Comparison of a Visual Analytics Simulation with Real World Manufacturing Lines

Rafael V. Ferrari

Course of Study: Computer Science
Examiner: Prof. Dr. Thomas Ertl
Supervisor: Dipl.-Inf. Dominik Herr

Commenced: May 5, 2016
Completed: September 19, 2016
CR-Classification: I.3.8, I.6.6, I.6.7
Abstract

Global competition between manufacturing companies forces them to offer ever more customizable products to their clients, while keeping high production levels. To assure companies thrive in this environment, they store large amounts of data from their production facilities and conduct simulations in order to optimize their operations. Analysts need to use techniques to deal with large amount of data efficiently and without being overwhelmed by it. Visual Analytics handles exactly that challenge. The goal of this thesis is to develop an approach that help the user gain knowledge that is useful for conducting simulation studies faster. In this thesis, a visualization assists analysts in calibrating and validating settings using historical data. The Critical Path Method was successfully applied to precisely trace bottlenecks in a simulated manufacturing line. In addition, new metrics were developed to guide the optimization of schedules where the analyst may control the performance of resources over an entire sequence of jobs, but not any of them individually, such as in manufacturing machines. The important metrics are visualized and the sequence of operations was displayed by an interactive lens.
## Contents

1 Introduction ................................................................. 9

2 Theoretical Foundations .................................................. 11
   2.1 Conceptual Model & Simulation .................................... 11
   2.2 Model Evaluation ..................................................... 12
   2.3 Critical Path Method (CPM) ........................................... 12
   2.4 Probability Distributions ............................................. 14
   2.5 Normal Distribution .................................................. 14
   2.6 Machine Learning ..................................................... 15
   2.7 Density Estimation .................................................... 16
   2.8 Information Visualization ............................................ 17
   2.9 Visual Analytics ..................................................... 18

3 Related Work ..................................................................... 21
   3.1 Simulator Systems ...................................................... 21
   3.2 Visualization Techniques ............................................. 21
   3.3 Bottleneck Identification Methods .................................. 22

4 Concept ........................................................................... 25
   4.1 Preliminary Description of the Simulator Software ............ 25
   4.2 Manufacturing Company Collaboration ........................... 27
      4.2.1 Expert’s Preliminary Considerations .......................... 27
      4.2.2 Provided Historical Data ....................................... 27
   4.3 General idea .............................................................. 28
   4.4 Extend the Conceptual Model Capabilities ......................... 28
      4.4.1 Introduce a New Station Type for Assemblies ............ 29
      4.4.2 Introduce Variability in Simulated Durations .............. 30
   4.5 Improve the Workflow of Specifying Common Manufacturing Settings .................................. 31
      4.5.1 New Editing Possibilities on the Project Tree and the Attributes View .......... 31
      4.5.2 Distribution Calibration and Validation ....................... 32
   4.6 Support the Task of Layout Optimization .......................... 32
      4.6.1 Analyzing Bottlenecks in Production ......................... 33
5 Implementation
5.1 Extend the Conceptual Model Capabilities
5.1.1 A new Assembly station
5.1.2 Minor correction of step durations
5.1.3 Time penalty to simulate production changeover
5.1.4 Introducing variability to step durations
5.2 Improvements in Settings Configuration Workflow
5.2.1 Reformulate the list of steps available for configuration
5.2.2 New Editing Possibilities on the Project Tree and the Attributes View
5.2.3 Distribution Calibration and Validation
5.3 Supporting the Task of Layout Optimization
5.3.1 Metrics Summary Table
5.3.2 Production Flow and Operation Details Visualization
5.3.3 Applying the Critical Path Method
5.3.4 Resource Drag

6 Evaluation
6.1 Create and Configure Stations
6.2 Calibrate Time Distributions
6.3 Analyze Layout Optimization Opportunities

7 Discussion
7.1 Layout Optimization Support
7.2 Assembly Station and Deadlocks
7.3 Simulation speed

8 Conclusion & Future Work
8.1 Future Work

Bibliography
# List of Figures

2.1 Visual Variables .................................................. 17  
2.2 Visual Analytics Process ........................................... 19  
3.1 HyperMoVal ......................................................... 22  
3.2 PlanningLines ....................................................... 22  
3.3 CloudLines .......................................................... 23  
3.4 Shifting Bottleneck .................................................. 23  
3.5 Shifting bottleneck detection ...................................... 24  
4.1 Main screen of the simulator ...................................... 26  
4.2 Workflows in a simulation study ................................... 29  
4.3 New Assembly Station .............................................. 30  
4.4 Distribution Calibration Window .................................. 33  
4.5 Station’s Load Table ............................................... 35  
4.6 Chart of operation density with a lens showing detailed operation timing 36  
5.1 Configure process steps window .................................. 41  
5.2 New attribute pages ................................................ 42  
5.3 The implemented Distribution Calibration window ............ 45  
5.4 Table as implemented ............................................... 46  
5.5 New proposed variant for the density chart with lens .......... 47  
5.6 Example of Resource Drags calculation .......................... 51  
6.1 Layout of the Manufacturing Line .................................. 54  
6.2 Calibration examples ................................................ 55  
6.3 Simulation result overview ........................................ 56  
6.4 Actors’ Performance Detail ......................................... 57  
6.5 Simulation results for the first analyzed product ............... 57  
6.6 Detail of Bottleneck Shift .......................................... 58  
6.7 Simulation result for the second analyzed product ............. 58
1 Introduction

The ever increasing complexity of manufacturing environments due to higher product customization together with global competition, which demands efficiency, require production lines to be simulated before being built or modernized. Similarly, the vast amount of data collected and simulated requires solutions to let analysts extract important information that help them improving productivity levels of manufacturing facilities.

The main objective of this thesis is to support the use of simulation for layout optimization by developing an approach that helps the user to gain knowledge for conducting simulations of production lines faster through visualization and interaction.

The bottlenecks have be identified to be able to quickly optimize the productivity in a manufacturing line.

Wörner and Ertl [WE13] developed a factory simulation software to support the investigation of alternative layouts for reconfigurable manufacturing. This thesis extends their work by implementing approaches to visually support analysts in data collection and validation, and in identification of improvement opportunities in simulated layouts.

The Critical Path Method is explored as an alternative to detect bottlenecks. The production flow is visualized, and the sequence of operations is displayed by an interactive lens. A new metric, called Resource Drag, is developed to estimate the sensitivity of the production time has to a given manufacturing station. This new metric is used to visually inform the analysts about where the efforts should be focused.

A use case is provided to demonstrate the approach’s functionality.

The important metrics are visualized and the sequence of operations was displayed by an interactive lens.
2 Theoretical Foundations

This chapter presents some of the background topics that are a requisites for the development of this thesis.

2.1 Conceptual Model & Simulation

According to Robinson [Rob04], a model is a simplification of reality. Models can vary in level of detail from the simpler, analytical models, which usually cannot model complex behavior or some variability, to the more complex simulation models which can be full featured in details. But, of course, there are trade-offs and additional details come at the cost of managing and executing a very complex model (data collection and calibration effort, execution time, validity, analysis difficulty).

Therefore, for a defined set of purposes, there is an optimal balance between cost and complexity, which might not be the same for another set of purposes. Because of that, most models should be developed to solve a given problem the modeler has in mind, but on some cases they are created to handle a few purposes, and the validity of the model needs to be reassessed for any new purpose found.

In a simulation study, the conceptual model development is held as already being responsible for 50% of the benefits of the study, by simply uncovering important information which was hidden. It consists in defining the complexity of the model, influencing the computational cost, the validity, the amount and type of data required to calibrate it.

Properly accounting for variability is vital to adequately predicting performance of a system.

According to Thiede et al. [TSAJ13], specialized simulation packages usually offer meta-models which help the user to implement a conceptual model faster and less error prone, and whose purpose is to assist in decision-making. But the meta-model employed by the tool must support the whole conceptual model that is being built. To cover capabilities not natively supported, scripting languages are common features of commercial simulation software.
During decision-making process people may take into account information and constraints not modeled or not available to the computer. People are good at devising strategies and heuristics to solving problems [CK09], much better than any current general purpose artificial intelligence algorithm. Analysts may develop deeper understanding by interacting with a simulation system.

2.2 Model Evaluation

According to Moriasi et al. [MAL+07] the model evaluation process should start by defining the metrics which will be analyzed and the performance threshold that separates failure from success. To this end it is always good to include quantitative metrics (usually statistical) as well as qualitative ones (which are also more subjective).

Foss et al. [FSKM02] points out that having more data leads to better results than only using more elaborated analysis methods.

On the qualitative end, Robinson [Rob04] argues the visual display of a simulation already contributes by convincing of the plausibility of the model and the results. This is called face validity.

2.3 Critical Path Method (CPM)

According to Fondahl [Fon62], the Critical Path Method (CPM) is successfully used in Project Management to determine which tasks belong to the critical path, that is, which tasks influence the total project duration.

The method is composed by a forward pass, a backward pass, and the calculation of additional metrics after both passes. In the forward pass the Earliest Start (ES) and the Earliest Finish (EF) times are calculated:

The Earliest Start (ES) time of an operation is equal to the latest Earliest Finish among all its predecessors, or the project start time in case current operation has no predecessors.

\[
ES = \begin{cases} 
\max_{p \in P} (EF_p) & \text{if } \exists p \mid p \in P, \\
\text{StartMilestone (0)} & \text{otherwise.}
\end{cases}
\]

Notation notice: $EF_p$ is the Earliest Finish of predecessor $p$, while $ES$, without a subscript, relates to the Earliest Start of the current operation being analyzed, $P$ is
2.3 Critical Path Method (CPM)

the set of all predecessors of the current operation, $S$ is the set of all successors of the current operation, $c$ is the current operation, and $\text{All}$ is the set of all operations.

The Earliest Finish (EF) time is equal to the Earliest Start time plus current operation’s Duration.

\[(2.2) \quad EF = ES + \text{Duration}\]

In the backwards pass the Finish Milestone, the Latest Finish (EF) and Latest Start (LS) times are calculated:

The Finish Milestone is equal to the latest Earliest Finish among all operations.

\[(2.3) \quad \text{FinishMilestone} = \max_{o \in \text{All}} (EF_o)\]

The Latest Finish (LF) time of an operation is the latest time that this operation can finish without delaying the Finish Milestone. It is equal to the earliest Latest Start among all its successors, or the project Finish Milestone, in case there are no successors.

\[(2.4) \quad LF = \begin{cases} \min_{s \in S} (LS_s) & \text{if } \exists s \mid s \in S, \\ \text{FinishMilestone} & \text{otherwise.} \end{cases}\]

The Latest Start (LS) time is equal to the Latest Finish time minus the Duration.

\[(2.5) \quad LS = LF - \text{Duration}\]

After both passes, one can then calculate the Total Float (TF) and the Free Float (FF):

The Total Float (TF) is equal to the Latest Finish minus the Earliest Finish. It corresponds to how long the operation can be delayed without displacing the FinishMilestone.

\[(2.6) \quad TF = LF - EF\]

When the Total Float is 0 (zero), the operation belongs to the critical path, and is said to be critical.

The Free Float (FF) is equal to the earliest successors’ Earliest Start minus the Earliest Finish, and corresponds to how long the operation can be delayed without affecting the Earliest Start of any successor. It can only be nonzero if the current operation is not a critical operation.

\[(2.7) \quad FF = \begin{cases} \min_{s \in S} (ES_s) - EF & \text{if } \exists s \mid s \in S, \\ \text{FinishMilestone} - EF & \text{otherwise.} \end{cases}\]
Devaux [Dev12] introduces another metric, called Drag, which is defined only for critical operations. The meaning of the Drag is how long the operation is already delaying the Finish Milestone, and can be thought of as an opposite of the Free Float. The drag is calculated as the shortest Total Float of a parallel operation, or the operation’s duration if the shortest Total Float is larger than the duration, or there are no parallel operations.

\[
(2.8) \quad \text{Drag} = \begin{cases} 
\min \left( \max_{o \parallel c} (TF_o), \text{Duration} \right) & \text{if } \exists o \mid o \parallel c, \\
\text{Duration} & \text{otherwise.}
\end{cases}
\]

The importance of the Drag is that by reducing the operation duration by that amount, the total process duration is reduced by the same amount.

### 2.4 Probability Distributions

Probability distributions \( p(x) \) have desirable properties not always found on more commonly used machine learning functions:

- Probability densities are non-negative [DHS01];
- \( \int p(x) \, dx = 1 \) [DHS01];
- Common probability distributions are already smooth and have few parameters, and therefore hardly ever would overfit, making unnecessary the use of smoothness terms in obtaining a fit [Dom12];

Duda et al. [DHS01, p. 618-619] states in a continuous distribution, the probability of it generating any specific value is zero. Nevertheless it is possible to calculate the probability that the generated value lies within a distance \( \epsilon \) from a specified value. If you divide this probability by the length of that interval, you obtain an approximated probability density. The smaller the range the better the precision, and at the limit where the range is zero, you obtain the Probability Density Function (PDF).

### 2.5 Normal Distribution

According to Duda et al. [DHS01, p. 621-622], the normal distribution, also called Gaussian distribution, is the most commonly used continuous probability distribution. It is a reasonable assumption in many situations because of the Central Limit Theorem, which states that the effect of successive application of other distributions tends to
approximate a normal distribution. But even when this property is not applicable, it
might still be used due to its simplicity and the wide range of methods developed for
it.

It is defined by its Probability Density Function:

\[
 p(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}
\]

where \( \mu \) is the mean and \( \sigma \) is the standard deviation.

The Cumulative Distribution Function of the normal distribution is given by:

\[
 CDF(x) = \frac{1}{2} \left( 1 + \text{erf} \left( \frac{x - \mu}{\sigma \sqrt{2}} \right) \right)
\]

and \( \text{erf}(x) \) is a transcendental function, but Bell [Bel15] created the following approximation:

\[
 \text{erf}(x) \approx \text{sign}(x) \sqrt{1 - e^{-\frac{2x^2}{\pi}}}
\]

Box and Muller [BM58] introduced the Box-Muller method to generate normally-
distributed values from uniformly-distributed inputs.

2.6 Machine Learning

Machine learning is concerned with learning a hypothesis about the structure present in
data. The field is subdivided into supervised learning and unsupervised learning.

According to Hastie et al. [HTF09, p. 1-2, 4, 486], in supervised learning problems
typically there is an outcome we want to predict based on some features. The outcome
can be either a value (in regression), or a category (in classification). To be able to make
those predictions a training data set is used to fit a prediction model, called learner.

On the other hand, unsupervised learning is devoted to uncovering the organization of
the data, either by estimating probability densities or through clustering.

Domingos [Dom12] explains that machine learning can be viewed as an optimization
problem where three dimensions can be chosen:

**Representation:** The hypothesis space, that is the set of function families which can be
selected as a result (best representative of the data);
Evaluation: The error metric, meaning what types of deviations are more undesirable and should be penalized;

Optimization: The optimization method, used to select the result with the least penalties;

The resulting function may perform well on training data, but perform poorly on other data. This lack of generalization is a symptom of what is called overfitting. Machine learning’s ultimate goal is to generalize to all situations, including those not used during training. Therefore besides fitting the function, it is necessary to validate it with another data set, called test data. Validation does not eliminate overfitting, it may only select the best among a small set of final results, which may all overfit.

Reducing the complexity of the hypothesis space is a way of avoiding or at least reducing overfitting. Regularization accomplishes that by penalizing high values in parameter space, which restricts the solutions to smoother hypothesis. Another approach is to use smooth functions with few parameters.

2.7 Density Estimation

Duda et al. [DHS01, p. 164-166] explains that the Parzen Windows approach estimates the density at a point by counting data points that lie within a distance of the desired point and dividing by the "volume" defined by that distance.

There is a fundamental tradeoff: if the distance is too large, the resulting estimation will be too smooth and cannot accurately represent fine details such as a sharp change on the underlying function, while if the distance is too small, the estimation error will derive from quantization due to insufficient number of points considered.

According to Hastie et al. [HTF09, p. 208-209], Kernel Density Estimation (KDE) is a variant of Parzen Windows, which instead of considering a hard distance limit, employs some function, the kernel, to weight the influence of a point, making it possible the have a smooth limit:

\[
(2.12) \quad f_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{d(x, x_i)}{h}\right)
\]

where \( h \) is a smoothing factor called bandwidth and \( K \) is some kernel function, like for example the Gaussian kernel:

\[
(2.13) \quad K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}}
\]
A practical rule of thumb exists for choosing a "good" value for the bandwidth:

\[
(2.14) \quad h = \left( \frac{4\sigma^5}{3n} \right)^{\frac{1}{5}}
\]

Histograms are commonly used in statistics, and employ binning. Binning is the discrete analogous to the continuous density estimation using Parzen Windows. The bins are defined such that the intervals are adjacent with no overlap, meaning each point is accounted for exactly once.

Binning alone is susceptible to large value changes caused by small center changes, one solution for that is to distribute the weight of a point to the closest bins, proportionally to the distance to the centers. This approach can be thought of as equivalent to KDE with a triangular kernel.

Duda et al. [DHS01, p. 174-176] present k-Nearest Neighbors as yet another approach to the problem at hand, but with a fundamental difference to Parzen Windows, instead of having a fixed size, it determines the distance needed to collect the chosen amount of points, and defines the corresponding volume. Therefore it can deal better with very low densities without the need to blur other regions.

2.8 Information Visualization

According to Zhao [Zha15], there are several possible mappings for a given information as proposed by Bertin, as is shown in Figure 2.1:

![Figure 2.1: Visual variables can be used to map data features. Image by Zhao [Zha15]](image)

There are scalability issues, with for example some representations taking up too much screen space. Only a small number of colors can be distinguished from each other. Another issue is excessive visual clutter.
Gestalt laws elicit the tendency humans have to perceive images as a whole, being used for guidance in information visualization design. Zhao [Zha15] describes them as follows:

"Proximity: Objects close to each other are seen as a whole;
Similarity: Similar objects tend to be grouped together;
Connectedness: Connecting different objects by lines is a powerful way of expressing relationships between them;
Continuity: Smooth and continuous lines are perceived before discontinuous ones;
Closure: A closed contour tends to be seen as an object;
Symmetry: Symmetrical object pairs are perceived more strongly than parallel pairs;
Relative size: Smaller components of a pattern tend to be perceived as objects;
Common fate: Objects moving in the same direction appear as a whole."

As the human vision tries to interpret images, it ends up creating illusions. Optical illusions show us that human perception about length, lightness and angle are subject to misinterpretation. So the task the user needs to perform is of great importance as a bad mapping may cause data to be misinterpreted, what is called a visual lie.

2.9 Visual Analytics

According to Keim et al. [KKEM10, p. 1-3, 11, 116-117, 119], one of the first uses of the term Visual Analytics was in 2004, but works with characteristics of the field already existed for a long time. Visual Analytics is an interdisciplinary research area which integrates independently born fields such as visualization, data mining, data management, data fusion, statistics, cognition science among others. Its view is to empower analysts to make better decisions by allowing them to have the right information at the right time, instead of being overloaded by the massive amount of data. It shifts the focus into exploratory data analysis though interaction of human and digital processing.

Most real problems are ill-defined, so often more specific questions are only asked after a general overview of the data is presented. This culminates in the guidelines proposed by Shneiderman "Overview first, zoom/filter, details on demand".

All this process is accomplished through interaction, which can be categorized into the following types:

select: mark data points, usually for performing other operations afterwards;
explore: show other data, for example: pan, zoom, resample;
reconfigure: change spatial mapping, for example: sort, change axis variable;
encode: change appearance, for example: change colors, sizes, shapes;
abstract/elaborated: show more or less detail, for example: details on demand, semantic zooming;
filter: show or hide data matching a criteria;
connect: highlight related data (on different views), for example: brushing;

According to [KKEM10], insights are pieces of discovery, and are unexpected by nature, nevertheless visualization can support it, specially by applying reconfiguration.

How and Why are usually related. Sacha et al. [SSS+14] proposes another model, comments about the small "work" memory of humans and fallacies such as the confirmation bias, which the computer system should help to overcome.

Provenance annotates discoveries made, thus allowing communication to other people, and/or future analysis of how conclusions were obtained and reverify its validity.
Aigner et al. [AMM+07] classifies time visualizations into seven category dimensions, although not all combinations are relevant, other complementary classifications are not included, so it is reasonable to imagine that well over 100 different problem configurations exist.

For specialized tasks it becomes necessary to use domain specific analysis methods and knowledge. But the specificity of that makes it hard for it to be adopted and implemented on other visual analytics frameworks, therefore

Tominski et al. [TGK+14] presented a taxonomy of interactive lenses. Lenses allow to look at details. They can have any shape, but the most common are circular and rectangular. They can simply magnify what is already being shown, or they can make any type of change (such as change the position of objects, filter what is displayed, or display more details) depending of their intended function The position, size and function of lenses may be parameterized.
3 Related Work

This thesis identified related work within a few topics, which can be organized in the following categories: simulator systems which use focus on manufacturing and employ visual analytics approaches, visualization techniques, and bottleneck identification methods.

3.1 Simulator Systems

The FACTS analyzer [MNS08] is a conceptual modeling system used for manufacturing simulation. It simplifies simulation models in order to decrease the cost of data collection and execution time. For modeling task duration, it uses an Effective Process Time (EPT) which is a probability distribution which captures process variability, and also other disturbances such as breakdowns. It helps the user locate bottlenecks by displaying the percentage of time each machine is a bottleneck, further subdivided into sole and shifting bottlenecks.

HyperMoVal [PBK10] uses general purpose multivariate visualization techniques, such as Scatter Plot Matrices and Parallel Coordinate Plots, to compare features generated by the simulation, as well as new derived features. See Figure 3.2. It uses a machine learning model to predict results for costly simulations. The models learned are also evaluated with the help of the tool.

Feldkamp et al. [FBS15] developed another manufacturing simulation system. Only a few metrics are analyzed, but it scales well for large numbers of simulation repetitions. It uses scatter plot matrices and parallel coordinate plots to convey information.

3.2 Visualization Techniques

PlanningLines [AMTB05] extends Gantt Charts to express the uncertainty in a task’s start time, finish time, and even its duration, as can be seen in Figure 3.2.
3.3 Bottleneck Identification Methods

The bottlenecks in a manufacturing system limit its performance. The approaches commonly used for detecting bottleneck are by analyzing workload and buffer levels, but they are sometimes inconclusive.

Roser et al. [RNT01; RNT02; RNT03] proposed a method for detecting bottlenecks that can be momentary and may shift from a machine to another over time. It defines the
3.3 Bottleneck Identification Methods

Figure 3.3: CloudLines by Krstajic et al. [KBK11]. With a visualization like this, it is possible to perceive how event-flow patterns evolve over time and over instances.

longest active period of a machine at a moment to be a bottleneck, then defines the period of intersection from bottleneck active periods to be a shifting bottleneck. See that in Figure 3.4.

Figure 3.4: Shifting Bottleneck example from Roser et al. [RNT03]

Later, Roser et al. [RLD14] developed another methodology for detecting bottlenecks. It consists in performing walks though the line, observing starved, and/or blocked stations, and the buffer levels. Buffers less than one third full are weak evidence the bottleneck is before, buffers over two thirds full are weak evidence the bottleneck is after it, and buffers about half full are uncertain. The detection is performed by analyzing the evidence collected in a table as in Figure 3.5.
Figure 3.5: Shifting bottleneck detection through line walks methodology developed by Roser et al. [RLD14]. Each line corresponds to a single walk. The areas marked with bold rectangles indicate where the bottlenecks are found.
4 Concept

This chapter describes the general ideas proposed in this thesis, but before that some preliminary information is provided.

4.1 Preliminary Description of the Simulator Software

VIS had previously developed a manufacturing layout simulation tool in C#. The user manages most functionality through the Project tree on the left of the main screen, see Figure 4.1.

The project tree contains some parameters used in the simulation, like:

- a list of actors which determine the behavior of the stations;
- a list of part types that can be handled. The parts can be organized in a hierarchical way so it simplifies setting up machine behavior;
- a list of production processes;
- a list of production orders;
- a list of layouts, which can be opened for viewing and editing, as well as simulated;
- a list of simulation runs;

When an item is selected on the project tree, its attributes are shown on the right panel. Items can be dragged from the tree and dropped on compatible fields in the attributes panel.

When a layout is opened, a new tab is created, filling the middle screen space. Then with the mouse it is possible to zoom, move objects eventually connecting close ones. A right-click on empty area allows for adding new stations, and a right-click on a station allows for rotating, configuring, deleting or disconnecting them.

The tool includes a layout discovery functionality. It is a generic algorithm which starts with some seed(s) and mutates it randomly until a new workable layout is generated. There is a probability of performing each type of mutation on each step. The entirety
Figure 4.1: Main screen of the simulator

of this probabilities together with the initial conditions determine how likely "good" outcomes are to be explored.

The kinds of stations which can be added are predefined, but the behavior can be changed by changing the underlying actor. This capability is not exploited by the layout discovery.

When they are simulated, a simulation run is created underneath. On simulation runs it is displayed some score features, the occupancy of each station, and the start and end times of each workpiece, as well as more detailed displays.

Station length, conveyor speed, scanner read time, switch position change time are all factors which are hard-coded on software and therefore are difficult for an end user to change to fit a preference or need.

Parts may have a list of restrictions, and hold many other kinds of child objects (including slots and attributes), but for real simulation usage it just seems like too much detail, more likely to confuse the user than to model a behavior in the end result of the simulation. It is used to animate the progress of the pieces during the simulation.
4.2 Manufacturing Company Collaboration

This thesis benefited from the cooperation with a manufacturing company. The collaboration consisted of granting access to historical data from a manufacturing line and through meetings and e-mail exchange with domain experts.

4.2.1 Expert’s Preliminary Considerations

The domain experts see potential application for the simulation tool to:

- Plan new layouts
- Detect drifts in production performance, so to trigger maintenance work.

The factors she considers important to be input to the simulator are:

**Cycle time:** the total time between a piece is completed and the next one is completed; The time between work on a piece is finished and work on the next piece is finished on the same machine. It is equivalent to the process-time plus the time to move the finished piece out and the next piece in and the time the machine is idle.

**Process time:** the part of the cycle time in which a machine spends actually performing work on a piece and the transport plate is stopped.

**Buffer capacity:** how many pieces can be waiting before the previous machines have to stop.

The definition of those terms as used in this thesis is important because they are used with various different meanings throughout industry.

A further consideration is that the user of the software should be able to change the parameters directly on it.

4.2.2 Provided Historical Data

Access to a copy of the database of a real production line was granted. The most relevant data that was made available were the cycle-times of the operations, annotated with station, product type, and timestamp.

The experts also provided a table containing the standard operation times for two product types.
4.3 General idea

For optimizing a manufacturing layout, first there are some common settings that need to be configured, only then a layout can be built and simulated, thus starting the optimization workflow itself. Those settings span specifications like:

- Which products will have to be produced;
- How is the production process, which operations are necessary, and in which order they can be performed;
- Which kinds of stations are available for building the production line, and which steps they can execute;
- How long the steps would take to complete;

In case there is data about a manufacturing line, which employs the same, or at least similar operations, these data can be used to enhance the realism of the simulation by calibrating the model parameters.

After reproducing an existing layout, the overall simulation may be validated, and in case some difference is noticed, it may be possible to adjust and correct for it. This constitutes a distinct workflow from the layout optimization one mentioned earlier.

In this thesis, it also became necessary to extend the simulation capabilities to incorporate features used in real layouts, which otherwise would not be able to be realistically reproduced. This is the case for introducing time distributions instead of fixed durations, and introducing a changeover delay. An overview of both workflows can be seen in Figure 4.2.

As the user interacts with the simulation, and the results, he or she gains insights of what is going on. Sometimes the insight comes in the form that the simulation is not doing it right, and he may have to correct some setting. Visualization supports optimization by highlighting opportunities for increased speed and/or reduced cost.

4.4 Extend the Conceptual Model Capabilities

To be able to simulate layouts similar to those in real factories there are some enhancements that need to be performed, extending the model capabilities.
4.4 Extend the Conceptual Model Capabilities

Figure 4.2: Workflows in a simulation study. The simulation study usually starts with some initial specifications. The top feedback cycle collects historical data to refine and validate settings of the problem under investigation. The lower feedback loop deals with optimization of the layout itself, which is the goal of the simulation study, but it can only be adequately done after the first workflow has generated consistent specifications. Many visualizations offer each a piece of knowledge, knowledge can be used on both workflows.

4.4.1 Introduce a New Station Type for Assemblies

To manufacture a complex product it is not rare to mount sub-assemblies before assembling them together in the final product. The original simulator software could not model a dependency in which the sub-assemblies must be ready before they could be used in the main production line. A new assembly station would fill that gap.

Such an assembly station should have two conveyors. On the main conveyor travels the piece on which the sub-assembly is going to be mounted. And the sub-assemblies are fed through the auxiliary conveyor. See how an assembly station looks like in Figure 4.3.
4.4.2 Introduce Variability in Simulated Durations

Variability is very important to capture the characteristics of real systems. By having variability and multiple runs, it is possible to have confidence that the simulation results are representative and do not reflect some rare situation, but really the expected behavior.

The proposition is to make a few probability distributions available to the user, which would suffice to imitate most of the behavior of the systems being modeled. The normal distribution is included here, since it can be used as an approximation for the sum of many random variables of any underlying distribution. The log normal distribution is also included, as the cycle-time data seems to have a broader tail than the normal distribution would produce.

Seed Strategy

With the introduction of the variability and the probability distributions it becomes necessary to have a Random Number Generator (RNG). A Pseudo Random Number Generator (PRNG) scrambles an initial value, the seed, generating deterministic sequences of numbers which look random.

A common approach is to derive the seed of PRNGs from the system clock, but this undermines repeatability. Repeatability of a simulation is a good feature for both debugging and presenting results, therefore it is desirable. The option to preserve repeatability is to use a fixed seed. But if multiple simulations are being executed
in parallel, such as, for example, how the layout discovery does, it becomes hard to consume the generated numbers repeating the very same order.

A solution for that is to make PRNGs be exclusive for each simulation run, as the order of the consumed values becomes then fixed. But to retain the variability between different simulation runs while keeping the other settings unchanged, the PRNGs have to be initialized by different, but yet deterministic seeds. So the final strategy is to have independent PRNG instances on each simulation run, and initializing them with the number of existing simulation runs.

4.5 Improve the Workflow of Specifying Common Manufacturing Settings

Specifying the simulation settings which are common for all layouts takes quite some time and can be somewhat distracting. Therefore, improvements are needed.

4.5.1 New Editing Possibilities on the Project Tree and the Attributes View

The user needs to specify what, and how, is to be manufactured, and by which stations. Originally, those settings could only be edited on the XML file, which required the file to be loaded. Although editing directly on XML file is not hard, specially if the constructs are already present and can be copied, or just have its values changed, learning how to do that takes some time. Even just searching for the right place wastes time and distracts the user, making him or her less productive.

The load file operation also comes with the cost of loosing the simulation runs that were previously executed, making a comparison between before and after the changes much harder. All those reasons follow the advise given by the expert to have those editing options easily available, therefore it makes sense to implement in this thesis.

So the project tree has to be complemented to display:

- the Steps each Actor can perform;
- the Operations each Process has to execute;
- and also the Ports each Actor have, which are needed to correctly setup the handling of production pieces;
Likewise, by selecting the items in the project tree, it is important to offer detailed editing abilities on the attribute pages. Attribute pages have to be created for Steps, Ports, Processes, Operations and Tasks, including the ability to edit all their main characteristics.

4.5.2 Distribution Calibration and Validation

To have realistic simulations, the process times need to be defined. With the introduction of time variability, the user can choose a distribution family and input the corresponding parameters. In case the analyst does not know already what the parameters should be, this information can be retrieved from past data, if available.

By clicking on the select button of a distribution control, a window opens. There the user can choose a station from the drop-down list and the corresponding data is filtered and loaded from the database.

Then a histogram is drawn, always starting from zero, but the outliers are grouped into a single bin of a different color, making the histogram readable. Additionally, the values are also displayed on another area below as a Cumulative Distribution Function (CDF), spanning vertically from 0 to 100%.

A few probability distributions are fitted, and the Probability Density Function (PDF) and CDF of each are also drawn. See Figure 4.4.

The user is able to highlight the curves of the selected probability distribution, hide it by unchecking, or edit the parameters, making the fit as he judges best. By moving the mouse over the histogram, more details are shown, including the range of durations of the underlying bar, the number of datapoints contained and the percentage it represents.

Also in case a simulation run has already been executed, it is possible to compare the histogram and the curves with the simulated cycle-time results of a station.

4.6 Support the Task of Layout Optimization

Manufacturers have pressures that manifest themselves in various ways, but all of them are interrelated somehow:

- When there is a product delivery deadline, total production time is important;
4.6 Support the Task of Layout Optimization

Figure 4.4: Distribution Calibration Window. The top left chart is a histogram and combines the PDF of the fitted distributions. The bottom left chart show the CDF of the data and of the distributions as well.

- The total cost is as important as the total revenue, because together they define the operational profit;
- When the sales are limited by the ability to produce, the productivity is important;
- The amount of turn capital in process is also important;
- When planning an expansion all those issues matter, together with investment cost;

So the single most important aspect of the simulation is time, although closely followed by money, the latter is crucially affected by the first, and at the same time is more complex and suffers influence from external factors which may be uncertain. Therefore in this thesis the main focus for optimization is on time.

4.6.1 Analyzing Bottlenecks in Production

The bottlenecks are the machines that determine the rhythm of the whole production line. They force other stations to stop either because the stations have no more workpieces to
process or because they have nowhere to store processed pieces since the exit buffer is full.

The analyst, who needs to improve time efficiency (throughput) in the simulation, should focus the attention to the critical path, which is ultimately what determines total time of the manufacturing process.

Bottlenecks may shift from one station to another over time due to machine breaks, product and process changes, other complex behavior, or simply due to inherent process variability.

The idea of critical path from the Critical Path Method (CPM) is analogous to the notion that bottlenecks can migrate from a station to another, so the CPM will be used in this thesis to identify bottlenecks. With CPM it is always possible to identify the critical path, and it also offers some other metrics which might be useful as well.

Displaying the total time spent by the stations in critical activities gives a better picture of what are the most important bottlenecks. After all, that much of the total simulation time really is spent because of that step.

The most important is to quantify how much is being lost in productivity due to each bottleneck. The drag is a metric which calculates that for a single operation. The first idea is to sum the drags of all operations of each station. This information would help the analyst decide what to do next. If the time lost is large, it is something that should be addressed or at least studied. But if the time lost is small, it could be considered good enough, as it may already be a local optimal solution and it’s not worth taking much effort.

### 4.6.2 Resource Drag

The importance of the drag is that by reducing the operation duration by that amount, the total process duration is reduced by the same amount.

But it does not work well for a repetitive production process. If there are many sequential operations on the critical path, the drag of all of them will add together, effectively meaning whichever operation you can reduce the duration will alleviate the whole process, but whenever you reduce the duration of one of those operations, you must recalculate before you know whether the other operations’ drag are still there to be harvested. And in repetitive production processes you do not act upon a single operation instance, but on the whole series of operations from that machine. Therefore something else is needed.
A new metric, called Resource Drag, is then developed. The term resource is preferred instead of station because, in other situations, it could be applied to any type of resource, not only stations. It reports how much time can be saved from the complete process execution if the operations of the analyzed station had zero duration.

4.6.3 Station’s Load Table and Bar

The analysts need to know which stations limit the performance of a layout and this translates into bottlenecks. So the amount of time each station is a bottleneck (identified as a bottleneck while performing operations in the critical path) has to be displayed.

The optimizable time might be even more important for the analysts than the bottleneck time, as it indicates possible productivity gains. In the bar, it is displayed aligned to the left, allowing the user to have a good impression of which stations should receive more attention, and whether it is worth the effort, or it is already good enough.

Another relevant information to be displayed is the idle time of the stations while in cycle, as it indicates cost reduction opportunities, since, at least for some cycles, the station could run slower without sacrificing productivity. See Figure 4.5.

![Station’s Load Table](image)

(a) Old appearance  
(b) Proposed bar appearance

**Figure 4.5:** Station’s Load Table: the displayed bar now indicates

To facilitate a deeper analysis, the percentage values are included as new columns in the table.

4.6.4 Visualization of the Production Flow and Detailed View of Operations

A chart relating operations to machines and time was missing. Two distinct views seem compelling:
• Display the flow of products on the machine over time. This helps identify transients.

• A detailed view of the operations and the exact start and finish times, as well as the path of the products.

The uses of the chart would range from validating the simulations, to identifying opportunities for improving throughput or reducing cost, or identifying root causes for disturbances.

The densities of operations are plotted, in as many vertically stacked charts as there are stations to show. A lens is also drawn, displacing the remaining of the density chart to the right. The lens shows individual operations as rectangles, connecting the operations from a single piece with line segments. The line segments are connected to the start and the end of the boxes, and therefore are more vertical than if they were connected together, making it easier to compare inclination shifts between similar line segments from neighbor pieces.

The background area from each station in the lens may be painted with a color to convey special information. Light red is used to denote a critical operation, a stronger red is used to indicate resource drags, a light blue indicates some idle time. See Figure 4.6 for a better understanding.

![Figure 4.6: Chart of operation density with a lens showing detailed operation timing.](image)
The background colors indicate whether the operation can be delayed without compromising total production time (light blue shades), whether the operation is critical (light red), and when an operation is critical, it may further indicate the time that could be saved by reducing the operation duration (red). Lines connecting operations represent a single workpiece.

Through interaction it is possible to move a clicked reference point on screen. If the clicked reference point lies within the lens, only the lens is moved such that the same reference point keeps under the pointer. Otherwise, if it lies outside the lens, the actual data is moved.
5 Implementation

This chapter describes the implementation details and in some cases complement the ideas already presented in Chapter 4.

5.1 Extend the Conceptual Model Capabilities

To be able to simulate layouts similar to those in real factories there are some enhancements that need to be performed, extending the model capabilities.

5.1.1 A new Assembly station

The following items need to be created or modified in order to create a new type of station:

A station class as a subclass of ITrameElement. This is responsible to define conveyor paths, work positions and ports. It also is responsible for holding an actor, changing the direction of elements when commanded, forward the requests to check whether a step can perform an operation, and generate a corresponding state object. In this first example deriving from a LinearITrameElement simplified the task;

A state class as a subclass of ActorObjectState. Beyond the constructor, it just have a read-only property which returns the original station object;

A model class, with exactly the same name as the station class appended with the "Model" suffix. It is responsible to add the corresponding 3D mesh from the .dae file located in the Sam\Deploy\models\ folder;

Editor menu: Change the GetContextMenu function from the Editor class to make the new type of station available for the user to insert it through the editor;

Layout discovery: Many other changes are needed to make the new station be considered by and work with the layout discovery;
A new type of station usually also requires a new type of functionality or behavior and then the following items need also to be created:

**A step class.** The *CanPerform* function checks whether the operation could be executed by this step instance, and whether the station is configured to execute operations like that. The *Execute* function is able to actually perform the step, but in practice it is only used in the simulation mode, which just verifies whether all settings are right and the step is ready to start on the piece currently at the work position. The *StartStep* function does the checking again as the *Execute* and really start the execution of the step. After it returns true, indicating the step started, the timer is set to trigger the *EndStep* after the duration time. The *EndStep* function is more concerned with cleaning up the started execution.

**An operation class** as a subclass of *Operation* class. The instance of this class is kept simply as a placeholder for requesting a step capable of executing such an operation.

The assembly station is more complex than other stations, because it needs to handle two workpieces simultaneously, but usually the pieces do not arrive at the same time, and the piece which arrives first needs to wait.

To make a piece wait, the *ConveyorStopper* from the *ConveyorWorkPosition* is automatically activated. But if a piece cannot do any work at the current *ConveyorWorkPosition*, as evidenced by no *Step* from the station’s actor returning true from *Execute*, the stopper is released.

To avoid changing general code, which deals with all kinds of steps, the implemented solution is to start the execution of the step for the first piece as soon as it arrives, but only end it when the end command from the other piece is triggered.

### 5.1.2 Minor correction of step durations

It was noticed that each operation execution was taking a time step too little. This effect although small, could potentially sum up causing larger deviations.

The simulation creates a time step to indicate the time interval currently being simulated. When checking for scheduled events the end time is used, but at this point in code, the time had not being updated yet on the simulation run, which is used to schedule new future events, effectively making each and every duration to be scheduled to finish one time step (10 ms) too early.

Placing the simulation time update before the actual execution of the events fixes the issue.
5.1 Extend the Conceptual Model Capabilities

5.1.3 Time penalty to simulate production changeover

By analyzing the delay between two consecutive production records of the same station it becomes evident that the first produced item of a new product type takes longer than other records. This extra time is due to changeover. One noticed detail is that all following stations also have a delay of about the same order of magnitude, but it does not accumulate.

For the simulator, it means that only the PutOnConveyor operations, need to be delayed in order to reproduce changeover behavior.

5.1.4 Introducing variability to step durations

To be able to handle multiple distribution types, an abstract Distribution class is introduced. Polymorphic subclasses implement the normal and the log-normal distributions.

The semantics adopted in the XML file format used to save and load the project settings do not allow instances that are from a subclass of the defined property type to be correctly saved unless they are hold by a collection such as a list. So the distributions are kept in a list even though only a single instance is used.

Gaussian Generator

The .Net Framework provides the class Random, which is an implementation of a pseudo-random number generator. When a seed is not specified, it uses a number derived from the system clock as a seed. The Random class is able to generate both integer and real random numbers, but it is not able to generate Gaussian distributions, which are so widely used.

To overcome this a new class RandomGen was created, extending Random. The function NextGaussian was created, following the naming pattern already adopted by the base class. This function retrieves a real number from a standard normal distribution. It follows the Box-Muller algorithm [BM58], modified to assure better performance as the original algorithm requires two fresh real random numbers between zero and one (which are already provided by the Random class) and outputs two independent Gaussian samples, but the usage requirements are that only one Gaussian might be needed at a time.

So the modification consists in saving the pre-computed values which can give rise to the second Gaussian, and consuming them whenever they are already stored, as can be seen
in Algorithm 5.1. This leads to defining the angle and the radius as class fields. There is another option which needs to keep in memory a single variable, by pre-computing and storing the second sample already when the first one is being requested, but it would anticipate the cost of computation, without being sure the result would actually be used in the future.

Algorithm 5.1 Modified BoxMuller algorithm

```plaintext
procedure NEXTGAUSSIAN
    if IsNaN(angle) then
        // Use Not a Number values to signal to draw new numbers
        radius ← √−2ln(NextDouble())
        angle ← 2π NextDouble()
        result ← radius cos(angle)
    else
        // Use values previously generated
        result ← radius sin(angle)
        angle ← NaN
    end if
end procedure
```

To implement the seed strategy, a random number generator instance is created and stored by each simulation run, assuring there are no inter simulation interference. The number of simulation runs in the project is used as the seed for the PRNG.

5.2 Improvements in Settings Configuration Workflow

The improvements in the workflow of configuring common settings are described in this section.

5.2.1 Reformulate the list of steps available for configuration

Originally the Configure process steps window did not show correctly which options should be available for selection and which were already set. The correction consisted in first getting the result from the CanPerform function from the step classes to check if the option was set, and in case it was not set, calling it again, but with the newly created optional parameter checkPermission equal to false, which would then indicate whether the step option should be available.
Another modification was to make the options from all the processes available on the window, as previously only options from the first process were shown. This basically just required to place the existing code within a for loop along with some minor index changes.

With the possibility of multiple sub-assembly lines, it became necessary to control which station were able to put which WorkPieces on conveyor. This is achieved by using the process' part in the cache to mean the station can start production of pieces of this process.

One should bear in mind that the rule for allowing or disallowing a step type relies only on the existence of a reference in the right port (such as for store in Output port), or parts in the cache (all other steps/operations). So, for example, there is no way to set a store step from one process and not set the store step from another process. Even if you do that, when it is saved, it will set the store step for all processes.

See how the new options look like in Figure 5.1.

![Configure process steps window](image)

**Figure 5.1:** Configure process steps window. The options are listed for all steps of all processes sequentially. Notice the numbers before the colons (:) are the Ids of the steps, whose default value is the sequence number within the process, but could be a name.

### 5.2.2 New Editing Possibilities on the Project Tree and the Attributes View

*Actors, Steps, Ports, Parts, Processes, Operations, Orders and Tasks* can be added, edited, deleted, renamed or have their attributes viewed. Each of those required some of the following modifications:
• Context menu additions
• Handling of "new" command, to include the appropriate item on the project and on the tree.
• Handling of "delete" command, to remove the item accordingly.
• Creation of new attribute page controls for handling specific functionality. See the new attribute pages in Figure 5.2.

![Step attribute page](image1)
![Operation attribute page](image2)
![Port attribute page](image3)
![Process attribute page](image4)
![Task attribute page](image5)

**Figure 5.2:** New attribute pages: The select button on the step page is selected.

The *TreeNode*'s *Tag* field is used to store a reference to the correct object instance. *Steps* and *Operations* come in many subtypes, and the switching from one subtype to the other is made through a drop-down list on the newly created controls. This operation is accomplished by creating a new instance of the correct subtype, filling the properties whose values can be inferred and changing all old-instance references to point to the new instance. This substitution includes updating object references on the project tree. An event delegate is created in the *Actor* class to be able to trigger the tree updates for updating the step instances.
5.2 Improvements in Settings Configuration Workflow

All those changes made use of the project property on the attributes window, making it necessary to also update this when loading a project from an XML file.

Many of those attribute pages contain a variable-size set of references. The list for selecting references always provides an empty extra control in the end, allowing new references to be inserted.

5.2.3 Distribution Calibration and Validation

The database which contains the cycle-time data is a SQL SERVER. The data is selected, filtering by station information. The data contains both a timestamp information as Datetime and the Cycle-time as a string, ordered by timestamp. Repeated timestamps mean the record was duplicated, so only the first one is considered. Even thought duplicated records exist, the reason why they exist is still unclear.

As identified by looking at sample values, the cycle-time string format follows the following pattern: T#[0d][0h][0m][0s][0ms], where 0 means one or more decimal digits, sequences surrounded by [ ] appear optionally, and any other characters are placed literally.

A regular expression is used to separate the available parts of the value, which are parsed and set (all of them have zero as the default value), and finally are converted into a 

$\text{TimeSpan}$.

The values are filtered, removing cycle-times stated as 0 ms, which were not measured. The values are then sorted, as it is needed for displaying the Cumulative Density Function (CDF) later on.

Filtering out Outliers

Besides removing outliers for curve fitting, the outlier limit is used to define the upper limit of the histogram’s range. Z-score limits do not produce satisfactory results, since both the mean and the standard deviation are affected by extreme outliers present in data.

Assuming the underlying distribution of the cycle time data were a normal distribution, then the 50th percentile would correspond to the mean ($\mu$), and the 84th percentile would correspond to the first standard deviate after the mean ($\mu + \sigma$). The outlier limit is then adopted to be twice the value at the 84th percentile, so values higher than that would be considered outliers. Identifying the 84th percentile is made easily since the list of values is already sorted.
5 Implementation

Histogram and Cumulative Densities Chart

The chart is implemented using the Graphics and Bitmap classes available on .Net framework. When the chart needs to be changed, the bitmap is updated and the PictureBox is invalidated and forced to update. The Paint event then only draws the previously generated Bitmap. This way, the hard work may be done less often, only when the chart changes, and not every time it needs to be repainted.

To assist in converting from data coordinates to screen coordinates a matrix multiplication in homogeneous coordinates could be used, but it was preferred to develop a helper Scale class. This class holds a reference value and a size value, and offers functions to convert values or sizes from another scale to its own, abstracting the calculations done. This is an intuitive way of working with several scales for the same axis.

The Cumulative Distribution Function (CDF) is easily drawn having the sorted list of values, since the cumulative percentage is proportional to the index on the list. Figure 5.3 shows the window implemented.

Fit of Distributions

To fit the distribution to the data the usual statistical methods (mean and standard deviation of the sample) are used as they already provide algebraical forms to optimal solutions of the problem, but not considering the presence of outliers. Therefore the outliers, whose limit was previously determined, are simply ignored.

Although fitting either the PDF or the CDF would lead to the desired distribution in a perfect fit scenario, the data does not fit perfectly. So we should consider which type of error is more or less acceptable. Fitting the PDF tries to represent real density of values, which is nice, but it may happen that all values lower than the mean have an small excess and all values larger a small deficit, this situation would lead to a biased mean which is specially undesired. Fitting the CDF leads to a more correct mean value, but may have some shifted density of a particular value in detriment to its close values, this could be an issue only in specific synchronization cases. And there is also the choice of error metric, which can be squared, linear, or some other function.

This task may be better accomplished by the user, which can have a grasp of how the curve behave overall and which distribution would have a better fit.

It should be noted that the database contains cycle time data, but not process time, which ultimatelly are the parameters that need to be set.
5.3 Supporting the Task of Layout Optimization

This section describes the implementation details of the approaches used to support layout optimization.

5.3.1 Metrics Summary Table

The Bars do not really exist inside the table cells, instead they are painted during the Paint event of the cell. The newly introduced color regions are then painted on top of the bar.
In the end, to keep color coding consistent across views and not to cause larger confusion it was opted to use the same light red color to indicate critical, but not readily optimizable. See the final result in Figure 5.4.

5.3.2 Production Flow and Operation Details Visualization

The density chart is implemented very crudely. It is constructed by binning the end points of operations into the two closest bins, proportionally to the distance to them. The bins are arranged to be spaced by 4 screen pixels. After binning, a one pass Gaussian smoothing is applied, to further smooth the curve, and then the maximum value is obtained, for scaling the height of all areas. The points that would appear to the right of the lens, are all displaced by the lens size, compensated for the length the data would occupy if it where drawn on the density plot, without the lens.

The lens part is drawn on a separated Bitmap, and later painted on top of the density view. This is to assure no shapes would exceed the borders.

The Scale class is also used for this charts. A Zoom function is included to help zoom a Scale around an arbitrary position, affecting both the value reference and the size.

After implementation and given the space available in the attribute page, it became apparent that a solution like from Figure 5.5 would be better suited, both because of a more intuitive interaction, and because the lens does not occlude the densities of the area whose details it is showing.

5.3.3 Applying the Critical Path Method

The model embedded in the CPM is simpler than the simulation model. The model itself does not know anything about the layout, so it could never be able to determine
what the transportation times would be, for example. Instead the detailed results of the simulation are used to parameterize the model.

The original simulator adopts an As Soon As Possible (ASAP) scheduling strategy. So, the resulting simulated times correspond to the Early- metrics of the Critical Path Method (CPM), kept the same order of events. For this modeling, the results obtained from the simulation are directly used as the Early Start, as well as the transportation times.

It was decided to model transportation times (also the station-blocking time) just as node transition delays, specific for each origin-destination pair, instead of new operation nodes. This has the advantage of reducing the number of nodes in the resulting graph, which simplifies display functions as no further filtering is needed, makes it clearer which free-floats should be reported, and might also improve computation cost of some tasks.

Some characteristic behavior of the simulation was manually calibrated into the model, such as the station blocking intervals, which originates from the need for the piece to move in and out of the working position, during this time other pieces cannot be handled. On some rare cases, the blocking intervals may be different than predicted, but then the value is adjusted locally in the model to match with the simulation results. Such a strategy would enable the use of this model with real data from a manufacturing line, where the times are expected to vary, even if only slightly. Unfortunately the available real-world data lacks some attributes such as the start time of processing an item and the relative item’s identity (or any other way to infer them).
A memoized and lazy implementation was used, where each metric is calculated only once, at the first time that they are required, and then the results are stored for future requests. This makes the implementation to be very efficient. The downside is that the metrics should be evaluated only after the simulation and the model are already complete. It would be possible to evaluate it while running by resetting all precomputed information between evaluations.

Each task have at most three predecessors:

- Piece’s previous operation;
- Station’s previous operation;
- Next station’s first operation in buffer, if buffer is full;

To calculate the drag, it is necessary to consider all nodes in parallel branches. A naive algorithm would search each node independently to determine whether it is either an ancestor or a descendant. A cleverer algorithm removes the sets of all ancestors and all descendants from the list of all nodes to determine the set of all parallel nodes, and then iterate over them, trading computation complexity for temporary memory usage.

Another class was created which groups all operations performed by each station. This eases the computation of metric summaries for the group.

Due to the repetitive nature of a production process, it is not useful, and would even cause occlusion to draw markers showing the Latest times from the CPM model (as done in [AMTB05]), therefore only the Early times are shown. Likewise, the total float is not shown as well, but the free-float is as it displays a gap that could be harvested in the process.

Again, due to the repetitive nature, the drag could be limited also by the largest parallel free-float. This new measure has a less proper underlying meaning, but looks more useful in spotting improvement opportunities.

Unfortunately there is still considerable idle-cycle time in activities that are not the last of a piece that still need to be shown. Therefore a yet new measure has to be quantified: the station in-cycle idle-time.

The main distinction that can be made over a station idle time are: if it's setup/ramp up or ramp down, if it's blocked time, or if it is in-cycle idle-time.

As the implementation is done, it is not safe to call for some properties before the simulation is completed and completely mapped to the CPM model. It may be possible to make an implementation that works real-time, by taking advantage from the fact the simulator already provides the partial-order between all relevant operations, and by using a marker to store the moment in time an estimate was computed.
5.3.4 Resource Drag

As said before in Section 4.6.2, the Resource Drag reports how much time can be saved from the complete process execution if the operations of the analyzed station had zero duration.

To calculate it the whole CPM algorithm is applied again, but in a different function which excludes the time contributions from the analyzed station. This function is called \textit{NoDragEarlyFinish} (NDEF).

And the Total Resource Drag equals how much the total execution time would be reduced if all the operations of the given station had zero duration, kept all the rest the same. More precisely, the total execution time corresponds to the latest Earliest Finish among all the operations, resulting in the following equation:

\begin{equation}
\text{TotalResourceDrag} \left( \text{station} \right) = \max_{p \in P} \left( NDEF_p \left( \text{station} \right) \right) - \max_{p \in P} \left( E_{F_p} \right)
\end{equation}

Calculating how this total time is distributed throughout the individual operations is a little more complex, and uses the \textit{NoDragLateFinish} (NDLF) function, which calculates the Latest Finish time for the operation, ignoring the duration of operations from the analyzed station, and also considering the total execution time as the shorter one, calculated using the \textit{NoDragEarlyFinish} of the analyzed station.

In case there is some drag which can be harvested either before or after an operation, for the purpose of displaying it, that is better to concentrate and show the shared drag after the operation, because the drag from the operations will be shown to the right.

The idea is first to calculate how much the total simulation time could be reduced, if the current and the descendant operations from the resource take no time. This value is the Accumulated Resource Drag (ARD), because it contains not only the reduction due to the current operation, but also due to the descendants. It is the difference from the Earliest Finish to the \textit{NoDragLateFinish}. For the critical operations, the Earliest Finish and the Latest Finish are equal, but for the noncritical the Latest Finish includes the Earliest Finish and some Total Float, but measuring accumulated drag should not include any float.

When the Accumulated Resource Drag is positive, the \textit{NoDragLateFinish} occurs before the Earliest Finish. That means the total simulation time has reduced, and the reduction from some ancestor operations is required to achieve the full effect.

Accumulated drags cannot be negative, but the \textit{NoDragLateFinish} can occur after the Earliest Finish, which would mean that the reduction from the descendant operations is already enough to produce the total reduction effect, and the excess reduction would
even become some total float instead. So the Accumulated Resource Drag (ARD) is calculated as follows:

\[
ARD_{\text{(station)}} = \max (EF - NDLF_{\text{(station)}}, \ 0)
\]

Finally, the result is the difference between the accumulated resource drag of the current operation and the largest accumulated resource drag from the predecessors.

\[
\text{ResourceDrag} = ARD_{\text{(station)}} - \max_{p \in P} (ARD_p_{\text{(station)})}
\]

So the drag that could be eliminated by speeding up either one in a sequence of operations is computed only by the later operations, while the drag that is exclusive to an operation stays at it, where it should. See Figure 5.6 for a visual example.

Like for implementing CPM metrics, the NoDragEarlyFinish and NoDragLateFinish can benefit from memoization, even though it needs to take care that the results are specific for the station whose drags are being ignored.
5.3 Supporting the Task of Layout Optimization

Figure 5.6: Example of Resource Drags calculation: First a schedule of a simple process consisting of 3 machines (PA, PB and PC) and 4 pieces, without any transport delays, is calculated, and the Earliest Finish of PB are retained for future computations (bluish lines). Then the NoDragEarlyFinish of station PB is calculated for all the schedule, determining the NoDragFinishMilestone. The Milestone is then used to calculate the NoDragLateFinish of station PB for the whole schedule. The Accumulated Resource Drag is then computed for the operations of each piece in station PB, and the results are used to calculate the actual Resource Drags. Finally the Resource Drags are painted red back on the bars used to determine the Accumulated Resource Drags.
6 Evaluation

The evaluation consists in performing a realistic use case where the functionality developed is tested.

6.1 Create and Configure Stations

The evaluation starts by creating the layout of an existing manufacturing line, and configuring it to produce two product variants it manufactures.

The following actions need to be performed to configure a station, which does a unique type of work, for two products with different process times:

1. Create the two parts to be added (one for each process type);
2. Create a new actor type;
3. Create a new port instance for the second process;
4. Set the parts in both ports (one for each process type);
5. Create a new step (for the second process);
6. Set the new port instance to the from port of the new step;
7. Set the time distribution for both steps;
8. Create a new operations in each process, and set the corresponding part;
9. Place and connect the station in the layout view;
10. Set the station’s actor to the created one;
11. Configure the both pieces to be produced (clicking the OK button creates a cache object);
12. Set the just created cache to the second input port, before doing anything else;
6 Evaluation

Figure 6.1: Layout of the Manufacturing Line: The main line has an "L" shape, and there are two other lines, each responsible to manufacture a different type of sub-assembly. After finished, a sub-assembly is just like a single regular part. In the real factory, there are no conveyor belts connecting the lines, but the picture depicts a model where there is no external stock of finished sub-assemblies.

Figure 6.1 shows a modeling possibility for the manufacturing line. The model actually used in this evaluation contained only the main line.

Even with the improvement that the edits can be done directly on the GUI, this process is still too long and complex. Without loosing time looking up for information or making mistakes, it takes around two minutes to configure a single station. It could be greatly shortened by simplifying the original simulator’s notion of port, but this is not a straightforward thing to do, and the functional requirements should be reaccessed.

6.2 Calibrate Time Distributions

While configuring the stations, the analysts might want to get estimates of duration distributions based on real historical data from a similar manufacturing line. The distribution calibration window is reachable through the select button from the distribution interface in the step attribute page.

Figure 6.2 show distribution calibration examples. The automatic fit, which is a maximum likelihood fit, appears to overestimate the variance, as in Figure 6.2a. Figure 6.2b shows manual fitting is specially useful when the analyst have a deep understanding of the effects appearing in data, and want to fit a special region. Manually entering parameter values is not a very good interaction experience.
6.2 Calibrate Time Distributions

(a) Calibration example of automatic fit for a single product. The automatic fit curve shown appears to overestimate the variance.

(b) Calibration example of manual fit: multimodal distribution present in data. The Gaussian distribution (black curve) is fitted manually to a specific mode, since the analyzed data is from cycle times and not process times, and the other modes could mean the cycle times are dominated by bottlenecks on other stations.

**Figure 6.2:** Calibration examples of automatic and manual fits.
The current implementation accesses the database on every update of the selected products, but even using indexes the database has high latency. The work overhead of opening the calibration window makes calibrating each time adjustment less productive than opening the window once and using the open window to do all the fitting, taking notes of the results and later go through all the steps just filling the values. Thus the calibration window should be incorporated into the middle area of the main screen.

### 6.3 Analyze Layout Optimization Opportunities

With the station settings properly specified, the layout optimization phase can start. It is time to simulate and evaluate whether the results presented help to identify bottlenecks and to suggest optimizations in the layout.

Small batches of 50 pieces were simulated, which are large enough to identify dominant behavior. Figure 6.3 shows an overview of how the results are arranged on screen, and Figure 6.4 shows an enlarged version of the details.

**Figure 6.3:** Simulation result overview. The table summarizing main results is on the top right corner, and a detailed view of the production flow and the operations follows bellow it.

Using the settings provided by the experts, simulation results show station A10 is the main bottleneck for the first product type analyzed. It a bottleneck for about 67% of whole simulation time, and exhibits an improvement room of 7.6%. The production line is starting or stopping the remaining of the time. While using settings calibrated with historical data reveals station A10 is the bottleneck only for about 15% of the time, and station A8 is the bottleneck for about 40% of the time. The room for improvement of each of the stations is only 1.6%, which together sum up to 3.2%. See those results in Figure 6.5.

For the second product type analyzed similar situation occurs. Here the detail of a bottleneck change is shown in Figure 6.6.
Figure 6.4: Actors’ Performance Detail. For large number of stations, it becomes necessary to have a minimum height for each station, and add a scrolling mechanism. Notice on the 5th row that the station is critical (light red background), but it can be barely seen. Above that row, notice how the connecting lines are very long and inclined, filling almost all the space. In the density part, on the right below, notice a throughput valley propagating through the stations.

(a) Results using fixed time settings obtained from experts.  
(b) Results from settings calibrated with historical data.

Figure 6.5: Simulation results for the first analyzed product. Notice results from the calibrated case indicate the bottleneck changes between stations A8 and A10.

The room for improvement in the line analyzed is small, indicating at least two stations limit the throughput. Nevertheless many stations are underutilized (seen as blue in Figure 6.7), at least for the products analyzed, meaning the cost may be reduced by running them slower, for example. In case of a production expansion, another option is to share the underutilized stations, duplicating only the busy stations.
Figure 6.6: Detail of bottleneck changing from station A10 (fourth row) to station A28 (second last row).

Figure 6.7: Simulation results for the second analyzed product. A few underutilized stations are seen (light blue region to the right in the bars). The in-cycle idle time of the stations shown ranges from 26.7% to 34%.
7 Discussion

7.1 Layout Optimization Support

The results that were obtained (identified bottlenecks, resource drag values and in-cycle idle time values) are evidence that the implemented methods work. CPM can be used successfully to identify bottlenecks in production simulation data.

Drags are useful in project management, but they are not so useful in a serial production setting. The model approximation which considers the simulated sequence the only valid sequence for calculating the parameters (total-float, free-float and drag) also contributes to the lack of accuracy.

In-cycle idle times showed to be a useful metric. They differ from Total Float in that they are limited to time that is available without dislocating the station’s next operation, similarly to Free Floats. On the other hand, they differ from Free Float in that the start time of the work piece in the next station does not restrict in-cycle idle times, just like happens with the Total Float.

The bottlenecks can be precisely traced, but while the production is not finished multiple paths may have no float. And more, float can be introduced to a path by future events, such as a future longer operation on a parallel path, transforming a critical path into noncritical.

7.2 Assembly Station and Deadlocks

Randomly generated layouts which include an assembly step are likely to fail, and even if they do work, they may reach a deadlock situation in which a piece keeps waiting forever for the correspondent piece to arrive but the other piece is also waiting for the first one on another assembly station. And although no new events are possible, the simulation does not terminate with a fail, because fail detection mechanism assumes the operation will finish and the simulation continue.
Here there would be other possibilities, but they are not how things work on most production lines:

**Do not block any workposition:** This makes it very unlikely for an assemble to be executed. It would require the piece to be able to return and have to try many times until there is a hit. This is not how things work in a real plant.

**Block a single workposition:** Even blocking only one piece, there are layouts which could create a deadlock.

The assembly station also introduces clutter in the detailed view of operations, because lines from different links cross each other.

### 7.3 Simulation speed

The simulation advances in time-steps of 10 ms. This rather than an approximation, might actually happen in practice too, due to Programmable Logic Controller (PLC) scan cycle.

But today, the simulations take much longer than they should to complete, because the tool computes the motion of each piece at each time step, even when it is done in max speed. This is done in order to calculate the time the pieces kept on each conveyor segment, for later being displayed in the layout view. Since both the conveyor speed and the segment length are known, it is possible to wait until the piece stops moving and then update all stay times in the path.
8 Conclusion & Future Work

This thesis presented new metrics to support the layout optimization of manufacturing lines, and ways to visualize them. It also extended what could be modeled with the simulation tool by introducing time variability and a new assembly station, and implemented a visualization that uses historical data to calibrate and validate time-related settings.

Time variability was added to the design, and effects derived from it appeared in the simulation. When desired, the exact same simulation results can be replicated under the same settings.

The interface implemented for configuring the simulation settings was able to create the realistic case without recurring to manual XML file editing.

The thesis successfully applied the Critical Path Method (CPM) to precisely trace bottlenecks in a simulated manufacturing line. It also developed the formulation for calculating resource drags.

The evaluation showed how the new metrics resource drags and in-cycle idle time may be used to help guide layout optimization effort.

8.1 Future Work

Some future work opportunities were identified. Visual Analytics could help identify Pareto-optimal layouts, a scatter plot could be used to display the mean values of multiple simulation repetitions, over cost and time scores, and when one of those means is selected, the individual simulation results it summarizes could be plotted as well. The density and the operations’ detail view should be separated, eliminating the occlusion of the density view, and improving interaction consistency. A scroll mechanism should be applied, synchronized for both vertical axes. The score system could be improved, by using metrics that scale with the number of pieces produced, like average cost per item, or by adding metrics such as profitability or return on investment. On the detail view, more lines should be shown, highlighting the active dependencies (those impacting on the schedule), and some of the other lines should become semitransparent according
to their length and the zoom level. On the calibration window, an interaction capable of selecting a range of values in data, and fit the distributions only to the selected data cases might be helpful. The buffer sizes should be made parameterizable. The layout discovery should be able to accept a layout from the layout tree as a seed. The simulator speed should be improved by transforming the simulator into a pure discrete event simulator, enhancing response times to favor exploration.
Bibliography


All links were last followed on September 19, 2016.
Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

______________________________

place, date, signature