTM models, it became clear that the light prediction was not sufficient and could be improved by a more straightforward calculation. I incorporated this calculation in a second greenhouse model. Here the the other climate variables are computed the same as in Listing 4.5, but light is computed by using the outside radiation, the shading screen including transparancy of the screen that is usually 0, and the artificial lighting with light intensities. This implementation is shown in Listing 4.6. Currently it only works for departments with one screen, so it is not possible to use this functionality for department 6.

Crop Lintul-3 Model in CP

The crop model that I implemented is the Lintul-3 model [50]. This model is already used at Delphy and was selected and implemented in [4]. It was later extended in [5] by specifically looking at the light element of the calculations. After some final adjustments by some of the colleagues at Delphy of the parameters used in this model, I implemented the Lintul-3 model embedding. It calculates the fresh-weight crop growth in grams by combining a temperature, CO_2, and light component. This model is quite basic and short-term based, but it demonstrates the usage of a crop model and could be extended or replaced by Delphy in the future.

```
function var float: par_calculation(
    var float: out_radiation,
    array[int] of var float: lights,
    array[int] of float: lights_intensities,
    var float: screen,
    float: screen_transparancy
    ) =
        % outside radiation to PAR
        (1.0 - screen) * out_radiation * 2.0 * 0.6 +
        % shaded part
        screen * screen_transparancy * out_radiation * 2.0 * 0.6 +
        % artificial lighting
        sum(i in index_set(lights)) (lights[i] * lights_intensities[i]);
```

Listing 4.6: Simpler calculation for PAR-light inside the greenhouse, to replace the LSTM prediction. The outside radiation is converted to PAR (*2), while accounting for glass transmissivity (*0.6), omitting the shaded part by the shading screen, and adding artificial lighting PAR.

4.2.3. Key points LSTM-CP embedding

Rebuilding the LSTM within a CP setting has been the main challenge in this thesis, and various key points in CP modelling contributed to its realisation. Especially the time-based structure of the LSTM required a specific approach. These key points and general lessons learned are explained in this section.

Firstly, the general way of modelling decision variables and relations within CP, is by defining them and comprise their dependence on other variables through constraints. Say we have variable a that is dependent on variable b in the sense that $b = 2 \cdot a$. A general way of modelling this in MiniZinc would be the following:

```
var int: a;
var int: b;

constraint b = 2 * a;
```

Listing 4.7: General approach of modelling decision variables in MiniZinc.

From this model, we see that b is functionally dependent on a. While for small models, the MiniZinc compiler is able to derive this functional dependence and annotate variable b accordingly, in larger models this is not always the case. Therefore we can help the solver by explicitly stating this functional dependence by directly instantiating variable b as two times a as follows:

```
var int: a;
var int: b = 2 * a;
```

Listing 4.8: Directly instantiating a decision variable in MiniZinc with a functional dependency.

This may seem trivial and unimportant, but in the case of a more complex model such as the LSTM, this direct instantiation ensures that the solver simply calculates variable b, and does not see it as a decision variable to take into the search. In the background, the MiniZinc compiler adds the annotation pair *is_defined_var* and *defines_var()*, which together indicate that a variable is functionally dependent on another, and should not be decided. In our specific case, it means that we can specifically tell the solver to only search on greenhouse action variables, all the other variables should simply be the result of the LSTM calculations. For the inside climate variables, the before-mentioned annotation pair was explicitly added, because the variables are defined in our complete model and sequentially computed in our greenhouse LSTM model. The *defines_var()* annotation of the inside climate variables can be seen in Listing 4.5. For small models, i.e. having a small prediction time window or few LSTM units, MiniZinc was able to arrive at a solution without using these direct instantiations. For larger models however, MiniZinc would crash as a result of the vast amount of decision variables. Using direct instantiations mitigated these crashes.

This advantage of direct instantiations was passed on in the choice of functions over predicates. Where predicates take in variables and constrain these, functions directly return a result. In the case of computing the gate functions in the LSTM for example, a function is preferred as this entails a direct computation instead of posing a constraint on a variable, where the solver would try to find values that satisfy the constraint. Also, an additional variable would be needed for such a predicate, needlessly making the model more complex. The same line of thought applies to not creating additional variables to hold the results of functions, but rather chain these function calls as much as possible. Chaining the functions not only reduced the amount of crashes as the problem size increased, it also improved the compiling and solving speeds greatly.

Most LSTM computations were done in compliance with this direct instantiation strategy, but for the cell and hidden state this was not possible, as their structure is dependent on the chosen lag. As mentioned earlier, the cell and hidden state vectors can not be updated directly, but are encompassed in separate vectors for each timestep. To programmatically make this time-based computation work, both the cell and hidden state vectors were defined in 2D arrays, with timesteps as rows, and the chosen LSTM unit size u as columns, corresponding to vectors of length u for each timestep. In an iterative constraint, the values of the vectors are decided by a calculation that includes values of the vectors of the previous timestep.

Because the solver needs to find values for these vectors, it helps to explicitly state bounds on these, as tight as possible. This is also recommended in MiniZinc's documentation [39]. We know that the hidden state in an LSTM holds values between -1.0 and 1.0, because of the tanh activation function. The bounds of the cell state are not that straightforward. In Equation 4.6, we see that c_t depends on f_t, c_{t-1}, i_t, and c'_t. The bounds for f_t and i_t are (0, 1), as a result of the sigmoid function, similarly c'_t has bounds of (-1, 1) due to the tanh function. Taking the cell state calculations in mind, the bounds for the cell state increase at each timestep. The resulting bounds thus correspond to the lag l, so $(-l, l)$. Deriving these bounds is more clearly visualised in Table 4.2:

	f_t	$\odot$	c_{t-1}	$+$	i_t	$\odot$	c'_t	$=$	c_t
t_1	$(0,1)$		$(0,0)$		$(0,1)$		$(-1,1)$		$(-1,1)$
t_2	$(0,1)$		$(-1,1)$		$(0,1)$		$(-1,1)$		$(-2,2)$
t_3	$(0,1)$		$(-2,2)$		$(0,1)$		$(-1,1)$		$(-3,3)$
...	$(0,1)$		...		$(0,1)$		$(-1,1)$		...
t_l	$(0,1)$		$(-(l-1),(l-1))$		$(0,1)$		$(-1,1)$		$(-l,l)$

Table 4.2: Bounds calculation of the cell state, showing the dependence on the chosen lag. Initialised with zeroes, the cell state bounds are propagated for each timestep up to *lag* timesteps.

4.3. Economic Greenhouse Decision Support

The greenhouse and crop model, their embedding into CP, and the CP model itself ultimately result in a complete economic greenhouse DSS. This system models the complete greenhouse environment and optimises the decisions based on the economic aspect of a tomato cultivation. How this complete system operates and how the various components interact is visualised in Figure 4.3. The parameters and variables in the system are denoted by the letters O, A, I, C, and E, respectively representing the outside weather, greenhouse actions, inside climate, crop production, and economics. As can be seen in the figure, some of these components are simply retrieved, others are predicted through the embedded models.

These parameters and decision variables each have a different purpose within the CP model, and should thus be modelled differently. Also, the relations between the variables and their constraints need to be modelled. In this section, the complete model implementation including the connection between the greenhouse and crop model is laid out, as well as the assumptions made for the costs accompanied with the greenhouse actions and the revenue following the tomato yield.

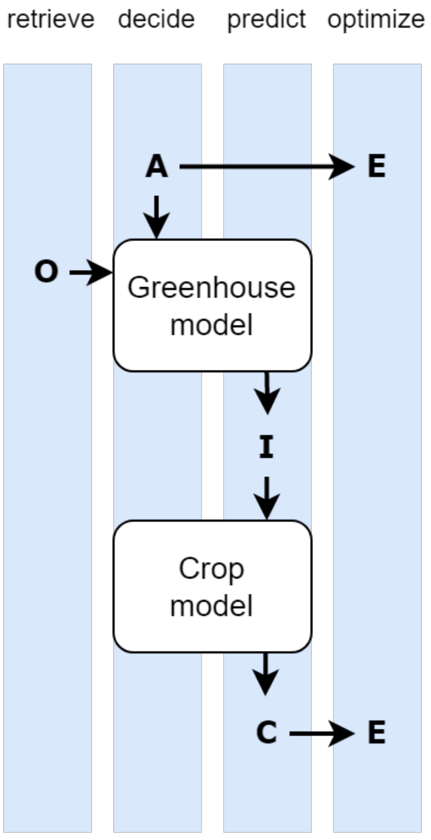


Figure 4.3: Overview of the CP model and its relevant parameters and decision variables. The outside weather (**O**) is retrieved from a weather forecast, the greenhouse actions (**A**) must be decided, so that the inside climate (**I**) and crop production (**C**) can be predicted through the embedded models, finally leading to an economic result (**E**) that is used for the optimisation process.

4.3.1. MiniZinc model

I built the complete CP model in MiniZinc and connected it to the greenhouse and crop model. In the previous section, we saw how the greenhouse LSTM model and Lintul crop model were embedded into a MiniZinc CP model through functions and predicates. To use these models, we need to model their inputs. Before modelling the variables, I initialised the amount of timesteps that follow from the chosen prediction period, which can be seen in Listing 4.9. One extra timestep is added, such that the decided actions of the last timestep are reasonable for the timestep that falls just outside of the chosen period. I modelled each of the three types of greenhouse variables (*out*, *act*, and *in*) differently.

```
int: timesteps = 25;
set of int: TIMESTEPS = 1..timesteps;
```

Listing 4.9: Initialisation of timesteps that follow from the chosen period.

First, I modelled the outside weather by arrays with a length of the amount of timesteps, with normalised parameters (bounds of (0, 1)). They are parameters, because to the DSS they are known values. They are retrieved from forecast data and will not be predicted or decided within the system. The values of the parameters are stored in the data file.

```
array[TIMESTEPS] of 0.0..1.0: out_rad_known;
```

Listing 4.10: Example of outside weather parameter array, in this example radiation.

Next, I modelled the actions in the greenhouse by a known and unknown part. The amount of known timesteps T_a for the actions is given as input, and we split the variable arrays on this timestep. This T_a is at least the chosen lag of the LSTM model, but can be larger. The LSTM model needs these first "lag" inputs to start the multi-stage prediction process. The known part is a parameter array with normalised real numbers that represent the first T_a actions that are already decided, the unknown part is a variable array with integers that represent the decisions that the system will make. We discretise using integers, to reduce the amount of possible values, improving the efficiency of the solver. There is a trade-off however, between efficiency and precision of the result. The precision is an input in the model, indicating how many possible values a greenhouse action decision variable can have. I use a base of 10 for the floating point operations to derive the integer values which is especially useful for the inside climate variables, as will be discussed later in this section. The bounds for the unknown part of one action variable are then $(0, 10^p)$, where p is the precision. Throughout the rest of this thesis, a precision of 1 is used for the action variables, to limit the search space of the DSS.

```
array[1..6] of 0.0..1.0: act_tube_under_known;
array[6..timesteps] of var 0..10: act_tube_under_unknown_int;
```

Listing 4.11: Example of greenhouse action parameter and variable arrays, in this example the heating undertube. A precision of 1 is used, leading to the bounds $(0, 10^1)$

Finally, I modelled the inside climate similarly by a known and unknown part. The known part is already normalised and is a parameter array with a length of the chosen lag of the LSTM model. The unkown part consists of a variable array with normalised real values for the remaining timesteps, accompanied by an array of the same length with integer values. The predicted values are rounded within the LSTM prediction process, as can be seen in Listing 4.5. By rounding the inside climate values within the LSTM, the solver is able to match the real numbers to their integer counterparts. To illustrate, consider the following example: Say for $t = 7$, the LSTM outputs 0.312 for *in_temp*, which is a normalised value corresponding to 17.8°C using the bounds (10, 35). If we use a precision of 2, this value will be rounded to 0.31. Since the integer counterpart of the variable has the bounds $(0, 10^2)$, it will take on the value 31. So instead of the full range of possible real numbers within (0, 1), the set of possible values now only consists of the integer values in (0, 100). Using integer values to match the predictions, improved the solving time substantially, because there are less possible prediction outputs. Throughout the rest of this thesis, a precision of 2 is used for the inside climate variables, to simplify the calculations but still give a reasonable result.

```
array[1..6] of 0.0..1.0: in_temp_known;
array[7..timesteps] of var 0.0..1.0: in_temp_unknown :: is_defined_var;
array[7..timesteps] of var 0..100: in_temp_unknown_int;
constraint forall(t in 7..timesteps) (
    in_temp_unknown[t] = scale(
        in_temp_unknown_int[t],
        0,   100,
        0.0, 1.0
    )
);
```

Listing 4.12: Example of inside climate parameter and variable arrays, in this example temperature. A precision of 2 is used, leading to the bounds $(0, 10^2)$

These outside, action, and inside arrays all come together in the greenhouse model input 2D array. Through a direct instantiation of this 2D array, for each timestep either the known or unknown part of the arrays is used. I normalise the unknown action variable arrays by scaling from $(0, 10^p)$ to (0, 1). As we saw earlier, the inside climate variables are already normalised.

```
array[1..timesteps, 1..15] of var 0.0..1.0: X_greenhouse = array2d(1..timesteps, 1..15, [[

    out_rad_known[t],
    [...], % other out_vars

    if t <= 6 then act_tube_under_known[t]
    else scale(act_tube_under_unknown_int[t], 0, 10, 0.0, 1.0) endif,
    [...], % other act_vars

    if t <= 6 then in_temp_known[t]
    else in_temp_unknown[t] endif,
    [...], % other in_vars

][feature] | t in TIMESTEPS, feature in 1..15]);
```

Listing 4.13: 2D input array for the greenhouse LSTM prediction, including outside weather parameters, greenhouse action decision variables, and inside climate variables. For readability, only one of each type of array (out, act, in) is shown.

Elsewhere in the CP model, I scaled the greenhouse action and inside climate variables back to their original bounds so that they could be used to derive the action costs and the crop production, as the prices and the Lintul-3 crop model are not normalised.

Additional constraints

The power of CP lies in posing constraints on the variables that limit the search space. Since the focus of this thesis was on building a framework, embedding an LSTM into CP, and studying the applicability of CP within a greenhouse DSS, not many additional constraints were added that could improve the search. Nevertheless I implemented a constraint that constrains variables consecutive in time to be similar. This was done in constraint *difference_leq*, which takes two variables, a maximum difference, and makes sure these consecutive variables have values within this range.

Another constraint that was desired by Delphy, is the dark period in which lights are turned off. For this, I added a parameter that indicates how many timesteps after midnight timestep 1 is and, depending on the chosen dark period, with this I was able to turn off the lights for timesteps that fall within this dark period.

More constraints could be added to limit the search space even further. For example, not turning on the lights when the outside radiation is high enough. Or turning off the heating when the inside temperature reaches a known maximum value. However, these constraints are very specific to the strategy of the grower and precisely the DSS should aid in making these strategic decisions.

Search strategy

One of the key points in building the CP model, was the implementation of a search strategy. In a search strategy, a CP modeller can indicate how the solver should perform the search for feasible and optimal solutions. This involves both the order of deciding variables and what values these variables should be initialised with.

The order of searching is essential in solving a CP model that involves time-based decision variables, as is the case with our multi-stage prediction with an LSTM. Since the inside climate of a certain timestep depends on a multitude of variables from previous timesteps, it helps the solver to know which variables to decide first, before moving on to the next. I implemented this order of searching using MiniZinc's *seq_search* annotation, in which I placed a list of *int_search* annotations, each including all greenhouse action variables for one timestep (see Listing 4.14). The solver thus first searches values for the decision variables of the first unknown timestep, before searching values for decision variables of the next timestep.

```
solve
    :: seq_search([

        % search actions first timestep after "lag"
        int_search([
            act_light_led_unknown_int[7],
            act_light_sont_unknown_int[7],
            act_tube_under_unknown_int[7],
            act_tube_grow_unknown_int[7],
            act_screen_shading_unknown_int[7],
            act_window_lee_unknown_int[7],
            act_window_wind_unknown_int[7],
        ], first_fail, indomain_random),

        ...,

        % search actions of last timestep
        int_search([
            act_light_led_unknown_int[100],
            act_light_sont_unknown_int[100],
            act_tube_under_unknown_int[100],
            act_tube_grow_unknown_int[100],
            act_screen_shading_unknown_int[100],
            act_window_lee_unknown_int[100],
            act_window_wind_unknown_int[100],
        ], first_fail, indomain_random),

    ])
    ::restart_constant(5)
maximize econ_profit;
```

Listing 4.14: Sequential search strategy, in which all greenhouse action variables are searched sequentially, starting with one timestep after "lag", ending with the last timestep. For each timestep, the search strategy includes an *int_search* annotation, but for readability the intermediate annotations are omitted in this listing.

Next to sequential search, I also implemented priority search [41], which provides a shorter and more clear search strategy. Essentially, this search annotation performs the same search as the shown sequential search in Listing 4.14, but more compactly modelled. Not all solvers implement priority search, so both implementations of sequential search and priority search are available in the DSS.

```
solve
  :: priority_search(

      % for each timestep
      [ t | t in 7..timesteps ],
      [
          % decide actions
          int_search([
            act_light_led_unknown_int[t],
            act_light_sont_unknown_int[t],
            act_tube_under_unknown_int[t],
            act_tube_grow_unknown_int[t],
            act_screen_shading_unknown_int[t],
            act_window_lee_unknown_int[t],
            act_window_wind_unknown_int[t],
          ], first_fail, indomain_random)

      | t in 7..timesteps ],

      % in ascending order
      smallest

    )
    ::restart_constant(5)
maximize econ_profit;
```

Listing 4.15: Priority search strategy, that uses a variable selection array of t in (7, timesteps), i.e. the unknown timesteps, in ascending order to traverse the list of greenhouse action integer search annotations.

What can be seen in both Listing 4.14 and Listing 4.15, is that a restart annotation was used. While traversing the search tree, the solver can get stuck in a branch from which no better solutions can be found in reasonable time. Therefore, a restart of the search may help in finding better solutions more quickly. One condition to using a restart annotation, is that some form of randomisation must be used in the search, otherwise the restarted search will be the same as the previous search [39]. This randomisation is encompassed in the constrain choice *indomain_random* that I placed in the *int_search* annotation. This constrain choice assigns the variables a random value from their domain during the search. If a restart is initiated, the solver returns to the top of the search tree, this time initialising the decision variables with other random values within their domains. No elaborate experiments were done to choose the type of restart annotation, but the chosen annotation *restart_constant(5)* seemed to work well.

4.3.2. Economics

While the specifics of the economic side of the DSS are not the focus of this study, reasonable assumptions of the costs and prices for operating the greenhouse need to be made. The costs are related to the actions taken in the greenhouse, while the revenue is simply a connection of the predicted crop production with a tomato price. The investment costs are not taken into account in this study.

In a real greenhouse cultivation, the tomato prices are not fixed and the gas and electricity prices can also fluctuate. This has become especially clear during the time of writing this thesis, where gas and electricity prices have risen enormously. However, for simplicity the costs and revenues associated with the variables in the system are the same for every timestep. Still, reasonable assumptions are made.

Also, many growers have a combined heat and power (CHP) unit that is used to power various equipment in the greenhouse, and to generate heat. It also serves as a CO_2 supply, as that is a byproduct of the unit. Again for simplicity, this will not be taken into account and it is assumed that each cost item is treated separately.

To make the economic aspect of the greenhouse DSS independent of the size of the greenhouse, each of the costs and prices are normalised such that they relate to one m^2 greenhouse area, e.g. kg/m^2 CO_2 injection or kg/m^2 fruit production. Also, we have to take into account the size of the timestep we use in the problem instance. Generally, hourly data is used, but in the case of 5-minute data we have to multiply by 5/60 minutes. Therefore, where applicable a factor of $f/60$ is added, where f is the timestep frequency in minutes. Below the total costs for each action and revenue for crop production are displayed, where T is the total amount of timesteps in the problem instance.

- **Heating tubes**
  The energy needed to heat the greenhouse is not solely dependent on the temperature setting of the heating tubes, but also depends on the diameter and thus water throughput of the tubes. Following an example from [51], we assume a water debit of 54 $m^3/ha/h$, which means 5.4 Liters of water need to be heated per m^2 greenhouse area every hour. We are interested in the amount of natural gas needed to

heat this water. The power P needed for and cost C of operating one heating tube for one timestep are then calculated as follows:

$$P = m \cdot c \cdot \Delta Tmp \cdot \frac{1}{3600} \tag{4.8}$$

$$C = P \cdot \frac{3.6}{r_V} \cdot p, \tag{4.9}$$

where m is the mass of the medium in kg to be heated, c the specific heat capacity of this medium in kJ/kg/K, ΔTmp the difference between the desired temperature and current temperature in K or °C, 3.6 MJ the energy of 1 kWh, r_V the heating value of natural gas in MJ/m^3, and p the price of one m^3 natural gas.

The tube system circulates the water that needs to be heated. When a certain temperature of the water in the tube is realised, this water will go through the tubes in the greenhouse, cool off depending on the air temperature, and finally return to the beginning point where it will be heated again. It is thus important to know how much the water has cooled off so that we know how much heat needs to be added to the water again. While this cooling down depends on a lot of factors such as the set-up of tubes in the greenhouse, we will make an assumption. Since in [51] a difference of tube temperature 50°C and air temperature 20°C led to a decrease of 10°C, the factor we need is (50-20)/10 = 3. We thus make the assumption that the water needs to be heated by the following amount:

$$\Delta Tmp = (Tmp_{tube} - Tmp_{air}) \cdot \frac{1}{3}, \tag{4.10}$$

where Tmp_{tube} is the temperature of the tube and Tmp_{air} is the greenhouse air temperature.

The specific heat capacity c of water is 4.18 kJ/kg/K, the heating value r_V of natural gas is 35.17 MJ/m^3. Putting it all together, without the possibility of negative heating and again accounting for the timestep frequency, the cost C_{tube} of operating a heating tube becomes:

$$C_{tube} = \sum_{t \in T} min(0, act\_tube_t - in\_temp_t) \cdot \frac{f}{60} \cdot 2.14 \cdot 10^{-4} \cdot p, \tag{4.11}$$

where act_tube_t is the decision variable indicating the tube temperature and in_temp indicates the air temperature.

Gas prices in horticulture are made out of the commodity price and some extra service costs. As an example we can look at the prices of 2019 in [52]. There we have a commodity price of €0.2/m^3 and additional costs like energy tax and handling fees that sum up to €0.0359/m^3. For 2019 we can thus assume a gas price of €0.24/m^3. At the time of writing this thesis, gas prices have risen enormously due to various developments in politics and economics, which becomes visible in the gas prices of households which have risen from €0.65/m^3 last year to about €2.66 per m^3[53]. Of course, there is a difference between gas prices for households and greenhouse growers, but it shows the substantial increase. The effects of these price changes on the decision making process will be evaluated in section 5.3. As an example, with a price of €0.24/m^3, a desired tube temperature of 60°C, and an inside temperature of 20°C, the cost for one tubing system would be €0.002/m^2/h.

- **Artificial Lighting**
  The cost of operating artificial lighting depends of course on the type of lighting. Each lighting installation has a certain light intensity, which can be translated to needed electricity, which can be linked to electricity prices.

  We have a light intensity I of the lighting system in µmol/m^2/s and an efficiency η in mol/J. Converting this to hourly values we need to multiply by 3600. Together with the knowledge that 1 MJ = 1/3.6 kWh and thus 1 MJ = 1/3600 MWh, the factor cancels out. All four lighting variables are discrete, so this will result in either the costs of turning on the lighting for one hour or no costs at all when the lighting is turned off. Again, the timestep frequency is taken into account. The resulting cost calculation for a lighting installation C_{light} thus becomes:

$$C_{light} = \sum_{t \in T} act\_light \cdot \frac{f}{60} \cdot \frac{I}{\eta} \cdot 10^{-6} \cdot p, \tag{4.12}$$

where act_light is the discrete variable of turning the lighting system on or off and p is the price in €/MWh .

The LED and SON-T lighting in department 3.5 have light intensities of respectively 120 μmol/m^2/s and 80 μmol/m^2/s. We assume an efficiency of 3.6 μmol/J for LED lighting and 1.85 μmol/J for SON-T lighting. For one-hour data, the LED lighting thus uses $3.33 \cdot 10^{-5}$ MWh/m^2/h and the SON-T lighting $4.32 \cdot 10^{-5}$ MWh/m^2/h.

For department 6, both lighting systems have a light intensity of 140 μmol/m^2/s. The first LED system has an efficiency of 3.4μmol/J, the second LED system 2.7 μmol/J. The resulting needed electricity for LED system 1 is $4.12 \cdot 10^{-5}$ MWh/m^2/h and for LED system 2 this is $5.19 \cdot 10^{-5}$ MWh/m^2/h.

The electricity prices for growers also exist of a commodity component and some extra costs like the energy tax. From an example in the KWIN 2019 document, we get a commodity price of €48/MWh and extra costs of €39.56/MWh resulting in a total of €87.56/MWh [52]. Fixed (monthly) costs are not taken into account. Similar to the situation regarding gas prices, the price of electricity is also substantially higher than normal. At the time of writing, the average commodity price (peak and off-peak hours) of electricity is €271.11/MWh, while in January 2021 this was around €65/MWh [54]. Again, it would be interesting for a decision maker to see the effect of such changes in pricing on their cultivation. As an example, taking the price of 2019 of €87.56/MWh, this would result in €0.0029/m^2/h for LED and €0.0038/m^2/h for SON-T in department 3.5.

- **CO_2 injection**
  The amount of CO_2 injection in department 6 is measured in kilograms per hectare per hour. Since we are calculating costs per m^2, we have to divide the injection by 10.000.

  In the case of using a CHP unit, the price of CO_2 could be set to € 0, as the costs of producing the CO_2 would be entailed in the costs of generating heat and power. However, for simplicity we assume that the CO_2 is bought at a certain market price. A price of € 0,08 per kilogram CO_2 is assumed [52]. These assumptions on CO_2 injection thus lead to the following calculation of costs for CO_2 C_{co2}:

$$C_{co2} = \sum_{t \in T} act\_co2_t \cdot \frac{f}{60} \cdot \frac{1}{10000} \cdot 0.08. \tag{4.13}$$

  As an example, an injection of 50 kg/ha in one hour would thus result in a cost of €0.0004/m^2.

- **Windows & Screens**
  Opening and closing the windows and screens are not costly operations. Small motors regulate the the opening and closing and do not require much energy. Therefore these costs are neglected, resulting in a cost of € 0. From a decision making point of view, it would be interesting to see how much these "free" actions can be utilised to regulate the climate without resorting to more expensive actions. The costs for operating the windows and screens thus become respectively C_{window} and C_{screen}:

$$C_{window} = \sum_{t \in T} act\_window_t \cdot 0 \tag{4.14}$$

$$C_{screen} = \sum_{t \in T} act\_screen_t \cdot 0 \tag{4.15}$$

- **Crop production**
  Continuing the example cultivation year of 2019, the prices for tomato in this year ranged from €0.30/kg fresh-weight to €1.13/kg, the average season price was €0.78/kg [55]. Clearly including a time-varying tomato price in the system would give more realistic results. Therefore the monthly prices are approximated by taking the seasonal average price and doing something similar as with the cyclical features in section 4.1. The price in both 2019 and 2020 is the highest in January and the lowest in June [55]. We calculate the cosine of the month of the year, multiply this by the average difference between the seasonal price and the extrema (min and max prices), and add this to the seasonal average tomato price. So this results in $p = p_s + p_d * cos(M/12 * 2\pi)$, where p is the approximated tomato price, p_s is the seasonal price, p_d is the average difference between p_s and the extrema, and M is the month of the year (counting from 1) of the prediction. Note that while the fluctuating behavior of tomato prices is thus incorporated in the DSS, the resulting tomato price is still the same for each timestep. A result of this approximation for both 2019 and 2020 can be seen in Figure 4.4.

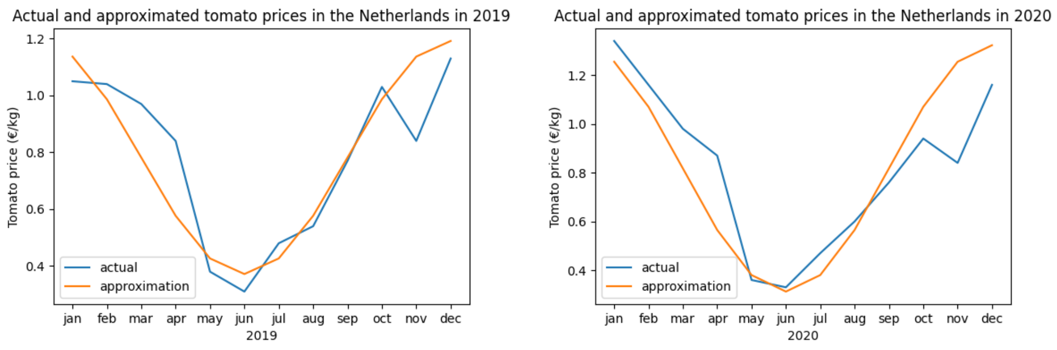


Figure 4.4: Comparison of actual and approximated tomato prices in the Netherlands for 2019 and 2020. The approximation is done with a cosine of the month number divided by the amount of months in a year.

The revenue R_{FW} is then simply the total fresh-weight crop growth times this approximated price p:

$$R_{FW} = \sum_{t \in T} crop\_growth_t \cdot (p_s + p_d \cdot cos(M/12 \cdot 2\pi)) \tag{4.16}$$

4.3.3. Running the DSS

When the complete CP model has been set up, including all greenhouse and crop variables, their prediction functions, additional constraints, and economic variables, the JaCoP solver can run the model. The objective function that is to be maximised is the profit realised within the given period, which I modelled by computing the revenue from crop growth, and from this revenue subtracting the costs that follow the greenhouse actions. A MiniZinc instance is created, including the greenhouse, crop, and complete model and data file. With a given timeout, the solver will search for solutions that satisfy all constraints, while maximising the profit. When an optimal solution is found or there is a timeout, the best solution will be saved as a data file that includes the outside weather, decided greenhouse actions, and predicted inside climate for the given period. Solving statistics and the objective values are saved as well.

To speed up the search, a step-size for assessing objective function values relative to the amount of timesteps was used. By using this step-size, the solver can skip intermediate solutions. A profit step-size of 0.5 eurocents per hour was used.

5 Experiments

To demonstrate and validate the implementations described in the previous chapter, various experiments were performed. For the LSTM and TCN greenhouse models, validation experiments were performed in both department 3.5 and 6. For the LSTM embedding in CP, more validation experiments were performed that demonstrate the correct implementation of the embedding. Lastly, some experiments were performed for runtime analysis of the DSS and its decisions. In order, the experiments for the LSTM and TCN models, the LSTM embedding in CP, and the CP DSS will be discussed in this chapter.

5.1. Time Series Neural Networks

To validate both the LSTM and TCN models, various experiments were performed to compare the predicted climate with the actual climate in the greenhouse. The LSTM and TCN models were trained with the parameters obtained in the grid searches in section 4.1 and predictions were done in both departments in two different season periods; summer and winter for department 3.5, autumn and spring for department 6. For the complete set of experiments in both department 3.5 and 6, please refer to Appendix C.

There is no year-round cultivation and therefore no year-round data is available. Also, some of the sensors in Zensie are removed in different cultivations. Lastly, because at Delphy each of the cultivations within one department are very different and involve different crops, we are limited to sample both the training and prediction data from the same cultivation. Normally, a grower would be able to use multiple cultivations as training data, as likely very similar cultivations are done in one grower's greenhouse. All of this results in only specific periods being usable for training and prediction. The prediction periods for department 3.5 were three days starting from 25/12/2020 for the winter experiment and three days starting from 5/8/2021 for the summer experiment. The training period is just within this range from 28/12/2020 to 4/8/2021. The training period for department 6 was 13/10/2021 to 9/4/2022. The autumn prediction period was three days starting from 10/10/2021, for spring this was three days starting from 10/4/2022.

5.1.1. LSTM greenhouse climate predictions

Firstly, I validated the LSTM greenhouse model for multiple seasons and departments. Light, temperature, humidity deficit, and CO_2 are predicted in a multi-stage prediction and compared to the real inside climate in the given period. Because no CO_2 injection sensor is available in department 3.5, I also performed experiments on department 6 to see the effect on CO_2 prediction.

Set-up

For department 3.5 I trained an LSTM model on all relevant features of department 3.5 (see Table 4.1), with a batch size of 512, a lag of 6 timesteps (6 hours), and 4 LSTM units. For the experiments of department 6 I used a batch size of 256, a lag of 6 timesteps (6 hours), and 4 LSTM units. I performed three-day multi-stage predictions of the inside climate using the prediction datasets of winter and summer in department 3.5 and autumn and spring in department 6. Each experiment was run 5 times and the result was averaged.

Results & Discussion

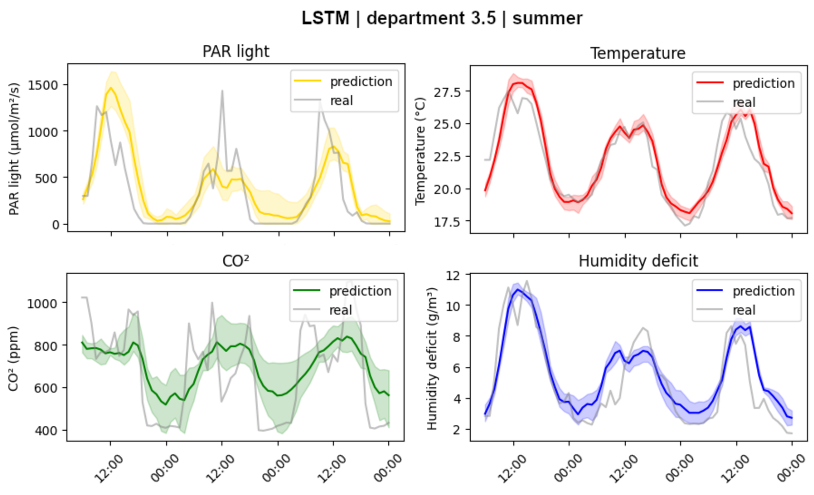


Figure 5.1: LSTM inside climate prediction 5/8/2021 00:00 - 7/8/2021 23:00, department 3.5, resulting in an RMSE of 0.065.

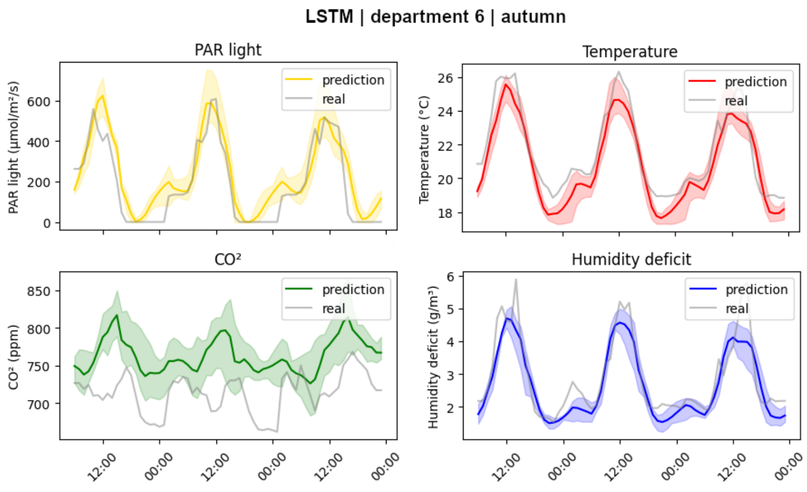


Figure 5.2: LSTM inside climate prediction 10/10/2021 00:00 - 12/10/2021 23:00, department 6, resulting in an RMSE of 0.0278.

Figure 5.1 shows the LSTM prediction in the summer in department 3.5. What we see is that the LSTM model is able to predict the temperature and humidity deficit quite well in all five runs, but it has some trouble with predicting the inside PAR light, which is probably caused by clouds or other shadows formed on the PAR-sensors. Because we have data of outside radiation and artificial lighting, I will perform a simpler computation instead which was explained in section 4.2. Lastly, CO_2 is the hardest target feature for the model to predict. Primarily, CO_2 levels recorded by the sensor are more fluctuating than that of the other climate variables. Furthermore, how much CO_2 is in the air also depends on crop processes, which is not a direct input feature. Unfortunately, the additional knowledge of amount of CO_2 injection did not help the CO_2 prediction, as we can see in Figure 5.2. The temperature and humidity predictions are still quite accurate.

5.1.2. TCN greenhouse climate predictions

Next, I performed validation experiments on the TCN greenhouse model. Again the model was validated on both departments and in multiple seasons.

Set-up

Similarly, the experiment of the TCN model included training a TCN model on all relevant features of department 3.5, with a batch_size of 256, lag of 6 timesteps (6 hours), [1, 2, 4, 8] as dilations, an nb_filter of 128, and a kernel size of 5. For department 6, the same hyper-parameters were used. Again the inside climate was predicted and I ran each experiment 5 times and averaged the result.

Results & Discussion

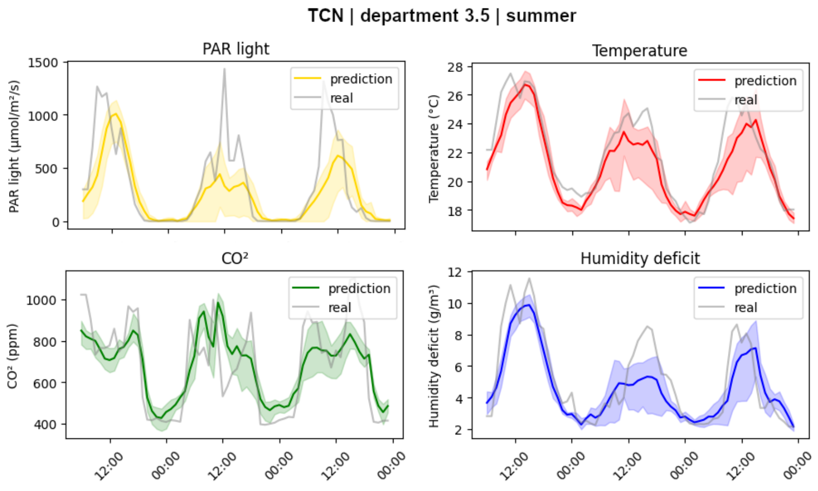


Figure 5.3: TCN inside climate prediction 5/8/2021 00:00 - 7/8/2021 23:00, department 3.5, resulting in an RMSE of 0.0598.

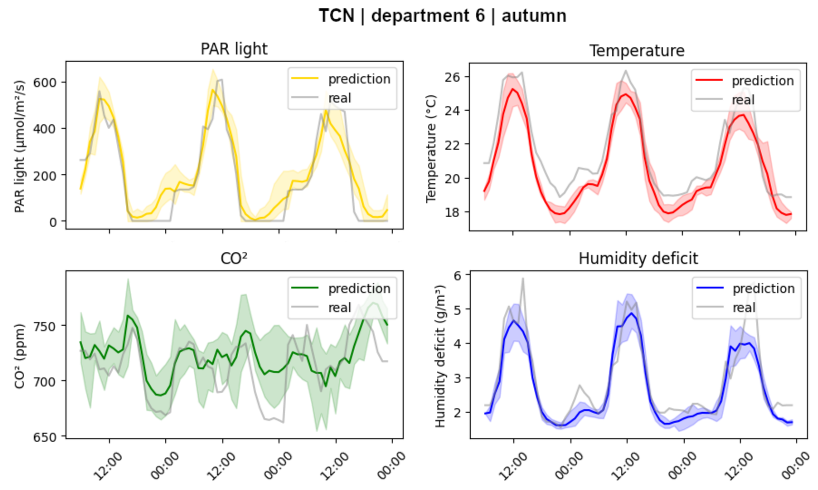


Figure 5.4: TCN inside climate prediction 10/10/2021 00:00 - 12/10/2021 23:00, department 6, resulting in an RMSE of 0.0215.

In Figure 5.3, we can see slightly worse results for temperature and humidity, while the CO_2 prediction was better. The light prediction is not better and additionally gives more variance in the results. In the autumn experiment in department 6 however (see Figure 5.4), the light prediction is better and temperature and humidity predictions are similar to that of the LSTM. CO_2 remains the hardest to predict. Overall, TCN achieves similar RMSE values as the LSTM models.

5.2. Model Embedding in CP

To make sure the LSTM embedding works properly, I performed validation experiments. The possible actions in the greenhouse like the heating tubes and window openings are the decision variables in the CP model, but for these experiments we fix these variables such that the actions are decided. In this way, the LSTM within the CP model should behave similarly as a regular LSTM. To validate this, I set up the following experiments:

5.2.1. LSTM-in-CP validation, 3 days

The most straightforward experiment to perform is using the same set-up as one of the experiments in section 5.1, to see if the results are similar. Therefore the first experiment is the three-day multi-stage prediction of the inside climate of department 3.5 in summer.

Set-up

I trained an LSTM model on cultivation data from 28/12/2020 to 4/8/2021, with all outside weather features, relevant action features for department 3.5, and inside features from Table 4.1. It was trained with a batch_size of 512, lag of 6 timesteps (hours), adam optimizer, loss function MSE, and one LSTM layer with 4 units. After training, the LSTM was embedded into a CP model. The chosen prediction period was 5/8/2021 00:00 to 7/8/2021 23:00 (72 timesteps), with the values of all action variables decided. Some discrepancy in the output is expected, as the action variables are discretised. The experiment was run 5 times and averaged.

Results & Discussion

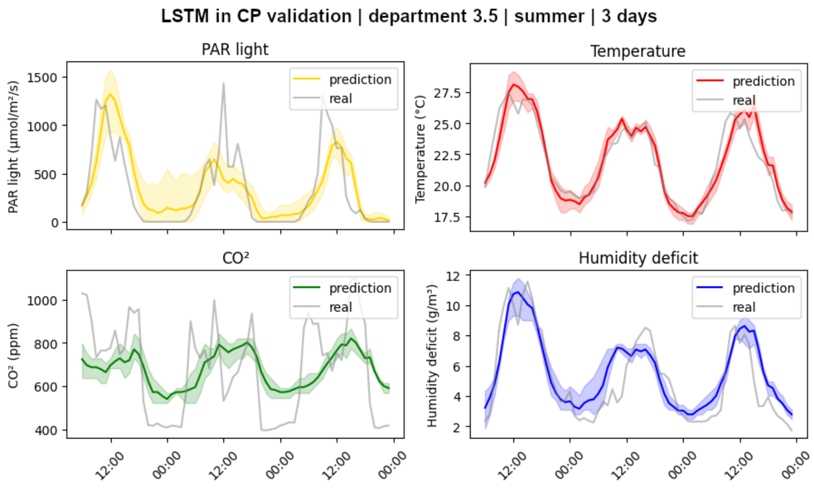


Figure 5.5: Validation of the greenhouse LSTM model embedded in CP for a 3-day prediction period on the inside climate. The first 6 timesteps (lag) of the inside climate are known, the remaining timesteps are predicted in a multi-stage fashion within CP. The outside weather and action variables of all timesteps are filled in.

In Figure 5.5 we see the LSTM prediction of each of the inside climate variables within CP for a prediction period of 3 days, with an RMSE of 0.0665. The figure shows similar results as the "regular" LSTM predictions in section 4.1 with a similar RMSE value as well, indicating that the embedding LSTM in CP is behaving as expected.

5.2.2. LSTM-in-CP validation, 10 days

Since ultimately we want to use the DSS to make long-term decisions and predictions, I did another experiment for a longer period. One could expect that any error in the calculation will be amplified with an increasing amount of timesteps, so the following experiment is set up to test this behavior.

Set-up

The set-up for this second experiment is almost the same as that of the three-day prediction, except that the prediction period is 5/8/2021 00:00 to 14/8/2021 23:00 (241 timesteps). The LSTM training and parameters remained the same. Again the experiment was run 5 times and averaged.

Results & Discussion

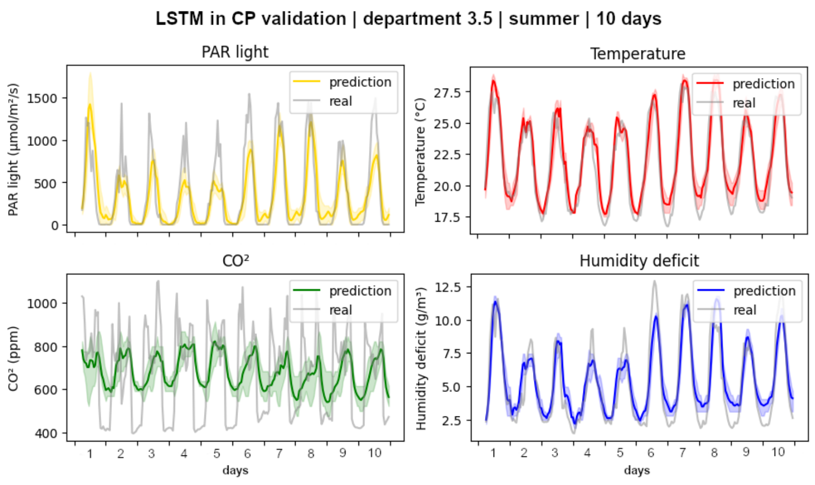


Figure 5.6: Validation of the greenhouse LSTM model embedded in CP for a 10-day prediction period on the inside climate. The first 6 timesteps (lag) of the inside climate are known, the remaining timesteps are predicted in a multi-stage fashion. The outside weather and action variables of all timesteps are filled in.

Figure 5.6 shows the embedded LSTM prediction for 10 days of the inside climate of department 3.5, with an RMSE of 0.0657. The figures show that for a larger amount of timesteps, the LSTM in CP embedding is still able to compute the right outputs. These experiments merely show the correctness of the LSTM embedding, the accuracy of the model remains the same, as we can see from the poor predictive power for CO_2.

5.3. Economic Greenhouse Decision Support

Ultimately, the goal of the DSS is for the growers to be supported in their decisions. Therefore it is interesting to see what decisions the system makes, how fast it comes to a solution, and how these decisions compare to decisions made by the grower. In this section first some general experiments on runtime are done, followed by the comparison between the system and a real grower for a specific cultivation period, and lastly the effect of changing some of the prices on the decisions is studied for small problem instances.

5.3.1. Runtime

There are various input factors that may influence the runtime of the decision support system. The most straightforward factor is the amount of timesteps that are taken into account in the decision making process, but also the amount of features used for prediction in the LSTM and the units within this LSTM affect the runtime. To understand exactly how these factors influence the runtime, I set up three experiments including these factors to see how the input size affects the time of finding a solution.

Set-up

For each of the input factors, I trained a greenhouse LSTM model, embedded this model in MiniZinc, and with the combination of the Lintul-3 crop model I set up the complete CP model. I then set the solver to find the first solution that satisfies all constraints and measured the runtime. Important to note here is that solving for optimality is thus not included in these experiments, these experiments merely show the runtime of finding a solution and computing all LSTM and other propagations that follow. I ran the experiments 5 times and averaged the result, each time with a newly trained LSTM to account for randomness in the training process. The training of the LSTM was not included in the runtime however, as in a real setting the training of the models would only happen once or occasionally. The total runtime thus includes the model embeddings, the set-up of the complete CP model, and running the decision support system. The default values in all three experiments for the amount of timesteps, features, and LSTM units were respectively 15, 15, and 4. In the timesteps experiment, the amount of timesteps to compute were then increased linearly with from 7 to 15 with a step-size of 2. For the features experiments the amount and step-size was the same. Lastly, in the LSTM units experiment, the units were increased from 2 to 10, with a step-size of 2.

Results & Discussion

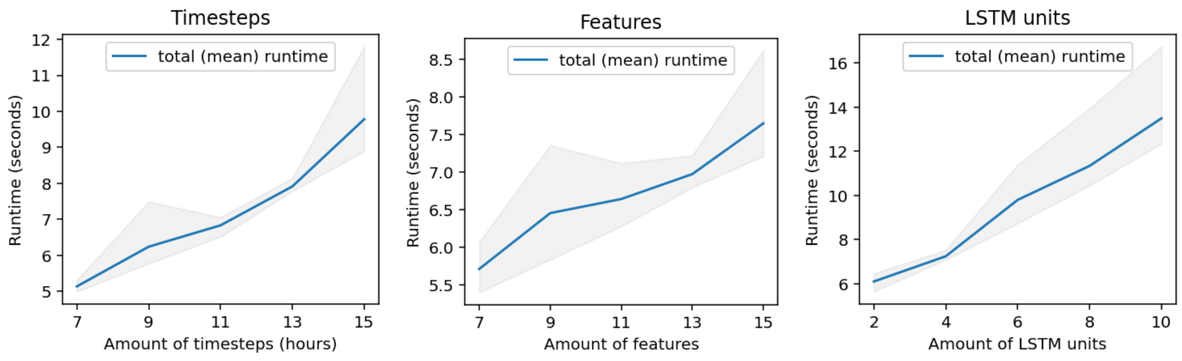


Figure 5.7: Runtime experiments for increasing the amount of timesteps, features, and LSTM units. The experiments were run 5 times and averaged. The blue line shows the average, the grey area are the bounds.

From Figure 5.7 we see that in all three experiments the runtime appears to grow linear with respect to the size of the input. When running the DSS, likely the LSTM units and amount of features will not be altered too much, as they follow from the preparatory hyper-parameter tuning and feature selection. The actual runtime thus mainly depends on the chosen amount of timesteps, which results in a trade-off between long-term prediction and runtime. Still, a linear growth in runtime is acceptable. As said, solving for optimality is not included in these experiments, so we can not say that the runtime of finding a better solution than the grower also grows linear with respect to the input size.

5.3.2. Grower versus Decision Support System

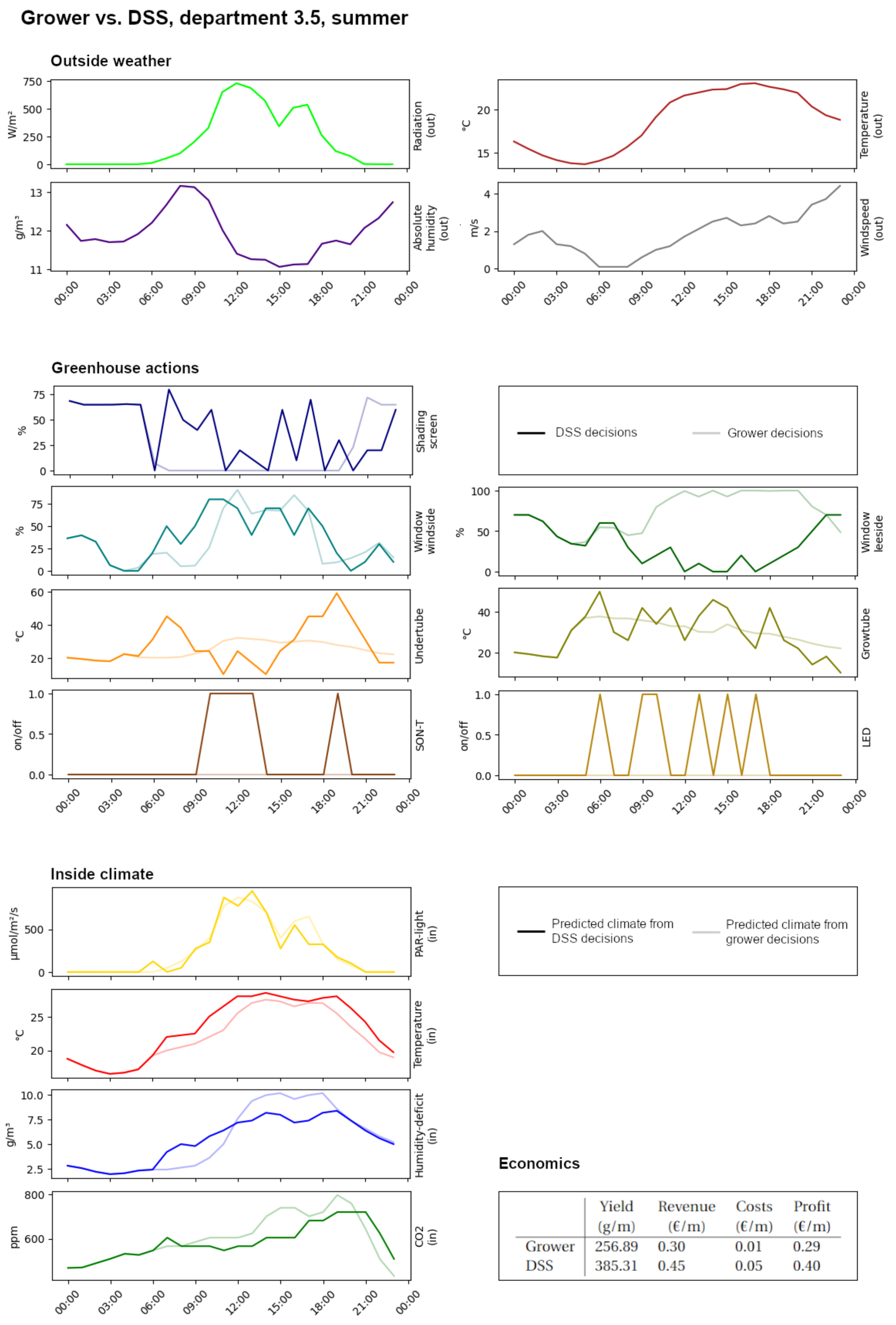
To understand the improvements of the system over a growers decisions, I performed a comparison experiment. In this experiment, I compare the decisions and economic outcomes of a grower and the DSS. Most importantly, the profit should increase, but it is also interesting to see the resources used and the production that was realised. After running some initial experiments with relatively short runtimes, it was clear that the DSS did not give solutions that are an improvement of the grower's decisions. Therefore in this experiment I used a very long runtime of 11 hours. In practice, this runtime would be too long for a grower to be useful, but I performed this experiment mainly for demonstration purposes of the capabilities of the DSS if it would have enough time. Running the experiment multiple times and averaging the result was not possible in this case, as the resulting intermediate solutions may have very different values for the decision variables because of randomisation.

Set-up:

A CP model including an LSTM model for the greenhouse climate and the Lintul-3 model for crop production was generated for department 3.5, using all the relevant features mentioned in Table 4.1, and the hyperparameters that were obtained in the grid search in section 4.1. The prediction period was one day in summer; 5/8/2021 00:00 to 5/8/2021 23:00. I chose such a small period so that the search space was not too large. I set a timeout of 11 hours. First, the CP model was run with action variables filled in, which represent the growers' decisions. The predicted climate and the resulting economic values are computed. The climate that followed the grower's decisions is thus also a prediction, it is not the actual realised climate as the LSTM does not have a 100% accuracy. Next, I ran the model with free action decision variables, so letting the DSS decide the actions. Again, the inside climate and economic results are computed to be compared to the grower.

Results & Discussion

In Figure 5.8 we can see the outcome of running the DSS for a prediction period of one day in summer with a timeout of 11 hours. What we can see is that the DSS decides to make more use of the shading screen and lower both the undertube heating and leeside window ventilation, while still reaching a similar temperature. Also, the lighting systems are used more, directly leading to more crop growth. The DSS thus sees the added benefit of turning the lights on compared to their costs. In the resulting economic values, we can see that indeed the system has higher costs, but we see a payback of this in the revenue. In this case, the DSS thus has an increased profit with respect to the grower's decisions. What we can conclude from this experiment, is that given enough time or a small enough search space, the DSS is able to find better solutions. This shows the potential of such a system. Still, the runtime is too long and the prediction period too short to be used in practice, so improvements are needed.



	Yield (g/m)	Revenue (€/m)	Costs (€/m)	Profit (€/m)
Grower	256.89	0.30	0.01	0.29
DSS	385.31	0.45	0.05	0.40

Figure 5.8: Results of running the DSS in department 3.5 in the period 5/8/2021 00:00 to 5/8/2021 23:00 (one day), with a timeout of 11 hours. The outside weather is retrieved, the greenhouse actions are decided by the DSS, and the inside climate is predicted. These actions and the predicted climate are compared to the actions taken by the grower and the resulting climate. In the figure, the dark lines represent the result of running the DSS, the brighter lines represent the grower's decisions and resulting climate. Also, a comparison of yield and economic outcomes is given.

5.3.3. Economics

As mentioned before, currently gas and electricity prices are extremely high, which shows the importance of an economic decision support system. Last year the commodity prices of electricity and gas were respectively €65/MWh and €0.65/m^3 [53, 54]. Using the additional fixed handling costs found in the KWIN 2019 document of €39.56/MWh respectively €0.0359/m^3, we see that the total price of electricity has become almost three times higher and that of gas almost four times higher. Seeing the effect of such price changes would be interesting to see for a grower, as the DSS should weigh in these costs and the expected profit that follows. In the following experiments, I will examine the effect of a price change of electricity and natural gas.

Set-up

I set up three experiments for both scenarios. One with the price taken from the KWIN 2019 document as explained in subsection 4.3.2, another experiment with a price three times lower, and lastly one with a price three times higher. Experiments are run for department 3.5 in the period 15/4/2021 5:00 to 15/4/2021 13:00. Such a short prediction period was chosen so that the DSS has enough time to come to a reasonable solution. Each of the experiments was run 5 times and averaged, each with a timeout of 10 minutes. A lag of 6 was used, which means that in each experiment the first 6 timesteps are already decided in advance. For clarity, only the last four timesteps are shown, of which the last two timesteps are thus newly decided by the DSS.

Results & Discussion

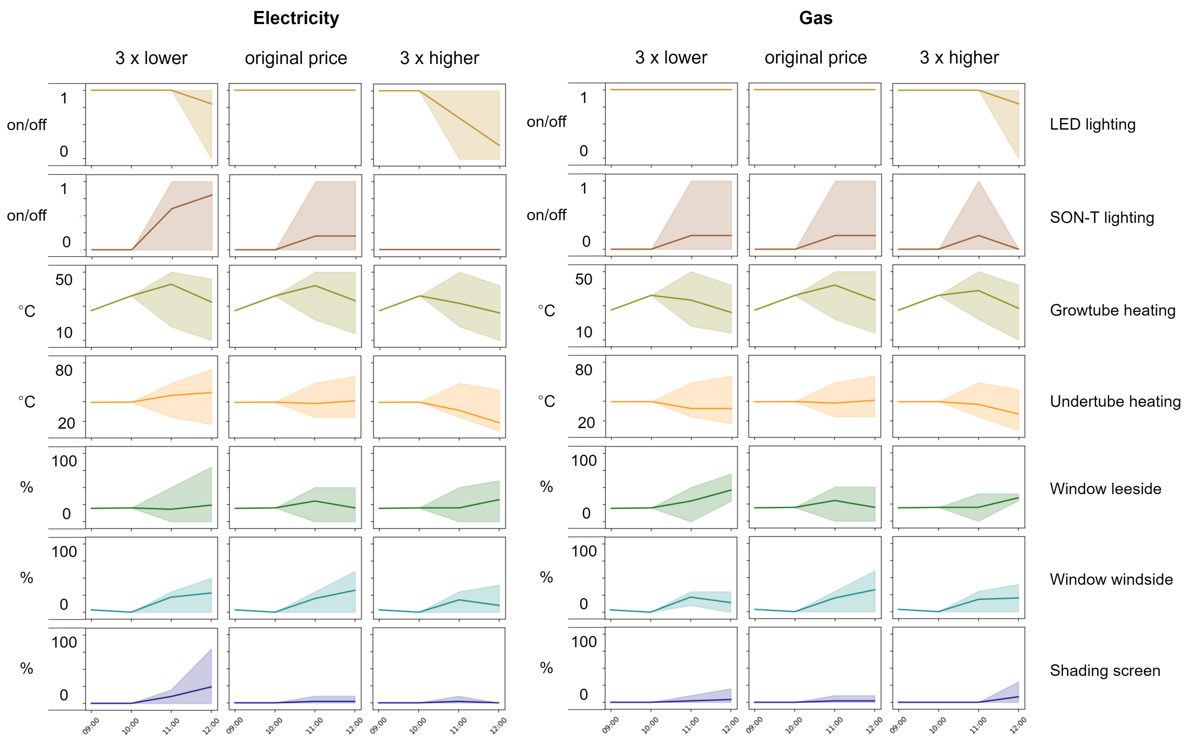


Figure 5.9: DSS decisions when varying the electricity (left) and gas (right) prices between the original price, a price three times lower than the original, and a price three times higher than the original.

In Figure 5.9 we see the actions taken by the DSS with varying electricity and gas prices. On the left we see the effects of decreasing and increasing the electricity price. Although generally we see subtle differences in the actions taken, in the case of lighting the effects of increasing the electricity price are evident. When the electricity price is higher, the DSS prefers to turn off the lights. Especially the SON-T lighting that has a lower efficiency is preferred to be turned off. From the resulting economic values in Figure 5.10, we see that with a higher price, the DSS gives up some of its revenue and thus yield, to cut the increased costs somewhat. Also, we see that with a lower price, the revenue may be lower to reach a similar profit as with the original price.

With the varying gas prices, we see very few differences in the actions in Figure 5.9 and similarly in economic results in Figure 5.10. What is clear from the economic results, is that the gas price has a substantially lower effect on the costs than the electricity price.

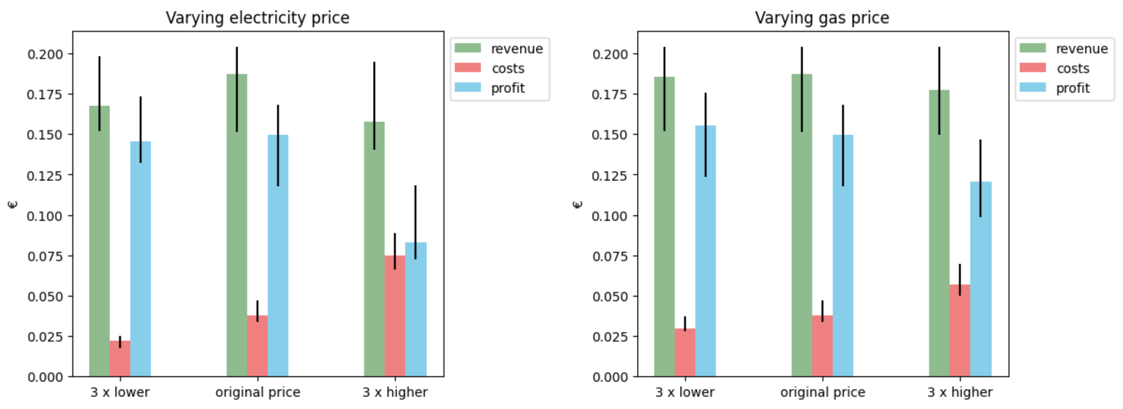


Figure 5.10: Economic results when varying the electricity (left) and gas (right) prices between the original price, a price three times lower than the original, and a price three times higher than the original.

6 Conclusion

In this thesis I studied the applicability of CP in greenhouse decision making, with the objective of maximising economic profits for the grower, by leveraging time-based NNs and embedding these in CP. In this concluding chapter I will first answer the research questions, followed by the limitations of my work with directions for possible future work, and lastly an initial set-up for a publication.

6.1. Research Questions

To guide the research, I set up three sub-questions that together helped in answering the main research question, all described in section 1.4. Each of the sub-questions will be discussed below, followed by an answer to the main research question.

1. **How can short-term decisions and their long-term profit in a greenhouse be modelled?**

   Many mathematical models exist that model the greenhouse and crop, some of which are discussed in chapter 2. Through multiple equations, the physical relations between the greenhouse and crop are modelled. However, these models are complex and need many parameters, making their implementation impractical. In this thesis, the greenhouse was therefore modelled by two types of time-based NNs; Long Short-Term Memory and Temporal Convolutional Network, of which the implementations were discussed in section 4.1 and their performances were shown in section 5.1. Unfortunately, a similar approach for the crop model was not realised in this study, so the simpler and more short-term based Lintul-3 model was used. In section 4.3 the DSS implementation was discussed, in which the short-term decisions were represented by the possible greenhouse actions that influence the inside climate, modelled as decision variables per timestep in a CP model. The long-term profit was modelled by the revenue obtained from the predicted yield in the Lintul-3 model that followed from the predicted climate, minus the aggregation of costs per timestep of each action decision variable.

2. **How can this model be used in combination with Constraint Programming?**

   The embeddings of the greenhouse and crop models, especially the LSTM-in-CP embedding, were discussed in section 4.2. An important aspect of this thesis was to implement a proof-of-concept of such an embedding. This embedding was realised through multiple MiniZinc functions, of which key points in its realisation were discussed. Firstly, by directly instantiating variables through their functional dependence on other variables, we explicitly tell the solver not to perform a search on these variables. This leads to the solver only performing a search on the input variables of the LSTM, not simultaneously on the output variables. Also, functions were preferred over predicates, as no additional intermediate variables are needed that the solver tries to perform a search on. Furthermore, since the internal cell and hidden state of the LSTM cell are dependent on the chosen time lag, predicates needed to be used instead of functions. To help the solver solve these computations, explicit bounds as tight as possible were used on these intermediate variables. Lastly, since the multi-stage prediction is a sequential process, it was essential to inform the solver the order of solving, which was shown in section 4.3. In two validation experiments in section 5.2, the correctness of the LSTM embedding was shown. The greenhouse model was used in combination with CP by embedding an LSTM, the crop model was used by embedding the Lintul-3 model as a single MiniZinc function.

3. **How does this Constraint Programming model aid growers in the economic decision making process?**

   The DSS was designed to optimise the grower's profit through minimising the costs that follow the actions taken in the greenhouse and maximising the revenue that follows from the predicted climate and subsequently the crop yield, which was shown in section 4.3. Using this DSS can help growers make better decisions in the future, as well as helping them understand what the DSS would do differently in past cultivation periods. Unfortunately however, the DSS is not always performing better than the grower in terms of resulting profit. This is due to the search space being too large for the solver to find a better solution within reasonable time. With the current state of the system, the DSS will thus not necessarily help growers make better decisions. For small problem instances however, we have seen improvements of the made decisions resulting in more profit in section 5.3. Here we have also seen that the DSS can help growers make well-informed decisions when gas or electricity prices rise, by turning off the lights for example. The potential of the DSS has thus been shown on small instances, but needs further improvement to more quickly find good solutions.

The main research question set out to answer in this research was formulated as follows:

> **How can short-term decisions on the climate in a greenhouse with certain costs be made such that it leads to a better long-term profit, using Constraint Programming?**

How Constraint Programming can be used to make better economic decisions in a tomato greenhouse was studied in this thesis and it was realised through embedding an LSTM greenhouse model and a Lintul-3 crop model in MiniZinc, and minimising the costs of operating the greenhouse climate while maximising the revenue following from the predicted crop growth. A sub-goal of this research was implementing a proof-of-concept of an LSTM-in-CP embedding, which was realised through specifically instructing the CP solver which variables should be taken into the search, and the order in which to perform this search. The applicability of CP in an economic greenhouse DSS was shown for small instances, but has room for improvement until it can be deployed properly in practice for larger instances.

6.2. Limitations and Future Work

While the applicability of CP in economic greenhouse decision making has been shown for small instances, before it can be applied properly on real-world instances some of the limitations need to be addressed. First of all, the implemented greenhouse LSTM model is not able to predict CO_2 and light well. CO_2 levels are fluctuating relatively fast and not as smooth as temperature or humidity and they depend on crop processes, making it harder to predict. The poor light prediction is likely the result of the PAR-sensors that are shadowed by clouds or other shadow-forming objects in the greenhouse. Secondly, the crop model is very basic and is does not include long-term relations. Lastly, the DSS itself is not yet usable in practice, mainly because the large search space makes it hard for the DSS to find good solutions. Possible directions for future work are laid out below:

Additional data

In this study, each time an NN model was trained it was trained on data of one cultivation which usually spans a period that does not cover the whole year. This means that the models do not see the full variety of weather data in a year during training. At Delphy, each greenhouse department involves different crops in different cultivation periods, and sensors are sometimes removed or replaced. Future work could include a more thorough data collection of multiple cultivations of the same crop variety, all in one large dataset. Such a larger dataset could improve the accuracy of the trained models. Also, because the crop substantially affects the environment, additional crop data such as the leaf area index (LAI) or age of the plant could improve the accuracy, if this data is available.

Improving the greenhouse model

We have seen that the LSTM and TCN models did not perform well on light and CO_2 prediction. Light was therefore modelled through a calculation using the outside radiation, artificial lighting systems, and the shading screen, giving more reliable results. Other screens, such as the energy saving screen in department 6, could be added in a more general approach, so that any number of screens can be applied in this calculation. Furthermore, the CO_2 prediction should be improved, possibly by including more features such as the LAI of the crop.

Improving the crop model

As mentioned, the simple Lintul-3 model was used to predict the crop yield. However, this model is very basic and does not take into account the long-term effect of decisions, as it simply calculates the effect of light, temperature, and CO_2 for each timestep. Also, the humidity is thus neglected in the current implementation. Initially, the intention was to implement a time-based NN for the prediction of crop yield as well, similar to that of the greenhouse. Unfortunately this was not realised in this study, but future work could include the implementation of such a model, which potentially can include the long-term effects of greenhouse decisions. The groundwork for the implementation of such a time-based NN crop model is laid out in this thesis, by the general implementation of LSTMs and TCNs.

Search strategy

The implemented DSS does not always give better results than a grower's decisions, so it needs improvement. While various search strategies were tried, future work could include experimenting more with the different combinations of search annotations and restart possibilities. Especially finding the best combination of variable and value selection annotations could help in finding solutions more efficiently. Also, any additional constraints that can reduce the search space would be very beneficial, as currently it is too large to find good solutions in reasonable time.

Warm start

Implementing a warm start, i.e. doing an initialisation with a known good solution, could improve the DSS greatly. For example, this initialisation could be decisions a grower has made in similar weather conditions. Firstly, since the grower almost always outperforms the DSS in large instances with short timeouts, starting with decisions similar to the grower would provide a head start. Secondly, when the DSS is provided with a solution that is already good, the bounds can be set tighter around this solution resulting in a reduction of the search space.

Iterative refinement

With an iterative refinement method, the DSS first can be run with low precision both in terms of climate and crop predictions as well as timestep frequency, so that a rough estimate of a good solution can be found. Then by using this found solution, the DSS can be run again with slightly higher precision but tighter bounds. By repeating this process, the system may converge to a good solution much quicker.

Accurate and time-based costs and prices

Currently, the costs of operating the greenhouse actions are assumed by looking at historical values. Similarly, the price of tomatoes is estimated by using a historical price, and additionaly it is increased or decreased according to the month of the year. However, these economic values are computed in advance and fixed for each timestep. A more realistic DSS would include arrays of costs and prices that show their varying values for each timestep. Also, future work could include predicting these values instead of making assumptions.

6.3. Research Paper

The potential of using CP in economic greenhouse decision making, as well as the novel embedding of an LSTM in CP, gave me and my supervisors the incentive to try and publish the results of this thesis. We have decided to apply for the Thirty-Fifth Annual Conference on Innovative Applications of Artificial Intelligence (IAAI-23). Since the thesis involves the use of inter-disciplinary AI techniques, namely LSTMs and CP, we will apply for track 4 of the conference; "*Innovative Inter-disciplinary AI Integration*". The submission date will be in September 2022. Please see Appendix D for a draft of this research paper.

Bibliography

[1] G. van Straten and E.J. van Henten. Optimal greenhouse cultivation control: Survey and perspectives. *IFAC Proceedings Volumes*, 43(26):18–33, 2010. ISSN 1474-6670. doi: https://doi.org/10.3182/20101206-3-JP-3009.00004. 3rd IFAC Conference in Modelling and Control in Agriculture, Horticulture and Post-Harvest Processing - Agricontrol.

[2] E. Iddio, L. Wang, Y. Thomas, G. McMorrow, and A. Denzer. Energy efficient operation and modeling for greenhouses: A literature review. *Renewable and Sustainable Energy Reviews*, 117:109480, 2020. ISSN 1364-0321. doi: https://doi.org/10.1016/j.rser.2019.109480.

[3] G. van Straten, H. Challa, and F. Buwalda. Towards user accepted optimal control of greenhouse climate. *Computers and Electronics in Agriculture*, 26(3):221–238, 2000. ISSN 0168-1699. doi: https://doi.org/10.1016/S0168-1699(00)00077-6. URL `https://www.sciencedirect.com/science/article/pii/S0168169900000776`.

[4] Z. Kang. Autonomous greenhouse model optimisation. Master's thesis, Wageningen University, the Netherlands, 2020.

[5] A. Giakoumatos. A decision support model for an optimal lighting strategy, 2021. As part of thesis Farm Technology course at Wageningen University.

[6] R.F. Tap. *Economic-based optimal control of greenhouse tomato crop production.* PhD thesis, Wageningen University, 2000.

[7] M. Wallace. Practical applications of constraint programming. *Constraints*, 1(1-2):139–168, 1996. doi: 10.1007/BF00143881.

[8] E.J. Van Henten, J. Bontsema, and G. van Straten. Improving the efficiency of greenhouse climate control: An optimal control approach. *Netherlands Journal of Agricultural Science 45, p. 109-125*, 45, 1997. doi: 10.18174/njas.v45i1.529.

[9] K. Nemali. History of controlled environment horticulture: Greenhouses. *HortScience*, 57(2):239 – 246, 2022. doi: 10.21273/HORTSCI16160-21. URL `https://journals.ashs.org/hortsci/view/journals/hortsci/57/2/article-p239.xml`.

[10] R.C. Morrow. Led lighting in horticulture. *HortScience*, 43(7):1947–1950, 2008. doi: 10.21273/hortsci.43.7.1947. URL `https://www.scopus.com/inward/record.uri?eid=2-s2.0-57049180834&doi=10.21273%2fhortsci.43.7.1947&partnerID=40&md5=4a33d49693845008b5a43b6369206ce3`.

[11] F. Rodriguez, M. Berenguel, J. Guzmán, and A. Arias. *The Greenhouse Dynamical System*, pages 9–97. 2015. ISBN 978-3-319-11133-9. doi: 10.1007/978-3-319-11134-6_2.

[12] P. van Weel, P. Geelen, and J. Voogt. *PLANT EMPOWERMENT The Basic Principles.* 2018. ISBN 978-90-829035-0-8.

[13] H. Challa. *Crop growth models for greenhouse climate control.*, pages 125–145. Simulation monographs. Pudoc, 1990. ISBN 9789022010044.

[14] C. Stanghellini. Transpiration of greenhouse crops : an aid to climate management. 1987.

[15] M.B. Kirkham. *Principles of Soil and Plant Water Relations.* Academic Press, 2014.

[16] B.H.E. Vanthoor. *A model-based greenhouse design method.* PhD thesis, Wageningen University, 2011.

[17] J.W. Jones, E. Dayan, L.H. Allen, H. van Keulen, and H. Challa. A dynamic tomato growth and yield model (tomgro). *Transactions of the ASAE*, 34(2):663–672, 1991. ISSN 0001-2351.

[18] I. Lopez-Cruz, E. Fitz-Rodríguez, J. Torres-Monsivais, E. Trejo-Zúñiga, A. Ruiz Garcia, and A. Arias. *Control Strategies of Greenhouse Climate for Vegetables Production*, pages 401–421. 2013. ISBN 978-3-319-03879-7. doi: 10.1007/978-3-319-03880-3_14.

[19] D. Katzin, E.J. van Henten, and S. van Mourik. Process-based greenhouse climate models: Genealogy, current status, and future directions. *Agricultural Systems*, 198:103388, 2022. ISSN 0308-521X. doi: https://doi.org/10.1016/j.agsy.2022.103388. URL `https://www.sciencedirect.com/science/article/pii/S0308521X22000245`.

[20] C.M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

[21] A. Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020. ISSN 0167-2789. doi: https://doi.org/10.1016/j.physd.2019.132306. URL `https://www.sciencedirect.com/science/article/pii/S0167278919305974`.

[22] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 1997. doi: 10.1162/neco.1997.9.8.1735.

[23] G. Chevalier. Schematic of the long-short term memory cell, a component of recurrent neural networks, 2018. URL `https://commons.wikimedia.org/wiki/File:LSTM_Cell.svg`. Accessed: 2022-03-01.

[24] A. Ali and H.S. Hassanein. Time-series prediction for sensing in smart greenhouses. In *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pages 1–6, 2020. doi: 10.1109/GLOBECOM42002.2020.9322549.

[25] D. Jung, H.S. Kim, C. Jhin, H. Kim, and S.H. Park. Time-serial analysis of deep neural network models for prediction of climatic conditions inside a greenhouse. *Computers and Electronics in Agriculture*, 173:105402, 2020. ISSN 0168-1699. doi: https://doi.org/10.1016/j.compag.2020.105402. URL `https://www.sciencedirect.com/science/article/pii/S0168169919317326`.

[26] J.W. Lee, W. Kang, T. Moon, I. Hwang, D. Kim, and J.E. Son. Estimating the leaf area index of bell peppers according to growth stage using ray-tracing simulation and a long short-term memory algorithm. *Horticulture, Environment, and Biotechnology*, 61, 2020. doi: 10.1007/s13580-019-00214-9.

[27] B. Alhnaity, S. Pearson, G. Leontidis, and S. Kollias. Using deep learning to predict plant growth and yield in greenhouse environments. *Acta Horticulturae*, 1296:425–431, 2020. doi: 10.17660/ActaHortic.2020.1296.55. URL `https://www.scopus.com/inward/record.uri?eid=2-s2.0-85097321834&doi=10.17660%2fActaHortic.2020.1296.55&partnerID=40&md5=ca0ab156f262258783dcf72d3f0fabce`.

[28] C. Lea, M.D. Flynn, R. Vidal, A. Reiter, and G.D. Hager. Temporal convolutional networks for action segmentation and detection. *CoRR*, abs/1611.05267, 2016. URL `http://arxiv.org/abs/1611.05267`.

[29] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A.W. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016. URL `http://arxiv.org/abs/1609.03499`.

[30] S. Bai, J.Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR*, abs/1803.01271, 2018. URL `http://arxiv.org/abs/1803.01271`.

[31] L. Gong, M. Yu, S. Jiang, V. Cutsuridis, and S. Pearson. Deep learning based prediction on greenhouse crop yield combined tcn and rnn. *Sensors*, 21(13), 2021. ISSN 1424-8220. doi: 10.3390/s21134537. URL `https://www.mdpi.com/1424-8220/21/13/4537`.

[32] J.R. Llera, E.S. Runkle, E.D. Goodman, and L. Xu. Improving greenhouse environmental control using crop-model-driven multi-objective optimization. pages 292–293, 2018. doi: 10.1145/3205651.3205724.

[33] C. Jin, H. Mao, Y. Chen, Q. Shi, Q. Wang, G. Ma, and Y. Liu. Engineering-oriented dynamic optimal control of a greenhouse environment using an improved genetic algorithm with engineering constraint rules. *Computers and Electronics in Agriculture*, 177:105698, 2020. ISSN 0168-1699. doi: https://doi.org/10.1016/j.compag.2020.105698.

[34] H. Pohlheim and A. Heißner. Optimal control of greenhouse climate using a short time climate model and evolutionary algorithms. 1997.

[35] J.P. Coelho, P.B. de Moura Oliveira, and J.B. Cunha. Greenhouse air temperature predictive control using the particle swarm optimisation algorithm. *Comput. Electron. Agric.*, 49(3):330–344, 2005. ISSN 0168-1699. doi: 10.1016/j.compag.2005.08.003. URL `https://doi.org/10.1016/j.compag.2005.08.003`.

[36] L. Chen, S. Du, Y. He, M. Liang, and D. Xu. Robust model predictive control for greenhouse temperature based on particle swarm optimization. *Information Processing in Agriculture*, 5(3):329–338, 2018. ISSN 2214-3173. doi: https://doi.org/10.1016/j.inpa.2018.04.003.

[37] G. Pesant. A constraint programming primer. *EURO J. on Computational Optimization*, 2(3):89–97, 2014. doi: 10.1007/s13675-014-0026-3. URL `https://doi.org/10.1007/s13675-014-0026-3`.

[38] N. Nethercote, P. Stuckey, R. Becket, S. Brand, G. Duck, and G. Tack. Minizinc: Towards a standard cp modelling language. pages 529–543, 2007. ISBN 978-3-540-74969-1. doi: 10.1007/978-3-540-74970-7_38.

[39] The minizinc handbook. `https://www.minizinc.org/doc-2.5.5/en/index.html`. Accessed: 2022-03-09.

[40] M. Wallace. *Building Decision Support Systems – using MiniZinc*. Springer, Cham, Switzerland, 2020. ISBN 978-3-030-41731-4. doi: 10.1007/978-3-030-41732-1. URL `https://doi.org/10.1007/978-3-030-41732-1`.

[41] T. Feydy, A. Goldwaser, A. Schutt, P.J. Stuckey, and K.D. Young. Priority search with minizinc. *The Sixteenth International Workshop on Constraint Modelling and Reformulation at CP2017*, 2017.

[42] M. Lombardi, M. Milano, and A. Bartolini. Empirical decision model learning. *Artificial Intelligence*, 244:343–367, 2017. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2016.01.005. URL `https://www.sciencedirect.com/science/article/pii/S0004370216000126`. Combining Constraint Solving with Mining and Learning.

[43] A. Bartolini, M. Lombardi, M. Milano, and L. Benini. Neuron constraints to model complex real-world problems. volume 6876, pages 115–129, 2011. ISBN 978-3-642-23785-0. doi: 10.1007/978-3-642-23786-7_11.

[44] S. Andringa. Applying constraint programming to enterprise modelling. Master's thesis, TU Delft, the Netherlands, 2021.

[45] The pandas development team. pandas-dev/pandas: Pandas, 2020. URL `https://doi.org/10.5281/zenodo.3509134`.

[46] F. Chollet et al. Keras. `https://keras.io`, 2015.

[47] P. Remy. Temporal convolutional networks for keras. `https://github.com/philipperemy/keras-tcn`, 2020.

[48] K. Kuchcinski and R. Szymanek. Jacop - java constraint programming solver. 2013. CP Solvers: Modeling, Applications, Integration, and Standardization, co-located with the 19th International Conference on Principles and Practice of Constraint Programming ; Conference date: 16-09-2013.

[49] J.D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[50] B. Rijk. Integration of sensor data in crop models for precision agriculture. Master's thesis, Wageningen University, the Netherlands, 2013.

[51] H.F. de Zwart. Lage temperatuurverwarming in de glastuinbouw. Technical Report GTB-1219, Wageningen UR Glastuinbouw, Bleiswijk, The Netherlands, 2013.

[52] M.G.M. Raaphorst and J. Benninga. Kwantitatieve informatie voor de glastuinbouw 2019. 26, 2019.

[53] Wat doet de gasprijs in 2022? https://www.overstappen.nl/energie/gasprijzen/#:~:text=In%20augustus%202021%20was%20de,2%2C10%20euro%20per%20m3. Accessed: 2022-03-16.

[54] Actuele beursprijzen. https://vanhelder.nl/zakelijke-energie/actuele-beursprijzen. Accessed: 2022-03-16.

[55] EU Fruit and Vegetables Market Observatory Tomato Subgroup. The tomato market in the eu: Vol. 2: Prices for fresh products. https://ec.europa.eu/info/food-farming-fisheries/farming/facts-and-figures/markets/overviews/market-observatories/fruit-and-vegetables/tomatoes-statistics_en, 2021. Accessed: 2022-03-16.

A
Pipeline

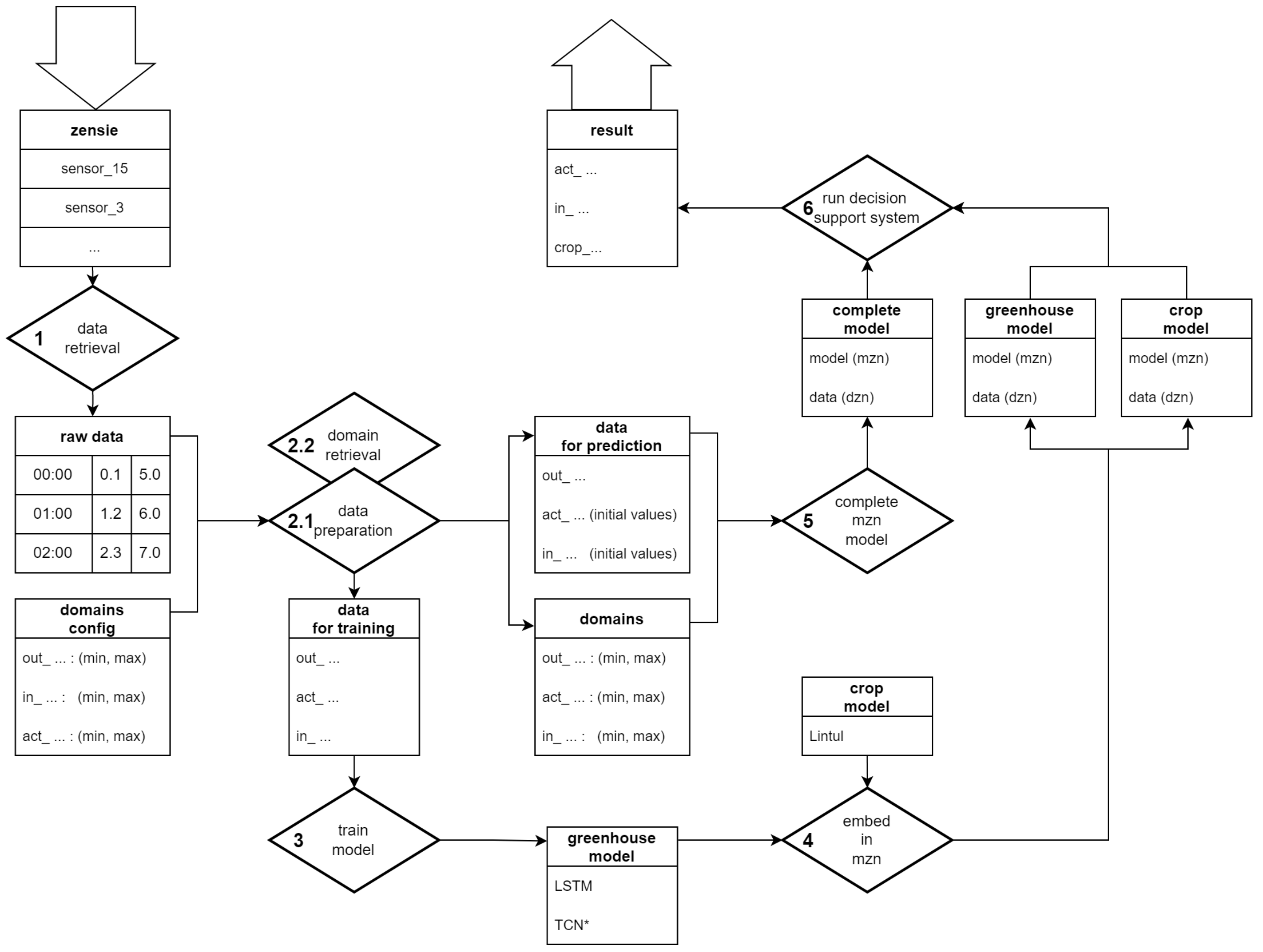


Figure A.1: Technical overview of pipeline from sensor data retrieval to the end result of the decision support system.
*TCN is not incorporated in the MiniZinc embedding.

B
LSTM functions in MiniZinc

```
include "activations.mzn";
include "linear_algebra.mzn";


% ---
% One forward LSTM calculation of one batch of features (timesteps x features).
% ---
function array[int,int] of var float: lstm_forward(
    array[int,int] of var float: X_batch,
    [ ... ] % array[int,int] of float: Wg, Ug, bg for g in [i, f, c, o]
    int: n,
    int: m,
    int: u,
    int: l
) =
    let {
        array[1..l, 1..u] of var -int2float(l)..int2float(l): c;
        array[1..l, 1..u] of var -1.0..1.0: h;
    } in
        let {

            constraint forall (t in 1..l) (
                let {

                    % cell state
                    array[1..u, 1..1] of var -int2float(l)..int2float(l): c_t =
                        lstm_forward_timestep_cell(
                            array2d(1..n, 1..1, [ X_batch[t, i]
                            | i in 1..n]),
                            array2d(1..u, 1..1, [ if t = 1 then 0.0 else h[t - 1, i] endif
                            | i in 1..u ]),
                            array2d(1..u, 1..1, [ if t = 1 then 0.0 else c[t - 1, i] endif
                            | i in 1..u ]),
                            Wi, Ui, bi, Wf, Uf, bf, Wc, Uc, bc, Wo, Uo, bo, n, u, l
                        );
                } in let {

                    % hidden state
                    array[1..u, 1..1] of var -1.0..1.0: h_t =
                        lstm_forward_timestep_hidden(
                            array2d(1..n, 1..1, [ X_batch[t, i]
                            | i in 1..n]),
                            array2d(1..u, 1..1, [ if t = 1 then 0.0 else h[t - 1, i] endif
                            | i in 1..u ]),
                            array2d(1..u, 1..1, [ c[t, i]
                            | i in 1..u ]),
                            Wi, Ui, bi, Wf, Uf, bf, Wc, Uc, bc, Wo, Uo, bo, n, u, l
                        );
                } in

                    % fill in cell and hidden state results of this timestep
                    forall(ii in 1..u, jj in 1..1) (
                        c[t, ii] = c_t[ii, jj] /\
                        h[t, ii] = h_t[ii, jj]
                    )
            );
        } in
            array2d(1..u, 1..1, [ h[l, i] | i in 1..u ]);
```

```
% ---
% Cell state calculations for next timestep.
% ---
function array[int,int] of var float: lstm_forward_timestep_cell(
    array[int,int] of var float: Xt,
    array[int,int] of var float: h_prev,
    array[int,int] of var float: c_prev,
    [ ... ] % array[int,int] of float: Wg, Ug, bg for g in [i, f, c, o]
    int: n,
    int: u,
    int: l
) =
    elementwise_addition(
        elementwise_multiplication(
            lstm_gate_calculation(Xt, h_prev, Wf, Uf, bf, "sigmoid", n, u),
            c_prev
        ),
        elementwise_multiplication(
            lstm_gate_calculation(Xt, h_prev, Wi, Ui, bi, "sigmoid", n, u),
            lstm_gate_calculation(Xt, h_prev, Wc, Uc, bc, "tanh", n, u)
        )
    );


% ---
% Hidden state calculations for next timestep.
% ---
function array[int,int] of var float: lstm_forward_timestep_hidden(
    array[int,int] of var float: Xt,
    array[int,int] of var float: h_prev,
    array[int,int] of var float: c,
    [ ... ] % array[int,int] of float: Wg, Ug, bg for g in [i, f, c, o]
    int: n,
    int: u,
    int: l
) =
    elementwise_multiplication(
        lstm_gate_calculation(Xt, h_prev, Wo, Uo, bo, "sigmoid", n, u),
        activation(c, "tanh")
    );


% ---
% Single LSTM gate calculation (can be used for i, f, c', and o gates).
% ---
function array[int,int] of var float: lstm_gate_calculation(
    array[int, int] of var float: Xt,
    array[int, int] of var float: h_prev,
    array[int, int] of float: Wg,
    array[int, int] of float: Ug,
    array[int, int] of float: bg,
    string: activation_name,
    int: n,
    int: u
) =
    activation(
        elementwise_addition(
            elementwise_addition(
                matrix_multiplication(Wg, Xt),
                matrix_multiplication(Ug, h_prev)
            ),
            bg
        ),
        activation_name
    );
```

Listing B.1: LSTM functions in MiniZinc, for calculating the cell and hidden states and for each of the gates i, f, c', and o. The weights and biases in the function definitions are truncated here for readability.

C
LSTM and TCN predictions

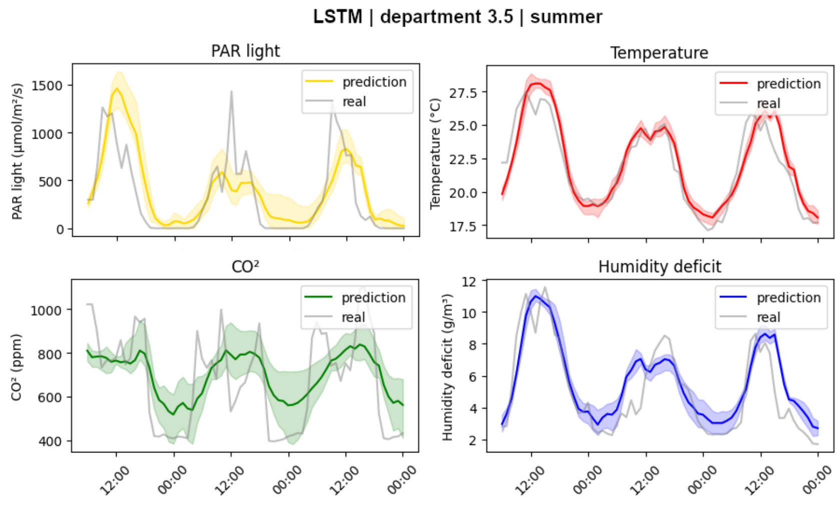


Figure C.1: LSTM inside climate prediction 5/8/2021 00:00 - 7/8/2021 23:00, department 3.5, resulting in an RMSE of 0.065.

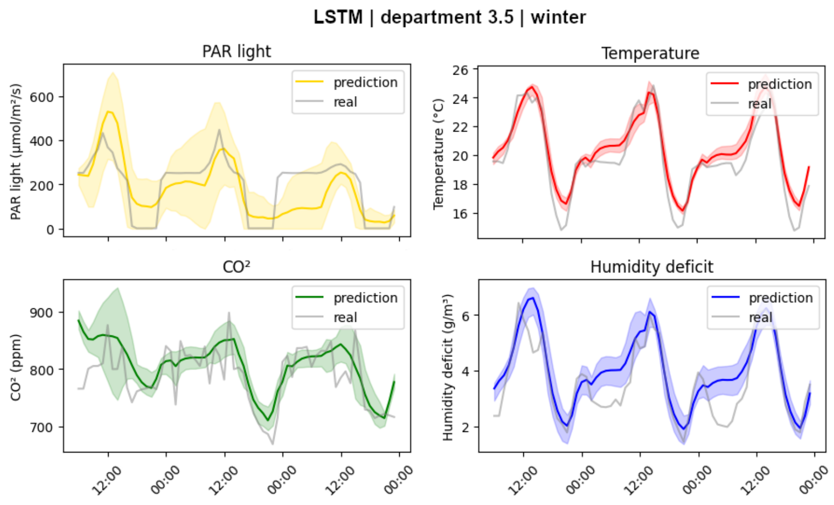


Figure C.2: LSTM inside climate prediction 25/12/2020 00:00 - 27/12/2020 23:00, department 3.5, resulting in an RMSE of 0.0314.

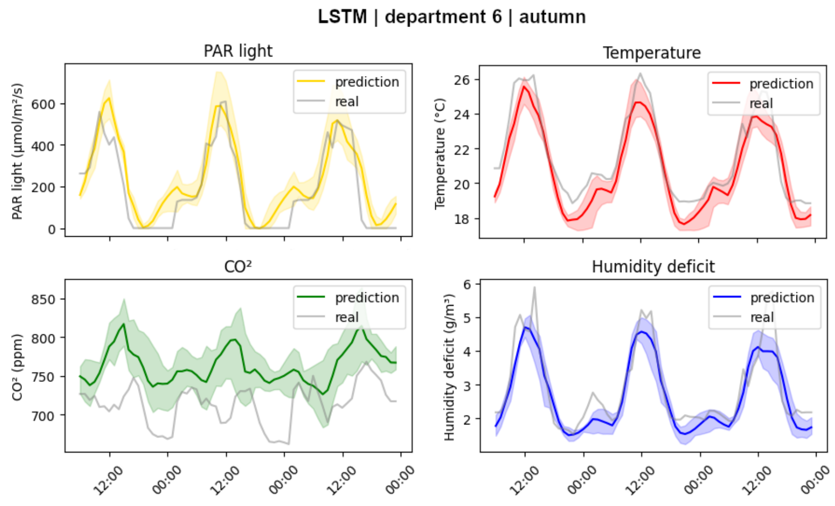


Figure C.3: LSTM inside climate prediction 10/10/2021 00:00 - 12/10/2021 23:00, department 6, resulting in an RMSE of 0.0278.

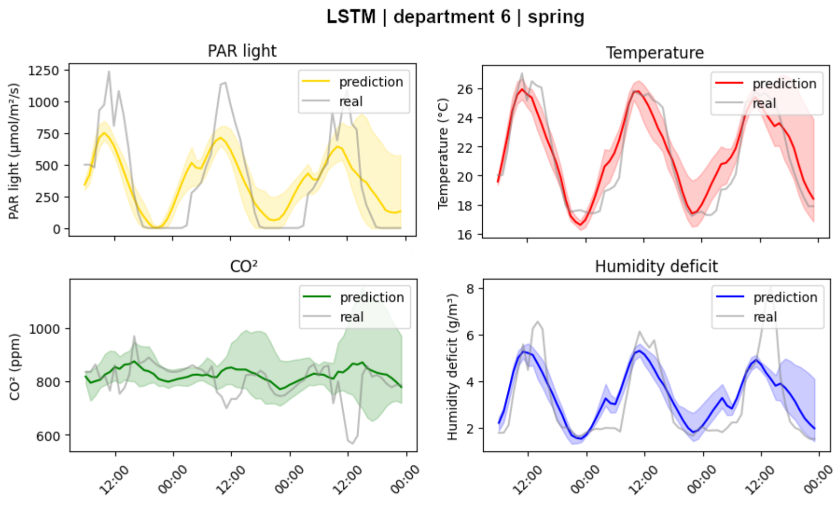


Figure C.4: LSTM inside climate prediction 10/4/2022 00:00 - 12/4/2022 23:00, department 6, resulting in an RMSE of 0.0524.

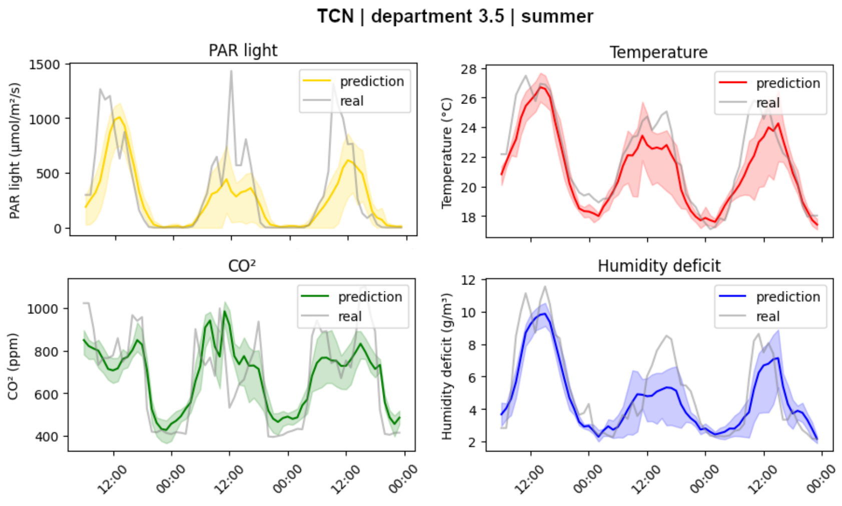


Figure C.5: TCN inside climate prediction 5/8/2021 00:00 - 7/8/2021 23:00, department 3.5, resulting in an RMSE of 0.0598.

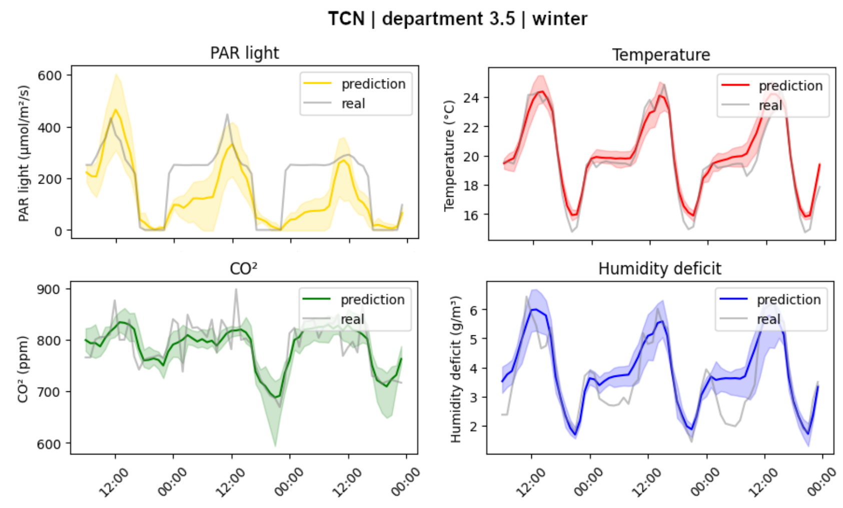


Figure C.6: TCN inside climate prediction 25/12/2020 00:00 - 27/12/2020 23:00, department 3.5, resulting in an RMSE of 0.0254.

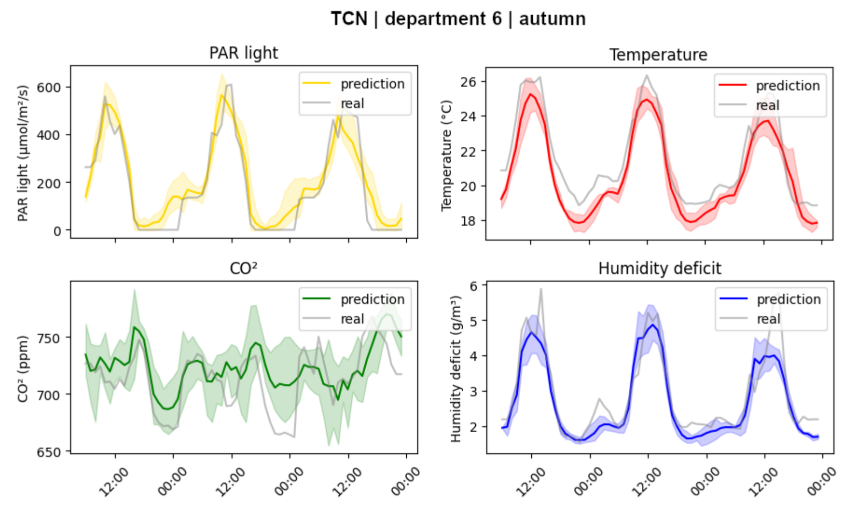


Figure C.7: TCN inside climate prediction 10/10/2021 00:00 - 12/10/2021 23:00, department 6, resulting in an RMSE of 0.0215.

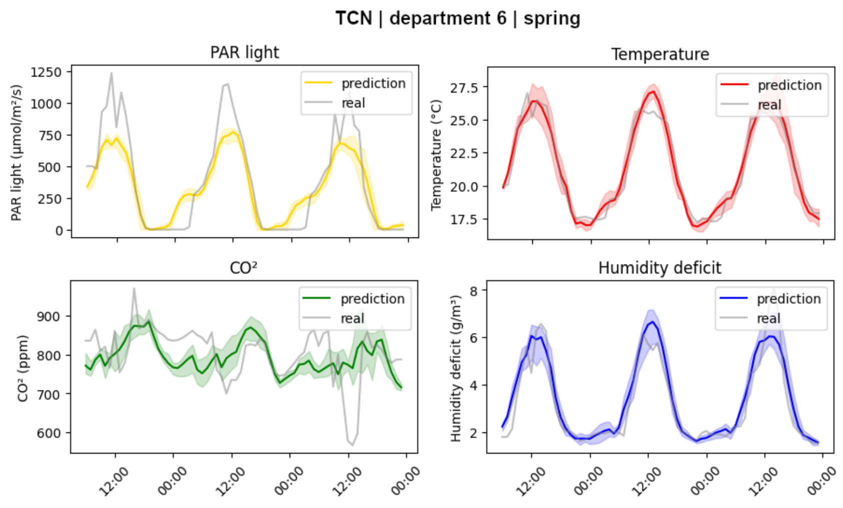


Figure C.8: TCN inside climate prediction 10/4/2022 00:00 - 12/4/2022 23:00, department 6, resulting in an RMSE of 0.0364.

D

Research Paper

Embedding a Long Short-Term Memory Network in a Constraint Programming Framework for Tomato Greenhouse Optimization

Dirk van Bokkem,[1] Neil Yorke-Smith, [1]Sebastijan Dumančić [1]Max van den Hemel [2]

[1] Algorithmics group, Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, The Netherlands

[2] Delphy Digital, Delphy BV, The Netherlands

d.vanbokkem@student.tudelft.nl, n.yorke-smith@tudelft.nl, s.dumancic@tudelft.nl, m.vandenhemel@delphy.nl

Abstract

The increasing global food demand, accompanied by the decreasing number of expert growers, brings the need for more sustainable and efficient solutions in horticulture. The controlled environment of greenhouses enable data collection and precise control. For optimally controlling the greenhouse climate, a grower not only looks at crop production, but rather aims at maximizing the profit. In this work, Constraint Programming (CP) is applied on the greenhouse economic optimal control problem, by leveraging a Long Short-Term Memory (LSTM) greenhouse climate model through a CP embedding. The contributions of this work are threefold; First, the greenhouse climate is modeled with an LSTM model. Secondly, this LSTM is embedded into a CP model. Lastly, the profit of the grower is optimized through this CP decision algorithm.

Introduction

With the rapidly increasing world population, effects of climate change, and decrease of expert grower knowledge, sustainable solutions in agri- and horticulture are needed to continue meeting the global food demands in the future. Data-driven greenhouses can play a big part in this solution, as the controlled environment allows for higher productivity, prolonged cultivation periods, leading to better crop yields (van Straten and van Henten 2010). Greenhouses are energy-intensive, but do provide a controlled environment for crops to grow closer to their location of consumption (Iddio et al. 2020). Data-driven cultivation in greenhouses can help both in meeting the increasing global food demands, as well as lowering the impact of said greenhouses on the climate.

Many user adjustable settings exist in modern day greenhouses, but there is little knowledge on the long-term economic consequences of these short-term decisions (van Straten and van Henten 2010). What could aid the grower in their decision making process, is a system that understands the effect daily greenhouse actions have on the eventual yield and profit. In this work, we present a decision support system (DSS) that leverages greenhouse climate and crop models.

While the potential benefits of data-driven cultivation seem obvious, this approach is not yet widely adopted by growers, who rely largely on their own experience in the field. Many decisions made by the grower are based on economic objectives, which is why an economic data-driven decision support system could aid in the adoption of such techniques. What is missing for growers, is an extension of their own experience; a tool that can help them understand long-term economic effects of short-term decisions (van Straten, Challa, and Buwalda 2000). Especially in times of uncertainty regarding energy and gas prices, growers are in need of economic decision support as experience alone might not suffice in making the best decisions.

An underexposed but promising method in developing such systems in a horticultural setting is Constraint Programming (CP). Also, modelling the greenhouse and crop is predominantly done with mathematical models, while the complexity of the greenhouse and crop provides a good playing field for machine learning models that can utilize sensor data. The interplay between such machine learning models and CP, is interesting in the complex greenhouse environment, as well as outside the horticultural domain.

In this work, a Long Short-Term Memory network (LSTM) is applied to the prediction of the greenhouse climate. The main contribution in this work is the embedding of such an LSTM in CP, such that the LSTM can be utilized in a CP DSS.

Background & Related Work

It is not the first time LSTMs are used in the context of greenhouse predictions. In this section, related work regarding greenhouse climate prediction using LSTMs is discussed, as well as other approaches for decision making.

A Long Short-Term Memory (LSTM) network is a recurrent NN (RNN), that finds its use mostly in language modelling, but is generally effective in domains with sequential data (Sherstinsky 2020). Its introduction was made in 1997, and its main purpose was solving the vanishing or exploding gradient problem, mainly by ”enforcing constant error flow through internal states of special units” (Hochreiter and Schmidhuber 1997). These ’special units’ are so-called LSTM cells. The propagation of values in the LSTM network goes through several gates (input, output, and forget gate) and updates the internal states (cell and hidden state) of the LSTM cell. The gates control the flow of information from the input to the output, and provide a way for the cell to ”forget” irrelevant information. Static weight matrices W and U for the input and hidden state respectively, along with bias vectors b are used in each of these gates, whose values represent the trained model. These weights and biases remain the same for each timestep. This results in the following calculations in an LSTM cell for one timestep t;

$$\begin{aligned}
i_t &= \sigma_r(W_i \cdot X_t + U_i \cdot h_{t-1} + b_i) \\
f_t &= \sigma_r(W_f \cdot X_t + U_f \cdot h_{t-1} + b_f) \\
c'_t &= \sigma_a(W_c \cdot X_t + U_c \cdot h_{t-1} + b_c) \\
o_t &= \sigma_r(W_o \cdot X_t + U_o \cdot h_{t-1} + b_o) \\
c_t &= f_t \odot c_{t-1} + i_t \odot c_t \\
h_t &= o_t \odot \sigma_a(c_t),
\end{aligned}$$

with input gate i, forget gate f, output gate o, cell state c, hidden state h, activation σ_a for the states and recurrent activation σ_r for the gates, where $\odot$ represents an element-wise multiplication.

In (Ali and Hassanein 2020), an LSTM is used to predict multiple greenhouse climate variables, such as temperature and relative humidity including their minimum and maximum values. Some experiments of finding the best parameters are discussed and the final predictions look promising. However, the figures suggest that there is some copying behaviour occurring. This tends to happen when the model relies too much on its previous value of the target variable, rather than also including other variables such as actions in the greenhouse.

Three time-serial deep neural network models are compared in (Jung et al. 2020). Out of an ANN, NARX model, and LSTM model, LSTM performed the best, with a standard error of the prediction (SEP) of temperature and CO_2 within 5%. However, the humidity prediction did not perform very well. An important but expected result of this research is that the accuracy of the time-based algorithm decreases as more time steps ahead are predicted. Nevertheless, it is shown that deep learning models can perform well in predicting the greenhouse climate.

Next to predicting the greenhouse climate, LSTM is also used for crop growth prediction. In (Lee et al. 2020), the Leaf Area Index (LAI) of bell peppers is predicted using LSTM. The LAI indicates the size of the leaf surface and in turn can be used to predict crop growth, but this is not explained in this paper. The model is trained on simulated data, but also validated with LAI data measured of a real crop, showing a high accuracy.

A more direct yield prediction approach is applied in (Alhnaity et al. 2020). Here LSTM is compared to a Support Vector Regression (SVR) implementation and an implementation of a Random Forest (RF). In all implementations, a one-step-ahead prediction was done. However, it is not clear if the full sequence shown is a multi-stage prediction (using the prediction of the current timestep to predict the next timestep), or if all predictions are simply based on the previous timestep. Of course, long-term yield predictions are more relevant. Since the environmental data is on an hourly basis and the yield data is gathered weekly, there is a mismatch which is dealt with by interpolation. While real hourly yield data is impractical to obtain, interpolating weekly data to hourly samples may not give accurate results of the effect of environmental factors on yield. Still, the results show the potential of yield prediction through an LSTM.

In (Lombardi, Milano, and Bartolini 2017) the concept of combining learned relations between variables from data and embedding these relations into an optimization model is coined Empirical Model Learning (EML). In an earlier study, the same authors explore this approach by embedding a trained Neural Network (NN) into a CP algorithm, for a ”temperature aware workload allocation problem”, with promising results (Bartolini et al. 2011). In their approach, a Neuron Constraint is added for each node in the network. The NN in this work is relatively simple, but the authors claim that the same techniques can be used for complex recurrent networks.

Economic Greenhouse Climate Control

The greenhouse environment is a complex, non-linear, multi-input-multi-output (MIMO) system. Multiple variables determine the inside climate, which in turn determines the state of the crop. The variables that determine the inside climate can be roughly divided into three categories; the current state of the inside climate, the outside climate, and the actions in the greenhouse such as opening the windows and injecting CO_2. The states are not defined by these dynamic relations only, but also depend on some static properties of the greenhouse and crop, such as the transmissivity of the greenhouse glass and the size of the greenhouse.

The main difficulty in modelling the greenhouse and crop and making decisions, lies in the varying time-scales. Opening the windows can cause a shift in the inside temperature within the hour, while the full lifespan of a tomato from flower to fruit ranges between one to two months (Tap 2000). To determine the best actions in the greenhouse, one has to understand the impact of short-term decision on the long-term yield.

Another challenge, is understanding the inter-dependencies between all these variables. Some decisions affect multiple state variables. For example, opening the windows has an effect on the temperature, humidity, and CO_2. But there also exist cyclical dependencies. The inside climate changes the state of the crop, while the crop in turn has an effect on the climate e.g. through transpiration (Tap 2000).

The greenhouse and thus the decision making process are also influenced by the weather and its uncertainty. In the near future, the weather can be forecasted with relative accuracy, but this becomes an issue when trying to make predictions and decisions for the long-term.

LSTM climate prediction

For the greenhouse climate a multi-stage prediction is done using an LSTM. This involves a step-by-step prediction where the prediction of one input-batch serves as part of the input for the next batch. This is especially useful in the context of CP, as the actions involved in the prediction need to be decided for each timestep, while the inside climate prediction depends on its previous state (see figure 1).

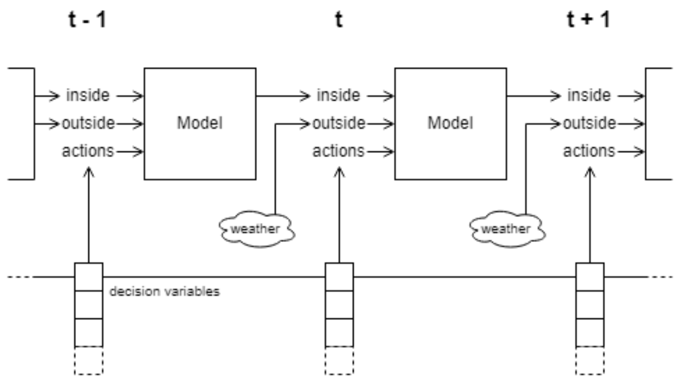


Figure 1: Multi-stage prediction in the context of greenhouse decision making. The input of each timestep is the inside climate, forecasted outside weather, and the decided actions of the previous timestep(s). The output is the inside climate for the next timestep.

The LSTM was trained using the following features:

Outside weather — radiation, temperature, absolute humidity, and wind speed.

Greenhouse actions — heating undertube, heating growtube, leeside ventilation, windside ventilation, LED lighting, SON-T lighting, and shading screen.

Inside climate — PAR-light, temperature, humidity deficit, and CO_2.

Combining LSTM with CP

We embedded the LSTM in CP through MiniZinc functions. These functions encompass the gate (input, output, forget) and state (cell, hidden) calculations. Pseudocode of the functions in MiniZinc is shown in Algorithm 1.

Various key points in CP modelling in MiniZinc contributed to the realisation of the LSTM-in-CP model. Firstly, the general way of modelling decision variables and relations within CP, is by defining them and comprise their dependence on other variables through constraints. While for small models, the MiniZinc compiler is able to derive this functional dependence and annotate variables accordingly, in larger models this is not always the case. Therefore we can help the solver by explicitly stating this functional dependence through a direct instantiation.

This may seem trivial and unimportant, but in the case of a more complex model such as the LSTM, this direct instantiation ensures that the solver simply calculates certain variables, and does not see them as decision variables to take into the search. In the background, the MiniZinc compiler adds the annotation pair *is_defined_var* and *defines_var()*, which together indicate that a variable is functionally dependent on another, and should not be decided. In our specific case, it means that we can specifically tell the solver to only search on greenhouse action variables, all the other variables should simply be the result of the LSTM calculations.

This advantage of direct instantiations was passed on in the choice of functions over predicates. Where predicates take in variables and constrain these, functions directly return a result. In the case of computing the gate functions in the LSTM for example, a function is preferred as this entails a direct computation instead of posing a constraint on a variable, where the solver would try to find values that satisfy the constraint. Also, an additional variable would be needed for such a predicate, needlessly making the model more complex. The same line of thought applies to not creating additional variables to hold the results of functions, but rather chain these function calls as much as possible. Chaining the functions not only reduced the amount of crashes as the problem size increased, it also improved the compiling and solving speeds greatly.

Algorithm 1: Pseudocode of the LSTM calculations in MiniZinc. The cell and hidden state are initialised to zero and for each timestep in the range 1 to lag, the cell and hidden state are calculated in separate functions. The gate calculation is generalised in a function as well.

```
function LSTM($X, W, U, b$)
    $c_0$ = 0
    $h_0$ = 0
    for t ∈ 1..l do
        $c_t$ = CELL($X_t, h_{t-1}, c_{t-1}, W, U, b$)
        $h_t$ = HIDDEN($X_t, h_{t-1}, c_t, W, U, b$)
    end for

    return $h_l$

function $CELL(X_t, h_{t-1}, c_{t-1}, W, U, b)$
    return
    $GATE(X_t, h_{t-1}, W_f, U_f, b_f, \sigma_r) \odot (c_{t-1}) +$
    $GATE(X_t, h_{t-1}, W_i, U_i, b_i, \sigma_r) \odot$
    $GATE(X_t, h_{t-1}, W_c, U_c, b_c, \sigma_a)$

function $HIDDEN(X_t, h_{t-1}, c_t, W, U, b)$
    return $GATE(X_t, h_{t-1}, W_o, U_o, b_o, \sigma_r) \odot \sigma_a(c_t)$

function $GATE(X_t, h_{t-1}, W_g, U_g, b_g, \sigma)$
    return $\sigma(W_g \cdot X_t + U_g \cdot h_{t-1} + b_g)$
```

Most LSTM computations were done in compliance with this direct instantiation strategy, but for the cell and hidden state this was not possible, as their structure is dependent on the chosen lag. As mentioned earlier, the cell and hidden state vectors can not be updated directly, but are encompassed in separate vectors for each timestep. To programmatically make this time-based computation work, both the cell and hidden state vectors were defined in 2D arrays, with timesteps as rows, and the chosen LSTM unit size u as columns, corresponding to vectors of length u for each timestep. In an iterative constraint, the values of the vectors are decided by a calculation that includes values of the vectors of the previous timestep.

Because the solver needs to find values for these vectors, it helps to explicitly state bounds on these, as tight as possible. This is also recommended in MiniZinc's documentation (mzn 2016-2020). We know that the hidden state in an LSTM holds values between -1.0 and 1.0, because of the tanh activation function. The bounds of the cell state are not that straightforward. From the LSTM state calculations, we know that c_t depends on f_t, c_{t-1}, i_t, and c'_t. The bounds for f_t and i_t are (0, 1), as a result of the sigmoid function, similarly c'_t has bounds of (-1, 1) due to the tanh function. Taking the cell state calculations in mind, the bounds for the cell state increase at each timestep. The resulting bounds thus correspond to the lag l, so $(-l, l)$.

One of the key points in building the CP model, was the implementation of a search strategy. In a search strategy, a CP modeller can indicate how the solver should perform the search for feasible and optimal solutions. This involves both the order of deciding variables and what values these variables should be initialised with. The order of searching is essential in solving a CP model that involves time-based decision variables, as is the case with our multi-stage prediction with an LSTM. Since the inside climate of a certain timestep depends on a multitude of variables from previous timesteps, it helps the solver to know which variables to decide first, before moving on to the next. We implemented this order of searching using MiniZinc's *priority_search* annotation (Feydy et al. 2017), in which we placed a list of *int_search* annotations, each including all greenhouse action variables for one timestep. The solver thus first searches values for the decision variables of the first unknown timestep, before searching values for decision variables of the next timestep.

CP Decision Algorithm

The greenhouse and crop model, their embedding into CP, and the CP model itself ultimately result in a complete economic greenhouse DSS. This system models the complete greenhouse environment and optimises the decisions based on the economic aspect of a tomato cultivation. How this complete system operates and how the various components interact is visualised in Figure 2. The parameters and variables in the system are denoted by the letters O, A, I, C, and E, respectively representing the outside weather, greenhouse actions, inside climate, crop production, and economics. As can be seen in the figure, some of these components are simply retrieved, others are predicted through the embedded models.

First, we modelled the outside weather by arrays with a length of the amount of timesteps, with normalised parameters (bounds of (0, 1)). They are parameters, because to the DSS they are known values. They are retrieved from forecast data and will not be predicted or decided within the system. Next, we modelled the actions in the greenhouse by a known and unknown part. The amount of known timesteps are the chosen lag of the LSTM model. The LSTM model needs these first "lag" inputs to start the multi-stage prediction process. The known part is a parameter array with normalised floating point numbers that represent the first actions that are already decided, the unknown part is a decision variable array with integers that represent the decisions that the system will make. We discretise using integers, to reduce the

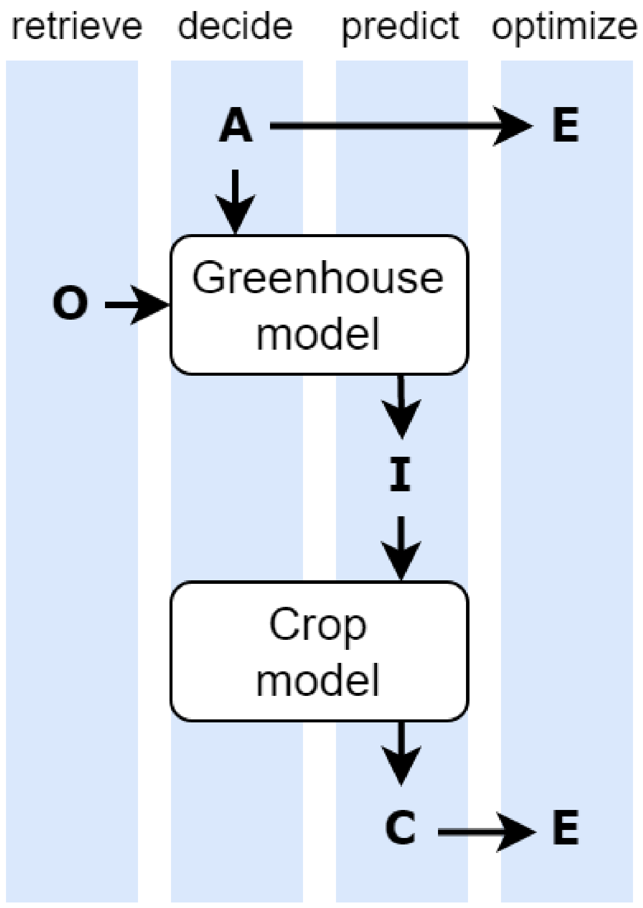


Figure 2: Overview of the CP model and its relevant parameters and decision variables. The outside weather (**O**) is retrieved from a weather forecast, the greenhouse actions (**A**) must be decided, so that the inside climate (**I**) and crop production (**C**) can be predicted through the embedded models, finally leading to an economic result (**E**) that is used for the optimisation process.

amount of possible values, improving the efficiency of the solver. We discretise based on the amount of values a decision variable may have, using a base of 10. The bounds for the unknown part of one action variable are $(0, 10^p)$, where p is the precision. The inside climate was also modelled as an unknown and known part, but both are floating point numbers. An additional integer array was added and connected to the floating point array through constraints. The predicted inside climate values are rounded within the LSTM prediction process. By rounding the inside climate values within the LSTM, the solver is able to match the floating point numbers to their integer counterparts. The outside weather, greenhouse actions, and inside climate variables all come together in the greenhouse model input 2D array, which is fed as input to the LSTM model.

When the complete CP model has been set up, including all greenhouse and crop variables, their prediction functions, additional constraints, and economic variables, the Ja-

CoP solver can run the model. The objective function that is to be maximised is the profit realised within the given period, which we modelled by computing the revenue from crop growth, and from this revenue subtracting the costs that follow the greenhouse actions. A MiniZinc instance is created, including the greenhouse, crop, and complete model and data file. With a given timeout, the solver will search for solutions that satisfy all constraints, while maximising the profit. When an optimal solution is found or there is a timeout, the best solution will be saved as a data file that includes the outside weather, decided greenhouse actions, and predicted inside climate for the given period. Solving statistics and the objective values are saved as well. To speed up the search, a step-size for assessing objective function values relative to the amount of timesteps was used. By using this step-size, the solver can skip intermediate solutions. A profit step-size of 0.5 eurocents per hour was used.

Results & Discussion

LSTM greenhouse climate prediction

To validate both the LSTM, various experiments were performed to compare the predicted climate with the actual climate in the greenhouse. The LSTM was trained with the parameters obtained in a grid search and predictions were done in both departments in two different season periods; summer for department 3.5, autumn for department 6.

For department 3.5 we trained an LSTM model on all relevant features of department 3.5, with a batch size of 512, a lag of 6 timesteps (6 hours), and 4 LSTM units. For the experiments of department 6 we used a batch size of 256, a lag of 6 timesteps (6 hours), and 4 LSTM units. We performed three-day multi-stage predictions of the inside climate using the prediction dataset of summer in department 3.5 and autumn in department 6. Each experiment was run 5 times and the result was averaged.

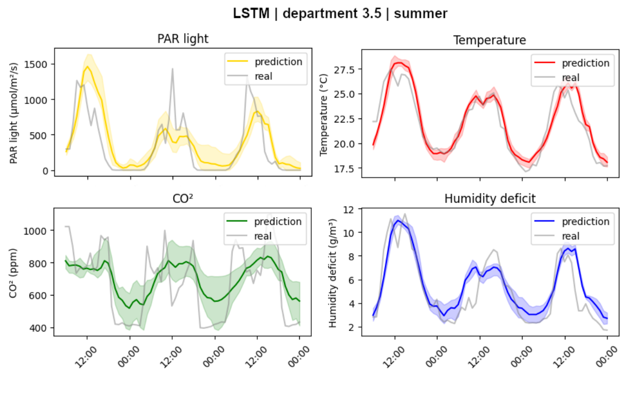


Figure 3: LSTM inside climate prediction 5/8/2021 00:00 - 7/8/2021 23:00, department 3.5, resulting in an RMSE of 0.065.

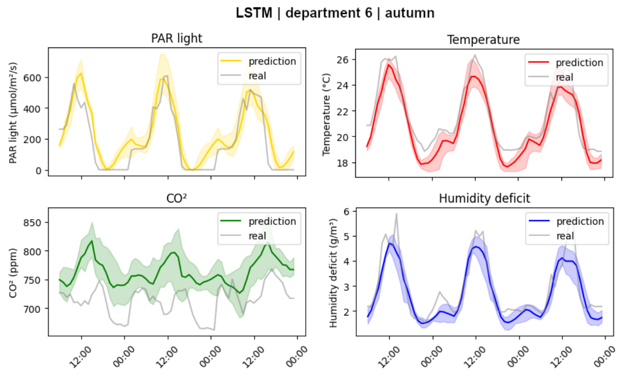


Figure 4: LSTM inside climate prediction 10/10/2021 00:00 - 12/10/2021 23:00, department 6, resulting in an RMSE of 0.0278.

Figure 3 shows the LSTM prediction in the summer in department 3.5. What we see is that the LSTM model is able to predict the temperature and humidity deficit quite well in all five runs, but it has some trouble with predicting the inside PAR light, which is probably caused by clouds or other shadows formed on the PAR-sensors. Because we have data of outside radiation and artificial lighting, we will perform a simpler computation instead. Lastly, CO_2 is the hardest target feature for the model to predict. Primarily, CO_2 levels recorded by the sensor are more fluctuating than that of the other climate variables. Furthermore, how much CO_2 is in the air also depends on crop processes, which is not a direct input feature. Unfortunately, the additional knowledge of amount of CO_2 injection did not help the CO_2 prediction, as we can see in Figure 4. The temperature and humidity predictions are still quite accurate.

LSTM-in-CP Validation

To make sure the LSTM embedding works properly, we performed validation experiments. The possible actions in the greenhouse like the heating tubes and window openings are the decision variables in the CP model, but for these experiments we fix these variables such that the actions are decided. In this way, the LSTM within the CP model should behave similarly as a regular LSTM.

We trained an LSTM model on cultivation data from 28/12/2020 to 4/8/2021, with all outside weather features, relevant action features for department 3.5, and inside features. It was trained with a batch_size of 512, lag of 6 timesteps (hours), adam optimizer, loss function MSE, and one LSTM layer with 4 units. After training, the LSTM was embedded into a CP model. The chosen prediction period was 5/8/2021 00:00 to 7/8/2021 23:00 (72 timesteps), with the values of all action variables decided. Some discrepancy in the output is expected, as the action variables are discretised. The experiment was run 5 times and averaged.

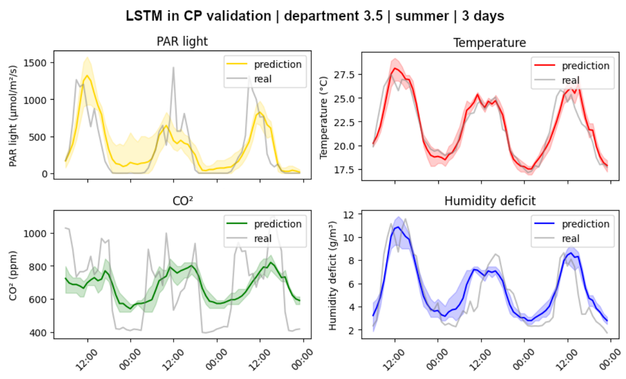


Figure 5: Validation of the greenhouse LSTM model embedded in CP for a 3-day prediction period on the inside climate. The first 6 timesteps (lag) of the inside climate are known, the remaining timesteps are predicted in a multi-stage fashion within CP. The outside weather and action variables of all timesteps are filled in.

In Figure 5 we see the LSTM prediction of each of the inside climate variables within CP for a prediction period of 3 days, with an RMSE of 0.0665. The figure shows similar results as the ”regular” LSTM predictions with a similar RMSE value as well, indicating that the embedding LSTM in CP is behaving as expected. These experiments merely show the correctness of the LSTM embedding, the accuracy of the model remains the same, as we can see from the poor predictive power for CO_2.

Conclusion

Many mathematical models exist that model the greenhouse and crop. Through multiple equations, the physical relations between the greenhouse and crop are modelled. However, these models are complex and need many parameters, making their implementation impractical. Therefore we modelled the greenhouse with a Long Short-Term Memory (LSTM). We embedded this LSTM in CP in this study and used this to model a CP decision support system. The short-term decisions in this DSS were represented by the possible greenhouse actions that influence the inside climate, modelled as decision variables per timestep in a CP model. The long-term profit was modelled by the revenue obtained from the predicted yield in the Lintul-3 model that followed from the predicted climate, minus the aggregation of costs per timestep of each action decision variable.

The embedding of an LSTM in CP was realised through multiple MiniZinc functions, of which key points in its realisation were discussed. Firstly, by directly instantiating variables through their functional dependence on other variables, we explicitly tell the solver not to perform a search on these variables. This leads to the solver only performing a search on the input variables of the LSTM, not simultaneously on the output variables. Also, functions were preferred over predicates, as no additional intermediate variables are needed that the solver tries to perform a search on. Furthermore, since the internal cell and hidden state of the LSTM cell are dependent on the chosen time lag, predicates needed to be used instead of functions. To help the solver solve these computations, explicit bounds as tight as possible were used on these intermediate variables. Lastly, since the multi-stage prediction is a sequential process, it was essential to inform the solver the order of solving.

The DSS was designed to optimise the grower's profit through minimising the costs that follow the actions taken in the greenhouse and maximising the revenue that follows from the predicted climate and subsequently the crop yield. Using this DSS can help growers make better decisions in the future, as well as helping them understand what the DSS would do differently in past cultivation periods. Unfortunately however, the DSS is not always performing better than the grower in terms of resulting profit. This is due to the search space being too large for the solver to find a better solution within reasonable time. With the current state of the system, the DSS will thus not necessarily help growers make better decisions. For small problem instances however, we have seen improvements of the made decisions resulting in more profit. Here we have also seen that the DSS can help growers make well-informed decisions when gas or electricity prices rise, by turning off the lights for example. The potential of the DSS has thus been shown on small instances, but needs further improvement to more quickly find good solutions.

While the applicability of CP in economic greenhouse decision making has been shown for small instances, before it can be applied properly on real-world instances some of the limitations need to be addressed. First of all, the implemented greenhouse LSTM model is not able to predict CO_2 and light well. CO_2 levels are fluctuating relatively fast and not as smooth as temperature or humidity and they depend on crop processes, making it harder to predict. The poor light prediction is likely the result of the PAR-sensors that are shadowed by clouds or other shadow-forming objects in the greenhouse. Secondly, the crop model is very basic and is does not include long-term relations. Lastly, the DSS itself is not yet usable in practice, mainly because the large search space makes it hard for the DSS to find good solutions. Possible directions for future work include improving the crop model by implementing an LSTM.

Implementing a warm start, i.e. doing an initialisation with a known good solution, could improve the DSS greatly. For example, this initialisation could be decisions a grower has made in similar weather conditions. Firstly, since the grower almost always outperforms the DSS in large instances with short timeouts, starting with decisions similar to the grower would provide a head start. Secondly, when the DSS is provided with a solution that is already good, the bounds can be set tighter around this solution resulting in a reduction of the search space.

With an iterative refinement method, the DSS first can be run with low precision both in terms of climate and crop predictions as well as timestep frequency, so that a rough estimate of a good solution can be found. Then by using this found solution, the DSS can be run again with slightly higher precision but tighter bounds. By repeating this process, the system may converge to a good solution much quicker.

Currently, the costs of operating the greenhouse actions are assumed by looking at historical values. Similarly, the price of tomatoes is estimated by using a historical price, and additionaly it is increased or decreased according to the month of the year. However, these economic values are computed in advance and fixed for each timestep. A more realistic DSS would include arrays of costs and prices that show their varying values for each timestep. Also, future work could include predicting these values instead of making assumptions.

References

2016-2020. The MiniZinc Handbook. https://www.minizinc.org/doc-2.5.5/en/index.html. Accessed: 2022-03-09.

Alhnaity, B.; Pearson, S.; Leontidis, G.; and Kollias, S. 2020. Using deep learning to predict plant growth and yield in greenhouse environments. *Acta Horticulturae*, 1296: 425–431.

Ali, A.; and Hassanein, H. 2020. Time-Series Prediction for Sensing in Smart Greenhouses. In *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 1–6.

Bartolini, A.; Lombardi, M.; Milano, M.; and Benini, L. 2011. Neuron Constraints to Model Complex Real-World Problems. volume 6876, 115–129. ISBN 978-3-642-23785-0.

Feydy, T.; Goldwaser, A.; Schutt, A.; Stuckey, P.; and Young, K. 2017. Priority Search with MiniZinc. *The Sixteenth International Workshop on Constraint Modelling and Reformulation at CP2017.*

Hochreiter, S.; and Schmidhuber, J. 1997. Long Short-term Memory. *Neural computation*, 9: 1735–80.

Iddio, E.; Wang, L.; Thomas, Y.; McMorrow, G.; and Denzer, A. 2020. Energy efficient operation and modeling for greenhouses: A literature review. *Renewable and Sustainable Energy Reviews*, 117: 109480.

Jung, D.; Kim, H.; Jhin, C.; Kim, H.; and Park, S. 2020. Time-serial analysis of deep neural network models for prediction of climatic conditions inside a greenhouse. *Computers and Electronics in Agriculture*, 173: 105402.

Lee, J.; Kang, W.; Moon, T.; Hwang, I.; Kim, D.; and Son, J. 2020. Estimating the leaf area index of bell peppers according to growth stage using ray-tracing simulation and a long short-term memory algorithm. *Horticulture, Environment, and Biotechnology*, 61.

Lombardi, M.; Milano, M.; and Bartolini, A. 2017. Empirical decision model learning. *Artificial Intelligence*, 244: 343–367. Combining Constraint Solving with Mining and Learning.

Sherstinsky, A. 2020. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D: Nonlinear Phenomena*, 404: 132306.

Tap, R. 2000. *Economic-based optimal control of greenhouse tomato crop production*. Ph.D. thesis, Wageningen University.

van Straten, G.; Challa, H.; and Buwalda, F. 2000. Towards user accepted optimal control of greenhouse climate. *Computers and Electronics in Agriculture*, 26(3): 221–238.

van Straten, G.; and van Henten, E. 2010. Optimal Greenhouse Cultivation Control: Survey and Perspectives. *IFAC Proceedings Volumes*, 43(26): 18–33. 3rd IFAC Conference in Modelling and Control in Agriculture, Horticulture and Post-Harvest Processing - Agricontrol.

Acknowledgments