Reducing the runtime of an NP-Hard algorithm using deep learning on historical data

Christoffer Lindkvist

September 22, 2025

Abstract

A mixed-model assembly line manufactures different product variants on a single line, where variations in tasks can create imbalances across workstations. When several labour-intensive models appear consecutively, some stations may exceed their capacity, leading to overloads. This challenge is formalized as the Product Sequencing Problem, an NP-hard optimization task concerned with arranging production orders into efficient sequences. This thesis investigates whether deep learning can complement heuristic methods for solving this problem. Using a Sequence-to-Sequence model trained to emulate tacit scheduling knowledge captured in historical data, and using its predictions to initialize a heuristic algorithm. By providing informed starting points, this approach aims to reduce the computation time of the algorithm.

Contents

1 Introduction **2**
1.1 Background . . . 2
1.2 Research Problem . . . 2
1.3 Objectives . . . 2
1.4 Visualization of the Problem . . . 3

2 State of the art analysis **5**
2.1 LSTM . . . 5
2.2 JSON to Vector Methodology . . . 5
2.3 Seq2Seq / Sequence to Sequence Models . . . 6
2.4 Transformer-based Approaches . . . 6

3 Methodology **7**
3.1 The Heuristic Approach . . . 7
3.2 Complementing the Heuristic Approach using Machine Learning . . . 7
3.3 System Architecture . . . 8

4 Experiments and results **9**
4.1 Dataset . . . 9
4.2 Evaluation Metrics . . . 9
4.3 Runtime Analysis . . . 9

5 Discussion **10**
5.1 Interpretation of Results . . . 10
5.2 Limitations . . . 10
5.3 Future Work . . . 10

Chapter 1

Introduction

1.1 Background

This thesis addresses the Product Sequencing Problem, an NP-Hard optimization problem arising in the planning of mixed-model assembly lines [6]. Traditionally, product sequences are determined manually by management staff, relying primarily on tacit knowledge accumulated through experience. While this often produces feasible solutions with relatively few overlaps, it remains ad hoc and sporadic.

To improve upon this, a heuristic algorithm is proposed. The algorithm first generates a feasible baseline solution from pre-defined constraints, and then it refines the baseline until an acceptable sequence is reached. The central question of this thesis is whether the runtime of such an algorithm can be reduced by initializing it with a baseline informed by historical sequencing data, rather than relying solely on rule-based construction.

1.2 Research Problem

The Product Sequencing Problem is classified as NP-Hard. Current approaches in practice rely entirely on human expertise and tacit knowledge, which limits scalability and consistency. If this knowledge could be systematically emulated using a model that mimics historical sequencing data, and in effect tacit knowledge, it may provide stronger starting points for the heuristic methods. Such informed baselines could potentially reduce the runtime required to reach high-quality solutions, especially for complex sequencing instances.

1.3 Objectives

The objectives of this thesis are:

1. To design a deep learning model capable of emulating the tacit knowledge of management workers using historical data.

2. To integrate the model as a preprocessing step in the existing heuristic algorithm in order to reduce its runtime.

3. To provide a visualization tool that intuitively illustrates the scheduling flow, highlighting overlaps, borrowed time, and bottlenecks across stations and time.

1.4 Visualization of the Problem

The UI will visualize the flow of the assembly line on two axes. One per station, and one in clockcycles. One clockcycle is the time it takes the theoretical items to make it from one station to the next. Hence the items must be displayed in a way that conveys that some stations take longer and shorter time to complete. In Figure 1.1 this is shown by stretching the items on the stations to better fit the clock cycles. One clockcycle is defined as a rudimentary unit of time, the size is arbitrary, and it does not reflect real life. For the sake of this thesis, we'll assume that one clockcycle is the time it takes for an arbitrary item X to make it from S_n to S_{n+1} in one clockcycle, i.e. T to $T + 1$.

Issues in visualizing this way start to appearing when we start to consider that different stations S_n and S_m may take different times to complete. If we then step a clockcycle for each possible item, then we can never keep our items in sync. The main issue is that; if we compare the station S_n and S_m, then we'll see that each station have a different time to finish, then the clockcycle system will not be perfect or even realistic as stations with differing times will each finish in different times and thus an item X might make it to the station S_{n+2} from S_n in the same time it takes item Y to make it to S_{m+1} from S_m.

Thus we find the difficulty in displaying it properly in an intuitive graphical user interface. If we wish to display each station as uniform sizes, then we also have to stretch the items to make up for it visually. But doing this we have no intuitive way of knowing that S_4 could be 20 seconds long in real life, while S_3 could be 90. *As luck would have it, each station in this specific case are each roughly 7 minutes long*, thus we will not run into any major desync problems using clockcycles on these stations.

Between each station lies a buffer zone called a "drift area". A drift area in this case is the transitional area between any given station S_n and S_{n+1}. Both of the stations can borrow time from each other within this area, but only one station may utilize that area at the time. This proves useful to help fit items that take longer on some stations onto the assembly line, but they are also the source of most problems.

As pictured in Figure 1.1, D will take a lot of time on S_4 and is forced to utilize time from S_3 and S_5. While this works well in a vacuum, the problems start to arise when

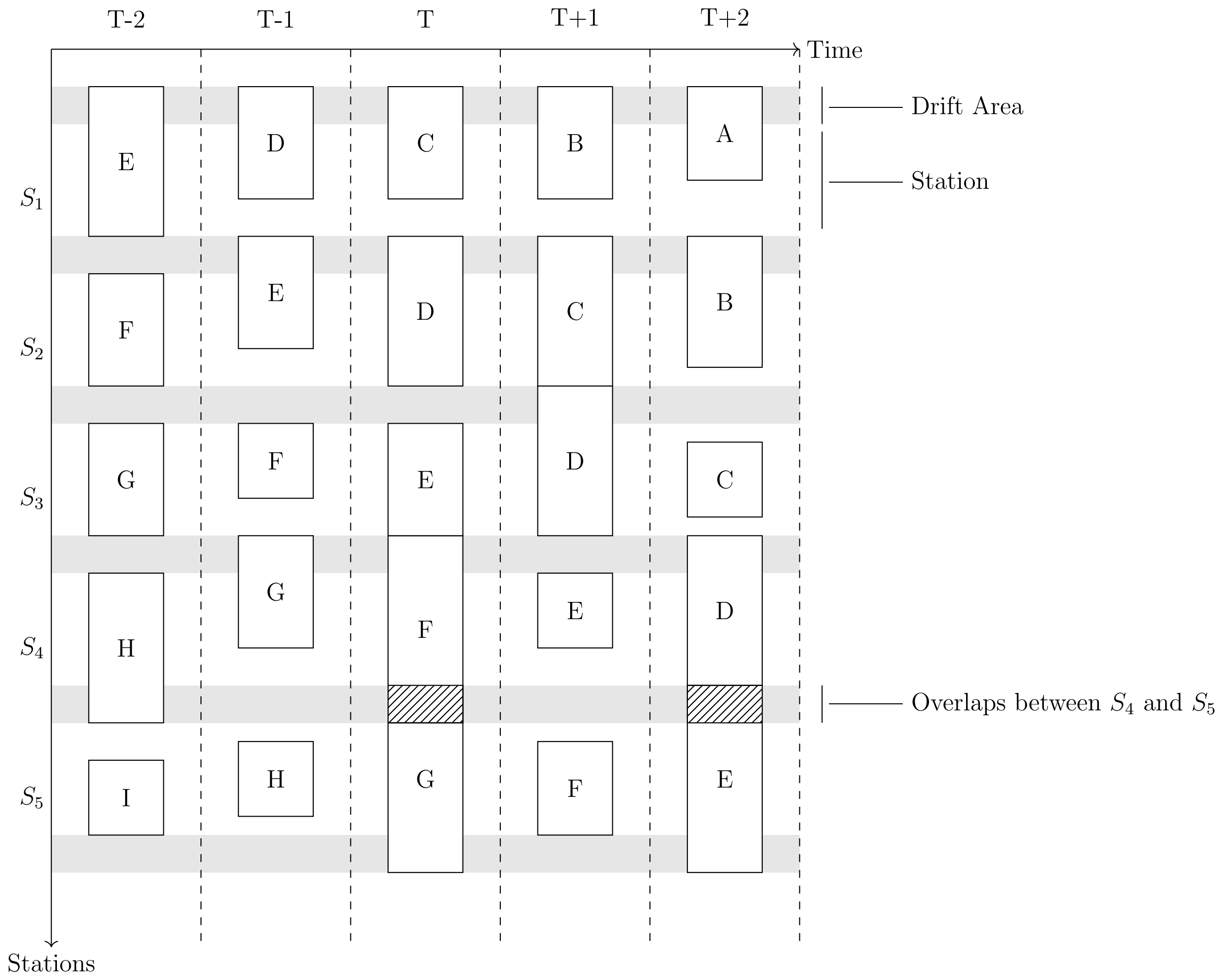

Figure 1.1: Assembly Line Example with Uniform Station and clockcycles

E also has to utilize additional time from its neighbouring stations, causing an overlap between D and E at $T + 2$ as they both require the use of the drift area.

The same problem arises with F and G at T as both items need to borrow time from the stations before and after. Thus we run into another overlap.

Do note that on T, E does not utilize the drift area which results in it sitting flush with F on the timeline, this may look good on paper but can result in overlap in practice due to the human workers at the assembly line occasionally taking a bit longer than presumed. This can be resolved by borrowing some time from S_2 and moving E into the drift area.

The same issue arises at $T + 1$ where C and D just barely get enough time, but it cannot get resolved by simply moving D forward, as D on $T + 2$ will require all of the time it can get on S_4.

Chapter 2

State of the art analysis

2.1 LSTM

In previous works addressing similar scheduling problems, researchers have applied Recurrent Neural Networks (RNNs), often with Long Short-Term Memory (LSTM) units, in a sequence-to-sequence (Seq2Seq) framework. Seq2Seq models are traditionally used in language-based tasks such as machine translation or text summarization, but the same architecture can be adapted to the assembly line problem.[2] Each item can be represented as a vector, and a day's worth of incoming items forms an input sequence of vectors. The Seq2Seq model can then be trained to output a corresponding placement sequence, effectively learning to replicate the tacit knowledge of the management workers in arranging the production items to minimize overlaps.[3]

The interesting part about LSTMs is that they can selectively forget irrelevant or outdated information through their forget gate. This helps them focus on more relevant patterns in the data over time, improving their ability to model long-term dependencies. [4]

2.2 JSON to Vector Methodology

Machine learning models operate on numerical vector data rather than raw JSON structures. [?] Given a list of JSON objects, such as trucks, each JSON can be encoded into a fixed-length vector x_t by extracting and normalizing its features. This transforms the entire list into a sequence of vectors $[x_1, x_2, \ldots, x_N]$. Using sequences from historical data, we can train a sequence-to-sequence model (Seq2Seq) which will learn to map an input order of JSONs to a desired output order.

2.3 Seq2Seq / Sequence to Sequence Models

Seq2Seq models, commonly based on encoder/decoder architectures of RNNs such as LSTMs, are designed to transform one sequence into another, hence the name. Usually these are found in language translation models, but can be as applicable in this thesis as we want to take one sequence of items and transform it into another. In our case, the input sequence represents the original order of vectorized JSONs, and the output sequence represents the target (reordered) sequence. [2][3] This approach allows the model to learn patterns in reordering and can assist the rearrangement process for new data when the tacit knowledge is emulated.

2.4 Transformer-based Approaches

While Seq2Seq models with LSTMs have been effective for sequence learning tasks, they suffer from limitations in handling very long sequences due to vanishing gradients and sequential processing bottlenecks. Transformer-based architectures address these shortcomings by discarding recurrence entirely and instead relying on a self-attention mechanism as an alternative to the Seq2Seq approach.

The key innovation of Transformers is the use of scaled dot-product attention, extended through multi-head self-attention which allows the model to weigh the importance of each element in the sequence relative to all others, regardless of distance. This parallelized computation not only accelerates training but also enables the model to capture global dependencies more effectively than RNN-based methods. Positional encodings are added to the input vectors to retain order information, since Transformers themselves are order-agnostic.[5]

For scheduling problems, this could translate into the ability to model complex interactions between items across the entire planning horizon. For example, the placement of one item can be directly conditioned on all others in the same day's sequence, not only its immediate neighbours. Such encompassing context awareness can prove particularly valuable in assisting assembly line scheduling.

Chapter 3

Methodology

3.1 The Heuristic Approach

The problem to properly order manufacturing assembly lines with as few overlaps as possible is considered an NP-Hard problem, as it is an optimization problem.

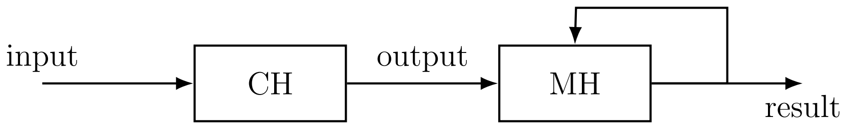

Figure 3.1: Heuristic solution

The Algorithm designed to solve this problem is a heuristic solution that will be made out of a Construction Hieuristic (*CH*) that produces a starting point based on pre-defined constraints, that feeds into a Meta Heuristic (*MH*) that finds a better solution starting from the output of the construction heuristic and self-improving until an acceptable result is returned.

3.2 Complementing the Heuristic Approach using Machine Learning

Due to the fact that management workers today place the items manually using tacit knowledge that they have accumulated over the years, then what this thesis proposes is to emulate that same knowledge by learning which placement patterns tend to work together and which do not.

The idea is that if they have knowledge of an adequate solution from the get-go with some risk of overlap, then we can train a Deep Learning Model (*ML*) on such previous data to give the algorithm a better starting point, thus (in theory) reducing the runtime of that algorithm.

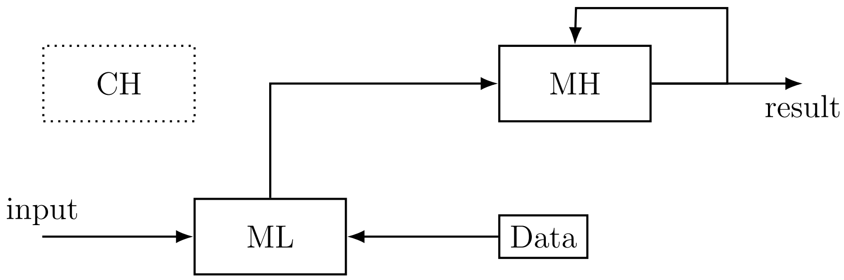

Figure 3.2: ML solution

However it is worth to consider that such an approach can prove redundant or yield worse results if the problem at hand is an easy problem where many solutions can be found quickly, as opposed to a hard problem where a desired solution may not even be found.

3.3 System Architecture

Chapter 4

Experiments and results

4.1 Dataset

4.2 Evaluation Metrics

4.3 Runtime Analysis

Chapter 5

Discussion

5.1 Interpretation of Results

5.2 Limitations

5.3 Future Work

Bibliography

[1] J. Abbasi, *Predictive Maintenance in Industrial Machinery using Machine Learning*, Master's thesis, Luleå University of Technology, Department of Computer Science, Electrical and Space Engineering, 2021.

[2] A. Dupuis, C. Dadouchi, and B. Agard, *A decision support system for sequencing production in the manufacturing industry*, *Computers & Industrial Engineering*, vol. 185, p. 109686, 2023.

[3] J. Lindén, *Understand and Utilise Unformatted Text Documents by Natural Language Processing Algorithms*, Master's thesis, Mid Sweden University, Department of Information and Communication Systems (IST), Spring 2017.

[4] Ian Pointer, *Programming PyTorch for Deep Learning*, O'Reilly Media, 2019.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is All You Need*, Advances in Neural Information Processing Systems (NeurIPS), vol. 30, 2017.

[6] C. Fink, O. Schelén, and U. Bodin, *Work in progress: Decision support system for rescheduling blocked orders*, Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, 2023.

[7] Author, Title, Journal, Year.

[8] Author, Title, Journal, Year.

[9] Author, Title, Journal, Year.

[10] Author, Title, Journal, Year.

[11] Author, Title, Journal, Year.

[12] Author, Title, Journal, Year.

[13] Author, Title, Journal, Year.

[14] Author, Title, Journal, Year.

[15] Author, Title, Journal, Year.

[16] Author, Title, Journal, Year.

[17] Author, Title, Journal, Year.

[18] Author, Title, Journal, Year.

[19] Author, Title, Journal, Year.

[20] Author, Title, Journal, Year.