Faculdade de Engenharia da Universidade do Porto

Self-adapting production control methodologies

Manuel Tomé de Andrade Silva

Submitted to Faculdade de Engenharia da Universidade do Porto in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Leaders for Technical Industries, supervised by Américo Lopes de Azevedo, Associate Professor of Faculdade de Engenharia da Universidade do Porto

2023

This research was supported by the Ph.D. grant PD/BD/128439/2017/ awarded by the Portuguese Foundation for Science and Technology under the MIT Portugal program.

Acknowledgments

First and foremost, I would like to thank Professor Américo Azevedo, I am deeply grateful for the invaluable guidance and direction. One of the aspects I most appreciate is the respect shown towards my unique style and my autonomy. The ability to offer insight and advice without imposing or overriding my personal preferences was very empowering. This delicate balance of supportive leadership has been truly inspiring.

To my cherished family, with a particular mention of my beloved sister and mother, whose unwavering support has been the bedrock of my resilience. Over the course of these past five years - which, by a twist of fate, manifested as the most challenging period of my life - you have steadfastly stood by my side. Through the tempests and triumphs, your love and encouragement have been the beacon that guided me. I am immeasurably grateful and deeply indebted to you for the sacrifices you have made and the boundless love you have shown me.

To Filipa, “my partner in crime”, my confidante, thank you for all the love and support. With every laughter we shared, and through every tear we wiped, you have enriched my life in immeasurable ways.

Abstract

In the past 60 years, the focus in manufacturing has shifted from low-cost mass production to low-cost mass customization. However, achieving mass production efficiency in complex manufacturing systems that support mass customization has proven to be a challenge due to inefficiencies such as frequent setup times, higher demand and processing time variability, and complexity management.

In this thesis we present research around the problem of higher variability in the processes that govern complex manufacturing systems. The problem of higher variability is often dealt with by increasing WIP levels, the most common buffer to mask inefficiencies. WIP parameter setting, a paramount design decision in any pull manufacturing system, has shown that we can substantially reduce organisations' WIP levels, without hurting productivity.

Through the integration of 3 large fields of study: production management, simulation, and deep learning; we have built a solution that is capable of outperforming current industry standards in WIP parameter setting. We have trained different reinforcement learning (RL) agents that take has input the real-time state of a production system and output a WIP cap limit at discrete time steps along a production system run.

This approach is particularly critical for low-volume, high-variety production systems where real-time conditions vary significantly. Our research is supported by an exploratory study involving simulation experiments. We employ a novel approach using deep reinforcement learning to demonstrate the possibility of substantially reducing WIP levels without jeopardizing throughput performance. While this method requires significant computational power during the training phase, it results in a policy that is resilient to changes in the random processes controlling the production systems. To prove the efficacy of the proposed approach we benchmark our RL agents against the most widely used WIP parameter setting methods: fixed WIP limits based on steady-state analysis and statistical throughput control. We have shown performance superiority for traditional flow lines, as well as mix-product production systems, in the setting of make-to-order environments.

Resumo

Nos últimos 60 anos, o foco da indústria tem vindo a mudar. No início, através de economias de escala, padronização de produtos, especialização de postos de trabalho, a produção em massa era vista como o sendo o principal objetivo. No entanto, nos dias de hoje, com a necessidade de satisfação de requisitos variados por parte de um grande conjunto de clientes, as organizações enfrentam o desafio de atingir as eficiências da produção em massa, num ambiente de alta customização. Estes ambientes são normalmente caracterizados por uma grande complexidade que resulta em ineficiências que acabam por tomar várias formas: longos e frequentes tempos de *set-up*, elevada variabilidade da procura e dos tempos de processamento, difícil gestão da complexidade, para nomear algumas. Nesta dissertação tomamos como objetivo abordar o problema da elevada variabilidade destes sistemas. Usualmente, a primeira linha de ação quando deparados com elevada variabilidade é aumentar a quantidade de WIP (*work in progress*). Esta estratégia permite manter elevados níveis de utilização dos recursos, que por sua vez permite manter altos níveis de produtividade. A definição inteligente dos níveis de WIP, um dos mais importantes parâmetros no design de sistemas de produção puxada, tem vindo a demonstrar a possibilidade da redução do WIP sem que as organizações sofram perdas elevadas da produtividade.

Através da integração de 3 grandes áreas de estudo: gestão da produção, simulação e aprendizagem profunda (deep learning) construímos uma solução capaz de superar os atuais métodos de definição do WIP utilizados na indústria. Foram treinados vários agentes de decisão com métodos de aprendizagem por reforço. Estes agentes recebem como input o estado em tempo real de um sistema de produção e prescrevem um nível de WIP de acordo com as necessidades do sistema. Os métodos resultantes deste estudo foram comparados com os métodos mais comumente utilizados na prática: WIPs fixos definidos através de análise dos sistemas em estado de equilíbrio, e o método denominado por *statistical throughput control.* Demonstramos melhor performance do método proposto tanto em ambientes de linhas de produção dedicadas, como em ambientes de linhas de produção mistas, no contexto de produção por encomenda.

List of Figures

List of Tables

Table of Contents

Chapter 1 Introduction

This chapter presents an overview of WIP (Work in Progress) parameter setting, which is a significant challenge in modern manufacturing and production systems. We will also discuss the historical development and limitations of the various solutions that have been proposed to address this problem. Finally, we present the research goals, formulate different research questions, and set the research expected outcomes.

1.1 WIP parameter setting on pull methods

In make-to-order production environments, especially where product variety is high and volume is low, the dynamics of work-in-progress (WIP) management becomes an important challenge. The traditional static WIP parameter setting methods often struggle to effectively balance throughput and inventory levels in these scenarios, as they cannot adjust to the rapidly changing demand and capacity conditions inherent in such systems. Therefore, there is a growing need for a more flexible, dynamic approach to WIP parameter setting. This would allow the system to adapt in real-time, improving efficiency and responsiveness, and ultimately leading to better performance and customer satisfaction. The following paragraph delves into the potential of such an approach, specifically using deep reinforcement learning to dynamically set WIP parameters in make-to-order production systems.

All pull methods were created with similar goals: controlling the amount of WIP within systems. The goal is to reap the benefits of operating with low WIP (work in progress) without losing TH (throughput). Pull methods try to address this problem by setting explicit WIP limits and observing the resulting TH. This is in direct opposition to push methods, where we try to control TH (release rate) and observe resulting WIP levels. The most significant advantage of pull is its ability to achieve the same TH with lower WIP levels, on average(Hopp & Spearman, 2011 sec 10). Striking this balance is a trade-off game. Reducing WIP abruptly decreases inter-workstation buffers to the point that starvation can occur more frequently and hurt TH performance, especially in systems with high variability. As a result, for practitioners, when designing their pull systems, it is safer to set comfortable WIP limits to guarantee that utilization and TH do not get affected, incurring in higher inventory costs but still reaping the benefits of having an explicit WIP limit. These WIP limits are also not changed frequently. It is apparent that there is a lot of room for improvement on two fronts: first, can we zbe greedier towards reducing WIP without affecting TH; second, in the advent of industry 4.0 and the internet of things, frequently re-setting WIP limits is now a possibility. Production systems are highly dynamic and

stochastic entities. Therefore, if we can track their real time-state, we can use that information to adapt our pull policies accordingly.

Nonetheless, there are conflicting opinions regarding the development of sophisticated methods for calculating WIP cap levels, a WIP cap is to be understood as the maximum WIP to be allowed in the system at any given time. Hopp and Spearman (2011, sec. 13.5.4) believed that it was more or less useless to put considerable effort into developing complex models to estimate WIP caps for pull policies. The argument is that WIP caps should not be frequently changed, as WIP is very much insensitive to those changes. To better understand why, we can look at the following example: if we set a WIP cap to be high—letting a large number of orders to come in—and right after there is a machine breakdown that bottlenecks our system, if we then set a small WIP cap after the fact, it will take a long time until WIP lowers to the point where that WIP cap is respected. Hence, it is understood that overestimating the system's current capacity makes subsequent WIP cap recalculations as useful as just dictating a request to stop the release of parts until the capacity is restored. On the other hand, if we underestimate capacity and set a small WIP cap, we may lose throughput temporarily. However, if we change the cap frequently, WIP will be very sensitive to those changes, as WIP can very rapidly be increased, just by the release of new orders. At this point, we can envision a method that learns to avoid capacity overestimation with frequent WIP cap recalculations.

Two different operation modes in production systems are relevant for WIP parameter setting. A system should be designed so that on average its available capacity C (units/time, after unplanned and planned downtime) is greater or equal to the arrival rate λ. Nevertheless, at any point, stochastic systems can be in a state where $C > \lambda$, or $C < \lambda$.

When $C > \lambda$, setting a WIP cap is pretty much irrelevant as WIP will not grow, and probably will never reach a given WIP cap. Maximizing TH, in these cases, will only be relevant, if there is WIP accumulation within the system, from a previous period where $C < \lambda$. Even in those cases, there is not much we can do to maximize TH, as increasing a WIP cap on a period where $C > \lambda$ is pretty much useless since WIP in these circumstances cannot grow.

When $C < \lambda$, setting the right WIP cap is very important. In these scenarios, maximizing TH is crucial, we want to make use of all the capacity available, as efficiently as possible, because the system is currently being flooded with orders. Hence, what is expected is a sufficient increase

in the WIP Cap so that process starvation is avoided, and TH is maximized. The question is how much should WIP be allowed to grow? Allow it to grow indefinitely and you have a pure push system, TH will be maximized at the cost of carrying a lot more WIP than necessary to guarantee maximum TH. Having it fixed at a given amount will reduce average WIP, with no guarantee of TH maximization, these are the traditional pull systems with fixed WIP caps. These WIP caps are usually found resorting to steady-state analysis, and hence assume a system operating in steady-state conditions. In reality, production systems beyond being stochastic can also be dynamic, hence steady-state behavior may not be easily observed. From this point of view, there may be gains to be had by letting the WIP cap react to state changes as they happen, without resorting to steady-state behavior assumptions.

Despite this new window of opportunity to improve WIP parameter setting methods, the still persistent trend is focused on improving pull, not by better WIP parameter setting, but by reinventing policy architectures to make the pull approach work for lower volume, high product mix, and high variability production systems (Fernandes & do Carmo-Silva, 2006; Lödding et al., 2003; Spearman et al., 1990; Suri, 1998). Each method, based on the key ideas of pull, introduced modifications that made it easier, in terms of complexity management, to extend the application of pull to production systems with the abovementioned characteristics. For example, CONWIP relaxed Kanban's requirement to keep stock for every part controlled within the policy; CONWIP only has one type of card, which is non-specific, thus reducing the WIP that institutions are required to maintain. The research into pull architectures is a well-explored problem that is still very relevant based on the fact that between 1998 and 2018, 13 new architectures have been proposed (Bagni et al., 2021), and because we live in the "age" of the Lean Enterprise, reducing inventory is still one of the top priorities.

Every single one of these pull methods has as a commonality the need for parameter setting, which is, as stated, an implicit or explicit limit for how much WIP can grow. We view parameter setting methods as one more catalyst to the implementation of pull if they help deal with the variability associated with lower volume, high product-mix production systems without the fear of losing TH performance. Nevertheless, these kinds of techniques have not gained much traction, probably because some of them rely on simplifications of real environments (e.g., deterministic demand and other relevant variables, or other axiomatic models), and require high computational

power combined with the fact that at decision time, these procedures, often relying on heuristics, cannot be run in constant time.

Considering these limitations, we investigate the potential of dynamically prescribing WIP limits to maximize throughput performance and minimize WIP levels according to the production system state, i.e., considering the real-state conditions arising from process variability, which are especially crucial in low-volume and high-variety production systems. An exploratory study based on simulation experiments is used to support our research. Using an innovative approach based on deep reinforcement learning, we demonstrate that we can significantly reduce the WIP levels without compromising our production systems' throughput performance. With this method, we attempt to remove, as much as it is possible, assumptions about the production systems while having a procedure that runs in constant time at every decision epoch. This method still requires high computational power during the reinforcement learning Agent training phase, but once trained, the agent's policy is robust to shifts in the random processes that govern the production systems. We use a similar definition of robustness as Yan, Lou, and Sethi (2000), although our focus is not measuring robustness. Techniques similar to the one presented in this study are possible stepping stones for the vision of fully autonomous production, where no human control is needed at the operational level.

1.2 Objectives and research questions

Considering the issues with current WIP parameter setting methods, we investigate the potential of dynamically prescribing WIP limits to maximize throughput performance and minimize WIP levels according to the production system state, i.e., considering the real-state conditions arising from process variability.

In the context of these objectives, we defined the following research questions:

RQ1: Can artificial intelligence methodologies be used in the context of WIP parameter setting method?

Dynamic parameter setting has not yet seen many approaches using Artificial Intelligence techniques, as we will see in section 2.5.7(b). Artificial intelligence has been setting new state-of-the-art performance records in many fields of study, from Medicine to Transportation. Nonetheless, industrial applications are still few and limited to a small cluster of international companies (Bertolini et al., 2021). A further aspect that makes Artificial Intelligence techniques interesting to explore is the fact that enabling technologies (i.e., sensors, public datasets, computing power) are maturing and becoming readily available. Furthermore, with the advent of Industry 4.0, many enterprises already have large amounts of data at their disposal often not giving it due treatment and analysis. Dynamic parameter setting is one more problem that can make use of this information.

This research question can be divided into 4 different steps:

- Formally describing the dynamic WIP parameter setting problem, its optimization goals, and constraints.
- Finding the Artificial Intelligence methods available that are suited to address the characteristics of the dynamic WIP parameter setting problem.
- From the set of methods found to be the best suited, find the best algorithms, adapting them as required.
- Provide a solution based on the selected methodologies.

RQ2: What scalability challenges are involved in extending the dynamic WIP parameter setting method to more complex manufacturing systems?

Most WIP parameter setting methods, when presented, are tested under simple production systems. Flowlines are the most obvious contender, for their simplicity of implementation. Debugging a new methodology over a very complex system is often counter-productive, as we have to deal with the complexity of not just coming up with a new solution, but also implementing it at scale. We want to avoid this while still keeping the research relevant for more complex systems.

To answer these research questions, the following general methodology will be applied: hypothetical production systems will be envisioned and simulated resorting to discrete event simulations; the system will be verified for stability to guarantee that arrival rates and overall TH

are aligned, and we do not have a system where WIP naturally grows. Reinforcement Learning Agents will be trained by interacting with these simulated environments, with the goal of maximizing TH and minimizing WIP inside the systems. Finally, we will benchmark performance against different WIP parameter setting methods.

1.3 Expected outcomes

O1: A solution that runs in constant time at decision time

Very computationally intensive procedures may be prohibitively slow for real-time decision-making, especially if decision epochs are short. For the WIP parameter setting the solution space is often very large – the set natural numbers. It would be huge progress if we could devise a methodology where even given an infinite solution space, we could find a good solution in constant time.

O2: Benchmark proposed solutions with other relevant WIP cap setting methodologies

The last expected outcome is to demonstrate performance improvement over current relevant solutions. Due to the large number of WIP parameter setting methods, we want to restrict these comparisons to other methods that prescribe WIP limits by taking as input the system's real-time information. There is a set of relevant methods that try to avoid any assumptions about random process distributions, these methods exist in opposition to others that rely on steady-state analysis. We will consider the production system as dynamic and stochastic entities therefore for comparison purposes we intend to prioritize these kinds of methodologies.

1.4 Thesis outline

This dissertation is organized into 6 main chapters and 2 complementary ones. Chapter 1 provides the reader with the definition of the goals of the dissertation and a short introduction to the theme of WIP parameter setting and its relevancy. Chapter 2 goes into detail describing the impact of production flow control on the performance of production systems on the fronts of cycle-

time, customer lead-time, and throughput. We present the different pull methods and define the goals of WIP parameter setting procedures in the context of the different pull methods.

In chapter 3 we present the research methodology. Going through its strengths and weaknesses and what kind of conclusions can we take in the face of the chosen research methods. In chapter 4 to 6 we reformulate the WIP parameter setting problem in the framework of a Reinforcement learning problem. Solving it using 2 different algorithms and benchmarking them against the most common WIP parameter setting methods used in practice.

In Chapter 7 we adapt the RL learning problem defined through chapters 4 to 6 to a setting of complex, mix-product manufacturing systems. We train new agents for the new setting and redo the benchmarks.

In Chapter 8 we present the main conclusions, followed by a brief overview of the challenges of the adoption of methods similar to the one proposed in this dissertation. And finally, we provide a very short reflection on how the proposed method could be extended beyond just CONWIP to other pull methods.

Chapter 2 Literature review and background

In this chapter, we lay out the background behind WIP parameter setting in the broader context of production flow control. Explaining the reasons for enforcing a WIP cap and its impact on production systems. To go beyond just citing the literature, we also make use of simulation to make brief demonstrations that corroborate the presented material. Finally, we show the literature around pull production, demonstrating that pull is essentially a reference to a group of methods that work by explicitly defining WIP limits. Finally, we review the various WIP parameter setting methods used within the context of pull production, including their benefits and limitations, and identify opportunities for improvement

2.1 Key concepts and definitions

In the literature flow control is often associated with flow shop scheduling problems and their variants (Johnson, 1954), where we have *n* machines and *m* jobs and each job goes through each machine sequentially. Specific operations will be performed in each machine, and those operations have to happen in that order, from machine *i* to *n*. Examples of these problems have been extensively studied for decades and research still perseveres in this field (see e.g Li, Luo, Xue, & Tu, 2011 and Peng & McFarlane, 2004).

The flow control that we will work upon in this thesis is related to the flow of parts that go through a product route within a production system, we are not interested in the scheduling of parts to workstations, but in ways of understanding if the flow of parts should be stopped.

Throughout this work, we will use some definitions like customer lead-time, cycle-time, throughput, and WIP. Some of these definitions are well established and universal within research communities, however some of them depending on whom you ask may be different, therefore we will define those terms as we want the reader to understand them for the rest of the thesis:

Customer lead-time (CLT): Time between accepting an order from a customer to the time it is delivered to the client.

Throughput (TH): Rate in which parts exit the system (units/time).

Cycle-time (CT): cycle-time of a production system measures the time it takes for an order that was released onto the shop floor to reach the finished goods inventory. Cycle-time can be used more broadly; for example, it may refer to the interval of time between process exits. This definition is the one that varies the most according to whom you ask, some researchers and practitioners will call this definition flow time or even lead-time. The latter causing confusion with customer lead-time.

Input-rate: Rate at which orders are released into the system (units/time).

In (1961) D.C Little provided a proof for the queuing formula, which is now more commonly known as Little's law. Little's law under certain conditions of stationarity of the processes, states that at the limit, when time goes to infinity this relation holds $CT = \frac{WIP}{TH}$. Of course, this can only serve as a guide because there is no guarantee that the processes are truly stationary, nor can we

observe it long enough so that this relation holds. Nonetheless, it is enough in practical terms to use this formula to approximate the long-term behavior of a system, given that we can observe at least two of these variables with consistent units.

From direct observation of this formula, it becomes apparent that to decrease cycle-time we can either decrease *WIP* or increase the throughput of the system. Increasing throughput may be done by increasing capacity on bottlenecks and decreasing *WIP* may be achieved by opting for order release strategies that consider the state of the system, like most pull strategies. (Spearman et al., 1990; Sugimori, Kusonoki, et al., 1977).

When using Little's law one should be careful, this formula gives the relation between these 3 quantities, but it does not tell us, for example, what is going to happen if we decrease WIP. If we decreased WIP knowing that TH will remain the same, then we could with confidence say that CT would decrease, but if WIP is decreased will the TH be the same? We have no guarantees of that being true as the amount of WIP that we allow will affect the TH of the system.

It also should be clear that the TH of a line is different from the capacity of a line (units/time) because TH is dependent on the utilization of resources and capacity is not. When using Little's Law we have to make sure that we are in fact using TH. When a machine is working below its capacity, beyond physical problems a machine may be experiencing, this behavior can be explained by two scenarios: the input rate of the system is not enough to maintain the machine working all the time, which is the obvious reason, or, there is variability on precedent machines, therefore sometimes the machine is waiting, idle, other times the machine has a queue in front of it.

TH can then be defined as $capacity \times real\ utilization$, where $real\ utilization$ is a function of release rate. Hence, it should be obvious that throughput can never be higher than the release rate. Of course, that momentarily the instantaneous release rate may be higher than the instantaneous output, or vice-versa, but on average, when time tends to infinity TH can only be less or equal to the release rate. Also, it is important to notice that the release rate is different from the rate at which orders come into the company, there is often a hierarchy process from demand to production plan, to scheduling (MRP, MRP II, Finite capacity algorithms), or other order release strategies (see e.g (Orlicky & Plossl, 1994), and (Almada-Lobo et al., 2015))

Finally, we can summarize three dimensions of flow within a production system:

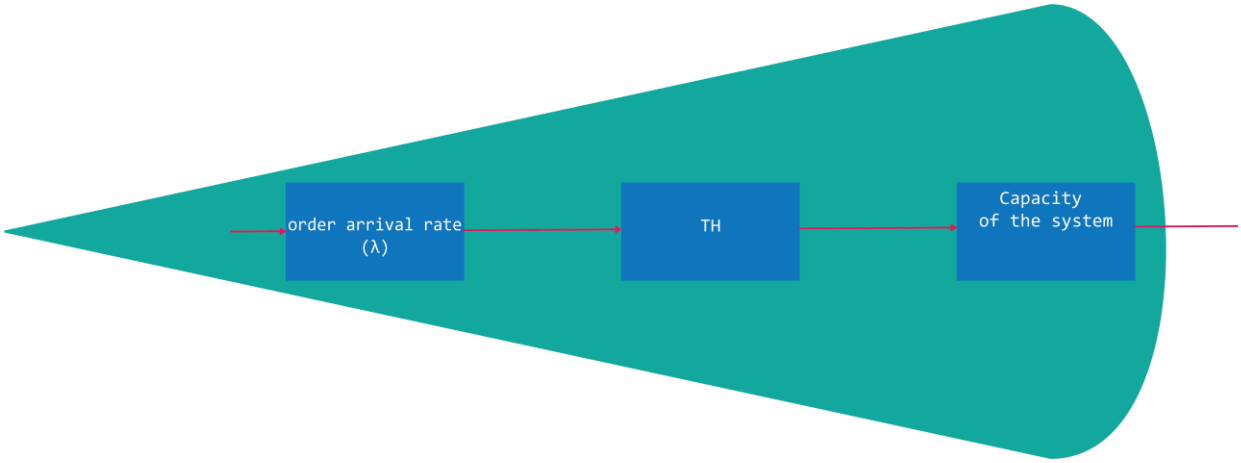


Figure 1- Dimensions of flow

In Figure 1 we first have order arrival rate, secondly, we have the TH, and lastly the potential capacity of the system. In Figure 1, ideally, on average we would have all the flows being aligned, but in practice, we always have more capacity so that we can account for planned and unplanned downtimes. TH is rarely equal to capacity as we will fail to release the orders in a way so that bottlenecks are not starved, especially in high variability scenarios and because this theoretical capacity is rarely available. TH tends to be higher than order arrival rate, or if it is not, we must resort to order rejection. The latter is not a very good long-term management strategy as it means that we are losing customers to potential competitors and throwing away a market share, that could be ours have we been prepared to meet demand.

2.2 Understanding the impact of production flow control on the performance of a system by enforcing a WIP cap

Some authors when advocating the reduction of WIP as a measure to control customer lead-time seem to equate cycle-time reduction to be equal to customer lead-time reduction or that reducing the first will reduce the latter. The problem is not that their conclusions are wrong, the problem is that both terms are used interchangeably when authors are referring to different

quantities. The effects of WIP on cycle-time were already extensively studied both analytically or by the use of simulations (Jodlbauer, 2008; Spearman et al., 1990).

Let us consider a plant that works in a make-to-order environment. Demand comes in the form of orders and customer lead-time is quoted at the time of the order. Demand is then turned into a production plan by a scheduling mechanism, let us assume that this mechanism includes capacity constraints. In this step, we just converted order arrival into a potential release rate, again we reinforce the difference between order arrival and release rate. From the previous step, an order release date is determined and orders that are due in the same time bucket are sequenced by yet another procedure, as to determine the sequence in which orders being released at the same time bucket will enter the shop-floor. Now, as to synchronize TH with release rate, let us assume that the company uses a CONWIP strategy where another order can only be released in the system if the WIP on its route is less than a fixed amount (Spearman et al., 1990).

In the production system just hypothesized there is a schedule and a sequence but what dictates the release rate is a CONWIP strategy. This means that if TH for some reason is less than expected the rate of release will also be less than expected, hence some orders scheduled for a given time bucket will have to be postponed to another time bucket.

Now, we ask the reader to imagine another plant that works exactly under the same circumstances, but without any WIP cap strategy, where the release rate is exactly as defined on the schedule and sequence, in other words, there is no synchronization between TH and release rate, this is a traditional push system. In this scenario, orders will always be released on schedule and wait as work in progress. The question that arises is, in what scenario is customer lead-time expected to be less?

To illustrate a conceptual answer let us look at the following diagram:

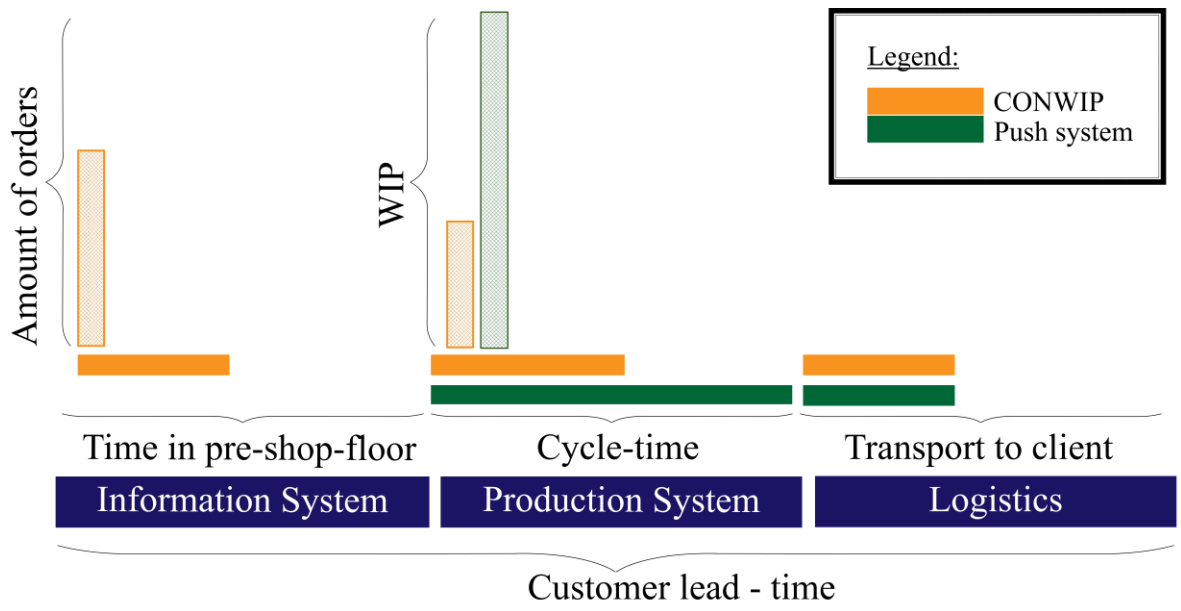


Figure 2- Waiting time of an order within a make-to-order production system for the CONWIP strategy and the Push system

In Figure 2 we divide the customer lead-time into 3 components: time on the pre-shop floor; cycle-time and transport to the client. We will not bother about the transport to the client, because that will include the time spent on finished goods inventory (FGI) and the time in transit, which are out of the scope of this work. But it is of value to conjecture what is going to happen to the other components of customer lead-time for the CONWIP and push systems we have just defined.

Starting with the CONWIP system: practically for any schedule, regardless of if it can be accomplished or not, in a CONWIP system with a limited amount of WIP allowed, if TH is the same, the cycle time will be lower by Little's law. Now, on the Push system, orders will be released into the system as established in the schedule, independent of WIP levels being very high or not, therefore cycle-time will be higher, again by Little's Law. In practice in the Push system the time spent in pre-shop floor is zero, as all the orders are going to be released at the beginning of the time bucket and then wait on the buffers between workstations to be processed as WIP. On the other hand, in the CONWIP environment orders will be released until the WIP cap is met, after that, a following order will only be released when another one comes out. Therefore, for the CONWIP system orders instead of waiting as materials on workstation buffers, wait as information

before the production system on what we call the pre-shop floor. From this perspective, it becomes apparent that given an X number of orders to process on a given time bucket the customer lead-time for the last order is always going to be X/TH. In other words, the time an order spends in production will always be equal to the time it takes to produce all the parts ahead of it, plus the time to produce the part in question. It does not matter if a part spends less time on the shop floor as it will have to wait elsewhere, given that throughput is the same.

It should now be clear that customer lead-time will be the same for both systems as long the TH remains the same. The only significant difference between the systems will be the form in which orders await. In the CONWIP strategy, we can postpone the commitment of materials to a given order and, in that way, you gain more flexibility to adjust the production schedule or to accommodate unpredicted changes. Also maintaining high levels of WIP can create all sorts of hassles like space management in the production system area, material deterioration, etc. It also means that if you can operate with fewer orders as WIP, you have less money invested in inventory, which makes a company more efficient.

2.3 Measuring the impact of flow control using discrete event simulation

To test the hypothesis just presented, a simple simulation can be performed to attest the conjectured behavior of the system.

For this simulation, our production system will have two workstations, and only one type of product will be produced out of the production line in Figure 3.

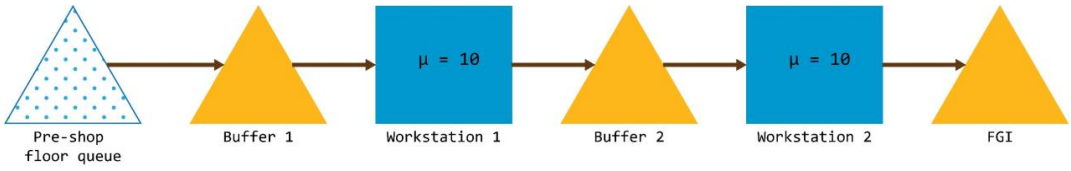


Figure 3- Simple flow line

In this system our order arrival rate will be constant, every 10 units of time an order will arrive, the release rate will be defined by a CONWIP strategy, and the sequence in which orders will be processed will always follow the FIFO rule. Note that the system just defined shares the

general characteristics of the one described conceptually in the previous section: the order arrival simulates the make-to-order environment, the release mechanism is defined by the CONWIP strategy, and the sequencing procedure is FIFO. The only difference between the system described, which is irrelevant for the analysis, is that we said that between release rate and order arrival rate there could be a scheduling procedure, in this case there isn't one.

In this example, no reentrant flows will be considered, and we will also consider that all parts that exit the system are completed with the required quality standards. There is no disqualification of parts. The goal of this simulation is to understand what the effect of WIP on customer lead time is. The reader should remember that costumer lead-time, under our definition presented in Figure 2 is the sum of the time spent on the pre-shop floor and cycle-time. For that effect, we are going to simulate scenarios for 10 different CONWIP strategies Table 1.

Table 1- Different WIP cap strategies used in the simulation

Strategy	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**
WIP	1	2	4	6	8	10	12	14	16	10000

In the last strategy, the CONWIP is defined as 10.000 units to simulate a pure push system, where parts are released into the system as orders arrive, we set the limit that high knowing that for such a WIP cap, there is no chance for that cap to enforce any restrictions into the system. The simulation was designed so that WIP will not get even close to the 10.000 unit limit.

We also want to pressure the system under different scenarios of variability, to make that possible we will simulate the processing times of our machines as pseudo-random processes that follow different uniform distributions with the same mean, but different coefficients of variation. This modeling choice was motivated because of the need to isolate the effects of having different levels of variability and still maintaining the same mean. The normal distribution could serve the purpose of isolating variability, but it would allow for the existence of negative processing times, therefore it fails to model the process adequately. Another distribution often used to model processing times is the exponential distribution, but because its mean is always equal to it's standard deviation, we cannot define the mean of the processing times independently of its

standard deviation. Or in other words, the coefficient of variation (CV) for the exponential distribution is always 1.

We will use 4 different levels of variability given by the uniform distributions in Table 2:

Table 2- Different uniform distributions for workstation processing times

Variability levels	**1**	**2**	**3**	**4**
Distribution parameters	[8, 12]	[5, 15]	[2, 18]	[0, 20]
Mean	10	10	10	10
Std	1.15	2.89	4.62	5.77
CV	0.12	0.29	0.46	0.58

Now we have 4 different scenarios of variability against which we can simulate our production system. We will perform a factorial experiment, meaning that every CONWIP strategy will be tested on 4 levels of variability. Under these conditions, we will make 40 simulations. Each simulation will be run over 3000 units of time, and no repetitions will be necessary as we are using pseudo-random variables. That means that when looking at Figure 4 below, for example, the blue line (UN Dist = (8, 12)), all the simulations with the different WIP caps have encountered the exactly same random values in the exact same order. That is why we can compare all the points on the blue line as they resulted from the same exact conditions. Or in other words, there is randomness within each simulation of the blue line, but there is no randomness between simulations. The same thing happens for the rest of the lines (different levels of variability).

Because we are testing the effect of WIP it takes some simulation time until we can see the full effect of the CONWIP strategy. As the simulation starts with 0 orders inside the production system some warm-up period was considered. We want to give the simulation enough time for the WIP cap to be defied before we start analyzing the data.

The variables we are going to observe are average customer lead-time, average cycle-time, average time on pre-shop floor, and average throughput.

2.3.1 Effect of a WIP cap on customer lead-time

When looking at Figure 4, we can observe both the effect of variability and WIP on customer lead-time for the simulated system. The first apparent behavior is that if WIP is too low average customer lead-time will be very high. This is not new, other authors made the same observations. We need a certain amount of WIP in the system as to guarantee that we have utilization near capacity, Hopp and Spearman (2011)coined this idea as critical WIP. In a system with balanced work, no variability nor randomness the critical WIP is equal to the number of workstations. For our simulation, that critical WIP, if variability and randomness did not exist would be 2. That would mean that every time a part would be finished in workstation 1 it could immediately start work in workstation 2. This is because a part is finished in workstation 2 at the exact same time a part is finished at station 1. Therefore, if WIP is less than 2 we will fall short of our capacity because release rate is not enough to keep the system utilized. Working with less than critical WIP would not make any sense if the resulting TH of that practice does not match demand. That would mean that we would be underutilizing our system when there were enough orders to make it work at full available capacity (C).

We can also see that if a certain level of WIP is achieved then adding more WIP will not decrease customer lead-time further. For the case with less variability, (blue line), that level of WIP happens at 3 units. The interpretation is that for this scenario a WIP equal to 3 is enough to keep the system with the highest utilization possible. Hence, adding more WIP is of no gain to one's TH. But the most important thing, which was the purpose of the simulation is that it shows that a higher WIP cap does not increase customer lead-time for a given demand (arrival rate(λ)) and production system available capacity(C). Now let us inspect the effects of variation on customer lead-time. The first thing to notice is that all the lines are pushed upwards as variability increases, which means that the average customer lead-time will be higher when variability is high.

The other interesting behavior is that as variability increases, the amount of WIP that establishes the point where no improvement is made by adding more WIP also increases. Or equating it differently, the amount of WIP necessary so customer lead-time can be as low as possible increases with higher variability. More WIP is needed for a lower customer lead-time to be achieved, even

though cycle time increases with WIP. To make it concrete with the data in Figure 4 for the 4 levels of variation simulated, from smallest to highest, the levels of WIP, from which an increase does not reflect any benefits on shorter customer lead times are, 4, 6, 8, 8, respectively.

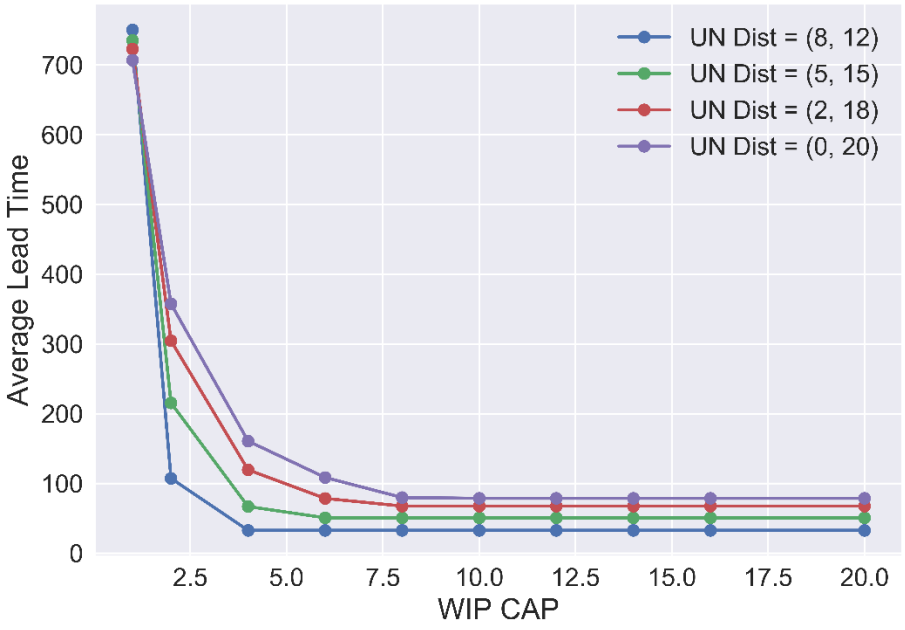


Figure 4- Effects of different WIP Cap Strategies on customer lead-time

2.3.2 Effect of a WIP cap on cycle-time

In Figure 5, as expected, higher levels of WIP allowed resulted in higher cycle-times, that is why most research points out that to reduce cycle-time we need to reduce WIP levels. A behavior that at a distracted glance might seem to contradict Little's Law is that it appears that after some point increasing WIP cap does not increase Cycle-time, again we should not confuse WIP with WIP cap. In our simulations, we are only restricting WIP so that it does not go above a certain limit not keeping WIP constant. So, naturally, for a given arrival rate and variability the WIP will not naturally grow indefinitely to higher levels, therefore higher WIP caps will never be

met, consequently, they will not affect the performance of the system. As we se, increasing variability will make WIP levels naturally grow if higher WIP caps are allowed, therefore average cycle time grows with variability.

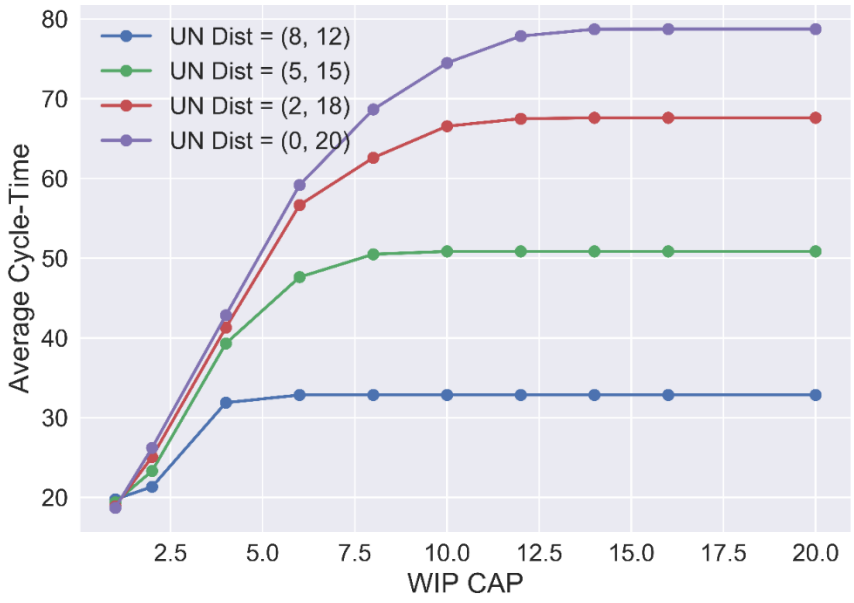


Figure 5- Effect of different WIP cap strategies on cycle-time

2.3.3 Effect of WIP cap on pre-shop-floor time

As we have seen in previous sections having higher WIP caps leads to higher cycle-times, but if cycle-times are lower, for a given level of variability, arrival rate, and TH, then if an order is spending less time in the shop-floor (cycle-time) then it should be spending that time elsewhere. This graph (Figure 6) shows the opposite behavior of that of the cycle time graph: as the WIP cap is increased orders spend less and less time on pre- shop floor; It should be noticed that the lead time in Figure 2 is the sum of the graph in Figure 6 and the one in Figure 5. It shows that on a CONWIP system, a part may spend less time on the shop-floor, but for a given arrival rate it means that that order must have waited before it entered production.

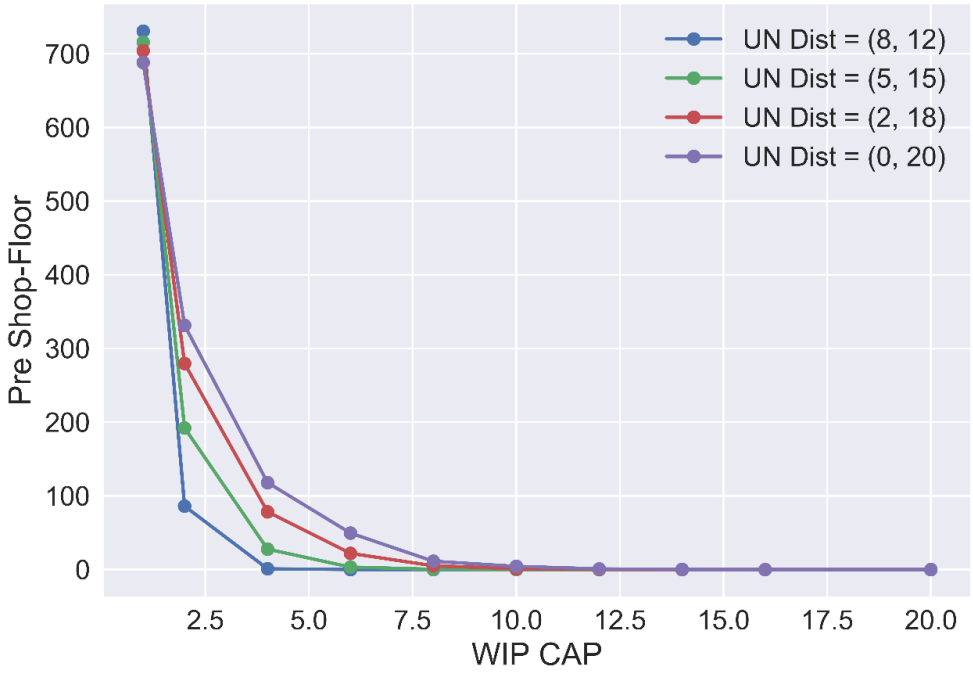


Figure 6- Effect of different WIP cap strategies on pre-shop-floor time

2.3.4 Effect of WIP cap on throughput

This is one of the most important measures for a company, there are very few reasons for a company to not want to maximize its throughput when there is a demand willing to absorb it. Therefore, the best we can achieve would be to have a TH as close to capacity as possible.

In Figure 7 we confirm the already presented idea that we need to allow a certain level of WIP, so it is possible to output orders at a higher rate. The higher the variability of a production system, the lower the overall throughput and the higher the levels of WIP necessary to obtain a given throughput. As observed before, TH does not increase forever as we increase the WIP cap, there is a point where no marginal gain in TH is obtained by adding one more unit of WIP into the system. In Figure 7 in the red color scheme, we witness that to achieve the highest throughput possible for the lowest variability we need a WIP cap of 3, for the highest level of variability that value grows to 8.

In Figure 7 we also display the average WIP for the different scenarios to make clear that having a WIP cap does not mean that the WIP will always be equal to the WIP Cap. For example, levels of WIP for the lowest variability scenario (UN DIST = (8, 12)) never grow past 3 units, even when the WIP cap is relaxed. This means that the system, with that arrival rate, capacity, and variability will never naturally retain a WIP past 3 units. Still, as soon as we increase the variability the WIP levels for the different WIP cap strategies grow as well.

To see that TH does not always grow proportionally with the growth of WIP, let us analyze the TH and WIP levels for the highest variation simulations (UN DIST = (0, 20)). For this example, TH stops increasing when the WIP cap is at 8 and when the average WIP is at a level close to 7. We can see that the average WIP for the scenarios with a WIP cap of 10, 12, 14, 16, and 10.000 grows until the average WIP is close to 8, after that, it remains constant. Consequently, we observe the average level of WIP growing from 7 to 8 with no gains in throughput.

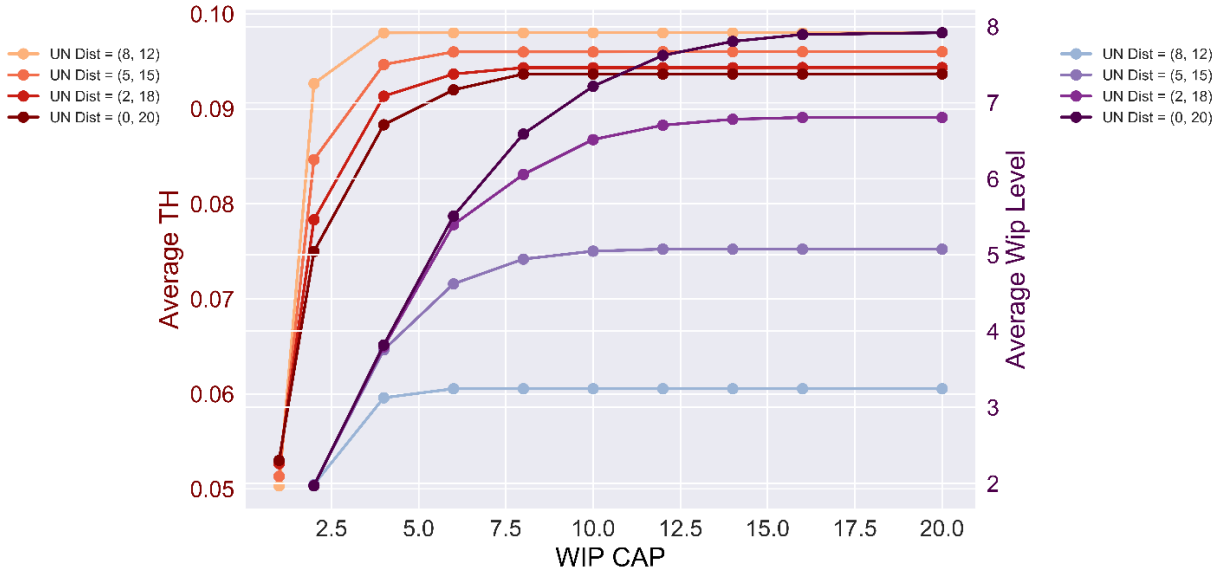


Figure 7- Effect of different WIP cap strategies on TH and WIP levels

2.4 Production flow research

Having understood the impact that controlling WIP has on a production system, it is important to catalog which methodologies have been proposed so we can find a path for improvement if there is one.

We understand as production flow control methods any method that tries to synchronize order release rate with the TH of a system. In the beginning, production planning and control technologies did not try to tackle the problem of flow control, production was scheduled, and enterprises relied upon MRP (material requirements planning). Here a Master Production Schedule (MPS) was the input necessary so that planned releases of components and end-items were computed, resorting to assumptions such as fixed processing and cycle times for all products and infinite capacity of the production systems. The flow of parts in these systems was dictated by a release plan that, if not feasible, would result in long cycle-times and high levels of WIP.

MRP-II (Manufacturing Resources Planning) was the technology that followed with the exact intention of dealing with the infeasibility of MPS plans that also brought a hierarchical framework for planning and control at different time horizons. The relevant improvements in MRP-II for this topic were the capacity checking modules: the rough-cut capacity planning (RCCP) (Berry et al., 1982) and capacity requirements planning (CRP) (Karni, 1982) functions. The first does a rough capacity requirement check to the MPS, by using the bill of resources of each item and checking if there is enough capacity available for the production of said items over the MPS plan horizon time buckets. The second, CRP, takes as input the MRP planned release orders, and together with information about cycle-times, WIP, routings and capacity for each workstation provides a capacity check for each process center. The result is an unreliable load map of each workstation over the planned time horizon. Although capacity checking is better than no capacity checking these two instruments rely on the exact same assumptions as MRP – fixed process cycle-times – which beyond giving an idea of the feasibility of a plan do not provide much else, as generated load maps are very unreliable. Hence, even though feasibility checks inform managers of infeasible plans, there is no mechanism for a systematic adjustment of those plans, so the potential for a production execution with high WIPs and long cycle-times is still high. That is why another function often offered alongside MRP-II was introduced – shop floor control (SFC). In this function, the idea of Input/Output control was offered as a potential solution. Input/Output

control was first proposed by Wight (1970) and later picked up by Fry and Smith (1987). The emphasis here is that to limit the growth of WIP within a production system there must be a mechanism that synchronizes TH with Input. If these two metrics are equal, WIP is bound to remain constant and any differences in those two metrics will result in a WIP increase or decrease, accordingly. Synchronizing TH with input is done by simple rules that establish that if WIP grows past its predefined limits, order release should slow down, and vice-versa. The authors report WIP reductions in the order of 42%, a decrease in order backlog of 93%, reduced customer lead-times by over 50%, and increased customer service to above 90%. These results seem to indicate the success of these methods. Input/Output control to the best of the author's knowledge was the first real attempt at flow control – independently of feasible or infeasible plans, this technique regulates the amount of WIP which enables shorter cycle-times and more rescheduling flexibility.

Later Spearman and Zazanis (1992b) demonstrated that controlling TH leads to less robust policies than if we were to control WIP. At first sight, it might seem like the two approaches will have the same result as they both try to cap WIP, but there is a nuance in using Input/output control because controlling TH is not trivial.TH is a function of capacity and utilization, meaning that to adjust throughput we would have to decrease the production rate of various workstations, which is done by increasing or decreasing input-rate. Due to production systems' complexity, making these adjustments can potentially lead to unpredictable changes in the overall throughput of a plant. Since then, a lot of research has sprung on methods that attempt to control WIP, by capping the number of items allowed within the overall and the different components of a system.

It is important to highlight that at the same time that MRP-II was evolving in North America, in Japan the idea of synchronizing TH with release rate was also developing and soon materializing into the idea of just-in-time and Kanban (Sugimori, Kusunoki, et al., 1977), which were part of a larger system called the Toyota Production System (Ohno, 1988). The idea of just-in-time is heavily inspired by the concept of the 1950s American Supermarket. In a supermarket, a customer takes products from shelves and this act creates a void in that particular shelf that needs to be re-stocked by the exact same quantity that a given customer has removed. The whole idea can be distilled in the following sentence: replenishing exactly what was removed in a timely fashion. To make this a reality, production environments had to change substantially so that efficiency was kept high even with the required smaller lot sizes. To that avail, Toyota had to reduce the set-up times dramatically which culminated in the practical guide, "SMED System" by Shingeo Shingo

(1985) who worked with Onho at Toyota. To implement just-in-time Toyota required a procedure to coordinate the movement of information and parts across the production system, that is the job of Kanban. The Kanban system was probably one of the first recorded attempts at synchronizing TH and release rate by controlling the amount of WIP allowed in a system. When JIT started to disseminate around professional and academic circles the term Pull Production emerged.

2.5 Pull production

In our research, we will adopt the definition of pull provided by Hopp and Spearman (2004). They state that any production system that has an explicit WIP limit can be called a pull system. The key here is the word "explicit" because in practice, every system has a WIP limit; it is unthinkable to imagine even in a MRP-II based system, that WIP would be allowed to grow indefinitely. Management most certainly would stop the release of orders and revise the Master Production Schedule and its planned order releases if WIP went out of control. The problem is that in these systems the WIP limit is arbitrary, implicit, and often too high. This definition gives us a clear distinction between Push and Pull systems, and from this vantage point, we can envision different pull mechanisms that work by enforcing WIP caps.

2.5.5 Pull advantages

Once the impacts of controlling WIP are understood some advantages of pull systems become apparent:

- Less congestion: we have shown through the use of simulations that the average WIP in these systems is reduced. This conclusion can be reached from the perspective of queuing theory by comparing an open queuing system (push equivalent) with a closed queuing system (pull equivalent) (Spearman & Zazanis, 1992b). In systems with great variability enforcing a WIP cap is even more important as WIP tends to grow in unstable systems. Less congestion means that inventories are low, which as stated in previous sections makes systems more efficient.

- Controllability: Pull is a more robust control mechanism than Input/output control or any other known methods that try to synchronize TH with input.

The aforementioned advantages are direct and quantifiable observations of comparing pull and push systems under the same operational conditions. But there are other desired qualities that enable pull system implementation that also deserve mentioning:

- Improved visibility: with less inventory in between workstations, any yield loss, would have a high probability of starving posterior workstations, this forces organizations to seek higher quality standards. Additionally, in case of undetected defects, stock inspections are faster.
- More efficient processes: as stated, with less inventory the buffer for errors and inefficiencies is very tight. If set-up times are long, there are unexpected machine breakdowns or any other external sources of entropy the result will be process starvation, which can dramatically impact TH. Ergo, pull requires and forces processes to be very robust and efficient.

Those perceived advantages are only the result of a successful implementation of pull. Just implementing pull control techniques will not automatically grant these advantages. Hence, as contradicting as it may seem those two advantages can as well be interpreted as implementation risks.

2.5.6 Pull methodologies

(a) Kanban

The classic version of Kanban is the two-card system (Figure 8): there are two kinds of production authorizations, one for the movement of parts and the other for its production. For every workstation there are two stock points: the inbound, where raw materials or subassemblies wait for processing; and the outbound where processed parts or subassemblies await movement to other workstation inbound stock points, or finish goods inventory.

The procedure starts first with a request for the movement of parts and every plant has material handlers that on a schedule are required to move parts around. In practice, these material handlers check the workstations for authorizations (cards) of movement, those authorizations signal that the current workstation needs its inbound inventory to get refilled by a given quantity of a specified part, that is located in a predetermined stock point. The job of this material handler is to go to that specified stock point, collect a container with the parts needed, remove its production card, transport it to the inbound stock point of the authorizing workstation, and re-attach the movement card previously collected. Every time a card is removed from a container it is placed in its appropriate board, being that each workstation has one board for production and another for movement cards. As we saw, every time a material handler collects a container from an outbound stock point he removes its production card and places it in that workstation production board, this provides the signal for that work center to start production and refill its outbound stock point with a specified part and quantity. The production will only start when there is a production card available, the workstation is idle and there is stock on the inbound stock point. Once those 3 requirements are met, the worker takes a container from the inbound stock point, removes the movement card from that container, puts it into the movement board, and starts production. In this system, the movement is the precursor for production and then the cycle repeats.

Movement cards are just a reflex of real-world scenarios where movement needs to occur between workstations outbound and inbound stock points. Theoretically, if an outbound stock point is the inbound for the next workstation, we can live with a one-card system, and for simplicity when referencing Kanban hereafter, the author will be referring to this simplified card version.

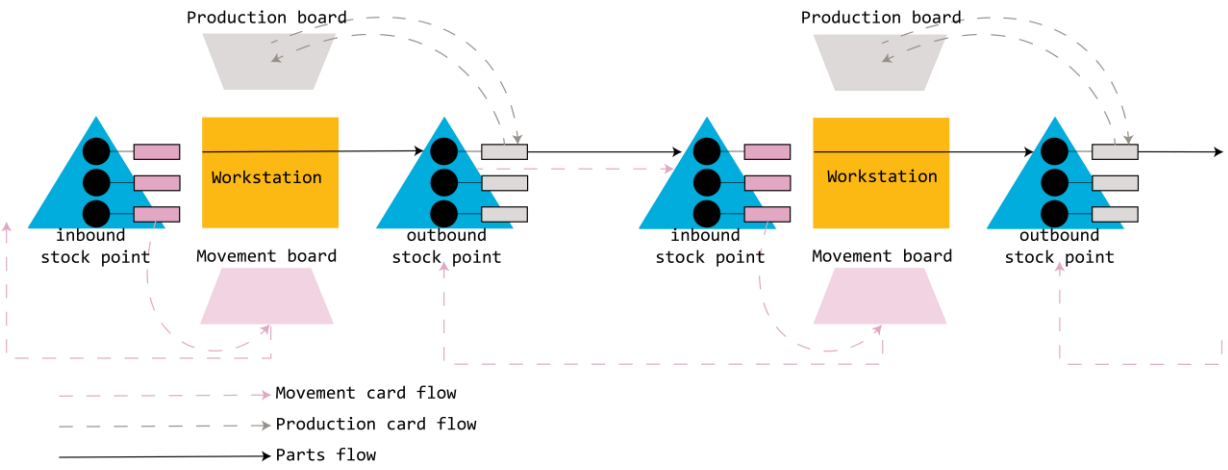


Figure 8- Kanban mechanics

(b) CONWIP

CONWIP was first introduced by Spearman, Woodruff, and Hopp in (1990)and was announced as an alternative to Kanban that enables a wider use of pull across different manufacturing environments. In CONWIP there are only production cards/authorizations. And the pull happens only between the last and first workstation of a production line. Every time an order exits the line one production card is now free, and able to authorize another part to enter the system. We can imagine a production line as a big stock point, once that stock point goes below a given level of inventory (WIP) a production authorization is released so that inventory is replenished.

CONWIP shares practically all the same advantages as Kanban, the biggest improvement over Kanban is the simplicity of implementation. First, because there is only one type of card, that is not particular to any specified part number, we can think of it as a unified production unit. Second, according to the authors' paper, in Kanban, we would need x amount of cards for every part number and we are required to maintain WIP for every part number. The idea is that in CONWIP, organizations would keep a backlog of orders, which can be thought out as a pool of scheduled production, resulting from an MPS, for example. CONWIP would then pull from that plan. Seen in this way CONWIP does not require a specific amount of stock for any part, it just establishes a limit for the sum of WIP for all types of parts in the system. To be fair, we can also adapt Kanban in this manner and instead of customers pulling on a system, we could have a Kanban-type

mechanic pull on a plan, where every workstation would pull on preceding stations until the most upstream one would pull directly from a backlog of orders. The only big difference then is that with this new type of Kanban mechanic, we would have to define the number of unspecific authorizations allowed for every workstation instead of a global amount, as defined for CONWIP. This is the reason why CONWIP can be seen as just a more general version of Kanban.

The abovementioned may cause some confusion, we are explaining pull methods and at the same time we are contemplating using them in environments where production is scheduled, or when there is a backlog of customer orders at the beginning of a line. Nevertheless, this is still coherent with the definition of pull adopted for this thesis – as being any system that explicitly limits the amount of WIP.

From this vantage point is still possible to incorporate pull within systems where production is scheduled, we can have a schedule and allow for a pull mechanism to determine if an order is processed as planned, or if it should wait –this is often the role of PFC (production flow control) policies.

With this explanation, we want to make apparent that there are two main pull alternatives: we can have a fully-fledged pull mechanism where the production system gets pulled by orders — these are the traditional pull systems that require minimal production planning (predominantly used in MTS) — or, the production system pulls the orders from a plan. Though, there is a big difference, if you would look only to the operation of the production systems from the first workstation to the last, the flow of parts in both would be identical. CONWIP is still better than MRP – II because even if CONWIP is pulling from an infeasible plan, WIP will not be allowed to grow past a given limit. In Figure 9, we can observe how the 1 Kanban card system and CONWIP card flow differ from each other. In CONWIP, the pull happens from the last workstation to the first and in Kanban pull happens at every workstation.

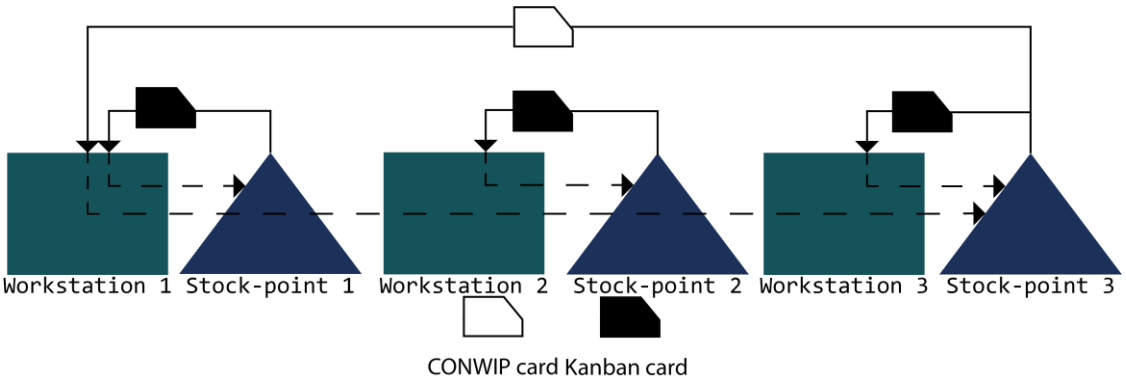


Figure 9- CONWIP vs 1 card Kanban system

(c) POLCA

The forces that led to the development of POLCA (Paired Cell Overlapping Loops of Cards with Authorization) were the same that led to the development of CONWIP – the necessity of making these systems work across a wide variety of production environments. In the case of POLCA, the effort was to create a pull mechanism that works well for production environments with a low volume, high product mix. Kanban is debatably the most popular pull mechanism, but its strengths are very stable, high volume, and small variability production environments. For example, for Kanban to work well each work center should work below its respective takt time, this is no problem when demand is rather stable as we can use average takt-times and the result will be a smooth production flow.

On the other hand, if we have a very unstable product mix and demand, then using average takt-times will be a recipe for disaster unless we were able to readjust our production capacity very frequently to face these underlying changes. The second problem with Kanban and its inadequacy for low volume/high product mix environments is the fact that its vanilla version requires us to keep stock for every part number we produce. This is impossible for production environments with a high product mix, but it is even worse if, on top of that, we have low-volume production. Keeping stock of a product with low and infrequent demand will hurt your inventory efficiency and push your inventory turnover down. This makes it apparent that the origin of Kanban is make-to-stock environments, if making-to-stock is a profitable endeavor for a given organization Kanban will

probably be a suitable pull solution. That is why most make-to-order companies still use MRP-based systems, where production is scheduled and we do not have to keep stock of a product. If that product is needed a production order release will be scheduled.

POLCA stands for Paired Cell Overlapping Loops of Cards with Authorization. In POLCA we think in terms of cells, that feed and are fed by other groups of cells (Figure 10). Cells as proposed by Suri, are a group of multifunctional tasks, but in the context of production flow, we can simply see them as work centers with a given cycle-time. These cells have production authorizations tied to other cells they feed. Hence, if cell A feeds cell B (Figure 10), cell A will have production authorizations of the form A/B, meaning that cell A has limited capacity available for providing products for cell B. In POLCA, for production to start, first we need a planned order release for the current or precedent time-bucket (in case there are delayed order releases), then if an order release is available we will check the routing of that order, and see if there are cards available for that route, in this step, we are only interested in the next cell of that route. If those two requirements are met and materials needed for production are available, then production starts. Using the example of the pair of cells AB, once, the product is finished in Cell A, it will be transported to cell B alongside its card, the card only returns to its original cell once it is finished at cell B. This card flow contrasts with the one of Kanban. In Kanban once a product is consumed cards automatically signal the need for replenishment, in POLCA only when the part is finished at the succeeding cell will a signal be sent to the preceding one. Suri points out that POLCA cards signal available capacity and not inventory depletion as in Kanban. The signals being sent at the time of consumption or at the time of completion do not lead to widely different scenarios, and if cell cycle-times were well balanced then there would not be any practical differences between them, most of the time. When we see a difference is when one cell is bottlenecking another.

Continuing with the example of cells AB, let us consider that there are in total 5 A/B cards and that currently, none are available at A, they are all on their way to cell B, in a consumption signal scenario, as soon as a part starts being processed at B, a card will be available at A for an A/B job, if the next scheduled job at A happens to be of type A/B, then we would have 6 A/B jobs on route, 5 on their way to B and one, still being processed at B. In the consumption signal scenario, there is a higher probability of having 1 more part on route than the number of cards available, especially if one cell is bottlenecking another. With the capacity signal that does not happen, the number of jobs on route will always be at most equal to the number of cards. Another advantage

of the capacity signal is evident when there is product disqualification and rework is needed. In these cases, POLCA cards will hold on to the job until a good product exits the production cell. Therefore, capacity signals lead to lower inventories. The last feature left to explain is the overlapping of cards, this is not a design feature but rather an emergent occurrence of obeying the other POLCA rules. If we have a job that on its route goes through cells ABC, in this sequence, at A it will have an A/B card, once it reaches cell B, it will have to wait for the time bucket where that job is scheduled and for a B/C card to be available, as soon as those conditions are met a B/C card will be attached to the job alongside with an A/B card. This behavior is a direct consequence of using capacity signals and means that a job while being actively worked upon in a cell will always have a pair of POLCA cards, except for the first and the last cells of its production route.

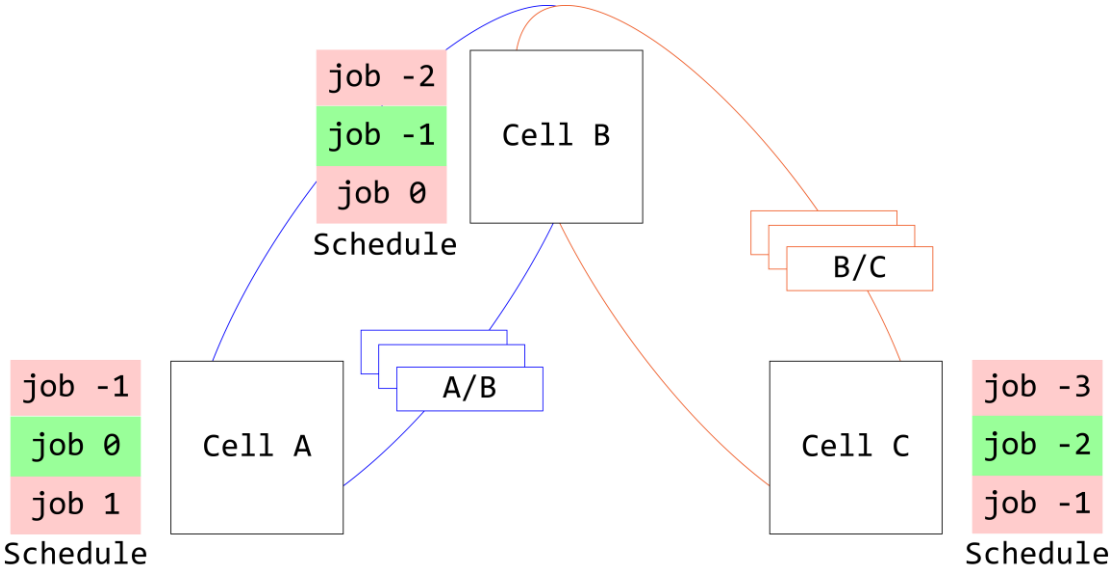


Figure 10- POLCA mechanics

(d) COBACABANA

COBACABA (Land, 2009) (control of balance by card-based navigation) is a card-based system that controls the functions of accepting and releasing orders. It is based on the concept of workload balance control (WLC) and was proposed as a method that is best suited for small to

medium size MTO organizations. To explain COCABANA we need first to introduce WLC. WLC benefits stem from the idea of keeping the workload of stations stable. If the load of a station is stable it is possible to estimate with more accuracy its cycle-time and consequently the entire process cycle-time for a given product, which may inform the due date quoting process. The loads are measured in time units and can be estimated using different methods (J. W. M. Bertrand & Wortmann, 1981; L. C. Hendry & Kingsman, 1989; L. Hendry & Kingsman, 1991; Kingsman et al., 1989), nevertheless, the goal at the decision time is to know what is the potential contribution that an order has on the workloads of the stations on its route. The decision on whereas a release should occur is dependent on workloads not exceeding a predefined limit – norm – on the workstations affected by the potential release. The orders considered for release are the orders whose planned latest release date falls within the current time bucket, and these orders are sequenced by their urgency, meaning that the orders whose latest release date is closest to the current date will be considered for release first. The latest release date is computed by subtracting the quoted due date with the estimated cycle-time of an order. The estimated cycle-time of an order is a function of the current load of all the stations on that order's route. The best methods from which cycle-times are estimated and workload norms computed remain an open research question. COBACABANA is a card-based system that allows the implementation of the WLC rules explained above.

The first step of COBACABANA is the acceptance of orders, here there is a card loop between the production planner and the marketing department, which is responsible for quoting due dates to clients. Before an order is released into production it awaits on a pool, the due date quoted to a client will be the time the order will spend at this pool plus its process cycle-time. Since WLC attempts to keep the load of workstations below a predetermined level, in theory, we will have an upper limit to the process cycle time, therefore we need only an estimation of how much time the order will spend at this pool, to be able to quote an approximately accurate due date. This is the reason why the order acceptance board was created. Every order that comes in, will request available capacity from the workstations on its route. The cards on this board will represent a given amount of work, once an order is received, we need to compute the number of acceptance cards required so that we cover the amount of work that is necessary from every workstation to complete the order. Following the example of Figure 11, if an order received from a client requests workstation 1 and 2 we need to look at the workstation which has more capacity already reserved,

in this case, workstation 2, this workstation will dictate when the order will be released. We can see that by accepting this order we can estimate that it will take at least 3 and 1/3 shifts until that order will be considered for release. Thereby we now have the two pieces of information required to quote a more or less accurate due date. As stated above, cards in this board represent work measured in time units, in the case at hand each card corresponds to 1/3 of a work shift. Once we have taken the amount of acceptance cards required for the order in question, we attach them to an order guiding form. These guiding forms will then be sent to the release planner, who will sort them in the order of ascending latest planned release date, and by that order consider them for release. For each guiding form, we need to compute the percentage of work that the job will require in relation to the established norms of the workstations on this job route. In the release board illustrated by Figure 12, each card specifies a percentage of a workstation's norm, if the job's amount of work – the number of cards it requires – goes beyond the number of cards available, then a release would push the workload for a given workstation over its predetermined norm, therefore the decision would be to not release the job at the current time, and consider the next guiding form in the queue for release, and then repeat the process until we find a release that does not exceed workstation norms. As soon we find a job that respects workstation norms, the planner will remove the release cards demanded for that job and attach them to the order guiding form, whilst concurrently removing its acceptance cards and sending them back to the acceptance board. At the point of release that job is no longer contributing to the acceptance's pool waiting time, but rather to the cycle-time of the workstations on its production route. Once an order finishes processing the release cards are sent back to the release board, signaling a decrease in the workstation's workloads.

Now, we have a full picture of the workings of COBACABANA, and its benefits come from having a fixed limit of release cards, which amounts to an implicit limit for WIP in the production system. In COBACANA the authors also provide a direct link between cycle-times of workstations – which the method tries to keep constant – and the process of due date quoting. This is not a new idea, but it is easier to realize under a system that limits WIP by limiting a workload measured in time units. The same approach can be done by having an explicit limit on WIP, but in an environment of high product mix, is harder to translate at any given moment what amount of processing time a given WIP corresponds to, as 1 unit of WIP could be equivalent to different processing times, depending on the product mix, nonetheless some attempts have been done (see

eg. Spearman, 1991). To finalize the discussion about COBACABANA, it is important that for consistency we explain its inclusion under the category of pull methodologies. As we stated before, we define pull methodology as being any procedure that explicitly limits the amount of WIP. COBACABANA, does limit WIP, but by means of establishing workload norms and not WIP caps. This is an implicit control of WIP, which seems to be in violation of the definition adopted in this thesis for pull methodologies. But, the fact of workload norms being strictly respected makes COBACABANA control WIP more predictably than for example Input/Output control, for that reason COBACABANA deserves to me mentioned in this category, even if not fitting squarely under its definition.

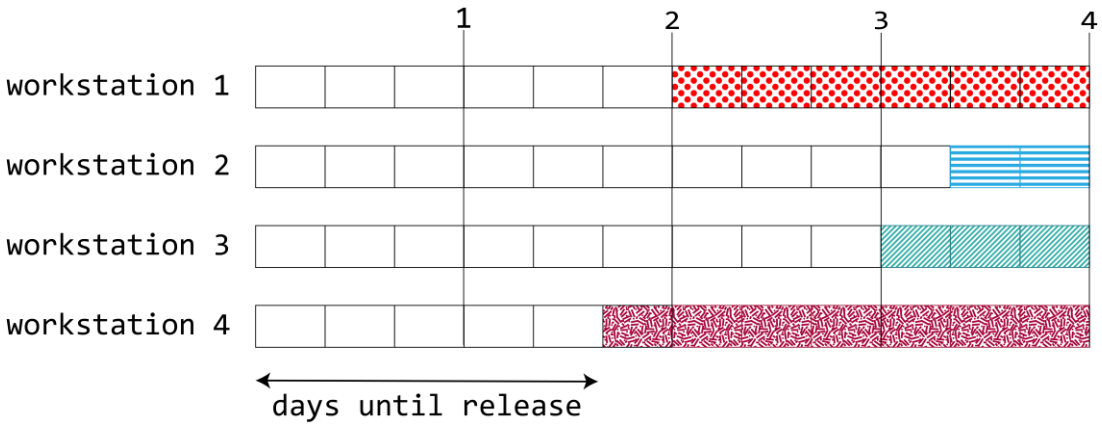


Figure 11- COBACABANA Acceptance board

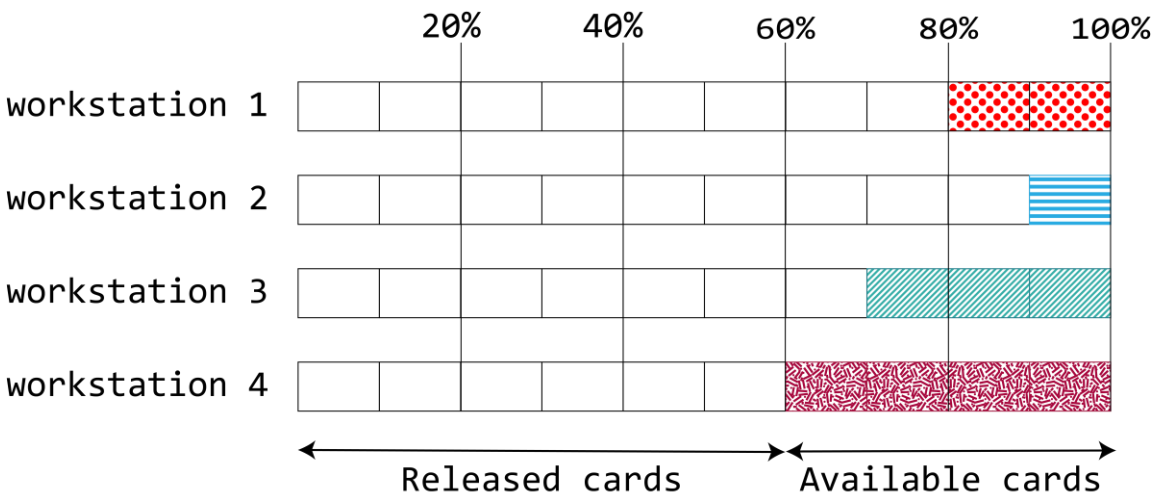


Figure 12- COBACABANA Release board

2.5.7 Challenges of current pull methodologies

All pull methodologies were created with similar goals in mind: controlling the amount of WIP within systems. The most modern methodologies were focused on trying to make the pull approach work for lower volume, high product mix and high variability production systems. Different mechanics were proposed, and each mechanic relies on the determination of parameters, in the case of the methods that explicitly bound the amount of WIP we need to determine that bound, in the methods where WIP is controlled implicitly, which is the specific case of COBACABANA we need to specify workload norms. These parameters should be defined with a goal in mind, and the questions to be answered are what that goal is, why should we opt for it, should the goal change for different production environments. Another type of question that follows from the first is how frequently those parameters should be revised if there is a case for revising them in the first place.

(a) Optimization goals

The flow control problem action is to make decisions on the release of orders into a production system, the decision is a temporal one: when to release. In Kanban, that decision occurs when there is a stock depletion of a specified amount. In CONWIP when WIP is below a predetermined level and there are scheduled orders awaiting release. In POLCA when there is available predefined capacity and, simultaneously, similarly to CONWIP when there are scheduled orders awaiting release.

In Kanban where we serve clients directly from stock, the goal for defining the number of Kanban cards should be maximizing customer service level, which corresponds to maximizing fill rate, and, at the same time, minimizing the amount of WIP and finished goods inventory. In other words, we want to carry the minimum amount of stock necessary for each product and sub-product so that we can fill all orders from inventory.

In CONWIP and POLCA there is a pool of orders that results from a previous schedule mechanism. The overall goal of any production system is to maximize customer service level, if you have two procedures, the first being the scheduling and the second being the release. The first decides the pool of orders that the second will act upon. The one which decides the pool of orders is responsible to maximize the customer service level and the second to minimize the amount of WIP. There is an argument for forcing both methods to maximize customer service level, but the release mechanism is constrained by the results of the scheduling procedure, at maximum the release mechanism can decide on the order of releases and on its timing. Although feasible this approach would have very little impact on the tardiness of orders, as in practice orders arriving at the same time bucket are expected to be finished on that same time bucket, and if for any reason a schedule is deemed to be unfeasible, any efforts to rush or delay orders will be a tradeoff decision between which orders we are willing to risk delivering after their due date, and which orders are we not. This argument shows that the potential for maximizing customer service level at release would be undermined by a poor schedule. Therefore, in this idea of separating scheduling and release, it makes sense to focus on customer service level during scheduling and reducing WIP during release. Having said that, reducing WIP cannot be the end goal for the release mechanism, as if it were then it would be the easiest problem to solve, the best solution would be to set the WIP limit to 1. This would not make any practical sense, because it would harshly reduce the TH rate

of our system and make an unfeasible plan even more unfeasible. A potential solution for this conundrum is to have as a constraint a given level of TH, but then we would have another parameter to define, and probably be restricting a problem to a point where there are no possible solutions, unless we constrain our TH by a very small amount, but even then, we could not expect a balanced trade-off between WIP and TH just by setting an arbitrarily small TH constraint. One possible solution would be to include the TH in our optimization goal, with this approach we would have a pareto front of possible solutions as seen in Figure 7. The matter now is to decide what level of WIP are we willing to sacrifice to achieve a given TH. If we assume that the goal of any production system is to maximize customer service level, this can inform how we balance the trade-off between WIP and TH, without directly enforcing customer service level optimization. This comes around to the idea of critical WIP, we know that at any given time there is an amount of WIP that is necessary to maintain our machines utilized, and that amount will be less when the process is stable and higher when the process is very unstable. We can envision the release function objective as being to keep the smallest amount of WIP necessary so that TH is the maximum it can be. This is a sound goal for a reinforcement learning agent, as we will explain in the next chapters because good performance is not dependent on external factors, as it would be if we chose to optimize for customer service level. If we would insist on using customer service level as an optimization objective, we would need context to evaluate the performance of a release mechanism because bad performance could be a result of either a subpar release mechanism or simply the result of a previous unfeasible production schedule. Therefore, a priori, we would need to have a measure of the customer service levels attainable from a schedule to make a fair evaluation of the customer service levels obtained after the release step. The problem with this idea is that we would need another procedure that pre-determines the feasibility of a schedule so that we have a frame of reference. We will expand on the option of maximizing customer service level as an objective for the release mechanism in section 5.2 p. 73.

On the other hand, maximizing TH with the least amount of WIP possible is implicitly the same as saying that we will execute a production plan as fast (high TH) and as efficiently (low WIP) as it is possible. The latter optimization goal is especially effective in make-to-order environments, where releases exist to directly satisfy demand, in this case maximizing TH will lead to shorter average customer lead-times. In make-to-stock systems with mix products the same parallel cannot be established since demand is not mandated to occur, we could be just producing

products with no demand, trading-off capacity with other products where demand is actually occurring.

Lastly, for COBACABANA, determining workload norms should be done with the same goal as CONWIP and POLCA, we would want to maximize the rate at which orders exit our production system (TH) with the least amount of WIP possible, this means that if our system is stable with very small variability, we could work with smaller workload norms, and if it is unstable, we would probably need higher workload norms to avoid process starvation. The difference between COBACABANA and CONWIP, for example, is that the pool of orders that await release in COBACABANA is informed directly from the acceptance step, which in turn is informed by workload norms. As we have seen in COBACABANA the acceptance step uses information of the expected process cycle-time of an order (which results from a given norm setting) at the time of due date quoting, if done correctly, this integrated procedure would result in feasible production schedules. In CONWIP we could apply the same concepts of using a WIP limit to inform due-date quoting, but as discussed, the process is not as direct, and there is no general accepted approach on how to do the translation between a WIP limit and an expected process cycle-time, especially if we have environments with high product-mixes.

After going over possible optimization goals for WIP parameter setting the next step would be to discuss which methods are available for determining those settings.

(b) WIP Parameter setting

The classical method to define WIP cap limits comes from Monden (1983) and it emerged with the necessity for determining the number of Kanban authorizations for each workstation and type of product (Equation (1)):

$$Number\ Kanbans = [DL(1 + \alpha)] \quad (1)$$

Where D corresponds to the average demand in units of standard containers per unit of time, L corresponds to the cycle-time for those containers for the workstation in question (processing

time + waiting time) and α is just a safety factor. The limitations of this approach are very apparent, first, L is taken as a constant on that formula, but in reality, it will depend on the actual number of Kanban, which makes the method contrived. Secondly, L is also dependent on factors such as available capacity, which if modified, would require Kanban recalculation. Albeit flawed the approach may have been sufficient for the kind of production environment of Toyota at the time, being it an environment of stable demand, product mix, and flexible workforce.

There have been some attempts on using Markov Chains to model hypothetical production systems, whose state representation usually involves the inventory levels of work centers, and state transition probabilities are the result of different random processes, such as machine reliability, demand, processing times, amongst others. The analysis is then derived from the steady state probabilities of system states and their respective costs computed from different performance metrics such as inventory levels (holding costs), process cycle-times, order backlog, etc. Deleersnyder et al. (1989) and Hunglin (1991) are authors of such studies, where different Markov Chains representing systems with different numbers of Kanban authorizations are compared to establish relations between Kanban levels and overall operational costs. These studies have common limitations, the first being the fact of not providing any method to search over the solution space other than iteratively increasing or decreasing the number of Kanban authorizations allowed and looking at the cost function behavior. Secondly, the fact that only considering a discrete and finite set of states to describe production systems may lead to an oversimplistic view of an actual production system. Li and Co (1991) proposed a dynamic programing approach to determine the optimal number of Kanban's for serial and non-serial production environments, the solutions are derived from deterministic models, where demand and other relevant production variables are deterministic. These kinds of approaches, although useful in terms of modeling a problem mathematically, suffer from a lack of generality and only work on premises not usually observable in practice. Moeeni and Chang (1990) propose a heuristic based also on a deterministic production environment. Another deterministic approach this time by using mixed integer linear programing was proposed by Bard and Golany (1991). By now, it is evident that there was a particular emphasis on finding solutions coming from deterministic models, these models even though deterministic, given the nature of pull manufacturing environments at the time, were enough to inform practical implementation, because demand was assumed very stable and smooth, set-up times were very small, and although there was increasing interest on multifunctional production

lines, the most common approach to production was dedicated lines, which tend to be more balanced. Hopp and Roof (1998) proposed a method named statistical throughput control, that was based on the idea of statistical process control, but now directed to the problem of finding the minimum amount of WIP necessary to achieve a given TH. The procedure starts by setting the amount of production authorizations at an arbitrary value, for this value it suffices a very rough approximation of what the optimal number of cards could look like. Then we need to set a target production rate λ, and a maximum cycle-time accepted for the process. After a warmup period, the method mandates that we start computing statistics about the process, namely, average interoutput time μ, which is the inverse of the average throughput rate, and the standard deviation of the interoutput time σ. Then, after each job completion, we need to add one more card if $\mu > \frac{1}{\lambda} + 3\sigma$, or in other words if our current interoutput time is above the target by 3 standard deviations or more, and we need to remove one card if $\mu < \frac{1}{\lambda} - 3\sigma$. When the average cycle-time goes above the predefined maximum an alarm is set so that decision agents either revise the target production rate λ or increase capacity. The cycle- time trigger is very important so that if λ does not happen to be a feasible target, the system does not procure higher levels of WIP in vain, in that case, it would not be an instance of not having enough cards, but simply a problem of not having enough capacity. Since, the method makes only one card adjustments to the overall card count, if the initial card count is very distant from the optimal count it can take a long time until the optimal value is reached. Therefore, the authors provide a method with a dynamic step which relies on the determination of expected cycle-times (Spearman, 1991) for different levels of WIP, and then by using Little's Law it is possible to go through potential card amounts until we find a WIP level that translates itself with a TH that satisfies λ. The dynamic step approach relies on the accuracy of the cycle-time estimation procedure, which as proposed by Spearman (1991). The procedure works well under systems where processing times coefficient of variation is less than 1, which most well-balanced manufacturing systems will probably be an example of. Comparisons between the one-step and the dynamic step methods show that the number of observations needed for achieving optimum card level for a given CONWIP line were 17247 and 374, respectively. Even though there is a considerable improvement in the dynamic step method, this may present a challenge for the adoption of this kind of approach as 374 observations mean 374 produced units. If TH targets are changed frequently, there is possibility of a production run ending before the

optimum level of cards is achieved. This methodology is very promising, because it is not based on deterministic assumptions, dynamically adapts itself to the current condition of the system, and the only perceived drawback, is a not so quick reaction to live changes in the system. More recently, there is been a focus on heuristics and meta-heuristics to tackle this problem, examples of this are Ryan et al. (2000), L. Wang and Prabhu, (2006), Ip et al. (2007) and Ajorlou and Shams (2013). Heuristics, although not guaranteeing optimal solutions, are a good way of tackling intractable problems, with the disadvantage of requiring, in general, more computing time, which may present a challenge when decisions have to be made under a time constraint, as it is the case of flow control decisions. The growth of heuristic-based solutions has been a consequence of the exponential increase of computing power in the last 2 decades, which opened the door for heuristic-based solutions even for time constrained decision making.

Finally, when it comes to workload balancing techniques as used in COBACABANA, very few attempts have been made to determine the optimal workload norms. Zäpfel and Missbauer (1993a) proposed a linear programming model to determine workload norms as a function of the capacity load of a system, the approach relies on the aggregation of production and capacity entities, as a result of the method having as its roots rough cut capacity checking methodologies. Due to this reality, there is a disaggregation procedure based on the assumption that each work center has the same probability of performing operation number 1, . . . , M on each order. This assumption makes the method less general, as in reality order routings are mostly deterministic and not random processes. Thürer et al (2011) use simulation and the OptQuest software to find optimal and suboptimal workload norms for the aggregate (L. Hendry & Kingsman, 1991; Tatsiopoulos, 1983) and the corrected aggregate load approaches with the optimization goal of reducing process cycle-time plus pre-shopfloor waiting time. Their findings show that to use the aggregate load approach it is necessary to dynamically adjust workload norms according to system state, and by using the corrected aggregate load approach, solutions are more robust, and there is less need for dynamic norm-setting. It is still left to demonstrate how dynamic norm setting under the aggregate load approach compares to fixed norm setting under the corrected aggregate load approach.

(c) Revising policies

Statistical throughput control integrates parameter setting with parameter revising. However, the vast majority of the presented methodologies do not provide any process to find when a recalculation of parameters is necessary.

As stated in the introduction there are conflicting opinions when it comes to the development of sophisticated methods for calculating WIP cap levels.

WIP is thought to be insensitive to control: if we set WIP to be high—letting a large number of orders come in—and right after there is a machine breakdown that bottlenecks our system, setting a small WIP cap after that fact, will not result in an immediate reduction in WIP, in fact, it will take a long time until WIP lowers to the point where that WIP cap is respected. Hence, it is understood that overestimating the current capacity of the system makes subsequent WIP cap recalculations as useful as just dictating a request to stop the release of parts until the capacity is restored. Conversely, if we underestimate capacity and set a small WIP cap, we may lose throughput temporarily, but if a WIP cap is changed frequently, WIP will be very sensitive to those changes, as WIP can very rapidly be increased, just by the release of new orders. At this point, we can envision a method that learns to avoid capacity overestimation with frequent WIP cap recalculations. This idea has sprung a new type of research for this problem, where methods that look to the real-time state of a system dynamically control the amount of WIP allowed (Gupta & Al-Turki, 1997; Takahashi & Nakamura, 1998; Tardif & Maaseidvaag, 2001).

Gupta and Al-Turki (1997), propose a method where a base number of Kanbans is computed and then adjusted according to the demand for a planning period with the goal of balancing inventory levels and backlog-associated costs. The shortcoming of the method is to assume known demand for a given period and not considering the possibility of demand changing during the planning period. Tardif and Maaseidvaag (2001) introduce an algorithmic-based solution where a base level of production authorizations is computed alongside an E number of extra authorizations to be used according to demand and predefined thresholds R and C. R, being a release threshold, below which an extra authorization is released, and C capture threshold above which an authorization is taken out of the system. Unfortunately, the authors do not present an efficient algorithm to find parameters E, R, C, they do provide a heuristic to find K*, the optimal base

number of authorizations for given E, R, C parameters. This is also the only approach that considers production environments where random process distributions are dynamic. These algorithmic-based solutions have as a limitation the need of running a procedure every time a decision is made, most of the algorithmic procedures proposed for this problem can be run in polynomial time as a function of the size of the solution space, even though, in most cases, the solution set is infinite (number of possible production authorizations). This happens because for production environments we can set a bound for possible production authorizations, either by making it arbitrarily large, by resorting to steady state analysis or simulation so that we have an idea about the maximum inventory levels that can be reached under different circumstances. Nevertheless, if decisions are to be made very frequently it would be a large improvement if at decision time procedures could be executed in constant time as opposed to polynomial, this is one of the goals of this thesis.

Concerning workload balancing techniques (Land, 2009; Thürer et al., 2015), with the exception of the already mentioned study from Zäpfel and Missbauer (1993a), to the extent of the authors' knowledge there are no more attempts on using dynamic norm-setting methodologies. Despite, some demonstrated success with fixed workload norms under the corrected aggregate load approach, there exists, nevertheless, an opportunity to explore the potential improvements of using dynamic workloads. Fixed workload limits work well if the throughput of workstations, for specific products, keeps close to capacity, but in the scenarios where it does not, these methods will be overestimating the capacity of their systems, potentially overloading workstations beyond norm specifications.

Chapter 3 Research methodology

In this chapter, we outline the research methodology we employed to develop and test the new WIP parameter setting method presented in this thesis. We also acknowledge the limitations of our methodology and the extent to which our conclusions can be generalized. Additionally, we provide the motivation for our choice to utilize reinforcement learning in our approach.

3.1 Applying the scientific method: hypothesis generation and testing in AI-based dynamic WIP parameter setting

Every rigorous research across any field of study tries to follow the general rules of the scientific method. Following the scientific method requires the formulation of a research question, deriving a hypothesis by induction, and finally, resorting to testing, experimentation, and analysis to prove or disprove a hypothesis or theory (Godfrey-Smith, 2009). These steps are not rigid, nor in order nor content, so much so that in practice the process is more often iterative and cyclic. Nevertheless, in retrospect, if a study is done rigorously, we should be able to identify these stages clearly.

In Chapter 1, we defined **RQ1** as : Can artificial intelligence methodologies be used in the context of WIP parameter setting method? Inductively, by the expansive use of AI in a plethora of different fields and by the development of AI-based stated-of-the-art solutions, we can formulate an abundance of hypotheses:

- Current AI technologies can outperform available WIP parameter-setting methods.
- If we narrow the first hypothesis and substitute “Current AI technologies” with subfields of AI, we can generate even more hypotheses:
    - Supervised Learning methods can outperform available WIP parameter-setting methods.
    - Deep Reinforcement Learning methods can outperform available WIP parameter setting methods.

We could take this process of hypothesis generation even further by narrowing the hypothesis to specific AI algorithms. This anecdote shows the iterative and cyclic process of research: as you go deeper into the study, it is normal to stumble across new hypotheses that you decide to take in place of others that you may decide to abandon. For the following chapters, we decide to present the research in an organic fashion that exhibits the research cycle as it occurred in practice. We hope that chapter titles are self-explanatory to the point that the reader can easily infer its content and the hypothesis being tested.

Having said that, hypothesis testing in this thesis is done resorting to computer simulation and statistical analysis, i.e. we implement and adapt different AI methods to train different decision

agents, that will then prescribe actions to these simulated environments. These simulated environments then serve as benchmark tests so that comparisons with other procedures can be made, and conclusions drawn. The obvious alternative would be having the real world as a testing bed. Unfortunately, this is often impractical due to the operational costs that those tests would imply – not many organizations would allow the testing of unproven methods in real-world environments.

3.2 Simulation as a scientific method

Simulation is the process of mimicking the operation of real-world systems over time. With simulation, we generate historic data that can then be used to infer something about the real systems being represented (Banks, 1998). For those reasons, simulation constitutes an indispensable tool for analyzing real-world problems, especially those considered to be intractable under mathematical analysis (J. Bertrand & Fransoo, 2002).

From hereafter when referring to simulation, unless explicitly expressed, we would be implying Discrete Event Simulation (DES). As the name suggests discrete-event simulation models the operation of a system as a (discrete) sequence of events in time. Each event happens at a discrete-time stamp and denotes a change in the system state. Between two consecutive events, no change in the system state is assumed to occur. The contrasting simulation paradigm is continuously tracking system state over time with a set of differential equations that encode the rate of change of each system variable. The choice over DES is straightforward, we intend to simulate production systems, where entities go through a sequence of operations, without directly interacting with each other. We do not care much about the physical properties of these entities, so things like geometry, acceleration, temperature, and so on, for our purposes, could with no loss of generality be abstracted away. The crucial modeling details of production systems are in-tune with the DES modeling framework, where what matters the most are arrival and departure timetables, which describe how entities go through the system, occupying resources for a given time delay. Agent-Based Simulation (ABS) could in theory be a choice, but it would be excessive to model the problem in this way, considering that entities in our system do not require the modeling of autonomous behavior and do not freely interact with other entities.

Several authors have identified several stages for the rigorous development of simulation models (Law, 2014; Pegden et al., 1995):

- Model Conceptualization: the real-world systems are divided into sequences of mathematical and logical relationships to the point where all the desired logic behind the real system is encapsulated. In our case, this conceptualization will encompass the logic behind some well-known manufacturing systems, like simple flowlines and multi-route production systems.
- Model Translation: the conceptual model from the previous step is coded into a given programming language. For our simulation, we will use Python allied with a simulation python package called Simpy (Team Simpy, 2023) .
- Verification/Validation: there are two main steps at this stage: figuring out if the conceptual logic was correctly coded into a programming language and if it is an accurate representation of the system we intend to imitate. The usual way of conducting the second step is by comparing the output of the simulation with the output of a real system. However, as in the case of this study sometimes there is no specific real system to be replicated. So, for validating we will resort to observations of our model's output to check against the predicted output, gathered from existing theory. For example, resorting to queuing theory we can predict relationships between cycle times, arrival rates, and WIP levels. Putting the system into edge conditions, where the system behavior is highly predictable is another tool to assess model accurateness, an example of this practice is loading the system with an increasing number of jobs, starting with just one job, and iteratively adding more and comparing system runs. If systems are too complex, we can divide the tests between blocks of logic where systems behavior is still predictable and test those modules separately.
- Experimental Design: for each simulation scenario decisions such as running length, replications, and initialization parameters should be made.
- Implementation: Implementation of the experiments designed in the previous step.
- Analysis: the models' output is analyzed, performance measures are computed for simulated scenarios, and confidence levels can be calculated.

3.2.1 Drawing conclusions from simulated environments to the real world

As stated, we are not replicating a real-world production system, we are using a simulated environment to train and compare different decision Agents. Therefore, any conclusions drawn from this research are axiomatic in the sense that evidence is shown for the particular simulated system and its random processes. If we test our decision Agents for a particular flowline as will be the case of section 5.1, and we show that our decision agent improves performance for that particular flowline configuration, we cannot claim with certainty that it will improve performance for every flowline. Any attempt at proving or disproving a hypothesis resorting to simulation would be in the form of proof by cases, showing that the results are true for every possible case. Resorting to the example of the flowline, in theory, we could have flowlines with hundreds, thousands, or even millions of workstations, making it infeasible to have complete proof by simulating all these different flowlines.

Therefore, we have to accept that the scientific contribution from this kind of simulation study comes from showing evidence that supports a theory or hypothesis, and not by absolutely proving it or disproving it. In this particular case, we will be testing our AI methods in the task of WIP parameter setting, and if the results suggest improvements over current methods, we will be providing evidence to support the hypothesis that in fact "Current AI technologies can outperform available WIP parameter setting methods". Nevertheless, it is out of our reach to actually provide a complete proof using simulation. If we accept Popper's theory (1963) on how science should be done, namely the requirement for any scientific hypothesis to be falsifiable, i.e. empirical science must yield a testable hypothesis, then this proposed hypothesis follows that rule. It is very much possible to test the proposed methods in different production system configurations until a refutation of the hypothesis is found. For the moment we tentatively try to provide evidence to support the proposed theory.

3.2.2 Assumptions in our simulation studies

All the simulated production systems in this study will have the following assumptions: they will be operated under make-to-order environments, a CONWIP mechanism will oversee

production releases, and there will be unlimited raw materials. Any other axiom not referenced here will be referenced in the respective chapters concerning said simulations. When no assumptions are referenced, and simulation is used these will apply.

From the provided assumptions only the last does not reflect a real-world condition. Nonetheless, in the context of WIP parameter setting under make-to-order systems, there is no practical implication, that would make a decision Agent trained under this assumption to not perform well when this assumption is relaxed. Relaxing this axiom would make workstation starvation due to the lack of raw materials possible. We should understand raw materials here as the components that we integrate into a product, and that are sourced from entities outside the production system under study (usually suppliers). This is a type of WIP for which dynamic parameter setting is typically useless. Under make-to-order systems, there should not be much variability in the delivery schedule of raw materials. If we agree on a delivery schedule with our supplier based on a production plan, and if materials are not available when they were agreed upon, increasing or decreasing WIP of these materials is not the first line of response. Rather accountability measures, where costs are transferred to suppliers are the first line of defense. In make-to-stock systems, when Kanban is used to trigger the resupply of raw materials, then the story changes. Defining these levels correctly will decrease or increase the risk of starvation. Usually, in production systems that use Kanban, material resupply relies heavier on consumption signals than in production planning, therefore properly defining WIP levels for raw materials is very important. In our simulated production systems, as we will be working under make-to-order CONWIP systems, including logic about the resupply of raw materials is superfluous as there is nothing a WIP parameter setting method can do to avoid starvation of this type. The implication of the transferring of knowledge from the simulated environments to the real world is that even for a perfect WIP parameter setting, where starvation never occurs due to the depletion of workstation buffers, starvation on an equivalent real system can occur due to a shortage of raw materials. What would be observed is then a loss in TH between simulated and real-world environments.

3.2.3 Random number generation

Usually, in simulation studies, modelers are safe in taking random number generation for granted, i.e., not much thought is spent on the methods used for random number generation. In our case, some consideration is needed. Since we are using relatively deep neural networks as the engine for the dynamic WIP parameter setting, and a reinforcement learning agent is learning from interacting with a simulated environment. There is the potential risk, that the agent, instead of learning how to adjust WIP levels according to current system state is learning the random number generation procedures. In the latter case, a good performance would not come from learning the task at hand, but by predicting the future output from the random number generation process.

All randomness required by a model comes from a random number generator, whose output is assumed to be a sequence of independent and identically distributed (IID) random variables. The validity of this approach relies entirely on this IID assumption, nevertheless, this assumption is false, since random number generators (RNG) are deterministic programs trying to produce deterministic sequences that appear random to a user that is unbeknownst to the whole procedure. Hence, these procedures are actually called pseudo-random number generators (PRNG).

A PRNG procedure is expressed as a system: $G = (S, s_0, T, U, G)$; Where S is a finite set of states, $s_0 \in S$ is the initial state (seed), the mapping $T: S \rightarrow S$ is the transition function, U is the finite set output values (symbols), and $U: S \rightarrow U$ is the output function (L'ecuyer, 1998). The state of the generator starts at s_0 evolving according to $s_n = T(s_{n-1})$, then the random number generation is done by $u_n = G(s_n)$, u_n is the random number generated by the procedure. Since S is finite, it is apparent that the sequence of states s_n is periodic, hence pseudorandom sequences can always be determined by observing outputs for long enough. From the above, it follows that PRNGs should have a long period ρ. If a simulation takes N random numbers from a sequence of length ρ, there is evidence to suggest the need for $\rho \gg N$ (L'Ecuyer, 1994; L'ecuyer, 1998; Maclaren, 1992).

The other major desired characteristic that PRNGs should possess is related to the uniformity of sequences generated by the procedure, i.e., if we are taking sequences of random numbers with length ρ, from the set U of possible outputs, then each of the $|U|^\rho$ possible sequences should have the same probability of occurring. There are different methodologies to test for the independence

and uniformity of sequences of random numbers. These methods range from structural tests that study the mathematical structure underlying the successive values produced by a generator, and statistical methods that view the random generation process as a black box, analyzing its output by hypothesis testing. To understand these testing methods, a deep understanding of heavy mathematics like lattice geometry is required, since we do not possess this knowledge, the intent of this general explanation is so that we can move forward and design experiments where the risk of confounding results – coming from artificial neural networks reverse engineering our random number generation procedures – is out of the picture.

We chose the Mersenne Twister pseudorandom number generator (MT19973) (Matsumoto & Nishimura, 1998) for the following reasons:

- Its large period length $2^{19937} - 1$, which is many orders of magnitude larger than the length of sequences generated in our studies.
- It passes the Diehard battery of tests (Marsaglia & Tsang, 2002) and passes most of the tests from TestU01 (L'ecuyer & Simard, 2007), which measures sequence uniformity and independence.

Some research is trying to understand if artificial neural networks can accurately predict the next output of a pseudo-random number generator. Usually, in these studies, researchers use datasets, where a sequence of random numbers generated by a given PRNG procedure is divided into sub-sequences of say length *n*, and then *n – 1* elements are used as input for a neural network, and the n^{th} element is used as the ground truth for training. Some studies have found success in demonstrating that simple feed-forward networks, trained by supervision, can indeed find patterns within sequences generated by different PRNGs, including the MT19973 procedure (Fan & Wang, 2018; Feng & Hao, 2020). Nevertheless, in these studies next output prediction accuracy, though statistically significant never goes much above 50%. In our study, if our neural networks find any pattern for random number generation it will be learned without supervision and indirectly. Furthermore, we will never directly provide the actual bits of information generated by the MT19973-based random processes to our networks as input. Our networks will see production system state information, that is derived from MT19973-based random processes, but never the direct outputs from this PRNG procedure.

As explained, our experimental set-up is such that it is very unlikely that our neural networks can indeed obtain good performance by suppressing the randomness from PRNG procedures. Nonetheless, to guarantee no performance bias due to the explained phenomena, every decision agent trained under MT19973-based random processes will go through performance tests, where states are derived from a true random number generator (TRNG).

TRNGs are procedures where numbers are generated from physical processes, rather than by means of an algorithm. Commonly, these procedures are based on microscopic phenomena that generate random noise signals, like thermal noise, or other sources of entropy.

During the training of our decision agents, generated sequences should be reproducible, for example: in training, we provide a reward to a decision agent based on how better it performs against another agent, for that reward to be meaningful both agents have to be experiencing the exact same system dynamics, hence the necessity for pseudo-random number generator. Notwithstanding, after training is done, we can test the decision agent in a system where randomness comes from a TRNG process. To generate our approximately true randomness we will use CryptGenRandom, which is a cryptographically secure number generator function that is included in Microsoft CryptoAPI. CryptGenRandom is not considered a pure True Random Generator because random numbers are still generated from an algorithm, but seeds are generated from random noise signals collected from the Windows Operation System (*CryptGenRandom Function (Wincrypt.h) - Win32 Apps | Microsoft Docs*, 2018). In section 6.4.6 we will show that a decision agent trained under random processes generated with the MT19973 algorithm, retains its performance when tested in an environment where random process generation was done through the CryptGenRandom algorithm. The idea is that if the algorithm indeed was able to reverse engineer the MT19973 algorithm during training, when presented with production systems where random processes come from a different procedure, such as CryptGenRandom, will see a considerable performance drop. If that does not happen, we have a guarantee that the agent is not relying on predicting the next random number to achieve good performance. So, after training, for each simulated production system, we will run simulations with the CryptGenRandom procedure and check for statistically significant performance changes.

3.2.4 A precedent for generalizing from simulation

In Reinforcement Learning an agent learns either by interacting directly with an environment or with a model of that environment. From this point of view, reinforcement learning methods can be either model-based or model-free. In our study, we are using a discrete event simulation as the model. This simulation is considered to be a sample model because from a starting state, and a given action, a new state will be sampled according to state transition probabilities that are embedded into the logic of the model and its random processes. Model-Based approaches are very tempting; interacting directly with an environment is expensive, and data collection takes longer and is harder – you cannot run real-world environments at 10x "normal" speed.

There are obvious problems with this approach: can these, or any type of model be a 100% accurate representation of real-world systems? The answer is not simple; this is above all a philosophical question, that nobody was yet able to provide an unequivocal answer for. In this thesis, we will opt for a practical answer and accept that a model is accurate enough if from it we can derive a human satisfactory solution to a given real-world problem. By satisfactory we mean a solution that has at least comparable performance with other methodologies if they exist. If we provide a solution based on simulation how well does it transfer to the real world? In artificial intelligence, this is still a fundamental question to be resolved. In computer science and more specifically in robotics there exists the concept of simulation-reality gap (Zhao et al., 2020). In this field, there is difficulty in transferring simulated experience to the real world. What is at stake is a discrepancy between reality and simulation that prevents simulated robotic experience from directly enabling effective real-world performance. Robotics deals with very complex control problems, a popular one being object grasping: a robotic arm performs the task of grasping and manipulating objects with a given goal. Simulations of this kind require very accurate approximations of the physical dynamics of the real world. Specifically, how objects interact with each other and with the physical environment. In robotics, models trained with purely synthetic data fail to generalize to the real world. The question that arises is: how can a robot utilize simulation to enable it to perform useful tasks in the real world?

Solving this problem is crucial because training robots with real data is too costly: robots are very expensive, during training they can damage themselves, training can result in catastrophic failure, learning takes many episodes and hence requires a lot of time (Muratore et al., 2021).

Something similar could be said about real-world production systems, it would be infeasible to train an Agent in real-world systems. If the system is very large and involves great CAPEX investment, stopping the system just for an hour could cost thousands or even millions of dollars, not to mention any catastrophic failures the learning process could enact. Nevertheless, to the author's knowledge, there is little to no research in production management addressing the simulation-reality gap, which begs the following question: are production systems simulating tools accurate enough for direct learning transfer, from simulation to reality? From the literature, we can find plenty of works (Bae et al., 2020; Lang et al., 2008; E. J. Williams & Ülgen, 2013) with successful direct knowledge transfer from sim-to-reality. A substantial difference between the simulation of manufacturing systems and physics simulators in robotics is the validation phase. Often, when simulating existing production systems, validation using historical input data is the preferred choice, this allows for the statistical comparison between results coming from the simulation and those from the real-world system.

In robotics validation of this type maybe not be sufficient. Let us take as an example the grasping of a random object from point A to point B, and we want to use historical input data as a validation method for simulation fidelity. In this scenario, a human controlling a real robot arm would be controlling the simulated version of that same arm, without knowing. For a simulation to be validated the same input given to the real robot arm should have produced an identical trajectory with no statistically significant discrepancies both in the real and simulated environments. Instead of controlling simulated and real-world versions of the robot arm simultaneously, we could simply store the system variables related to the real-world environment and robot arm actuators and feed that data into the simulation at a later time. Whilst this type of validation may have been sufficient to guarantee direct transfer learning for simulations of production systems; in robotics, it appears to not have been sufficient. An explanation for this is related to the duality between control and perception in robotics. Tests like the abovementioned, check for simulation fidelity concerning variables associated with the control problem. What about the simulation fidelity concerning the perception problem. In other words, what if instead of controlling the real robot arm and feeding that information into the simulation, the experiment was done the other way around, i.e., we perceive the simulated world, actuate the simulated robot arm, and send the control inputs to the real-world robot arm. This is relevant because a human perceiving a simulated world can produce different control policies than if it was perceiving the

actual real world. An example of this, are problems where control is derived from the visual perception of the environment; synthetic images are still far away from being indistinguishable from real-world images (Beche & Nedevschi, 2020; Huang et al., 2018; Isola et al., 2017; T. C. Wang et al., 2018), therefor generated policies can be biased towards this simulated world based on perception discrepancies.

Looking into the techniques used in robotics that enable knowledge transfer from simulation to reality may offer some insight into the reason why knowledge gained during production system simulations can successfully be transferred to the real world. In robotics, there are two main techniques for sim-to-real knowledge transfer: domain randomization and domain adaptation.

In domain randomization instead of modeling all the parameters of the real world as they are, we highly randomize the simulations to cover the distribution of the real-world parameters. The randomization happens on two fronts: visual randomization and dynamics randomization. In robotic vision, for the tasks of object localization (Sadeghi & Levine, 2017; Tobin et al., 2017), object detection (Tremblay et al., 2018), pose estimation(Sundermeyer et al., 2018), and semantic segmentation (Yue et al., 2019), the training data from the simulator always has randomized textures, colors, lighting angles, and camera positions. The goal is to provide enough simulated variability over the visual parameters at training time, such that at test time the model is able to generalize to unseen real-world data, without the need for very accurate simulators. Dynamics randomization as the name indicates is the same idea applied to physics and physical parameters of the simulation, namely randomizing object dimensions, masses, centers of gravity, friction coefficients, actuator gains, and other relevant parameters (Peng et al., 2018; Van Baar et al., 2019).

Interestingly enough simulation environments for production systems are often highly randomized. In production system simulation, input data is often in the form of arrival and departure tables that describe the movement of parts across the system. These tables are often modeled using random processes, where arrivals and departures are sampled from a given distribution. An agent trained in such environments is prone to learn robust policies, that can generalize to very randomized environments. We can increase the chances of the Agent learning robust policies by using random processes with high dispersion, or dynamically changing the parameters and distributions of those random processes if needed. This evidence can help explain

why production system simulation, by default, can more easily enable direct learning transfer from simulation to reality.

In domain adaptation, we use data from a source domain to train an agent to perform well in a target domain, the catch is the fact that source and target domains live in different feature spaces. The most recent domain adaptation techniques are adversarially based. In James et al. (2019) an adversarial network (Goodfellow et al., 2014) learns to translate real visual perception features into simulated visual features. The goal is that an agent trained to perform well in a task by perceiving vision-based signals coming from a simulation, can, by the means of translating real-world signals into the simulation domain, perform well in the real-world task. The adaption comes from this translation procedure that takes features from the real-world domain into the simulated domain. It is harder to see a direct application of domain adaptation within the realm of production system simulation, nevertheless for completeness, and as a reference for future work, we decided to include it.

3.3 A motivation for reinforcement learning

The field of artificial Intelligence (AI) is concerned with the study of how complex cognitive functions associated with the human brain can be mimicked by machines. The ultimate goal of the field is to be able to achieve artificial general intelligence (AGI). Through the pursuit of this ultimate goal, narrow applications of AI have been demonstrated, from natural language processing (Brown et al., 2020) to playing highly complex strategic games (Berner et al., 2019; Vinyals et al., 2019) to self-driving cars. These last applications, not aiming to solve artificial general intelligence represent a subdivision of AI into another subfield called machine learning. The latter has as its goal to address solvable problems of a practical nature, as exemplified above.

From Machine learning, we can identify 3 main classes of approaches: supervised learning, unsupervised learning, and reinforcement learning.

Reinforcement learning is different from supervised learning used for example in computer vision and natural language processing applications. There are a lot of different supervised learning algorithms, some of them currently amid rapid development like artificial neural networks (ANNs), particularly deep neural networks, which gave rise to a new subfield of machine learning

called deep learning. Supervised learning and reinforcement learning are fundamentally different because as the name indicates, in supervised learning, there is a need for an external supervisor to tell what action is best in any given situation. The algorithm learns how to generalize by looking at examples that were classified by this external agent. The performance of these algorithms is constrained as we might expect by the performance of this external agent. The external agent might be a human, an expert in a given domain, or a group of experts. Nevertheless, to use supervised learning we need a large database for our problem and as the reader can imagine building databases for complex problems may present itself unfeasible or even impossible. Imagine creating a database where we recorded all management decisions (actions) of a plant manager during as many production system scenarios as we could. Here, production system scenarios could be any kind of random process, demand, machine availability, or as many parameters as we choose to consider for the problem. It would be a daunting task to generate this kind of database, only to find that the performance of these human experts is not very good, and therefore training our supervised algorithms with that database will lead to subpar performance.

In fields of research where humans also lack an extended understanding of how to perform a task, there is no external agent qualified to supervise the learning of these algorithms. It is for this and other reasons pointed out, for example by Silver et al. (2017) that there is interest in the use of learning mechanisms that do not require human supervision. Even though reinforcement learning can be used for problems where human knowledge is scarce, humans are the only ones that can define what a good action looks like. In other words, we might not know what the best action to take is, but once an action is executed, we can understand if its impact on the environment goes according to our goal so that the agent is rewarded or punished. To put it in the context of production management if, for example, we want to maximize TH, because we can observe it, we know when to punish or reward a given action.

In unsupervised learning, no supervisor is needed, these algorithms take in data with no annotations, i.e., not categorized, classified, or with any other form of human input. The goal of these algorithms is to find patterns in the input data. A common application of this technique is anomaly detection (Qu et al., 2018). Unfortunately, these algorithms cannot be used to generate policies that map from states to actions, therefore they are not useful for our problem. They can be used to generate descriptive knowledge about a system, but to generate prescriptive Knowledge, with no human supervision we have to resort to reinforcement learning.

Within RL there are two different approaches: the most popular paradigm is based on the Markov decision process (MDP) framework and the concept of value functions (Mnih et al., 2016; Ng et al., 2006; Silver et al., 2017) and evolution strategies (ES) (Rechenberg & Eigen, 1973).

The first class of methods uses backpropagation to directly update policy parameters, and the latter uses evolutionary algorithms or black-box optimization tools to directly search over the space of policy parameters.

Evolution Strategies (ES) is a category of black-box optimization inspired by natural evolution. During N iterations ("generations"), a population of parameter vectors ("chromosomes") is perturbed ("mutated") and their objective function value ("fitness") is evaluated. The highest-scoring parameter vectors are then recombined with the next-generation population. The cycle repeats itself until the optimization or fitness level is satisfactory.

In the setting of reinforcement learning problems ES is used to optimize the parameters of ANNs and is commonly referred to as neuro-evolution (Risi & Togelius, 2015). If we let F denote the objective function and θ the parameter vector on which F acts upon. This kind of algorithm represents the population with a distribution over parameters $p_{\varphi}(\theta)$ – on its turn parameterized by φ – and proceed to maximize the average objective value $E_{\theta \sim p_{\varphi}} F(\theta)$ over the population by searching for φ with stochastic gradient ascent.

These methods are highly parallelizable, run faster, and require less memory, as there is no need to do error backpropagation directly through the policy network (Salimans et al., 2017). Furthermore, they sidestep the problem of vanishing and exploding gradients when optimizing deep neural network parameters. Nevertheless, to fully take advantage of this technique it only makes sense to use them when a large set of machines and computer cores is available, otherwise, their performance is on par with state-of-the-art RL algorithms. The fact that these ES algorithms are easily and highly parallelizable, in opposition to most RL algorithms, is what makes them excel (Salimans et al., 2017). In fact, the time to solve a problem has been shown to decrease linearly with the number of CPU cores, when using ES techniques. It should be pointed out that in terms of sample efficiency, ES techniques are not as good as most modern RL algorithms, although the difference is not preposterous.

In our study, the hardware available is comprised of 8 CPU cores and TESLA K80 GPU, not enough computing power to take advantage of the ES algorithms parallelization power. For This reason, reinforcement learning from the panoply of techniques referenced is the tool of choice for the problem of WIP parameter setting.

Chapter 4 Reinforcement learning conceptual framework

This chapter presents the basic structure of a reinforcement learning (RL) problem. Formulating a problem in a way that it can be solved using RL is not always straightforward, as is the case with WIP parameter setting. Therefore, this chapter provides context for the modeling decisions discussed in subsequent chapters.

"Reinforcement learning is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal" (Sutton & Barto, 2018). This is a very simple definition that encompasses in a very generic form the idea of reinforcement learning.

Reinforcement learning presupposes the interaction of an agent with an environment, the agent will take actions that will affect the environment and yield a reward. The problem gets complex because immediate actions can affect not only the immediate rewards, but also all the rewards that come afterwards. The agent must then learn the actions that lead to higher rewards.

4.1 Agent-environment interface

To more formally, but still generally define a reinforcement learning problem we will look more carefully at the interaction of a decision agent and a particular environment by inspecting Figure 13. The agent will interact with the environment at a sequence of discrete time steps, $t = 1, 2, 3, ...$ At each one of these time steps the agent will receive a representation of the environment state, $s_t \in \mathbb{S}$, where $\mathbb{S}$ is the set of all possible states. Knowing s_t the agent will take an action $a_t \in \mathbb{A}(s_t)$, where $\mathbb{A}(s_t)$ is the set of possible actions available at the state s_t. As a result of the previously taken action in the next time step, $t + 1$, the agent receives a numerical reward, $r_{t+1} \in \mathbb{R}$, and the environment is now at a new state, s_{t+1}. The just described decision step process then repeats itself in a cycle. During this cycle, the agent is continually mapping states to actions according to what action has the highest probability of bringing the highest rewards. This map is called the agent's policy and is denoted as π_t. Hence, $\pi_t(s_t)$ is a model that will give us what action a_t will lead to the highest expected reward when at state s_t. This is a very general framework: in here we are considering that this policy is dependent on t. What we mean is that by depending on *t*, being in a given state at time *t* may lead the policy to select a different action than being in that same state at time *t+1*. This condition may be relaxed, and this mapping can be even more simple, where for given a state the best action will always be the same regardless of the time step we are in. These policies can also be stochastic, where instead of always choosing the highest reward action, we may choose other actions according to a given probabilistic procedure.

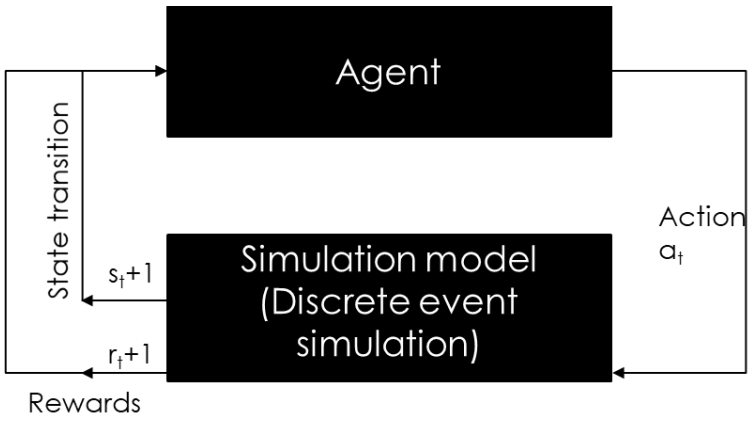


Figure 13- Interaction between an environment and a reinforcement learning agent

4.2 Modeling and solving reinforcement learning problems

Reinforcement learning problems are modeled as Markov Decision Processes (MDPs), hence we assume that action and state spaces follow the Markov property. This is the case when to know the state of the process at t+1, s_{t+1}, we only need the information about the state and the action taken at time t, $s_t \; and \; a_t$. In other words, s_{t+1} depends only on the previous state and action, we do not need information of all the previous states the environment was in until it got to s_t . This assumes that the state representation compactly retains all the information necessary to predict the future state. A good example taken from Sutton & Barto (2018), is the example of the cannonball. The current position and velocity vector of a cannon ball is all that it is needed to know the future of its trajectory.

A Markov decision process is a tuple made out of 5 components $(\mathbb{S},\ \mathbb{A}, \{P_{sa}\},\ \gamma,\ \mathbb{R})$ where:

- $\mathbb{S}$ is a set of states. For example, in a production system, a state could be a vector with different metrics of the system, like WIP, average utilization, average processing time, different measures of variability, etc. The set of states will then be made out of all the possible combinations of values for those metrics.
- $\mathbb{A}$ is a set of actions.

- $P_{sa}(s')$ are the state transition probabilities. This gives us the probability of transitioning to state s' being at state s and performing action a. This concept is applied only if we intend to use model-based algorithms to solve the reinforcement learning problem. What we mean by model-based is that we either have a model of the environment or we can extract one by finding these probabilities via running the environment many times and recording state transitions resulting from applying different actions. Doing this will generate a map of probabilities for state transitions. This map can be used as a model of the environment; thus algorithms can be executed without the need to interact directly with the environment. On the other side of the spectrum, we also have model-free algorithms, therefor part of the algorithm's job is performing an action and waiting for the environment to see what the consequence of that action is going to be. In the latter case, decision agents learn a model of the environment and a policy simultaneously.
- $\gamma \in [0,1]$ is called a discount factor. Represents relative importance for the value of future rewards.

We have said that the goal of reinforcement learning is to maximize the rewards the agent receives in the long term. This idea can be defined as the sum of rewards of each time step (Equation (2)):

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_{t+T} \tag{2}$$

In this simpler form, we are considering a finite number of time steps, T, but that restriction can be relaxed to an infinite sum. If that were to occur in the way in which we defined R_t (Equation (2)) then that sum would be unbounded. The discount rate solves this problem by making this sum convergent:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{3}$$

With this new representation (Equation (3)) the value of a reward in the future is worth less than one in the present. The value we decide to choose for the discount factor will determine how differently we value future rewards. If we choose the value to be 1 then we are saying that all rewards have the same importance, but then you will run into a problem: if you are modeling an infinite MDP then the sum will be infinite, and your algorithm will never converge. If we model the discount rate as 0, that is the same as saying that the only thing that matters is the immediate rewards you get, all rewards afterward are of no importance.

- $\mathbb{R}: \mathbb{S} \times \mathbb{A} \rightarrow \mathcal{R}$ is the reward function, the reward function has to encapsulate the intuition of what a good decision is. In this case, we defined the reward as being a function of the state and action, but we can also define it to be only a function of the state, depending on the problem. If the reward could be defined only as a function of the state then the equation (3) can be written as:

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \gamma^3 R(s_3) + \cdots \quad (4)$$

We have described all the components of an MDP, but we have yet to describe the problem in terms of its optimization goal. We are going to describe it with the use of value functions and Bellman's equation. The value function is a way of representing the expected return of starting on a given state and following a given policy. This is the standard procedure, as most reinforcement algorithms are based on estimating value functions (Sutton & Barto, 2018).

The goal of any reinforcement learning algorithm is to estimate a function that maps states to actions, $\pi: \mathbb{S} \rightarrow \mathbb{A}$. When following any given policy, if currently on state s to find the next action we use the policy function, $a_t = \pi(s_t)$. Hence, the value function for a given policy when we start from state s and follow policy π can be written as:

$$V^{\pi}(s_0) = E[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots] \quad (5)$$

Equation (5) is the expected sum of the discounted rewards, upon starting on state s_0 and following policy π. We talk in terms of expected rewards because most environments are non-deterministic, meaning that when in state s performing action a may lead to range of possible states according to the state transition probabilities, P_{sa}(s').

Given a fixed policy, equation (5) is equivalent to the Bellman equation (6) :

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\,\pi(s)}(s')\, V^{\pi}(s') \quad (6)$$

Looking closely at equations (5) and (6) we can see that they are identical: the first part of the equation (6) is just the immediate reward of starting on state s and the second term is the sum of the discounted expected rewards. In the second term we have summation of the probabilities of starting on *s* and applying the policy $\pi(s)$ and reaching state s', over all the possible states ($s' \in \mathbb{S}$) that could be reached from s and applying $\pi(s)$. This part is represented as $P_{s\,\pi(s)}(s')$. To have an expected value, besides having the probabilities of occurrence we need to multiply them by the expected value of starting from the state we have just reached s', and then following policy π upon reaching s' onwards, this is the term $V^{\pi}(s')$. We now understand the recursive nature of Bellman's equations we are defining $V^{\pi}(s)$ as a function of all the value functions for any other possible state.

We can now write a value function, $V^{\pi}(s)$ for every state. Hence, we will have a set of $|\mathbb{S}|$ linear functions with $|\mathbb{S}|$ unknowns. We will have $|\mathbb{S}|$ $V^{\pi}(s)$'s , one for each state. This linear system of equations can be solved resorting to different algorithms.

We only defined value functions in terms of a given policy, but what we want to find is what that policy is. In that direction, we can define the value functions for the optimal policy as:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{7}$$

In equation (7) we are essentially saying that there is a policy π that maximizes the value function for all the states. Using again the Bellman's equations we can rewrite equation (7) as:

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s') \tag{8}$$

In equation (8) the first term is the immediate reward as before. The second term is very similar to the second term of equation (6) with the difference that instead of choosing an action according to a policy, now we will try every single possible action available, compute the expected value of future returns for that action and then choose the action that leads to a higher return. In this way, we define the optimal value function for every possible state and try to solve the system of linear equations as explained previously.

Once we have solved the linear system of equations we will have the value function for every state, therefore we can determine the optimal policy as follows:

$$\pi^*(s) = arg \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s') \tag{9}$$

Having all the $V^*(s)'$, with equation (9), we can determine which action will lead to the highest return. At this point, we have determined the best policy.

The essence of solving reinforcement learning problems is determining the value functions just described. There is not a single way of defining value functions, different algorithms will work with different function definitions. There are various categories of algorithms to solve for value functions and policies, we will not explain them as it is out of the scope of this thesis.

4.3 Reinforcement learning and industrial management

The field of industrial management has demonstrated some interest in using supervised, unsupervised, and reinforcement learning methods.

Focusing on reinforcement learning, Cao et al. (2003), used a reinforcement learning algorithm together with a Monte Carlo simulation to solve a capacitated scheduling problem for a hybrid manufacturing process. They solve the problem in two phases: in phase one they take a plan generated from the MRP explosion and determine, first, the periods where there will be spare capacity; second, on those periods, what percentage of that capacity should be used to anticipate production for other periods. In the second phase, they decide what products will be produced with the previously determined spare capacity. They used a variation of the Q – learning (Watkins & Dayan, 1992) algorithm and had success validating the approach.

Creighton and Nahavand (2002) applied reinforcement learning to the problem of economic lot sizing with the added complexity of considering stochastic failure, which randomly changes the capacity available to execute a production plan. Another example of reinforcement learning applied to economic lot sizing is the work of Wang et al. (2012), they use the Q-learning algorithm to train their reinforcement learning agent and introduced a heuristic to balance the trade-off between exploration and exploitation during the algorithm's learning phase. Dranidis and Kehris (2001) tackle approximately the same problem as Cao et al. even though production environments (simulations) are different. The biggest difference between the two studies is that Dranidis et al. use estimated values for value functions that are provided by neural network approximators. This technique is often used to address "the curse of dimensionality" that emerges from trying to discretize a continuous state space. This is a current trend, as we have introduced before a state is generally represented by a vector of metrics, those metrics can be continuous or discrete in nature. Most commonly they are continuous, for example in the case of the cannonball, velocity is a continuous variable, so defining the state of the system by using velocity will translate into infinite state space. Hence, we would have to compute an infinite number of value functions, making the problem intractable. The solution is sampling over a large range of states and then using an estimator to compute value functions for other states as we require. Neural networks are now the most common estimator, but other supervised learning algorithms can be employed.

Arredondo et al. (2010) used reinforcement learning in the context of dynamic order acceptance in make-to-order manufacturing systems. This is the issue of managing order acceptance taking into account the opportunity cost of committing capacity to a lower profit order when a higher profit order may arrive at any moment. This is a very important problem when demand surpasses capacity, and we cannot accept every order that arrives into the system. They used an average reward, which represented the average revenue generated per order per capacity (ρ). Then, they defined the reward function of the MDP by comparing ρ with the generated revenue per capacity of a particular order, r. In other words, the agent would receive a positive reward if it accepted an order where $r > \rho$ and negative if it accepted an order where $r < \rho$. They created a novelty algorithm based on Q – Learning, named ARLOA that used locally weighted regression to estimate value functions. The novelty of the algorithm is primarily based on the fact that they sort the list of orders by the expected value of accepting them, and then the RL agent will make the decisions of accepting or rejecting those orders by the order defined in that list. The exploration and exploitation trade-off is balanced by allowing with a given probability, $1 - \varepsilon$, the RL agent to decide first on orders at the bottom of that list. That probability is dynamically adjusted during the learning process. Their algorithm showed comparable results to heuristics proposed by other authors.

Another use of reinforcement learning for management purposes comes in the form of goal regulation in an evolutionary manufacturing system. The work of Shin et al. (2012) proposes a learning approach where an agent will learn how to form goals that are compatible with the current state of the system. This is an interesting problem to manage the goals of individual resources.

4.4 Reinforcement learning for production flow control

The research into production flow control through the use of reinforcement learning it is still recent. Xanthopoulos et al.(2019), used Q-learning to train an agent that authorizes production within a CONWIP environment. To use Q-learning the authors had to discretize the state and the action space. This approach artificially reduces the complexity of the problem which can, but not necessarily, lead to suboptimal policies. Q-learning also restricts the use of continuous and infinite action spaces.

In CONWIP, for example, we control WIP by setting a cap, theoretically, that cap can be any number between 0 and infinity, the vanilla Q-learning algorithm cannot deal with continuous and infinite action-spaces, and for that reason, the authors had to find a way of representing actions in a discrete manner: do nothing, release a production authorization and capture a production authorization. Defining the problem in this manner changes the focus from controlling WIP to controlling the release-rate. It has been shown that controlling WIP is a more robust control problem than controlling release-rate (Spearman & Zazanis, 1992a). This phenomenon has been shown mathematically under certain assumptions, but intuitively we can think of a WIP cap as a limit to an error, for example: for any state a given system is in there is a minimum amount of WIP that maximizes TH, if we overestimate it, we will have that maximum TH with an higher amount of WIP, but there is a limit imposed by the cap; if instead, we control release rate, we may release at a higher rate than TH, and there is no limit to the amount of WIP that can be accumulated in the system. Both control problems are valid but controlling WIP generates more robust policies.

There are some perceived shortcomings in the current research on the topic that can potentially be overcome, under contemporary developments of the reinforcement learning field, namely: allowing for continuous state and action spaces; including in the state representation, information about random processes like processing times, demand and TH.

Chapter 5 Using deep reinforcement learning as a self-adapting WIP parameter setting method for flowlines.

In this chapter we make our first attempt at solving the problem of WIP parameter setting using RL. We start with a simple problem, using a simulated flowline as testbed, present the results and set the ground for further improvements.

5.1 Experimental set-up

The first step to solve a reinforcement learning problem is defining its Markov decision process. We will use as environment the flow shop described in section 2.3 (Figure 14).

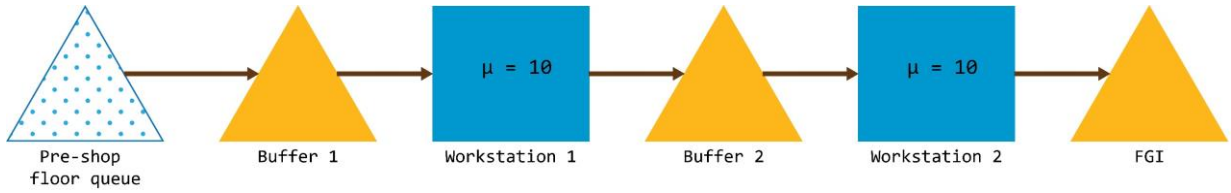


Figure 14- Simple flow line

This system is comprised of 2 workstations with intermedium buffers, a finished goods inventory, and a pre-shop floor queue. The pre-shop floor queue is where orders will wait for a release authorization and can be understood as the result of a previous schedule mechanism, or just simply a queue formed by orders that came in and should be processed in a FIFO paradigm. An episode is defined as a production system run with 3000 units of time. Workstation processing times will be defined by a uniform distribution with parameters [0, 20]; we opted for this distribution with a 0.58 coefficient of variation to test if an agent trained under high levels of variability can still perform well under scenarios of less variability stress. This might appear a strange choice over a Poisson process, for example, but we will relax this assumption later. The goal of using uniform distributions is to be able to test the RL agent against environments where variability is increased, but average values are kept the same. Exponential distributions do not allow to independently manipulate the coefficient of variation without affecting expected values.

To simulate a production schedule that respects the capacity of the line, order interarrival time will be set by a uniform distribution with parameters [0, 22]. This makes comparisons fairer between a pure push system and a pull system. This is so because in unbalanced systems, where demand exceeds the TH rate, the pull system just by setting an arbitrarily large WIP cap will probably perform better, and we want to avoid this bias. Similarly, if variability is very low and

demand is aligned with the TH rate, then we will not see much difference in performance between a pure push system and a pull system when it comes to average WIP levels and TH rates. The simulations will be programmed with a discrete event simulation python package called Simpy.

5.2 Optimization goals and assumptions

- The environment will operate in a make-to-order fashion, where demand D is a random process given by a uniform distribution with parameters: $\mathrm{D} \sim U[0,22]$.
- Processing times will be given also by uniform distributions $P_i \sim U[0,20]$, $\forall i$, where i stands for the i^{th} workstation.
- There will be unlimited raw materials, so that production never has to wait for supplies.
- A CONWIP mechanism will oversee production releases, any other pull methodology could potentially be employed, but for simplicity of implementation, CONWIP is the perfect test bed for a new dynamic parameter setting procedure.
- There will only be one type of product.

The optimization goal is given by Equations (10):

$$\begin{aligned} &\text{maximize:} \sum_{t=0}^{T} \mathrm{TH_t} \\ &\text{minimize:} \sum_{t=0}^{T} \overline{\mathrm{WIP_t}} \end{aligned} \tag{10}$$

Where,

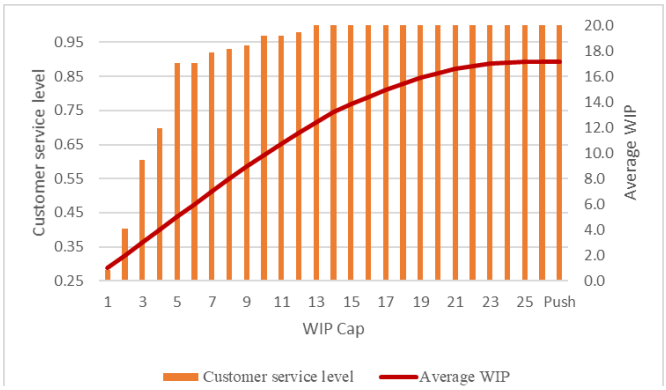
$$TH_t = \mathrm{f}(a_t) \qquad\qquad \overline{WIP_t} = g(a_t) \tag{11}$$

This optimization goal falls from the discussion in section 2.5.7(a) p. 37, where in a CONWIP make-to-order environment we assume a pool of scheduled orders from which the release mechanism will act upon. Assuming that a schedule is built upon the optimization goal of maximizing costumer service level, the release will be responsible to execute that schedule as fast and as efficiently as possible, by maximizing TH and minimizing average WIP at every decision

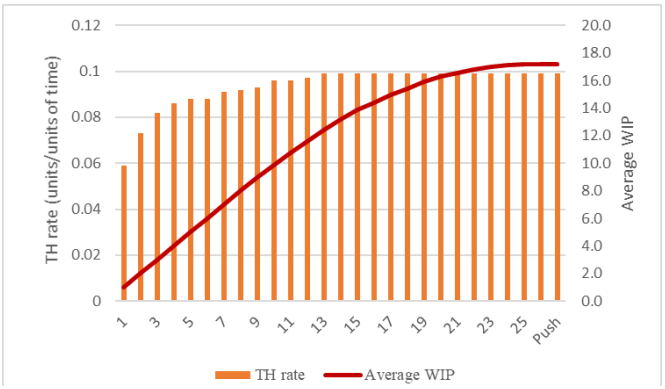
epoch *t*. We assume that $\mathrm{TH_t}$ and $\overline{WIP_t}$ are both functions of the action, a_t, prescribed at time *t* by our Reinforcement learning agent.

The focus on maximizing TH may appear short-sighted, as there are plenty of situations where higher TH is not the end goal, for instance, maximizing customer service level (CSL), minimizing set-up times, or other relevant metrics. The idea is that by creating a production schedule or sequence, you optimize for those metrics, and the RL Agent will take care of making sure that a given plan is executed as fast and with as little WIP as possible. As we will see, our RL Agent will not change a production sequence; it will only delay production order releases – that otherwise would wait in-between workstations in pure-push systems – so that when those releases actually happen, orders spend less time waiting as WIP, significantly reducing their cycle-time. This fact makes sure that our approach will work for any pull policy that assumes a pool of orders awaiting production release, no matter how that pool was created. As for pull policies where that pool does not exist, as in Kanban, where orders are filled from finished goods inventory, the procedure will have to be slightly changed, as we will show in Chapter 8. For a matter of concreteness, we will demonstrate how, given a feasible production schedule, maximizing TH whilst minimizing WIP will still maximize CSL for a CONWIP system. First, we have to take into account that in a pure-push system, a schedule is strictly respected since orders are released as they were planned without any delay and with no regard to current system state. When this is the case, a pure push system will provide the most loyal execution of a production plan. If a production plan is feasible and created with the intent of maximizing CSL, then a pure push system, by the same logic, will maximize CSL. If a production plan happens to not be feasible, the pure-push system will still be the benchmark for achieving the best CSL from a given schedule. The only thing CONWIP does is delay order releases, only affecting the level of WIP within the system, and not improving CSL. There is nothing that CONWIP can do to improve customer service level by just delaying order releases. CONWIP's goal is to delay order releases according to the system state so that orders wait less time in buffers. What we now have to show is in what conditions is CONWIP able to attain the same CSL that a pure-push system does, given that they are executing from the same production schedule. We will not offer a mathematical proof, but resorting to simulations, we can show empirical evidence of those said conditions. We will use the flow line of Figure 14, and execute the same production schedule under different scenarios: in a pure-push fashion and with a CONWIP policy with different WIP cap levels. The results can be seen in Figure

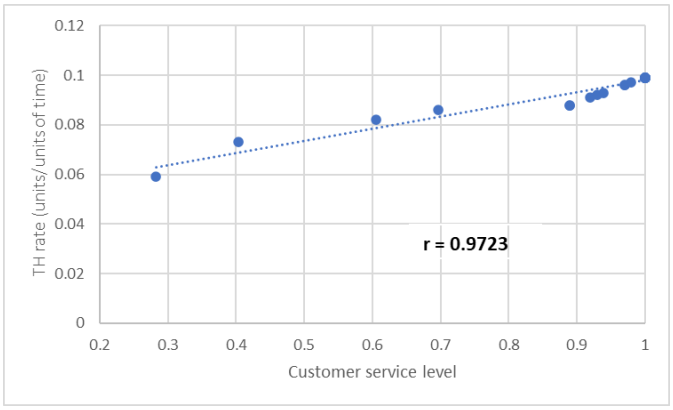
15: in a. we observe that the larger the WIP cap, the higher the CSL will be. No WIP cap at all will produce the maximum possible CSL attainable from a production schedule (see Appendices A and B for the order due dates and the production schedule used in this demonstration). Most importantly, we can observe that WIP caps greater than 13 units, under this production schedule do not provide a better CSL. Therefore, having a WIP greater than 13 units is a waste of resources. We observe the same exact relationship between WIP caps and TH rate (b.). Finally, we can empirically demonstrate that these two variables (CSL and TH rate) are almost perfectly correlated, with a coefficient of correlation of 0.97. At last, with respect to the conditions that make CONWIP attain the same service level as a pure-push system while executing from the same schedule, we can now ascertain that as long TH rate does not decrease, we can safely reduce WIP while still maintaining CSL. The reason why CSL is not a good performance metric to maximize in our reinforcement learning problem is that two systems can have the same CSL at time *t*, but the lateness of completed orders may be smaller for one of those systems, which can be an indicator of better performance. Looking at minimizing lateness alone is also not a very solid approach, as delivering an order ahead of time would offset the tardiness of other orders, and therefore you can have two systems with the same lateness but different CSL. By maximizing TH under a production schedule, we abstract away the scheduling layer, still respecting production sequences and respecting CSL and lateness (Figure 15, c. and d.) oriented goals.



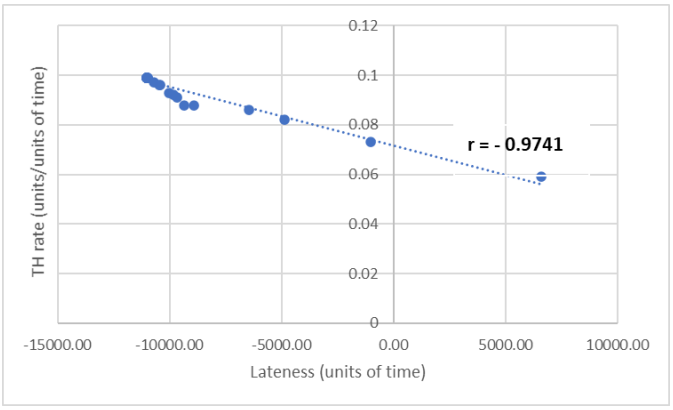
a.



b.



c.



d.

Figure 15- Relation between Customer Service Level and Lateness with TH rate in a system with different WIP caps and a pure-push system

5.3 State space

Our state vector has 7 dimensions, comprised of the following variables:

- Average WIP Level;
- Average interarrival time
- Standard deviation of interarrival time
- Average utilization for workstation 1

- Average utilization for workstation 2
- Average processing time for workstation 1
- Average processing time for workstation 2

These variables are continuous; therefore, we can have an infinite amount of states. As stated, our state representation is comprised of average and standard deviation values, which means that a state needs a sample of values so these statistics can be computed. Hence, a state in our system is not a discrete time step but the result of running the system for an interval of time and computing those statistics. Consequently, what we have is a scenario where our decision epochs happen at a constant interval of time. This decision is important, and the authors think is compatible with real-world cases because when we are defining a WIP cap limit for a production system, we cannot change that limit without thinking about the specificities of our process, if we change the system too often, there won't be enough changes in the system state that justify a change in the WIP cap, or we don't have the necessary processes to enforce that change so frequently. For our study, we used a decision epoch interval of 30 units of time, but this is considered as a parameter to be adjusted depending on the problem (hyperparameter).

To compute these statistics, we use exponential moving averages with bias correction to increase the precision of our estimates. We perform the updates by using equation (12) for all state variables. In equation (12) we show the average WIP as an example. The decision of using exponential moving averages comes from trying to instill the model with the sense of relative importance of values read at different times. The larger the β the lesser the importance we are giving to the past values of the decision epoch is. This is another parameter that can be tuned according to the problem. For this study, we used 0.9 for every state variable.

$$WIP \coloneqq \frac{\beta\, WIP_t + (1 - \beta)\, WIP_{t-1}}{1 - \beta^t} \tag{12}$$

5.4 Action space

Another important decision to make is to define the action space appropriately, there are a lot of possibilities, and one of them could just be to consider a continuous and unbounded action space i.e. allowing an infinite amount of possible WIP caps. Doing that would increase the complexity of the problem and decrease the spectrum of algorithms we could choose from, at optimization time. For this reason, we chose to start by discretizing the action space, this discretization presupposes some knowledge of the production system. Allowing for a continuous action space is an interesting pursuit as it removes the need of introducing domain expertise and makes the learning problem more general. Although that is the case, there are not many reinforcement learning algorithms that deal with continuous action spaces. In the next chapters, the authors will try to explore this idea and apply policy-gradient methods like the ones first introduced by Williams (1987, 1992), and more recently enhanced by Mnih et al. (2016) where a new actor-critic method was introduced.

We bound the maximum WIP cap to be 100 parts as statistically is very unlikely for the WIP to grow past this amount for the simulated system exemplified in Figure 14. Therefore, we allowed for the following vector of possible actions: [1, 2, 3, 4, 5, 6, 8, 10, 12, 14, 16, 18, 19, 30, 40, 50, 60, 70, 80, 90, 100]. From this vector we can see that action space is more sparsely defined after 20 units, again as the designers of the system, we know by running simulations in advance that the best policies will be found for WIP cap levels mostly below 20 units, therefore we do not need as much refinement above 20 units. The focus here is on using domain expertise to define the problem in such a way that the training of the algorithm takes less time. In this particular case running our environment and observing natural WIP occurrences gives us a very general idea of what the solution space might look like.

5.5 Reward function

Defining a reward function is often the most important part of a learning problem, and we must make sure that the reward signal we decide to implement translates itself into the desired behavior we want to observe from our policy. In other words, we may be maximizing the expected total reward at every episode, but our system may still be performing badly, this is often a sign that the reward signal is not representing effectively the desired behavior.

As we discussed previously, we have a multi-objective goal, we want to maximize throughput whilst minimizing WIP. To aggravate the problem, we have two goals that conflict with each other, therefore we know that there exists a pareto front of non - dominated solutions. The shape of this pareto front can be observed in Figure 7 as the red-hued curves. For a given level of variability, there is a curve where we have all the possible trade-offs between TH and WIP and we observe the best possible TH for a given WIP level. We can only operate below this curve, and we know that if variability changes this curve is pushed down, i.e. for a given WIP level the maximum possible TH will be lower. The field of reinforcement learning that tries to address these multi-objective problems is quite recent and is called MORL (multi-objective reinforcement learning).

There are two main classes of methods to optimize this kind of problem: single-policy or multi-policy methods. In single policy methods, we try to find a policy that maximizes a measure of all the possible objectives, this is often a linear weighted sum of the different reward signals. Although of usually simple implementation it requires the user to specify his preferences towards each one of the objectives, if those preferences were later to be changed then the RL agent has to be trained again under those new preferences. In the multipolicy methods, we try to learn different policies that approximate the pareto front, then at decision time the decision agent can decide what policy is best according to his preferences. Presenting the pareto front to the user provides them with the trade-off information among the objectives as well as the interaction among the competing goals. Although appealing, multi-policy methods are very computationally expensive because we are learning more than one policy at the same time, and they still need the input of an external decision mechanism (or agent). A more in-depth overview of multi-objective reinforcement learning methods can be found in (Chunming Liu et al., 2015), and a framework on how to apply multi-objective methods with deep reinforcement learning can be seen in Nguyen (2018).

As stated in section 2.5.7(a) we want to balance these two objectives so that we operate near the pareto front location where the improvement in TH over the increase of a unit of WIP is close to 0. As the reader can see this definition although reasonable is not precise, we want to find the critical WIP for a given variability, but is not easy to say, for example using a weighted sum of rewards, what combination weights will produce the critical WIP.

To run away from the complexities explained above we devised a competition structure where we trained our agent to perform better than another agent. By default, we know the two policies that maximize the two objectives individually. The policy that maximizes TH is always to choose the highest possible WIP cap, and the policy that minimizes the average WIP is to always set the WIP cap limit to 1. This means that we can devise a reward structure where we provide rewards when our agent performs better than an agent that maximizes TH. In other words, we will reward our agent when it makes decisions that result in the same TH as the agent that tries to maximize TH, but with lower levels of WIP. Because the policy that maximizes TH is given for free the only added complexity is that every time we want to evaluate the performance of our agent we have to run two simulations, one for each of the policies.

When trying to measure TH for a given epoch, we find that measuring it directly led to problems like: if during a decision epoch interval, no part came out of the system the TH would be 0, but that doesn't mean that system is not performing well, it means that processing times may be unusually long and this problem does not, necessarily, result from starved workstations. Therefore, we measured TH indirectly by finding what was the average bottleneck for that given epoch and looking at the utilization of that station. If the utilization of bottleneck is high, it means that the TH will be high as it is possible, under the current operation conditions, and vice-versa. The average WIP was measured directly. Let us define u_π as the utilization of the bottleneck station for the simulation controlled by our policy, and $u_{\max TH}$ as the utilization of the bottleneck station for the simulation controlled with the *maxTH* policy. For the rest of this sub-section let the subscript π mean that we are referring to a variable coming from a simulation controlled by our policy and the subscript *maxTH* as referring to a variable that comes from a simulation controlled by the policy that maximizes TH.

Algorithm 1- Reward structure for RL agent	
1	If $(u_{\pi} \geq u_{maxTH})$:
2	Reward $= (u_{\pi} - u_{maxTH}) + (WIP_{maxTH} - WIP_{\pi})$
3	Else:
4	Reward $= \frac{u_{\pi}}{WIP_{\pi}} - \frac{u_{maxTH}}{WIP_{maxTH}}$

In **Algorithm 1** we can observe the reward structure used in the problem. With this structure we not only wanted to reward our agent when it performed better than the *maxTH* policy but wanted to give rewards proportionally to how much better it performed. Therefore, there are two main scenarios: if the utilization of the bottleneck was higher or equal in π than in maxTH, it means that our policy was able to have at least the same TH, therefore we will reward it with the difference between u_{π} and u_{maxTH}, but then, we need to look at the WIP of the two policies. If $(WIP_{maxTH} - WIP_{\pi})$ is negative it means that our policy achieved good results with respect to the utilization due to higher levels of WIP, consequently we will subtract this value from the reward that came from the utilization advantage. If $(WIP_{maxTH} - WIP_{\pi})$ is positive it means that we not only had higher bottleneck utilization, but also achieved it with lower WIP, consequently we will increase the reward, which constitutes the highest possible reward our agent could receive. The second scenario is when the utilization under π is lower than under the *maxTH* policy, when this happens, we want to compare how efficient are the two policies with their WIP. Dividing utilization by WIP gives us a metric of how much utilization was achieved per unit of WIP, we want to give a reward to π, even if it could not match the utilization of *maxTH*, if it was more efficient with its WIP. This reward will have less impact than the first one, which reinforces the idea that we are more interested in achieving the same levels of TH as the *maxTH* policy is able to, but with the smallest amount of WIP possible.

5.6 Deep-Q Network optimization algorithm

For the optimization of this learning problem, we used the DQN algorithm introduced by Mnih (2015). This algorithm uses Q value functions, which represent the future expected cumulative reward for a given state an action, assuming that we follow a given policy π , as in equation (13):

$$Q^{\pi}(s,a) = E[r_t + \gamma r_{t+1} + \gamma^2 \mathrm{r}_{\mathrm{t+2}} + \cdots |\mathrm{s}_{\mathrm{t}} = \mathrm{s}, \mathrm{a}_{\mathrm{t}} = \mathrm{a}, \pi] \quad (13)$$

The goal is now to find a policy π that maximizes the expected sum of rewards. As we have seen when we have a state space that is continuous, we will have an infinite number of value functions, therefore we need to use a value function approximator as we cannot compute all the value functions analytically. If we have this estimator at decision time we can just solve for:

$$\pi^{*}(s) = \underset{a}{argmax}\, Q_{\phi}(\mathrm{s}, \mathrm{a}) \quad (14)$$

Where $Q_{\phi}(\mathrm{s}, \mathrm{a})$ is a value function estimator, parameterized by ϕ, which in our case will be the vector of weights of a neural network.

The job of the DQN algorithm is to find the parameter vector ϕ. The basic idea of most reinforcement learning algorithms is to estimate value functions by an iterative update that uses the Bellman equation as a target for that update. In DQN, the bellman operator depends on the estimator, as in equation (15):

$$Q(s,a) = \mathrm{r}(\mathrm{s}, \mathrm{a}) + \gamma \max_{\mathrm{a'}} \mathrm{Q}_{\phi}(\mathrm{s'}, \mathrm{a'}) \quad (15)$$

Despite that fact, in DQN the bellman operator is still used as the target for the update, therefore the cost function for the NN estimator is:

$$L(\phi) = E_{(s,a,r,s')}\left[\mathrm{r(s,a)} + \gamma \max_{\mathrm{a'}} \mathrm{Q}_{\phi}\,(\mathrm{s'},\mathrm{a'}) - Q_{\phi}(\mathrm{s,a})\right]^2 \qquad (16)$$

Notice, that in equation (16) we take the expectation of the error over state transitions. As in any supervised learning problem, we will need sample data with the ground truths, these ground truths come in the form of the bellman operator. But because the bellman operator is dependent on the function approximator, we have a situation where we are learning not just the approximator, but also the ground truths. This fact makes the distribution of our sample data to be constantly changing, which can make the learning process unstable. Intuitively it can be perceived as trying to hit a target that is always moving.

Reinforcement learning is known to be unstable and often diverges when using nonlinear approximators such as deep neural networks (Tsitsiklis & Van Roy, 1997). For that reason, DQN introduces some changes in Q-learning with function approximation that help the algorithm converge. One of the problems we have already pointed out is that the target values change from each parameter update to the next, this can create instability and make the approximator diverge. For example, if our training data has a considerable number of examples of transitions of the form of *(s, a, r, s'),* for a particular state s and action a, if we train our network over them, despite the generalization power of the network, we would expect that the network will be biased to perform well when trying to approximate the *Q* value for *(s, a)*, but, because ground truths depend on the function approximator, we cannot be sure that this will happen. The problem arises because there is no guarantee that performing updates over the transitions *(s, a)* will make estimations for $max_{a'}\, Q\,(s', a')$ better. In fact, the opposite may happen, and if that is true, we will make successive updates to *Q (s, a)* based on ground truths that are consecutively worse, making *Q (s, a)* to consequently run towards a worse estimate. With this problem in mind Mnih et al. (2016) introduced the idea of having a target network to compute the target $r(s, a) + \gamma \max_{a'} Q_{\phi'}\,(s', a')$. Notice that we now have a new network with parameters ϕ', used to generate the ground truths that is different from ϕ. The parameters ϕ' in practice are a lagged version of ϕ , i.e. over *C*

updates to ϕ we set $\phi' \leftarrow \phi$. Now that the target moves slower than the approximator we decrease the probability of divergence.

Another technique introduced by Mnih et al. (2016) to Q-learning algorithm is experience replay. The idea of experience replay is to store sample data in the form of *(s, a, r, s')* transitions into a replay buffer, then at each updating step of parameters ϕ we draw a minibatch at random from this buffer and use it to compute the bellman operator. This is a better alternative than to use consecutive state transitions that come from the same simulation, as this will lead to updates based on sample data that is heavily correlated. With experience replay we collect data from different episodes (simulations) and then randomly select from this buffer a sample that will serve as the target of the update. This sample is constituted by data that comes from many different episodes, ergo reducing the variance of the updates.

The algorithm used to optimize the policy for the flow control problem introduced in this paper can be observed in **Algorithm 2**.

Algorithm 2 – DQN algorithm used to optimize Flow control problem	
1	Initialize replay memory D with capacity for N state transitions
2	Initialize NN approximator parameters ϕ
3	Set target NN parameters $\phi' \leftarrow \phi$
4	Fill replay memory D
5	**For** k = 1 , K do:
6	
7	**Every 100 steps of k, Run Until episode (simulation) ends** With probability ε select a random action a_t, otherwise, select
8	$a_t = arg\max_a Q(s_t, a; \phi)$
9	Execute action a_t, observe reward r_t and state transition s_{t+1}

10	Store transition (s_t, a_t, r_t, s_{t+1}) into D
11	**End Run Until**
12	
13	Sample a random minibatch of size *m* with transitions of the type $(s_t, a_t, r_t, s_{t+1})_j$ from D, using a Uniform distribution.
14	
15	Set $y_j = r_t + \gamma \max_a Q_{\phi'}(s_{t+1}, a)$
16	Perform a gradient descent over the minibatch with respect to parameters ϕ on: $$\frac{1}{m}\sum_{j=1}^{m}\left(y_j - Q_{\phi}(s_{tj}, a_{tj})\right)^2$$
17	Every C steps of k set $\phi' \leftarrow \phi$
18	**End For**

5.7 Agent training phase

The architecture of the neural network used to estimate the values Q- values is comprised of an input layer with 8 units, 7 of those used for the state representation, and one unit for the action we want to evaluate. The output layer is comprised of a single linear unit, that outputs the Q-value for a given state and action. The network has 4 fully connected hidden layers, with 100, 80, 60, and 30 units respectively, all activated by leaky linear rectifiers (Figure 16).

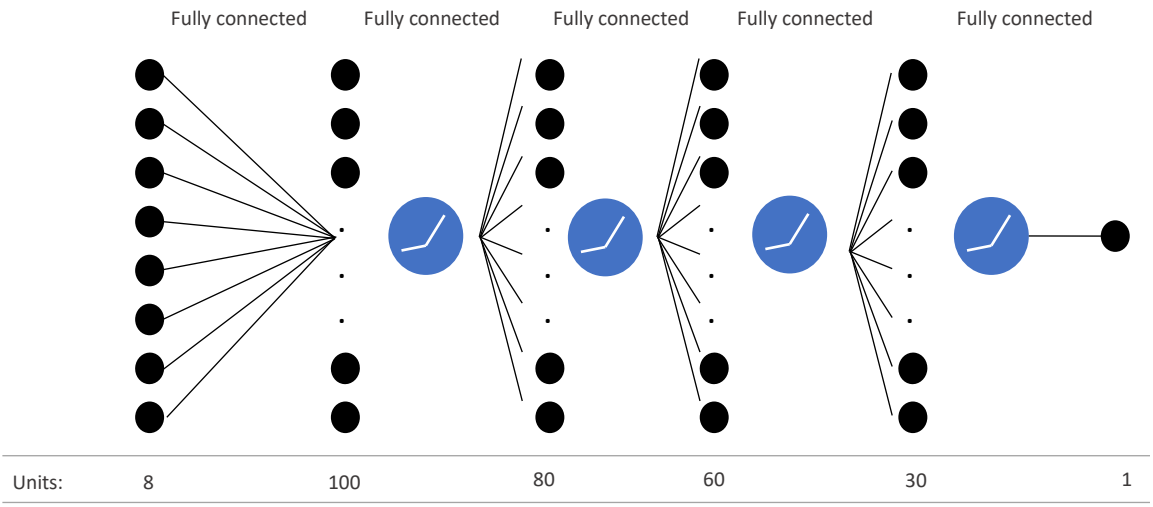


Figure 16- Architecture of the Neural Network used as Q- value approximator

One of the crucial parts of making the reinforcement learning approach work is to collect experience that will be used as a basis for the network to learn. Our goal from the beginning is to let the network decide what is the right WIP limit so that we can have the highest TH at the lowest level of WIP possible. Knowing that the best WIP limit will change according to the state of the system, more precisely with its variability in terms of processing and interarrival times, the intention is not so that the agent learns how to perform under specific random processes. In this case, we trained the agent for an environment where demand and processing times were defined by uniform distributions with given parameters, ideally, we are interested in a decision agent that is robust to changes both in the distributions and parameters of those random processes. That is the reason behind using sample statistics in the state representation so that the agent never sees the random processes distributions and parameters directly, learning to act without having full information about the random processes that govern system dynamics. In the next chapters, we will test system performance against changes in the random process distributions.

We collected data by running episodes, those episodes (simulations) had random durations and random warmup times, we only started collecting state transitions *(s, a, r, s')* after the warmup time expired.

One limitation of the examples generated is that they all come from state transitions that are dictated by a fixed decision epoch interval. This may affect the performance of the policy as we will show in the following sections. We are training a policy that will perform well considering that it can make a decision every 30 units of time. This influences the decision's efficacy, for example, if we make decisions with high frequency, it means we can be greedier in trying to achieve a certain goal because if for some reason the result is not positive, we can adjust our strategy soon enough. If we had to stick for longer with each decision we made, we would probably be conservative and think about the long-term effect of those decisions. A good decision in the short term may reveal itself to be a poor one in the future. We could allow for varying decision epoch intervals in the training phase of the algorithm to make our policy more robust to changes in the decision-making frequency. Robustness often comes at the expense of lower performance across the spectrum of possible decision epoch intervals, in opposition to high performance on given intervals of that spectrum and low performance on others. If the decision-making frequency was out of the control of production system managers as variability often is, it would be probably a good idea to make our policy robust to different decision-making frequencies. But, because this is often a decision made by managers, we find that we should opt for a system that performs as best as it can for a specific interval, at maximum requiring retraining when that interval is changed.

All the hyperparameters used in **Algorithm *2***, and those used to train the neural network are presented in Table 3. We point out that those parameters were not the result of a thorough grid search. Due to limitations of the set-up used to train the algorithm, a grid search would be computationally too expensive. Instead, we performed a search starting with some random parameters, observing the performance after some time, and trying to tune the model towards faster and more effective learning.

Most of the tuning was directed toward improving the reward structure, having a complex enough NN without requiring too much training time, and controlling the exploration parameter ϵ.

Table 3- Hyperparameters used in **Algorithm *2***

Hyperparameter	**Value**	**Description**
Minibatch size(m)	64	Number of training examples over each stochastic gradient descent (SGD) update is computed
Episode length	Randint (1100,3000)	Duration of each simulation in units of time
Target network update frequency	100	Frequency measured in number of SGD updates. Parameter C in **Algorithm *2***
Discount factor γ	0.98	Discount factor used in Q-learning update
Learning rate	0.001	Learning rate used in Adam Optimization algorithm
1st gradient moment	0.9	Parameter β_1 of Adam optimization algorithm
2nd gradient moment	0.999	Parameter β_2 of Adam optimization algorithm
δ	1e-8	Constant added to the denominator of the Adam optimization algorithm gradient. For numerical stability
Exploration	0.7	ϵ- value for $\epsilon -$ greedy exploration
Replay buffer size	100,000	Number of examples in the experience replay buffer
Decision epoch interval	30	Units of time between two agent decisions. Can also be seen as the time between two state transitions

5.8 Results and discussion

After training, the policy was evaluated by controlling 150 simulations, we used a random and the *maxTH* policy of section 5.5, as a baseline for performance comparison. The random policy takes a random action at every decision step from the set of possible actions [1, 2, 3, 4, 5, 6, 8, 10, 12, 14, 16, 18, 19, 30, 40, 50, 60, 70, 80, 90, 100] and the *maxTH* sets the WIP cap limit to 10,000 units at every step. The 10,000 WIP cap simulates not having a WIP cap at all, i.e., a pure push system.

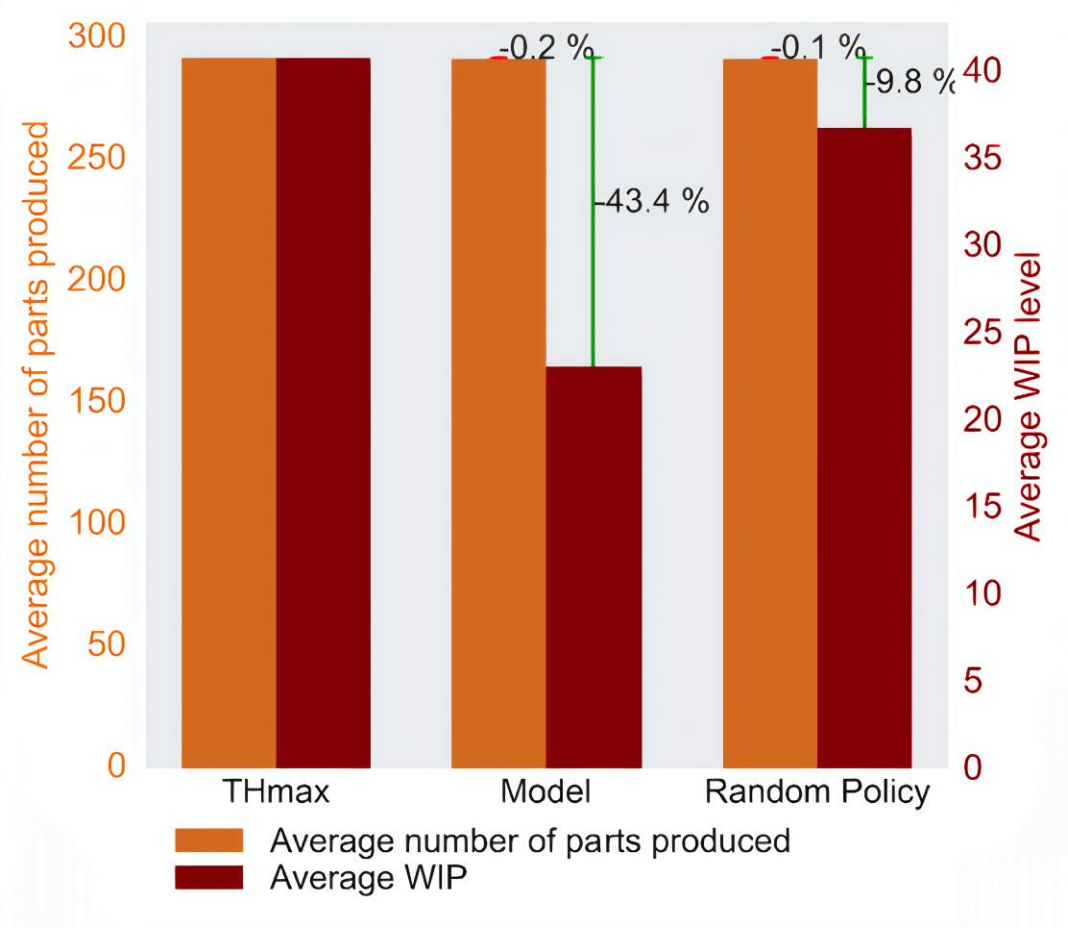


Figure 17- Performance comparison between the learn policy THmax and a Random

If we look at Figure 17 we can see how each policy managed the trade-off between the WIP level and TH. In the graph, the policy “Model” stands for the policy learnt through the use of the reinforcement learning algorithm, notice as well that TH in this graph is being measured as parts produced. Our policy was able to achieve practically the exact same TH (a difference of -0.2 %) with -43.4% less WIP when compared to the system with no WIP cap. This attests to the success of our approach. Although we cannot guarantee that our policy is in fact capable of determining exactly the critical WIP, our data shows that at least it pushes the WIP levels towards it. Another interesting behavior is that random policy was, as well, able to decrease the average WIP Level without any loss in TH. This behavior is part of a bias that we unintentionally introduced when discretizing the action space. The highest possible WIP cap limit in our action space is set to be 100. WIP levels above 100 have very little probability of occurring but given enough simulations it will eventually happen. Which means, that both our model and the random policy have a free advantage compared to the *THmax* policy.

Chapter 6 Reducing domain expertise required for designing reinforcement learning approaches to WIP parameter setting

In the previous chapter, we showed that RL-trained agents can effectively reduce WIP in the flowline without compromising throughput. However, the discretization of the action space can influence the performance of the algorithm. This means that in order to properly model the problem, domain knowledge about what constitutes a good solution is still necessary. In this chapter, we address this disadvantage and then proceed to compare our approach with previously existing methods in more detail

6.1 Experimental set-up

This chapter will start by explaining the motivation to use a new type of algorithm called Proximal Policy Optimization. The previous chapter showed the potential success of the reinforcement approach applied to the problem of dynamic parameter setting for a CONWIP system, nevertheless, there are still limitations to this approach, namely the fact that DQN requires the design of a discretization strategy for the state space. This step presupposes some degree of expertise about the system, as there is a need for intelligent discretization based on the characteristics of the production system. This step can be skipped by using Reinforcement learning algorithms that allow for continuous and infinite action spaces. PPO (Proximal policy optimization) (Schulman et al., 2017) is one such algorithm.

The production environment for this new algorithmic approach will be the same used in section 5.1. The MDP will be slightly different in terms of action space, and reward function.

6.1.1 Action space

The action space is the WIP cap to be applied to the production system, in this case, it would be the set of natural numbers, but to speed up the convergence of the RL algorithm we chose to cap the limit to the subset of natural numbers $A = \{a \mid a \in \mathbb{N}, a < 100\}$. This step can be skipped, but setting a limit is very easy, as it does not have to be precise, it only needs to be large enough to accommodate all the best potential actions available at every state. We can find this limit by simulating the system for some time and observing the maximum WIP recorded. This might seem contradictory; we expressed our preference for an infinite action space, where the discretization strategy will not have influence on the performance of the policy but at the same time, we are restricting our action space to be a discrete and finite set. Being discrete is a requirement to prescribe WIP caps, we cannot have fractional units of WIP, bounding the state space to 100 is imposing a rough limit, this value could easily be exchanged by an arbitrarily large number. Moreover, the described process is only done after the agent prescribes an action, in fact, the agent will prescribe an action from a continuous and infinite action space, only after that action will be rounded and clipped.

6.1.2 Reward function

We will continue with the same principle presented in section 5.5, aiming for the critical WIP, by comparing the performance of the policy with no WIP cap, the pure push policy, $maxTH$, with the Agent policy, π in the same way as previously described. If the agent makes decisions that result in the same TH as the agent that tries to maximize TH, but with lower levels of WIP a positive reward will be given (equation (17)).

$$\begin{aligned} \text{Reward} = & \; number of \; parts \, produced_{\pi} \\ & - number \; of \; parts \; produced_{maxTH} \\ & + \left(\overline{WIP}_{maxTH} - \overline{WIP}_{\pi}\right) \end{aligned} \tag{17}$$

Even though the general rationale behind the reward function is the same we now present a simpler version of the reward function previously proposed. The need for redefining the reward function came from the very sluggish performance improvements observed in the first training approaches with this new algorithm. Instead of comparing bottleneck utilization to attest throughput rate, we measured output directly by counting the number of parts produced at the end of each period. This reward function does not contain as much information as the other, and if the output for a specific decision epoch is 0 we will be immediately penalizing a given action, without taking into account that the occurrence may be due to unusually long processing times at the bottleneck station and not a starved bottleneck. So, in this particular case, we would be penalizing the agent both when processing times at bottleneck are longer than the decision epoch interval and when the bottlenecks are actually starved, equally. Ideally, only the latter occurrence should be penalized as the Agent has no say on the processing time of the workstations. There are possible explanations for having to change the reward function in this new manner, though it is not easy to say exactly why the old version doesn't work as well. One possible explanation may be that the previous reward function does not provide a strong signal towards the maximization of throughput, this is because utilization rates vary between 0 and 1, and WIP levels can be as high as 100 units or more, therefore without some normalization, WIP levels will have a higher impact on the

objective function, making the algorithm prone to exploit this by trying to reduce WIP more aggressively. For some reason, PPO performed better with the simpler version. It might not be apparent why we use two different metrics WIP and TH in the same function without weighing both metrics. The reason is related with the following: the difference between the two throughputs will never go beyond 0, meaning no pull system can get higher throughputs than a pure-push system, by definition. In this case, where both systems have more or less the same TH, we are interested in rewarding our agent proportionally to the amount of WIP it was able to reduce. If TH of the pull-system is lower than the TH of the pure-push system, we will get a negative value for the first part of the reward. And it may be the case that this difference is so large, in magnitude, that the second part of the score has little to no influence. As it is, this is not really a problem because learning can still happen in two phases. A phase where the agent learns to define WIP limits that are enough to obtain the same TH as the pure-push system, this learning is characterized by rewards where TH difference has higher magnitude than average WIP difference. And the second phase, where the TH of both systems is rather similar and higher rewards are generated mostly by decreasing the WIP of the system without reducing TH.

6.2 What is PPO

In contrast to value-based, there is a class of RL algorithms where instead of learning action values and extracting a policy, we learn the policy directly. In these cases, we parameterize a policy by $\pi(a \mid s;\theta)$ and perform gradient ascent on $\mathbb{E}[R_t]$. This quantity is usually approximated by $\widehat{\mathbb{E}}\left[\log \pi(a_t \mid s_t;\theta)\hat{\mathrm{A}}_t\right]$, where $\hat{\mathrm{A}}_t$ is an estimator of the advantage function at timestep *t*. This advantage function is usually the difference between a sampled return and a learnt value state function, therefor $\hat{\mathrm{A}}_t = R_t - V^{\pi}(s_t;\phi)$. Because R_t is a sampled return for a given episode it has been shown that subtracting $V^{\pi}(s_t;\phi)$, reduces the variance of the estimate of $\widehat{\mathbb{E}}[\mathrm{R_t}]$, (Greensmith et al., 2004). This value function is also parameterized by a vector ϕ which is updated by minimizing the mean squared error between the sampled returns and $V^{\pi}(s_t;\phi)$ equation (18). The sample returns used to train the value function estimator can be full episode returns, for every state visited during an episode we sum all the rewards that come afterward until

the end of the episode. Or, we can use n-step returns, where we use actual rewards up to step *t+n-1*, and thereafter the rewards up to the end of the episode are computed with the estimator itself, $V^{\pi}_{\phi old}(s_{t+n})$, but by using a previous version of its parameters ϕ_{old}.

$$L_{critic}(\phi) = \widehat{\mathbb{E}_t}\left[r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) \dots \gamma^{n-1} r(s_{t+n-1}, a_{t+n-1}) + \gamma^n V^{\pi}_{\phi old}(s_{t+n}) - V^{\pi}_{\phi}(s_t)\right]^2 \quad (18)$$

To clear the notation, $\widehat{\mathbb{E}}$ represents an empirical average over a finite batch of samples obtained by running episodes of the environment. This type of model architecture where we have a parameterized policy and a state value function is called actor-critic, where the policy is the actor and the critic is the state value function approximator (Sutton & Barto, 2018).

The PPO algorithm (Schulman et al., 2017) comes from the realization that performing multiple optimization steps on $\widehat{\mathbb{E}}\left[\log \pi\left(a_t \mid s_t; \theta\right)\hat{\mathrm{A}}_t\right]$ as is the case in other policy gradient algorithms (Sutton et al., 2000), using the same batch of samples led to destructively large policy updates; PPO goes around this fact by penalizing large policy changes. In PPO, the actor's optimization function is given by equation (19).

$$\mathrm{L}_{\mathrm{actor}}(\theta) = \widehat{\mathbb{E}_t}\left[min\left(r_t(\theta)\,\hat{\mathrm{A}}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{\mathrm{A}}_t\right)\right] \quad (19)$$

	Where, $r_t(\theta) = \frac{\pi(a_t \mid s_t; \theta)}{\pi(a_t \mid s_t; \theta_{old})}$	(20)

We can see that $r_t(\theta)\,\hat{\mathrm{A}}_t$ is the importance sampling version of $\widehat{\mathbb{E}}\left[\log \pi\left(a_t \mid s_t; \theta\right)\hat{\mathrm{A}}_t\right]$, where samples are collected under $\pi(\theta_{old})$. This objective function when $\hat{\mathrm{A}}_t > 0$ removes the incentive for $r_t(\theta)$ to go beyond $1 + \epsilon$ and when $\hat{\mathrm{A}}_t < 0$ to go beyond $1 - \epsilon$. This is made so that we do not update θ too much just based on one batch of sample episodes. ϵ is another hyperparameter to be tuned according to the problem. To this objective function, we also added

the policy Shannon entropy (Shannon, 1948) so that exploration is encouraged, and the policy does not converge rapidly to one with poor performance, this entropy is weighted by a coefficient *c*.

In policy gradient methods $\pi(a \mid s;\theta)$ will give us the probability of choosing action *a* at state *s*. Often, in RL problems π would be modeled as a deep neural network where the output layer will have $|A|$ units, which is just the number of actions available at state s, where A is the action space. As explained above our action space is $A = \{a \mid a \in \mathbb{N}, a < 100\}$, a discrete and finite set, although it is only finite because we artificially decided to restrict caps above 100 units. We want a method that works even when no limit is imposed. Therefore, in our model, a deep neural network will output 2 quantities an average μ and a standard deviation σ so that our policy will be a given by normal probability density function. That is, we have $\mu(s;\theta)$ and $\sigma(s;\theta)$ and our policy is parameterized as in (21):

$$\pi(a \mid s;\theta) \approx \frac{1}{\sigma(s;\theta)\sqrt{2\pi}} exp\left(-\frac{\left(a-\mu(s;\theta)\right)^2}{2\sigma(s;\theta)^2}\right) \tag{21}$$

This way we can deal with continuous and infinite action spaces. Hence, when determining the action for a given state, we sample it from a normal distribution $N\sim(\mu(s;\theta), \sigma(s;\theta))$ and only after we round up and clip the value to be a natural number below 100.

The pseudo-code for the PPO algorithm implementation can be seen in **Algorithm 3**.

Algorithm 3 – PPO Algorithm	
1	• For iteration = 1… to N1:
2	○ For actor = 1… to N2:
3	▪ For Episode = 1 to N3:
4	• Run policy $\pi_{\theta old}$ until episode (T)
5	• Compute advantage estimates $\widehat{A_1}$ ... $\widehat{A_T}$, using, V_ϕ^π
6	○ Optimize L_{actor} with respect to θ for $k_1$steps
7	○ Optimize L_{critic} with respect to ϕ for k_2 steps
8	○ $\theta_{old} \leftarrow \theta$
10	○ $\phi_{old} \leftarrow \phi$

6.3 Agent training phase

The PPO algorithm needs two Neural Networks one to estimate state value functions $V^\pi(s_t;\phi)$, the critic network, and another to prescribe the normal distribution parameters, $\mu(s;\theta)$ and $\sigma(s;\ \theta)$, from which an action is sampled $\pi(a|s;\theta)$. The architecture of the Critic network has 5 fully connected hidden layers, with 100, 100, 100, 80, 40 units activated with a linear rectifier, and an output layer with a linear unit that provides state value functions estimations (Figure 18).

The actor network has one shared trunk with 3 hidden fully connected layers activated with exponential linear units, with 100, 80, and 70 units, respectively (Figure 19). The actor-network also has two heads, the first responsible for outputting $\mu(s;\theta)$, with 2 hidden layers, with 60 and 50 units, respectively, activated by exponential linear units. The just-described head has an output

layer with a rectified linear unit (*relu*). The second head, responsible for outputting $\sigma(s;\ \theta)$ is equal to the first one in terms of its architecture.

During the training phase, we used multiprocessing techniques to run different agents in parallel, each of these agents had a copy of the actors' neural network and was responsible to control several production runs. This was done to accelerate data collection. At the end of an epoch the data from the different actors was collected and used to compute loss functions, gradients and finally update the parameters of the neural networks, those updates were then passed to the actors' processes so that data collection for the next epoch could commence.

The hyperparameters used for this problem are shown in Table 4. Analogously, to the approach used in the DQN algorithm, those hyperparameters were tuned iteratively, without a well-defined systematic procedure. During training, it became indispensable to monitor the learning process in real-time, especially at the early stages of learning, by collecting information such as gradients after each epoch, the distribution of prescribed actions, and the distribution of weights at the various layers. The training of the actor and the critic networks took around 23 hours of wall time, excluding the time used for debugging the reward function and the algorithm implementation.

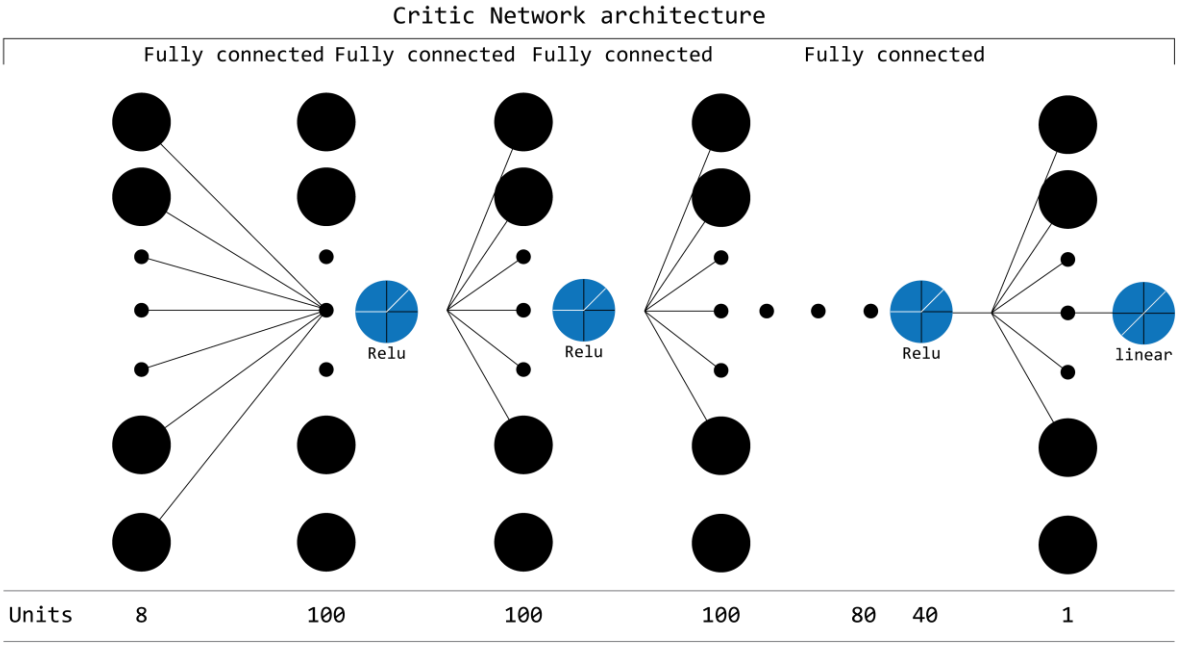


Figure 18- Critic Neural Network Architecture

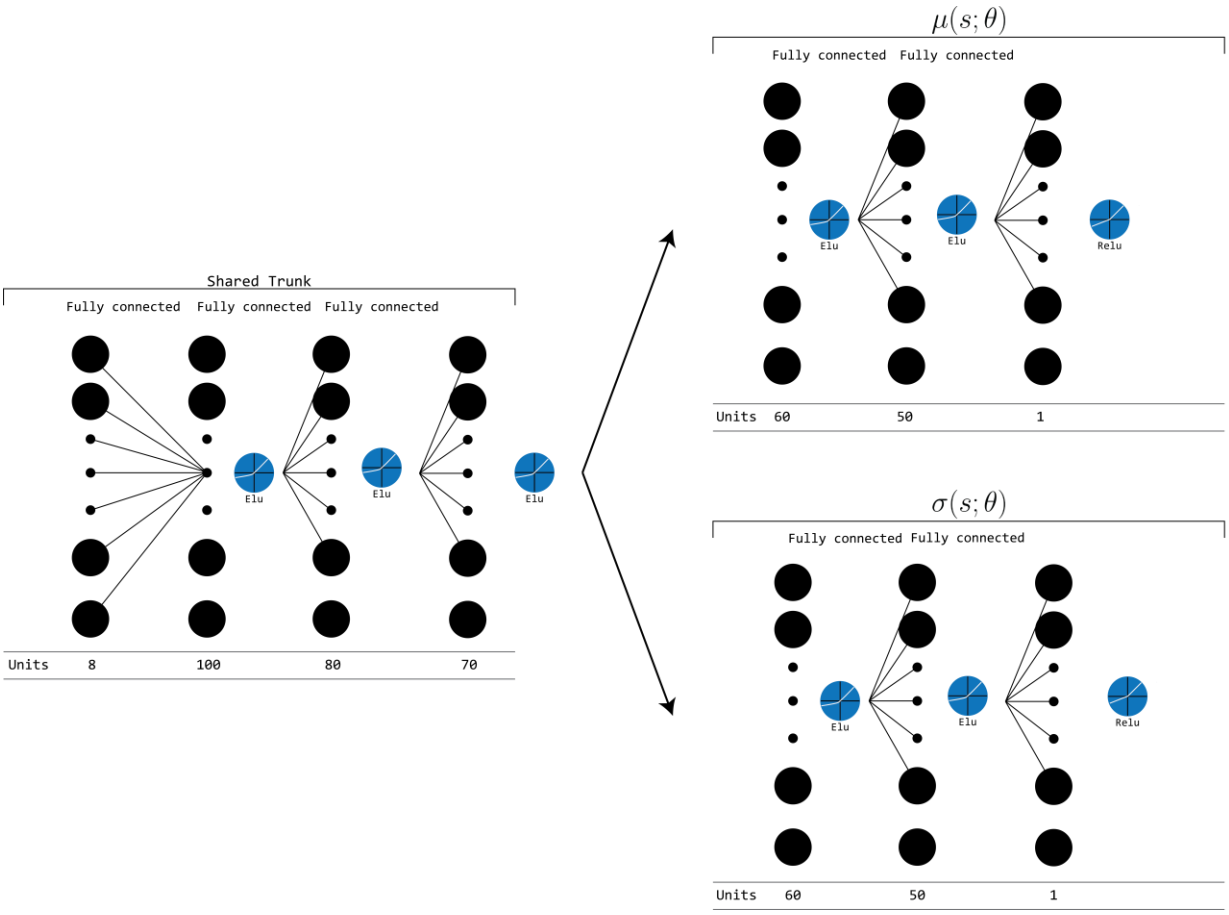


Figure 19- Actor Neural Network Architecture

Table 4- Hyperparameters used in **Algorithm 3**

Hyperparameter	**Value**	**Description**
N2	8	Number of Actors running in separate CPU cores
Episode length	Randint (1100, 2000)	Duration of each simulation in units of time
Critic optimizer	SGD(learning_rate =0.0001,momentum=0.9)	Optimization algorithm and parameters for critic network
μ optimizer	SGD(learning_rate=0.0001, momentum=0.9)	Optimization algorithm and parameters for the μ actor network
σ optimizer	GD(learning_rate=0.0001, momentum=0.9)	Optimization algorithm and parameters for the σ actor network
γ	0.99	Reward discount factor
C	From 1 to 0.005	Entropy coefficient
Gradient Scaling	L2 Norm scaling = 1	After gradients are computed they are scaled to avoid exploding gradients and large updates in the direction of a given batch of samples
ϵ	0.2	
Critic number of step returns (n)	20	
Number of episodes per batch	40	Each actor run 5 episodes por learning epoch
Actor number of gradient steps per batch (k_1)	10	After an epoch we compute the loss function, its gradient, and then update parameters. We repeat this process k_1 times based on the data from the same epoch,
Critic number of gradient steps per batch (k_2)	5	The same explanation for k_1 serves for k_2

6.4 Results and discussion

We are interested in knowing 3 things: how our approach compares to pure push systems, if it can preserve TH and reduce WIP, and by how much; if it is robust to changes in variability without model retraining; and lastly if changing the decision epoch interval would change performance, and how. The latter 2 elements were introduced in previous chapters, but not tested before, we could very well use DQN to test these elements, but because PPO is a better algorithm in terms of allowing for more powerful representations of production environments, which are very often made of continuous random processes, it makes more sense, for reasons of brevity, to only carry out this analysis once.

6.4.3 Performance

In Figure 20 we compare the trained policies to a pure push system and a random policy that chooses actions at random (WIP cap = random integer between (0, 100)). We present three models all trained with the PPO algorithm discussed in the previous section. The only difference is that "Model" was trained to outperform "THmax"; "Competition Model 1" to outperform "Model"; and "Competition Model 2" is just a more mature version of "Competition Model 1" meaning that it was run for a larger number of episodes. As discussed in section 6.1.2 the reward structure is the result of a direct comparison between a push system and a model to be trained. We use a pure push system as a basis for that comparison because, naturally, if no WIP cap is enforced we expect no losses in TH related to processes starvation from an imposed WIP limit. The idea is that because THmax controlled environments will have large levels of WIP, there is the danger of the competing policy plateauing to lower levels WIP, but that is still considerably high. This means that a policy A that was trained by competing with a policy B, which, in its turn was trained under THmax policy, will receive on average lower rewards if compared to B than if compared to THmax. This train of thought leads us to the possibility of making policies more and more greedy towards a diminishing potential for improvement. An analogy would be for example the difference between

training a high jump athlete to jump over a crossbar at a fixed height, but encouraging him to jump higher and higher, so that he increases the distance to the bar at each trial. Or, by raising the crossbar at every trial making it actually harder to perform a task successfully. Both methods of learning seem valuable, nevertheless, both are of relevant examination.

By looking at Figure 20, we can discern that our approach is capable of reducing WIP in the system by up to 56.3% with minimal TH losses. We noticed that the competitive approach, where we trained policies to outperform previous versions of themselves works well and increases the performance of the models.

This is important because when we stopped training "Model", it was stuck, and performance was not increasing consistently with more algorithm iterations. But when we started to train a new model from scratch to outperform "Model" we saw that it reached "Model"s performance very quickly, and was able to surpass it. The random policy shows a reduction in WIP of 4.7%, this is just a reflection of the imposed limit of 100 when choosing randomly, again, this goes to show how effective and robust controlling WIP is, even with a very generous WIP Caps.

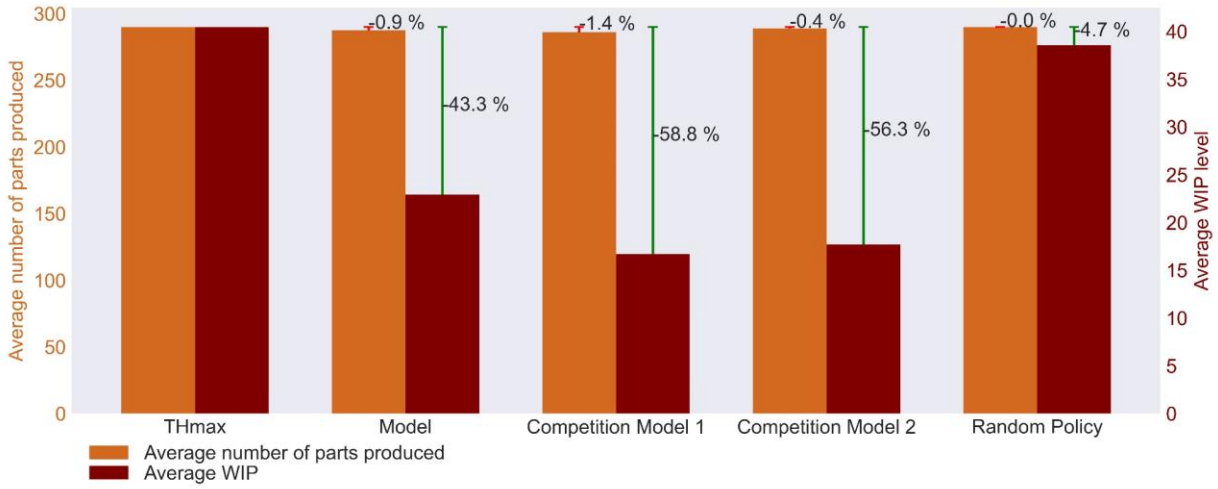


Figure 20- Performance of trained models (Model, Competition model 1, competition model 2), against pure push system (THmax), and Random Policy.

6.4.4 Robustness of policies to changes in variability and impact of decision epoch intervals

Figure 21 is the result of the study on the effects of variability and decision epoch interval. Our models were all trained under a uniform distribution (0, 20) and to take actions every 30 units of time. Without any re-training, we wanted to test how robust "Competition Model 2" performance was to these underlying changes. The first observation is that by decreasing the decision epoch interval we lose performance, and by increasing it we gain it, but only up to a given limit, which when surpassed, performance starts decreasing. That limit happens to be between 5/3 and 1/3 of the total running length (3000 units of time). This behavior, although not entirely intuitive, may be explained by the fact that the longer the decision epoch interval, the greater is the information contained in state representation that the policy acts upon. It seems to express the trade-off of acting very reactively with very little information or acting sparsely with better information.

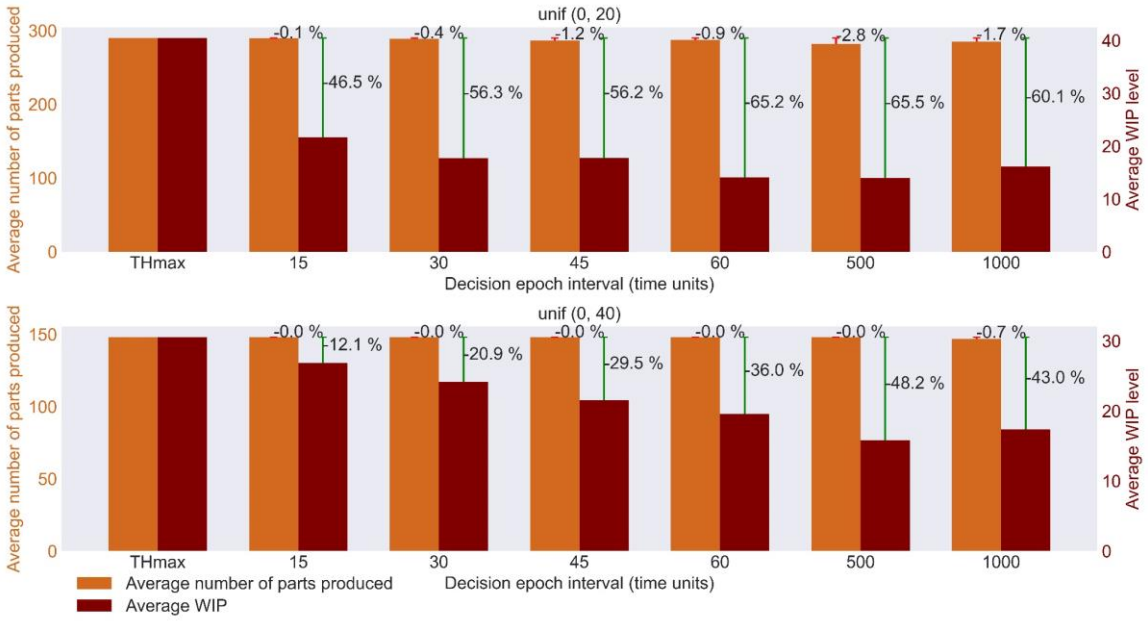


Figure 21- Effects of changing decision epoch interval and increasing variance (comparison between THmax and Model Competition 2

In Figure 21 the effect of increasing variance by a factor of 4 reduces the performance of our policy, on average by 40.6%, but that loss of performance can be compensated until a certain point by increasing the decision epoch interval, and we can see that if the decision epoch interval is set

to 500 units of time we get a 48.2% decrease in WIP with no loss in TH. This result shows the potential robustness of the learnt policies even when the system is under great variability stress without the need for retraining the policy under these new changes. It shows that there is evidence that the Agent is not learning to act under a given random process but is retaining some elementary knowledge about how to deal with variability.

6.4.5 Benchmarking against Statistical Throughput Control and Fixed CONWIP

The last stage needed to demonstrate the utility of a new method is to directly compare it with existing methodologies that propose to solve the same problem.

In section 2.5.7(b) we have examined, the most well-known procedures used to determine the parameters of the pull control mechanisms. For this benchmark, the aim is to not make an exhaustive comparison with all methods available in the literature, as that would be a very time-consuming task, with probably unfair results.

Following the abovementioned rationale, we can exclude analytical methods that are deterministic, for the reason that our approach being stochastic, approximates the stochastic nature of real manufacturing environments better, which is a bias for better performance. The analytical methods that resort to steady-state analysis, although allowing for stochasticity, do not consider dynamic changes in the underlying random processes that govern the production systems. For that reason, they assume as the name suggests stable random processes, perturbations, or changes in the hypothetical production systems, will require new steady-state analysis, therefore, comparisons with these techniques will be biased towards favoring our approach. Additionally, some methods that resort to steady-state analysis and queuing theory consider close queuing networks, which results in assuming demand to be infinite, a process we model as a random variable. The last type of methods available are those that use heuristics and meta-heuristics, those would-be direct competitors to our methodology, however, there is a detraction that makes comparisons hard: they assume stable random processes; To lift those assumptions the heuristic would have to be run every time a shift in the system would occur. This is a very computationally intensive procedure, as at decision time we need to quickly prescribe actions. Our methodology has a training phase where it learns to react according to system state, which is, as well, a computationally expensive process,

but at the decision time, because it is not dependent on steady-state analysis, actions can be prescribed in constant time, as no re-training is necessary. Our RL agent, as explained in the previous sections, sees only sampled values of the random processes, from which it computes statistics, that are used to infer the best actions to take. We can think of it as training an agent to continuously react to random variable distribution changes, even when the underlying processes from which those sampled values are taken remain the same.

To the author's knowledge, there is only a method that does not rely on steady-state assumptions, and that can be run in constant time at decision time (Hopp & Roof, 1998). This method makes for a perfect comparison as it relies on the same principles, them being, not requiring much knowledge of processing time distributions – the procedure will see sampled values of these metrics and compute statistics – and automatically respond to changes in the system. The presented procedure using reinforcement learning techniques is much more complex, but that complexity is hidden way in the training phase, at decision time both techniques make up for good comparisons.

The above-mentioned method was proposed by Hopp and Roof (1998) and uses the idea of Statistical Throughput Control (STC). In this method after a warm-up period, statistics about the interoutput time (average and standard deviation) are computed and compared to the inverse of a target throughput rate λ. Whenever average interoutput time reaches above the inverse of target throughput by more than three times the standard deviation, the system is declared to be "out of control on the low side" and the card count is increased by one. If average interoutput time goes below the inverse of target throughput by more than three times the standard deviation, the system is coined as being out of control on the high side and the card count is decreased by one. The procedure also requires a fail-proof control for when a throughput target rate is infeasible. If that is the case, with the described mechanism the TH rate target would never bet met and the system would be in a state of positive feedback ad infinitum, adding more and more WIP to no avail. There can also be situations where due to high variability, even if feasibility questions are out of the way, we can reach states where TH target rate cannot ever be met. For that reason, we need to control average cycle-time (CT) and establish a limit for this metric, so that λ, or production capacity is adjusted before the system "overflows" with WIP. The adapted version of the procedure taken from Hopp and Roff (1998) can be seen in **Algorithm 4**.

In **Algorithm 4** we consider a new parameter τ. Originally the authors of the procedure considered the categories of being out of the target, as deviations above or below 3 standard deviations from a specified target. From experiments, we noticed that changing the number of standard deviations as a trigger for parameter adjustment, changed the performance of the production system, smaller values of τ, made the system react quicker, which in the context of adapting to ever-changing system characteristics may be a desired feature.

Algorithm 4 – WIP setting Statistical Throughput control	
1	Set initial card count *m* and warm-up period *n*
2	Set target production rate λ and maximum cycle-time CT_{max}.
3	After each job completion, calculate:
4	Average interoutput time
5	Standard deviation of interoutput time σ
6	Average cycle time CT
7	**If**
8	$CT > CT_{max}$ then revise capacity and/or λ; Go to 2
10	$\mu > \frac{1}{\lambda} + \tau\ \sigma$ then $m \leftarrow m + 1$; Go to 2
11	$\mu < \frac{1}{\lambda} - \tau\ \sigma$ then $m \leftarrow m - 1$; Go to 2
12	Go to 3

The authors also present another similar procedure, where instead of incrementing one unit of WIP at a time, they increment it in dynamic sizes, based on a congestion model authored by

Spearman (1991), called ACM. This model provides a formula to estimate cycle-time given a WIP level *m*, a congestion parameter α, the bottleneck rate r_b , the raw processing time T_0 and another parameter b_0 as seen in equations (22), (23), (24).

$$\overline{CT}(m) = \frac{1}{r_b}\left[m + \frac{\alpha(M_0 - 1)}{ln(1 + b_0)} ln\left(1 + b_0 e^{\frac{-(m - M_0) ln(1 + b_0)}{\alpha(M_0 - 1)}}\right)\right] \quad (22)$$

$$M_0 = r_b T_0 \quad (23)$$

$$b_0 = 1 - \frac{1}{(1 + b_0)^{\frac{1}{\alpha}}} \quad (24)$$

The logic for this new procedure is seen in **Algorithm 5**. In summary, after we compute b_0 and α we gain the ability to estimate the average cycle-time for hypothetical levels of WIP (m). Then by using Little's law, we can estimate the average TH rate, and now it is a matter of increasing *m* until the estimated TH rate is above λ.

Algorithm 5 – WIP setting Statistical Throughput control – dynamic step size	
1:	Solve for α using equations (22), (23) and (24) with the current estimate of CT and value of *m*
2:	Calculate $\overline{CT}(m)$ for various values of m.
3:	Find m^* as the minimum value of *m* such that $m/\overline{CT}(m) > \lambda$ ¸ (i.e., by Little's law, $m/\overline{CT}(m)$ is the throughput rate for a WIP level of *m*

4:	Set $m \leftarrow m^*$

In Figure 22 and Table 5 the benchmark results are presented. We can see that "Model" is the agent with the smaller loss in TH, WIP reductions are only relevant if not at the cost of TH, this is consistent with the idea of critical WIP. As explained earlier, a well-designed WIP cap is crucial when $C < \lambda$. In those moments we want our systems to be working at full available capacity with the least amount of WIP. Therefore, when comparing different agents, for one agent to be better than another it needs to have at least the same TH with less WIP than its competitors. The second-best agent is the CONWIP with a WIP cap fixed at 25 units. We can see that none of the fixed WIP cap strategies performs better than our Model. Fixed WIP caps above 25 units result in higher levels of WIP than our model, below 25 units they result in lower levels of TH. This shows that dynamically changing WIP caps improves performance.

The single-step STC agent does not react quickly enough. Since it only adds one unit of WIP at each time, it produces lower levels of WIP, at the cost of reducing TH by 9 %. "Model" on its turn adjusts WIP levels in dynamic sizes, according to system state, therefore it reacts quickly, with minimal loss in TH. In the results shown in Figure 21 "Model" is performing against a new version of the Figure 14 production system now designed with exponential random processes with the same expected values as those in Figure 14. In other words, the system was trained under uniform random processes, and now is performing against a production environment where arrival rates and processing times come from exponential random processes. Even though in both cases, the expected values for the random processes are the same, their respective coefficients of variation are different, during training, they were 0.58, and in this exercise, they are 1. This goes to show learned policies' performance resilience. Even with an input data shift from uniform to exponential distributions, performance did not degrade, and no agent retraining was necessary.

This change in random process distribution was done for fairness of comparison with the STC methods, as they were introduced in the setting of exponential distribution environments.

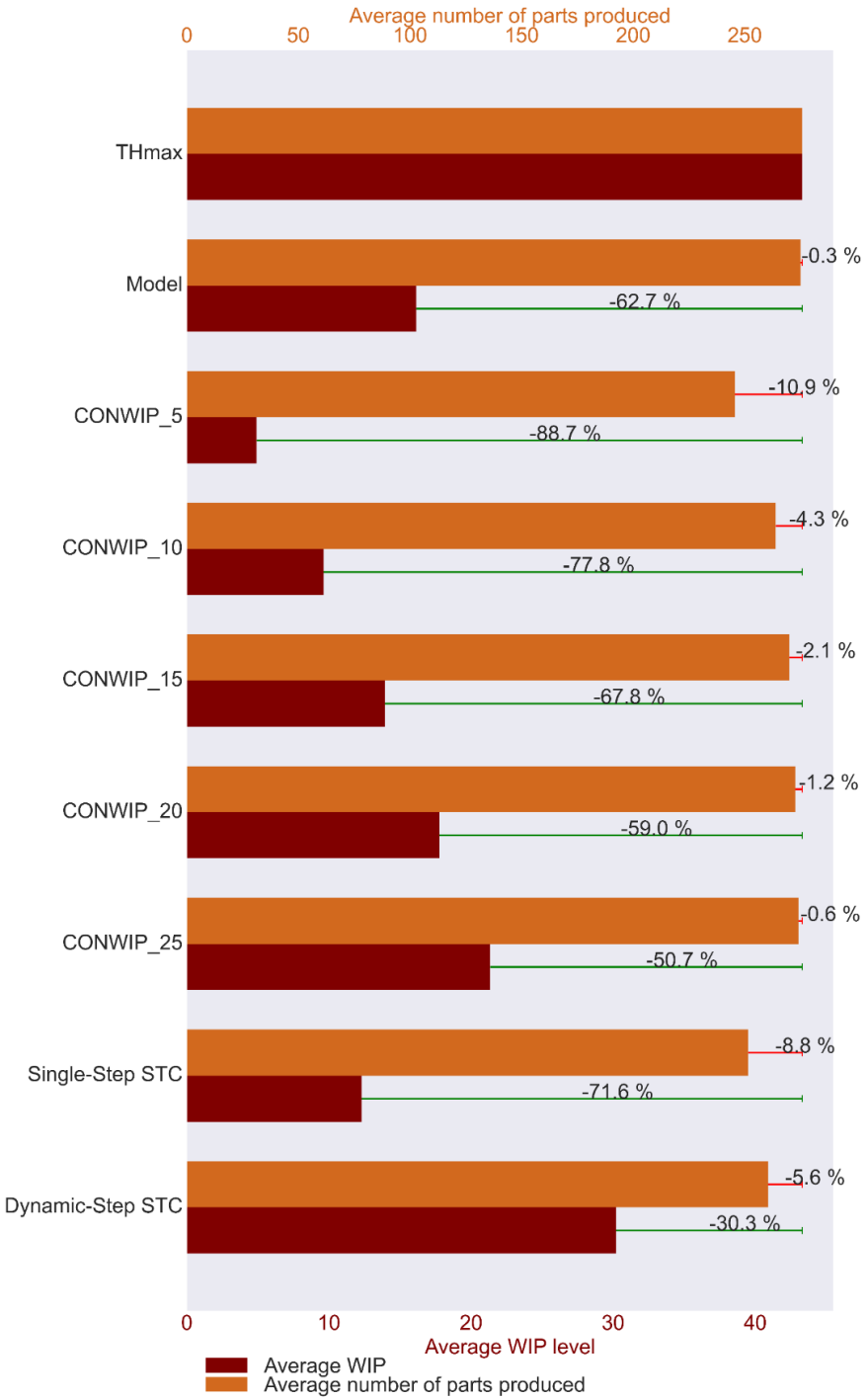


Figure 22- Comparison of RL Agent vs Statistical Throughput Control Agent in the setting of flow lines exponential random processes

Table 5- Comparison of RL Agent vs Statistical Throughput Control Agent in the setting of flow lines exponential random processes (confidence intervals for $\alpha = 0.05$)

	Number of parts produced		WIP	
Agent type	$\bar{X}$	CI	$\bar{X}$	CI
THmax	276.39	8.75	42.88	7.39
Model	275.67	8.73	16.29	0.84
Conwip 5	245.83	4.60	4.92	0.05
Conwip_10	264.50	6.79	9.63	0.17
Conwip_15	270.67	8.34	13.95	0.37
Conwip_20	273.33	8.74	17.84	0.67
Conwip_25	274.78	8.83	21.43	1.04
Single-Step STC	252.50	3.71	12.30	0.78
Dynamic-Step STC	261.06	7.70	29.38	4.61

6.4.6 Is the algorithm reverse engineering the random number generator behind the simulation random processes?

In section 3.2.3 we discussed the potential risk that our RL agent was able to reverse engineer the simulations' random number generation procedure. We had come to the conclusion that this was highly unlikely. First, because the period length of MT19973 is many orders of magnitude greater than the length of the sequences generated in our studies. Therefore, our agents would never be able to observe a full period of numbers. Second, the RL agents did not directly see the generated numbers from the random processes. Instead, the input to our agents are statistics derived from the generated random numbers. For completeness in this section, we show the performance of the "Model" agent with no retraining on a simulation environment with the same random processes used in section 6.4.5, but where the number generation procedure was the *CryptGenRandom*, referenced in section 3.2.3.

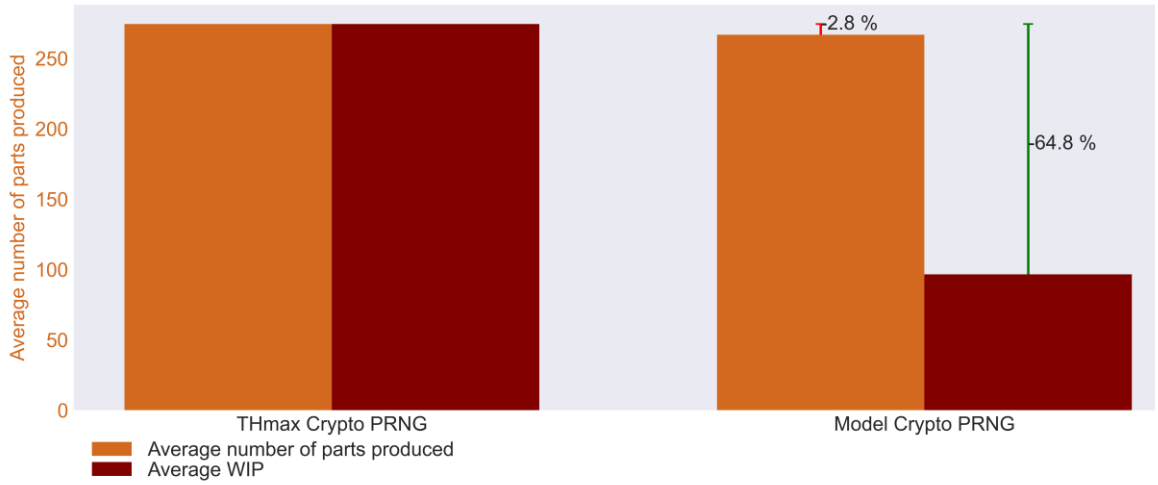


Figure 23- Performance comparison between the learn policy THmax and a Random

As the reader can see in Figure 23 the performance of the "Model" agent under this new random number generation procedure is similar to the performance obtained under MT19973. This shows that the RL agent is not reliant on learning the random number generation procedure to obtain good performance, otherwise we would observe a substantial loss in performance.

Chapter 7 Scaling the reinforcement learning approach to complex manufacturing systems

Until this point, we used a very simple flow line to prove the efficacy of the proposed methodology. However, to really make the approach interesting to the audience that would care the most, being them the managers of production systems with low volume and high product mixes, we need to test the approach in a proxy for those types of environments. By a proxy, we still mean a simple system, but one that encapsulates at a lower level the complexity of one of those production systems. The main goal is to define the challenges of scaling the approach and provide possible solutions.

7.1 Mix-product route system

To simulate the complexities of a low volume, high product mix, we propose a hypothetical production system with 4 Machines, shared between 3 types of products, that make up for 3 production routes. Each part will have random processes dictating its demand, and processing times. Figure 24 is the illustration of such a production system.

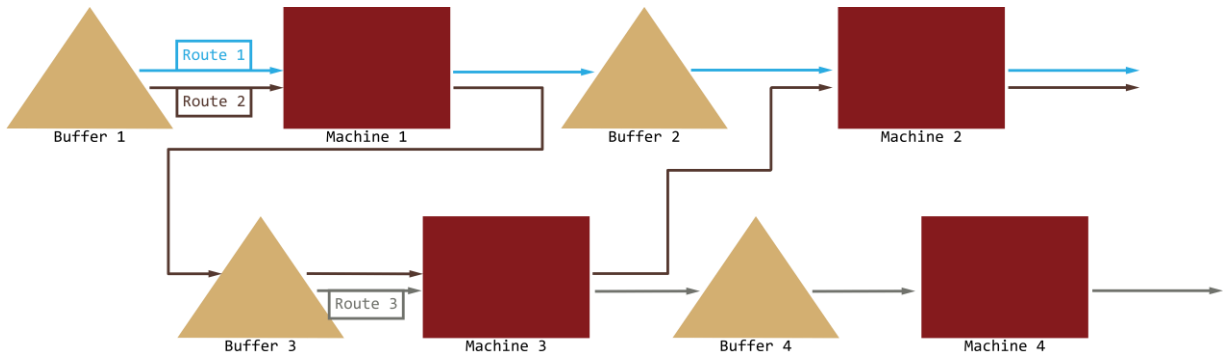


Figure 24- Mix product production system

Table 6- Random processes for demand and processing times of the different parts

Part type	**Demand (interarrival time)**	**Machine 1**	**Machine 2**	**Machine 3**	**Machine 4**
1	$U \sim [0, 60]$	$U \sim [0, 30]$	$U \sim [0, 30]$	_____	_____
2	$U \sim [0, 60]$	$U \sim [0, 15]$	$U \sim [0, 30]$	$U \sim [0, 30]$	_____
3	$U \sim [0, 60]$	_____	_____	$U \sim [0, 30]$	$U \sim [0, 60]$

When deciding the distributions of the different random processes, it is crucial to pressure the system, still keeping it balanced. The idea is to simulate the variability of a complex manufacturing system, where demand is still aligned with internal capacity, as previously

explained. To test the design of the random processes, we run the pure push system for 30.000 units of time to check if WIP converges. This is a sign that the system is balanced and WIP does not grow indefinitely. Also, to increase the level of variability inside the system, random demand processes were dynamic, meaning that its distribution parameters changed at every decision epoch. At the beginning of a decision epoch, interarrival times will be given by equation (25):

$$N \sim (X, 20), \qquad where \;\; X \sim U(a, b) \tag{25}$$

In equation (25), X is given by a Uniform distribution, whose parameters are defined on the demand column of Table 6, for the corresponding part type. For consistency and comparison purposes, we will keep using CONWIP as the pull method that will exert control over our system. But with this selection, there are more design decisions available, owing to the fact that we now have more than one production route. The decisions are: controlling a general WIP level, controlling WIP at each route, or controlling the WIP of each type of product.

Let us start with controlling a general level of WIP. Controlling an overall WIP level, though of simple implementation, can lead to undesirable situations. If we imagine that we start with a given WIP cap, and we execute orders from a given schedule, we may encounter a circumstance where a WIP cap is saturated with parts for a given route, while an order for another route is blocked, awaiting release, for a route that is empty, meaning that we have process starvation due to unbalanced schedule fueled by an insufficient WIP cap. This shows that with this approach when we are before a mix product system, a WIP cap can severely exacerbate an unbalanced schedule in terms of the product mix.

Controlling WIP at each route would signify that a WIP cap for each route had to be defined, this would, apparently, resolve starvation problems caused by unbalanced schedules/production sequences of the type portrayed above, nevertheless, there are two problems with this approach:

- The way we measure a WIP level in a route is usually by counting the number of orders/parts awaiting processing on the workstations that constitute the said route. The problem arises when we have shared resources; when that happens, if a product is waiting on one of those shared resources, that unit of WIP will be counted as many times as the

number of routes that resource is present in. This method of measuring WIP is vital because it measures the true level of congestion at a given route by considering that more than one type of product may contribute to that congestion. Nevertheless, simultaneously, it is not an accurate representation of true levels of WIP inside the overall production system.

- The second problem derives from the first, product routes that share the majority of their allocated resources with other routes will have the bulk of their WIP caps saturated with product types from other routes. This can translate into an unbalanced release of parts. To illustrate this, we can check route 2 in Figure 24, it shares machines 1 and 2 with the first route and machine 3 with the third route. If for a given period there are n orders destined to each of the routes, and there is a WIP cap of n, also, for each of one those routes, we would observe that both routes 1 and 2 would have allowed for fewer releases than route 3. The situation is such, that balanced WIP cap levels across routes will lead to unbalanced releases between the different types of products.

The remaining option is to control WIP not at each route but for each product type. Thereby, the goal is to determine the number of units of each type of product that are allowed in the system during a given period. Meaning, that if at the beginning of a production period, we define the WIP cap for product A to be 10, we would never release orders that would push WIP levels for that type above 10. Thus, by summing the WIP levels for each product type, we would realize the actual maximum level of WIP to be observed, in opposition to the previously explained method, where the true level of WIP had to be determined by accounting for repeated counts. For the above-stated reasons, controlling the level of WIP for each product type will be the strategy employed in this section.

With a mix product CONWIP configuration, the release mechanism has now the power to change the sequence of releases, something that was not possible in the single product, flow-shop CONWIP line. In the flow-shop, the release mechanism makes decisions that can delay the release of a part according to the state of the system, but it does not alter the order in which parts are processed. Suddenly, in a mix-product system, if product A is scheduled to be released before product B, but A's route is full at planned release time, and B's route still has space available, there is a chance that B will be released first. This can lead to a new scenario where the production of a given product takes priority over another. This is another challenge with this new mix-product

configuration: to respect the order established by a schedule and sequence. In practice, the decision agent may have to decide to restrict the WIP of Part A, not just because there is congestion on its route but also because Part B, whose route shares some resources with Part A's, is getting behind in schedule. Essentially, the decision agent has to guarantee that we do not produce some product types ahead of schedule at the expense of producing other product types behind schedule. The urge to respect a schedule and sequence comes from routes having shared resources, which presupposes that any release of Part A ahead of schedule could potentially increase the cycle-time of some Part B. If a schedule and sequence are done with the goal of maximizing customer service level (or other relevant metric), there is a risk or hurting this performance metric.

As in section 6.1, we will assume unlimited raw materials. This time, optimization functions will include information of the 3 routes (equation (26)), and f and g (equation (27)) are now vector-valued functions, $f, g: \mathbb{R}^3 \rightarrow \mathbb{R}^3$, that take as input the vector $\overrightarrow{a_t}$, which is the action vector prescribed by our Reinforcement Learning Agent at time t. Here, each component of the vector $\overrightarrow{a_t}$ will be a WIP cap for a corresponding product type. As explained above we need to add another goal to our optimization problem in the form of minimizing work ahead, WA_t, which will be a count of the number of parts of any type that were released ahead of schedule, or by disrespecting a predefined sequence, this count is also dependent on vector $\overrightarrow{a_t}$.

$$\text{maximixe:} \sum_{t=0}^{T} TH_t^1 + TH_t^2 + TH_t^3$$

$$\text{minimize:} \sum_{t=0}^{T} \overline{WIP_t^1} + \overline{WIP_t^2} + \overline{WIP_t^3} \tag{26}$$

$$minimize: \sum_{t=0}^{T} WA_t$$

Where,

$$\begin{bmatrix} TH_t^1 \\ TH_t^2 \\ TH_t^3 \end{bmatrix} = \mathrm{f}(\overrightarrow{a_t}) \qquad \begin{bmatrix} \overline{WIP_t^1} \\ \overline{WIP_t^2} \\ \overline{WIP_t^3} \end{bmatrix} = \mathrm{g}(\overrightarrow{a_t})$$

$$WA_t = \mathrm{h}(\overrightarrow{a_t}) \text{ , h: } \mathbb{R}^3 \to \mathbb{R} \tag{27}$$

7.2 State and action spaces

The state space will be a vector with 17 dimensions, comprised of the following variables:

- Average WIP Level of route *i, for i = 1, 2, 3*;
- Average interarrival time for product *j, for j = 1, 2, 3*;
- Standard deviation of interarrival time for product *j, for j=1, 2, 3*;
- Average utilization for workstation *k, for k = 1, 2, 3, 4;*
- Average processing time for workstation *k, for k=1, 2, 3, 4;*

This is in line with all the previous definitions used in section 5.2 and 6.1, being the only difference the fact of now tracking a larger number of parts, machines, and routes. We should remark that we are controlling the level of WIP by part type, and, by its state representation, the Agent will observe the average WIP per route. This is so, because the agent to make decisions about the number of parts of each type to be allowed in the system needs to take into account congestion, and average WIP level at a route is the variable that provides that information. We could consider using average WIP level per type of product, but then, for the Agent to infer congestion, we would need to provide it with data about the interaction between routes. In the end, it would make state representation more complex.

The action space will be very similar to the one defined for section 6.1.1, but now extended for 3 types of products. Action space, $A \in \mathbb{R}^3$, will be the set of 3D vectors, whose components,

$a_j \in \mathbb{N}, a_j < 100, \text{for j} = 1,\ 2,\ 3$. As discussed previously, the algorithm will consider the action space as a set of all possible positive real numbers in the 3D real space after its components are clipped and rounded to be natural numbers below 100.

7.3 Reward function

If we look at the optimization goals for this new control problem, we can anticipate that a more complex reward function will be needed. As in the previous sections, we will avoid going into the route of multi-objective reinforcement learning approaches and instead use the pure push system to establish performance comparisons and derive rewards from those juxtapositions.

We first need to understand what information a pure push system can give us and how to apply it to evaluate the decisions of our agent. In a pure push system, where no WIP cap is defined to any route, and there is no release policy, a production schedule and sequence are strictly respected – regardless of the state of the production system, orders will be released as planned. This means that TH rates for the different types of products could never go beyond those observed in a pure push system. The only way that would be possible is if we change the schedule/sequence, by favoring a particular type of product (working ahead) at the expense of another, what we would then observe is a higher TH rate for product A, at a cost of a lower TH rate for product B, given that they share resources. The TH rates for each product type given by the pure push system are the target TH rates for our decision agent (see. Figure 25), we do not want to go above them, as that would imply losing performance on the TH rates of other product types, and at the same time, we do not want to operate below them, as that would mean working under critical WIP for a given product type. This described incentive pattern encompasses the goal of maximizing TH and the goal of minimizing work ahead at the same time. What is less apparent is that this incentive pattern also covers the minimizing WIP part of our optimization goal. This happens because now by increasing the amount of WIP for a given product our agent could be indirectly punished through TH rate comparisons. If, that increase shifts a production schedule and sequence, to a situation where we allow work-head for a given product at the cost of another, then, the agent would be penalized without even having to compare WIP levels. In the single product, flow line, increasing TH would always be regarded as good, therefore, with no direct WIP level penalty, WIP would be

incentivized to grow, which is why a WIP level-related penalty was added for those circumstances. Here, as we will see a WIP level penalty would be redundant and lead to decreased Agent's performance.

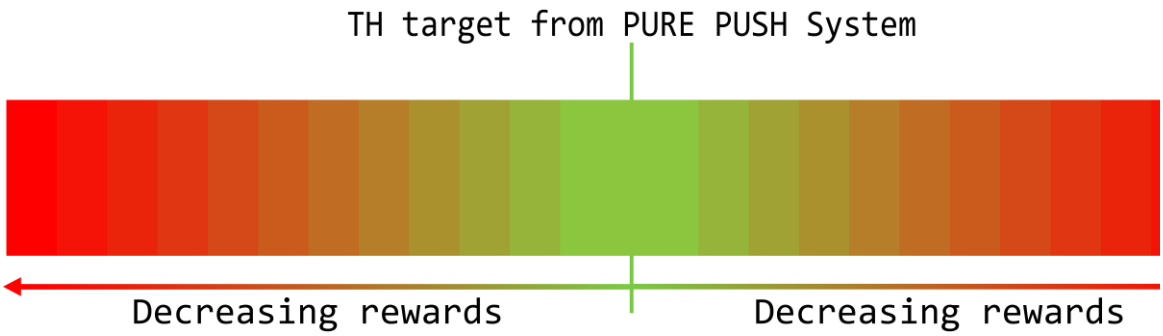


Figure 25- Reward structure illustrated

Algorithm 6 – Reward structure for the mix- product route system	
1	*For i = 1 to N, where N = number of product types:*
2	• *If current_throughput_difference > 0 and previous_throughput_difference > 0:*
3	• *Reward += previous_throughput_difference - current_throughput_difference*
4	• *If current_throughput_difference < 0 and previous_throughput_difference < 0:*
5	• *Reward += current_throughput_difference – previous_throughput_difference*

6	• Else:
7	• $Reward += |previous\_throughput\_difference| - |current\_throughput\_difference|$

Algorithm 6 represents the reward structure that we found incentivizes the Agent in the direction of our optimization goals. The main idea behind this structure is measuring the throughput difference for each product type between our CONWIP-derived control Agent and a pure push system. This measure allows us to track how far our agent is from the TH targets set by the pure push system. As a result, there are 3 main potential events:

- The agent is going in the direction of reducing the distance between the throughputs it is achieving and those set by the pure push system. In line 2, of **Algorithm 6** we measure *current_throughput_difference,* which is the difference of the cumulative number of parts finished, between the Agent's controlled system and the pure push system at time *t,* we also measure *previous_throughput_difference*, which is the same quantity, this time measured at time *t – 1*. If these two quantities are positive, for some product type, it means that our agent is working ahead of schedule, but instead of immediately penalizing this behavior, we use information about the previous decision epoch to check if the Agent is trying to improve. Therefore, we reward cases where the agent is out of target, but is shrinking the TH gap, and penalize the contrary behavior, where the agent is out of target and is increasing the TH gap between the systems.
- The second case happens when our system is operating below target TH. In this case, we would want to reward our system if it is below target, but is decreasing the gap, and penalize it, if it is doing the opposite.
- The third and last case happens when our system is below target and moves too much in the right direction that at the current time is above target or the other way around. In this instance, we will look at the absolute values of the current and previous TH differences and penalize our agent if that absolute distance is

increasing and reward it otherwise. To make it concrete, if our Agent at time *t – 1*, was below target, and now at time *t*, is above target, we want to express our preference by states where that difference is shrinking in absolute value.

As the reader may verify the reward structure presented in **Algorithm 6** could be simplified to just line 7 of the same algorithm. For exposition purposes, so that we make explicit the different situations our system may be in, we decided that is better to have the algorithm in its redundant form. Otherwise, its simpler version, though containing the same information, makes it less transparent.

7.4 Agent training phase

The algorithm used for training the RL Agent was PPO as in section 6.3. In terms of modeling the only differences between having a single product flow shop line versus a mix-product production system are the architecture of the critic and the actor's neural networks. By in large, the major changes consisted of increasing the number of layers and the number of units of each layer. To that avail, the actor's network (Figure 26) has a shared trunk with 3 layers, with the sizes, 200, 160, and 140 units, respectively, activated with exponential linear units. Still, with respect to the actor's networks, as previously, we have two heads. The first outputs 3 different averages and the second 3 unique standard deviations, that will be used to parameterize 3 normal distributions, from which the prescribed WIP levels for each product type will be sampled. The first head has layers with sizes 120, 100, and 80, all activated with exponential linear units, and, as explained above, the output layer has 3 exponential linear activated units. Focusing on the second head, we now have layers with sizes 120, 100, 80, 60, and 40, activated with exponential linear units. We should notice that to achieve good results we had to make heads asymmetric, the σ head has 2 more layers than the $\mu's$, and finally the second head has 3 output units activated with rectified linear functions.

The critic's Network (Figure 27) suffered a more severe increase in the number of layers and units, compared with the flow shop line. It has 7 layers with sizes, 200, 200, 200, 200, 100, 80, and 40, activated with the exponential linear units, and an output linear layer, that offers an estimation for the value of a given state.

The training approach for this version of the control problem was in everything similar to the others, starting with random hyperparameters and tuning them by observing the behavior of the networks in real-time. The group of hyperparameters utilized to successfully train the presented neural networks can be found in Table 7. The training of the actor and the critic networks took around 73 hours of wall time, excluding the time used for debugging the reward function and the algorithm implementation.

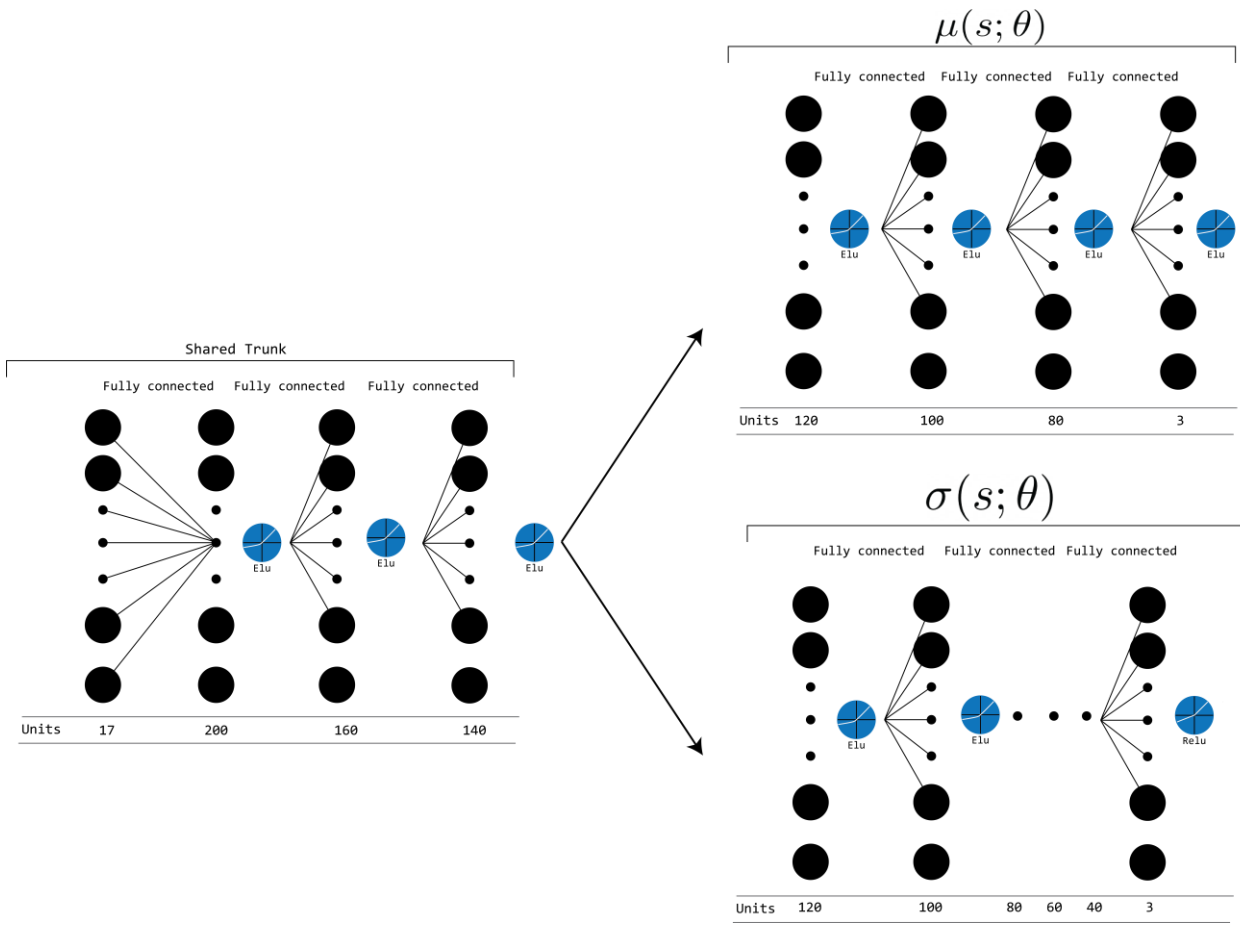


Figure 26- Architecture of Actor's Networks

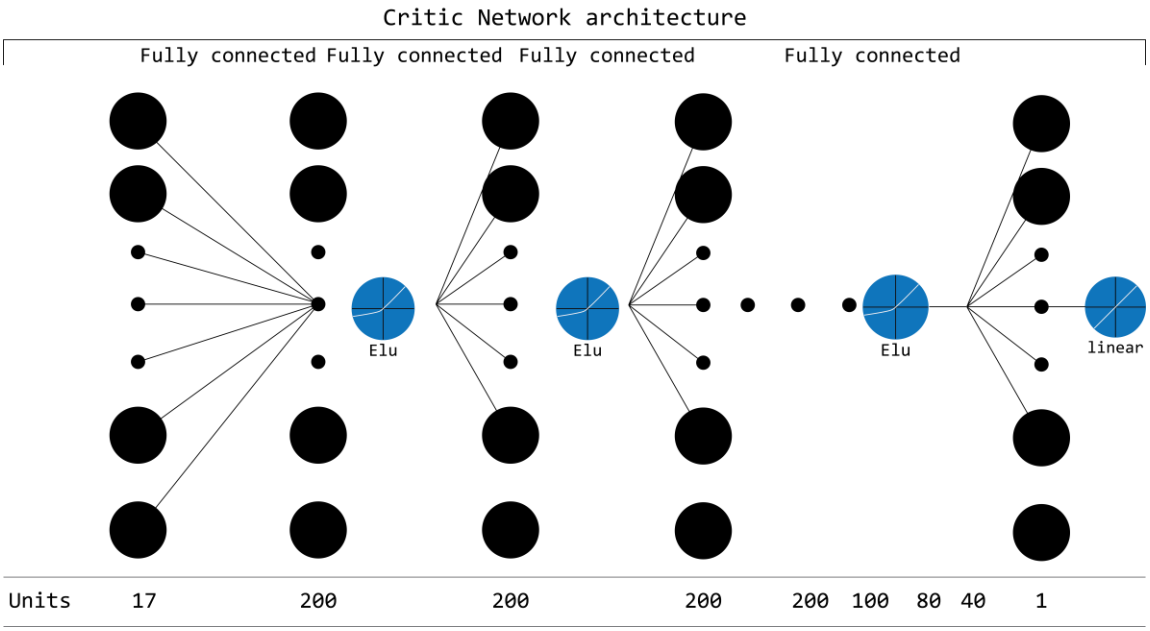


Figure 27- Architecture of Critic's Network

Table 7- Hyperparameters used to train the actor's and the critic networks for the mix-product route system.

Hyperparameter	Value	Description
N2	8	Number of Actors running in separate CPU cores
Episode length	Randint (1100, 2000)	Duration of each simulation in units of time
Critic optimizer	Adam (learning_rate =0.00001)	Optimization algorithm and parameters for critic network
μ optimizer	Adam (learning_rate=0.00001)	Optimization algorithm and parameters for the μ actor-network
σ optimizer	Adam (learning_rate=0.00001)	Optimization algorithm and parameters for the σ actor-network
γ	0.99	Reward discount factor
C	0.8	Entropy coefficient
Gradient Scaling	L2 Norm scaling = 1	After gradients are computed they are scaled to avoid exploding gradients and large updates in the direction of a given batch of samples
ϵ	0.08	
Critic number of step returns (n)	20	
Number of episodes per batch	100	Each actor runs 5 episodes per learning epoch
Actor number of gradient steps per batch (k_1)	10	After an epoch, we compute the loss function, and its gradient, and then update the parameters. We repeat this process k_1 times based on the data from the same epoch,
Critic number of gradient steps per batch (k_2)	5	The same explanation for k_1 serves for k_2

7.5 Results and discussion

In Figure 28 we can observe the comparisons between a pure push system (THmax Agent), our RL Agent's CONWIP system (Model), and a random policy that chooses product type WIP cap levels randomly. We can verify that our agent achieves considerably lower levels of WIP for every production route, with reductions in WIP of 24.4%, 57.7%, and 61.4%, for each route, respectively. There is a new behavior observed for the mix-product production system, we now have a higher number of parts produced in the system controlled with the RL Agent for routes 1 and 3, they show gains in TH of 1.3 % and 1.0%, respectively, when compared to the pure push system. In the single product, flow-shop line, our agent could never achieve larger TH levels than a pure push system for the obvious reason that if a small WIP cap can restrict TH, no WIP cap at all will maximize it. In the particular case of the mix-product production system, with shared resources, we now have the possibility of changing the schedule and the sequence of production, at release time, by prioritizing a product type at the cost of delaying another. This is exactly what we observe, to gain TH in routes 1 and 3, our agent had to delay the production of product type 2, manifesting in a loss of TH for this product type. This shows that our agent, which has good performance indicators does not perfectly respect the production schedule and sequence, and given the stochastic nature of the actions taken by our decision agent, we cannot expect it to strictly do so. What the RL agent tries to do is minimize the deviations from the schedule.

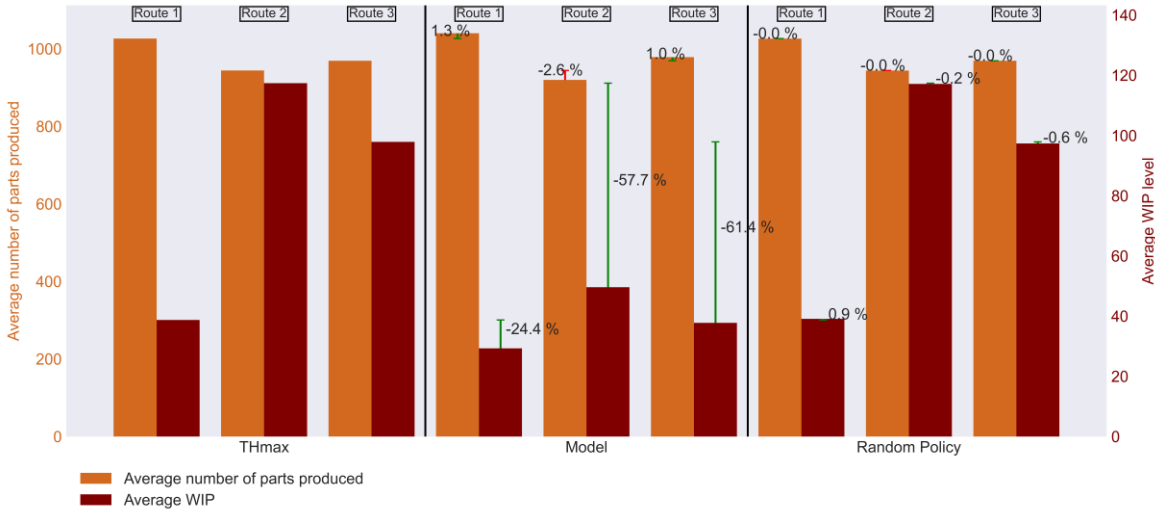


Figure 28- Comparisons between the performances of a pure push system, the trained agent (Model), and a Random Policy

7.5.1 Benchmarking against Statistical Throughput Control and Fixed CONWIP

To compare the STC procedure and the RL approach in the setting of mix-product route systems, we need to use the STC method in a slightly different form from the one it was debuted. Hopp and Roof (1998), when faced with these kinds of systems they tested their procedure, assuming a repeating production sequence. In our mix-product setting, there is no repeating production pattern. We see the production sequence as a FIFO plan for order release, or a result of a more complex scheduling and sequencing mechanism that does not presuppose a repeating pattern of production. What they found is that for each pattern of production, there is an optimal level of WIP for each production route. On the other hand, it is expected, when there is no pattern of production and orders arrive according to a random process, that stable optimal levels of WIP will not be achieved. For the stated reasons, the single-step method will struggle to adapt to a non-repeating pattern of production, as it can only add one unit of WIP at a time for each production route.

As with the case of the simple flowline, an agent is considered to have better performance if it achieves at least the same level of TH with lower levels of WIP, the comparisons should be done at each route. When an agent obtains higher TH than a pure-push system, that excess of parts produced is not counted towards comparisons with other agents, as that is the result of working ahead on a product type at the cost of working behind in another.

Taking this into account our "Model" performs better than any other agent for Routes 1 and 3, only losing by a small margin to the agent with CONWIP fixed at 80 units (Figure 29 and Table 8). The 1.8% excess in parts produced by this agent, is not taken into account.

In Figure 29 and Table 8, for the fixed CONWIP models, WIP caps are set for the overall level of WIP in the system and not, for each specific product type. Choosing different fixed WIP cap levels for each product type would increase the number of simulations needed for comparisons. If we chose to test *n* WIP cap levels for each product type we would need to run n^3 simulations.

We see that the single-step STC Agent harshly reduced the average WIP levels across all production routes but with not-so-good results in achieving the desired TH. This is a similar behavior to what we have already observed on the single product flow-line, with a minor aggravation that can potentially be attributed to the fact of not having a repeating production cycle. The dynamic step variant of STC casts improvements in TH levels over its single-step version. Nonetheless, it cannot reduce WIP as much as the RL Agent while preserving TH. It should be noted that the Dynamic Step variant of the STC agent is reliant, as previously stated, on a congestion model, which was developed under the assumption of exponential random processes. For that reason, the results in Figure 29 were taken from a new version of the random processes of Table 6, where every process was substituted by an exponential process with the same expected value. The RL agent, who was trained under uniform random processes, performed well, even under these new exponential processes (see Appendix F for the hyperparameters used during the training of the RL agent for the mix-product route system.

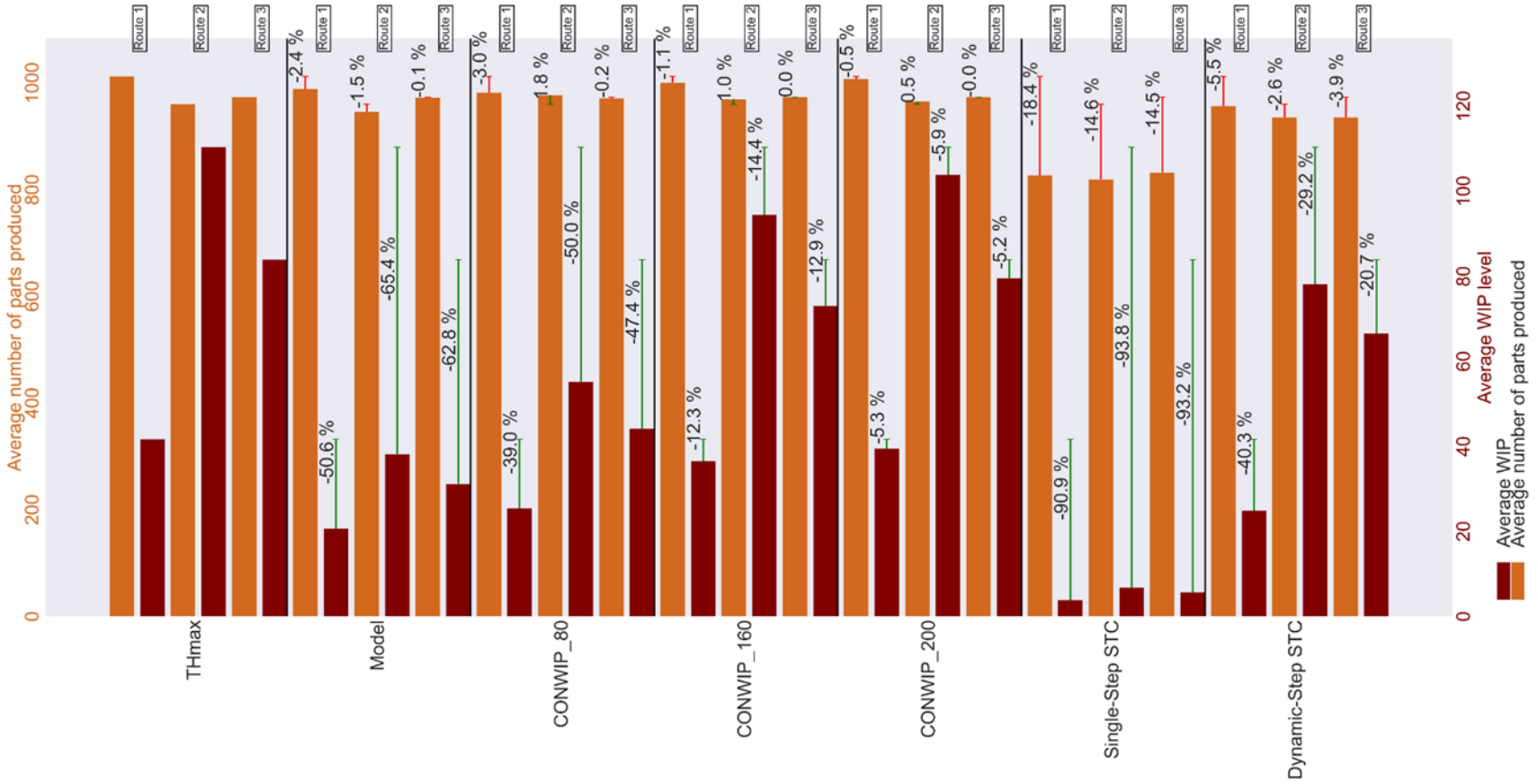


Figure 29- Comparison of RL Agent vs Statistical Throughput Control Agent and Fixed CONWIP configurations, in the setting of a mix-product production system

Table 8- RL Agent vs Statistical Throughput Control Agent and Fixed CONWIP configurations in the setting of a mix-product production system (confidence intervals for α = 0.05)

	Number of parts produced						WIP					
	Route 1		Route 2		Route3		Route 1		Route 2		Route3	
Agent type	$\bar{X}$	CI	$\bar{X}$	CI	$\bar{X}$	CI	$\bar{X}$	CI	$\bar{X}$	CI	$\bar{X}$	CI
THmax	1013.75	13.50	958.15	13.29	974.20	7.35	42.42	6.18	111.25	13.06	83.88	13.89
Model	990.10	11.52	944.50	13.93	973.50	8.80	20.87	1.90	38.45	2.51	31.40	2.45
CONWIP_80	983.20	16.70	976.00	10.82	972.40	7.40	25.78	4.96	55.45	3.45	43.92	5.30
CONWIP_160	1002.15	14.24	967.90	11.31	974.20	7.31	37.12	6.27	94.93	7.48	72.78	10.07
CONWIP_200	1008.45	13.64	963.45	12.28	974.15	7.31	40.14	6.23	104.55	9.93	79.41	11.97
Single-Step STC	825.90	7.15	821.25	9.41	833.10	6.63	3.81	0.41	6.88	0.50	5.80	0.39
Dynamic-Step STC	956.05	17.85	934.75	18.47	931.35	12.31	25.06	5.17	78.97	11.36	67.37	9.27

Chapter 8 Conclusion and future research directions

This chapter summarizes the key findings of our research and discusses the managerial implications of incorporating reinforcement learning techniques into real production systems. We also outline potential avenues for future research on using RL for flow control within pull production

8.1 Conclusion

The deep RL learning method is an effective way of adjusting the number of production authorizations in CONWIP systems, with the goal of maximizing throughput and minimizing WIP levels over a pool of orders. It has demonstrated significantly better performance when compared with similar methods, i.e., methods that are free from assumptions about processing time distributions and other random variables, usually considered when simulating production systems, hence we provided a solution towards **RQ1**. Since we wanted to show how this methodology could be extended to more complex environments as stated in **RQ2** we started with simple flow lines and extended to multi-product systems, with shared resources. The trained decision agent's performance was robust to changes in the random variable distributions of processing and order interarrival times, which seems to indicate that retraining may not be an issue unless major changes in the system occur, such as product suppression or order routing modifications,

With this method, another procedure that runs at a constant time when decisions need to be made is presented, in line with **O1**. Indeed, building this kind of solution is somewhat of a complex problem when compared, for example, with the STC method, hence there is a trade-off between ease of implementation and better performance. This is the primary limitation of these techniques, beyond requiring real-time monitoring of the systems, it adds a new layer of complexity, where we need to maintain a digital twin of a manufacturing process and train an automated decision agent. Benchmarking against the most common WIP parameter setting methods (**O2**) has demonstrated the performance superiority of the proposed method.

To further study the effectiveness of RL approaches, real production system tests are the logical next step. It should be considered that the training of RL agents is done under simulated environments. Therefore, it is reasonable to worry about performance loss when the agent is tested against real environments. Ergo, the importance of showing performance robustness under data shift circumstances as we have shown in this research.

8.2 Managerial implications and barriers to adoption

Most pull methods have as assumptions stable and well-balanced manufacturing systems; low variabilities are one of the core requirements for those methods. We provide a technology that makes enterprises more flexible and able to work under higher variabilities without resorting to the most common safety net – higher levels of WIP. We showed, through the use of simulations, that it is possible to drastically reduce WIP levels with the right parameter setting.

Regarding implementation, the approach can be adapted easily to most pull methods that control WIP and execute from a plan, without changing optimization goals. Nevertheless, these kinds of approaches come with their subset of requirements. Our approach relies on the fact that to adopt this kind of self-adapting pull strategy, industry 4.0 related technologies should already be a reality, namely the real-time tracking of system state and automatic system actuation (Lasi et al., 2014). The aforementioned, compounded by the fact that this is not a solution fits all kind of problem, may present itself as a barrier to entry on the adoption of this kind of technology. This approach requires the development and validation of a digital twin, training of a decision agent, and possible retraining efforts if systems undergo major operational changes regarding demand variability, overall process stability, and re-routing arrangements, even though, as we have shown, the learned policies are rather robust. The prospects of training an RL agent from scratch, online, i.e., interacting directly with a real production system, seems, at the time, unrealistic. An Agent needs a lot of exploration to learn, and that exploration can happen in regions of the state space with poor performance. In any case, online learning may have a role in the training of the RL Agent. Once the system starts having good performance under the simulated environment, we can put it online. If simulated performance translates into real performance, the rest of the learning can be done online as well, interacting directly with the environment, enabling continuous improvement, as systems change. We can also control the exploration parameter while the agent is learning online so that policies do not change drastically and end up in system states with lower performance. Since the agent's performance is being tracked in real-time, if for some reason performance deteriorates significantly – TH drops or WIP rises abruptly – by looking at system state, namely, demand interarrival-time, workstations processing times and utilization, we can identify if the agent is the culprit. For example, if interarrival times do not change, and we suddenly

observe a huge loss in TH, by looking at workstation utilization, we can check for low WIP induced process starvation.

By far, the largest obstacle to the dissemination of these approaches is the fact that we are still far away from the commoditization of artificial intelligence technologies and their approaches to production and operations management issues. This is another case of the intersection between information systems and operations management, as pointed out by Kumar, Mookerjee, and Shubham (2018), and in these circumstances, adoption is heavily dependent on how new information systems integrate with existing ones. Otherwise, organizations will have a set of compartmentalized blocks of software that are hard to manage and maintain. This is where commoditization is crucial, most likely, only when AI solutions start being offered as part of services provided by large software providers, for example, ERP vendors, will we see practical applications of approaches similar to the one presented. Commoditization will allow similar approaches that once required dedicated teams of data scientists to be purchased and consumed on-demand as a service.

8.3 Future work: extending the presented method to other pull strategies

As stated, CONWIP, due to its simplicity, is the perfect test bed for this new approach, nonetheless, this method can be adapted to any pull policy, where WIP level limits need to be prescribed. For example, in Kanban, instead of determining one card count per route, we would output as many card counts as there are workstations on that same route, the goal here would also be shifted from maximizing TH to maximizing fill-rate, as now we would be fulfilling orders from finished goods inventory.

Another class of methods that could potentially reap the benefits of this new approach is Workload Control (WLC). WLC benefits stem from the idea of keeping the workload of stations stable. If a load of a station is stable, it is possible to estimate with more accuracy its cycle-time and consequently the entire process cycle-time for a given product, which may inform the due date quoting process. The loads are measured in time units and can be estimated using different methods (J. W. M. Bertrand & Wortmann, 1981; L. C. Hendry & Kingsman, 1989; L. Hendry & Kingsman, 1991; Kingsman et al., 1989), nevertheless, the goal at the decision time is to know what is the potential contribution that an order has on the workloads of the stations on its route. The decision on whereas a release should occur is dependent on workloads not exceeding a predefined limit – norm – on the workstations affected by the potential release. As we can see, WLC techniques limit WIP, not by establishing a WIP cap but by means of establishing workload norms. This is an implicit control of WIP.

Many authors have emphasized the importance of determining the right workload norms (L. C. Hendry et al., 1998; Land & Gaalman, 1998). Nevertheless, there is not much research focusing on this particular problem (Breithaupt et al., 2002; see eg., Thürer et al., 2011; Zäpfel & Missbauer, 1993b)

Determining optimal workload norms for each workstation would be the subject of control in the setting of the reinforcement learning approach, and a reward structure that incentivizes TH, whilst minimizing WIP could be a good approach to solve this problem.

As is the case for WIP control methods, we think that this new dynamic parameter setting method could potentially catalyze the implementation of WLC (Thürer et al., 2012) and its card-based techniques such as COBACABANA and its variants (Land, 2009; Thürer et al., 2020).

References

Ajorlou, S., & Shams, I. (2013). Artificial bee colony algorithm for CONWIP production control system in a multi-product multi-machine manufacturing environment. *Journal of Intelligent Manufacturing*, *24*(6), 1145–1156. https://doi.org/10.1007/s10845-012-0646-5

Almada-Lobo, B., Clark, A., Guimarães, L., Figueira, G., & Amorim, P. (2015). Industrial Insights Into Lot Sizing And Scheduling Modeling. *Pesquisa Operacional*, *35*(3), 439–464. https://doi.org/10.1590/0101-7438.2015.035.03.0439

Arredondo, F., & Martinez, E. (2010). Learning and adaptation of a policy for dynamic order acceptance in make-to-order manufacturing. *Computers & Industrial Engineering*, *58*(1), 70–83. https://doi.org/10.1016/J.CIE.2009.08.005

Bae, K.-H., Feng, B., Kim, S., Lazarova-Molnar, S., Zheng, Z., Roeder, T., Thiesing, R., Gyulai, D., Bergmann, J., Lengyel, A., Kádár, B., & Czirkó, D. (2020). Simulation-Based Digital Twin of a Complex Shop-Floor Logistics System. *Proceedings of the 2020 Winter Simulation Conference*.

Bagni, G., Godinho Filho, M., Thürer, M., & Stevenson, M. (2021). Systematic review and discussion of production control systems that emerged between 1999 and 2018. *Production Planning and Control*, *32*(7), 511–525. https://doi.org/10.1080/09537287.2020.1742398

Banks, J. (1998). Handbook of Simulation. In *Handbook of Simulation*. John Wiley & Sons, Inc. https://doi.org/10.1002/9780470172445

Bard, J. F., & Golany, B. (1991). Determining the number of kanbans in a multiproduct, multistage production system. *International Journal of Production Research*, *29*(5), 881–895. https://doi.org/10.1080/00207549108930108

Beche, R., & Nedevschi, S. (2020). Narrowing the semantic gap between real and synthetic data. *Proceedings - 2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing, ICCP 2020*, 361–367. https://doi.org/10.1109/ICCP51029.2020.9266246

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., Pinto, H. P. d. O., Raiman, J., Salimans, T., Schlatter, J., … Zhang, S. (2019). *Dota 2 with Large Scale Deep Reinforcement Learning*. http://arxiv.org/abs/1912.06680

Berry, W. L., Schmitt CPIM, T. G., & Vollmann, T. E. (1982). Capacity planning techniques for manufacturing control systems: Information requirements and operational features. *Journal of Operations Management*, *3*(1), 13–25. https://doi.org/10.1016/0272-6963(82)90018-3

Bertolini, M., Mezzogori, D., Neroni, M., & Zammori, F. (2021). Machine Learning for industrial applications: A comprehensive literature review. In *Expert Systems with Applications* (Vol. 175, p. 114820). Elsevier Ltd. https://doi.org/10.1016/j.eswa.2021.114820

Bertrand, J., & Fransoo, J. (2002). Operations management research methodologies using quantitative modeling. *International Journal of Operations and Production Management*, *22*(2), 241–264. https://doi.org/10.1108/01443570210414338

Bertrand, J. W. M., & Wortmann, J. C. (1981). *Production Control and Information Systems for Component-manufacturing Shops* . https://books.google.pt/books/about/Production_Control_and_Information_Syste.html?id=IdBTAAAAMAAJ&redir_esc=y

Breithaupt, J. W., Land, M., & Nyhuis, P. (2002). The workload control concept: Theory and practical extensions of load oriented order release. *Production Planning and Control*, *13*(7), 625–638. https://doi.org/10.1080/0953728021000026230

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020). Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, *33*, 1877–1901. https://commoncrawl.org/the-data/

Cao, H., Xi, H., & Smith, S. F. (2003). A reinforcement learning approach to production planning in the fabrication/fulfillment manufacturing process. *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693)*, 1417–1423. https://doi.org/10.1109/WSC.2003.1261584

Chunming Liu, Xin Xu, & Dewen Hu. (2015). Multiobjective Reinforcement Learning: A Comprehensive Overview. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *45*(3), 385–398. https://doi.org/10.1109/TSMC.2014.2358639

Creighton, D. C., & Nahavandi, S. (2002). The application of a reinforcement learning agent to a multi-product manufacturing facility. *2002 IEEE International Conference on Industrial Technology, 2002. IEEE ICIT '02.*, 1229–1234. https://doi.org/10.1109/ICIT.2002.1189350

CryptGenRandom function (wincrypt.h) - Win32 apps | Microsoft Docs. (2018). https://docs.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptgenrandom

Deleersnyder, J.-L., Hodgson, T. J., Muller-Malek, H., & O'Grady, P. J. (1989). Kanban Controlled Pull Systems: An Analytic Approach. *Management Science, 35*(9), 1079–1091. https://doi.org/10.1287/mnsc.35.9.1079

Dranidis, D., & Kehris, E. (2001). A production scheduling strategy for an assembly plant based on reinforcement learning. In *Proceedings of the 5th World MultiConference on Circuits, Systems, Communications & Computers, Crete*.

Fan, F., & Wang, G. (2018). Learning from Pseudo-Randomness with an Artificial Neural Network-Does God Play Pseudo-Dice? *IEEE Access*, *6*, 22987–22992. https://doi.org/10.1109/ACCESS.2018.2826448

Feng, Y., & Hao, L. (2020). Testing randomness using artificial neural network. *IEEE Access*, *8*, 163685–163693. https://doi.org/10.1109/ACCESS.2020.3022098

Fernandes, N. O., & do Carmo-Silva, S. (2006). Generic POLCA-A production and materials flow control mechanism for quick response manufacturing. *International Journal of Production Economics*, *104*(1), 74–84. https://doi.org/10.1016/j.ijpe.2005.07.003

Fry, T., & Smith, A. (1987). A procedure for implementing input/output control: a case study. *Production and Inventory Management*, *28*(3), 50–52.

Godfrey-Smith, P. (2009). *Theory and Reality: An Introduction to the Philosophy of Science*. University of Chicago Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial networks. *Proceedings of the International Conference on Neural Information Processing Systems*. https://doi.org/10.1145/3422622

Gupta, S. M., & Al-Turki, Y. A. Y. (1997, January 1). An algorithm to dynamically adjust the number of kanbans in stochastic processing times and variable demand environment. *Production Planning and Control*, *8*(2), 133–141. https://doi.org/10.1080/095372897235398

Hendry, L. C., & Kingsman, B. G. (1989). Production planning systems and their applicability to make-to-order companies. *European Journal of Operational Research*, *40*(1), 1–15. https://doi.org/https://doi.org/10.1016/0377-2217(89)90266-X

Hendry, L. C., Kingsman, B. G., & Cheung, P. (1998). The effect of workload control (WLC) on performance in make-to-order companies. *Journal of Operations Management*, *16*(1), 63–75. https://doi.org/10.1016/S0272-6963(97)00011-9

Hendry, L., & Kingsman, B. (1991). A Decision Support System for Job Release in Make-to-order Companies. *International Journal of Operations & Production Management*, *11*(6), 6–16. https://doi.org/10.1108/01443579110144655

Hopp, W. J., & Spearman, M. L. (2004). To pull or not to pull: What is the question? *Manufacturing and Service Operations Management*, *6*(2), 133–148. https://doi.org/10.1287/msom.1030.0028

Hopp, W. J., & Spearman, M. L. (2011). *Factory physics*.

Hopp, Wallace. J., & Roof, Mark. L. (1998). Setting WIP levels with statistical throughput control (STC) in CONWIP production lines. *International Journal of Production Research*, *36*(4), 867–882. https://doi.org/i

Huang, X., Liu, M.-Y., Belongie, S., & Kautz, J. (2018). Multimodal Unsupervised Image-to-Image Translation. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *11207 LNCS*, 179–196. http://arxiv.org/abs/1804.04732

Ip, W. H., Huang, M., Yung, K. L., Wang, D., & Wang, X. (2007). CONWIP based control of a lamp assembly production line. *Journal of Intelligent Manufacturing*, *18*(2), 261–271. https://doi.org/10.1007/s10845-007-0021-0

Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, *2017-January*, 5967–5976. https://doi.org/10.1109/CVPR.2017.632

James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., & Bousmalis, K. (2019). Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2019-June*, 12619–12629. https://doi.org/10.1109/CVPR.2019.01291

Jodlbauer, H. (2008). A time-continuous analytic production model for service level, work in process, lead time and utilization. *International Journal of Production Research*, *46*(7), 1723–1744. https://doi.org/10.1080/00207540601080498

Karni, R. (1982). Capacity requirements planning— a systematization. *THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH*, *20*(6), 715–739. https://doi.org/10.1080/00207548208947800

Kingsman, B. G., Tatsiopoulos, I. P., & Hendry, L. C. (1989). A structural methodology for managing manufacturing lead times in make-to-order companies. *European Journal of Operational Research*, *40*(2), 196–209. https://doi.org/https://doi.org/10.1016/0377-2217(89)90330-5

Kumar, S., Mookerjee, V., & Shubham, A. (2018). Research in Operations Management and Information Systems Interface. *Production and Operations Management*, *27*(11), 1893–1905. https://doi.org/10.1111/poms.12961

Land, M. J. (2009). Cobacabana (control of balance by card-based navigation): A card-based system for job shop control. *International Journal of Production Economics*, *117*(1), 97–103. https://doi.org/10.1016/j.ijpe.2008.08.057

Land, M. J., & Gaalman, G. J. C. (1998). The performance of workload control concepts in job shops: Improving the release method. *International Journal of Production Economics*, *56–57*, 347–364. https://doi.org/10.1016/S0925-5273(98)00052-8

Lang, T., Williams, E. J., & Ülgen, O. M. (2008). Simulation improves manufacture and material handling of forged metal components. *Proceedings - 22nd European Conference on Modelling and Simulation, ECMS 2008*, 247–253. https://doi.org/10.7148/2008-0247

Lasi, H., Fettke, P., Kemper, H. G., Feld, T., & Hoffmann, M. (2014). Industry 4.0. *Business and Information Systems Engineering*, *6*(4), 239–242. https://doi.org/10.1007/s12599-014-0334-4

Law, A. (2014). *Simulation Modeling and Analysis* (5th ed.). McGraw-Hill Education. https://www.amazon.com/Simulation-Mcgraw-hill-Industrial-Engineering-Management/dp/0073401323

L'Ecuyer, P. (1994). Uniform random number generation. *Annals of Operations Research*, *53*(1), 77–120. https://doi.org/10.1007/BF02136827

L'ecuyer, P. (1998). Random Number Generation. In Jerry. Banks (Ed.), *Handbook of Simulation*.

L'ecuyer, P., & Simard, R. (2007). TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, *33*(4). https://doi.org/10.1145/1268776.1268777

Li, A., & Co, H. C. (1991). A dynamic programming model for the kanban assignment problem in a multistage multiperiod production system. *International Journal of Production Research*, *29*(1), 1–16. https://doi.org/10.1080/00207549108930045

Little, J. D. C., & C., J. D. (1961). A Proof for the Queuing Formula: $L = \lambda W$. *Operations Research*, *9*(3), 383–387. https://doi.org/10.1287/opre.9.3.383

Lödding, H., Yu, K. W., & Wiendahl, H. P. (2003). Decentralized WIP-oriented manufacturing control (DEWIP). *Production Planning and Control*, *14*(1), 42–54. https://doi.org/10.1080/0953728021000078701

Maclaren, N. M. (1992). A limit on the usable length of a pseudorandom sequence. *Journal of Statistical Computation and Simulation*, *42*(1–2), 47–54. https://doi.org/10.1080/00949659208811409

Marsaglia, G., & Tsang, W. W. (2002). Some difficult-to-pass tests of randomness. *Journal of Statistical Software*, *7*, 1–9. https://doi.org/10.18637/jss.v007.i03

Matsumoto, M., & Nishimura, T. (1998). Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Transactions on Modeling and Computer Simulation*, *8*(1), 3–30. https://doi.org/10.1145/272991.272995

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). *Asynchronous Methods for Deep Reinforcement Learning*. http://arxiv.org/abs/1602.01783

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236

Moeeni, F., & Chang, Y. -L. (1990). An Approximate Solution to Deterministic Kanban Systems. *Decision Sciences*, *21*(3), 596–607. https://doi.org/10.1111/j.1540-5915.1990.tb00337.x

Monden, Y. (1983). *Toyota Production System: Practical Approach to Production Management* (Industrial Engineeering and Management Press, Ed.; 1st ed.). https://www.amazon.com/Toyota-Production-System-Practical-Management/dp/0898060346

Muratore, F., Gienger, M., & Peters, J. (2021). Assessing Transferability from Simulation to Reality for Reinforcement Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *43*(4), 1172–1183. https://doi.org/10.1109/TPAMI.2019.2952353

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., & Liang, E. (2006). Autonomous inverted helicopter flight via reinforcement earning. *Springer Tracts in Advanced Robotics*, *21*, 363–372. https://doi.org/10.1007/11552246_35

Nguyen, T. T. (2018). *A Multi-Objective Deep Reinforcement Learning Framework*. http://arxiv.org/abs/1803.02965

Ohno, T. (1988). *Toyota Production System - beyond large-scale production*. Productivity Press.

Orlicky, Joseph., & Plossl, George. (1994). *Orlicky's material requirements planning*. McGraw-Hill. https://www.goodreads.com/book/show/969801.Orlicky_s_Material_Requirements_Planning

Pegden, D., Shannon, R., & Sadowski, R. (1995). *Introduction to Simulation Using Siman*. McGraw-Hill College. https://www.amazon.com/Introduction-Simulation-Using-Dennis-Pegden/dp/0070493200

Peng, X. Bin, Andrychowicz, M., Zaremba, W., & Abbeel, P. (2018). Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. *Proceedings - IEEE International Conference on Robotics and Automation*, 3803–3810. https://doi.org/10.1109/ICRA.2018.8460528

Popper, K. (1963). *Conjectures and Refutations: The Growth of Scientific Knowledge*. Routledge. https://www.routledge.com/Conjectures-and-Refutations-The-Growth-of-Scientific-Knowledge/Popper/p/book/9780415285940

Qu, Z., Su, L., Wang, X., Zheng, S., Song, X., & Song, X. (2018). A Unsupervised Learning Method of Anomaly Detection Using GRU. *Proceedings - 2018 IEEE International Conference on Big Data and Smart Computing, BigComp 2018*, 685–688. https://doi.org/10.1109/BigComp.2018.00126

Rechenberg, I., & Eigen, M. (1973). *Evolutionsstrategie. Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. https://www.jstor.org/stable/23679080

Risi, S., & Togelius, J. (2015). Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, *PP*(99). https://doi.org/10.1109/TCIAIG.2015.2494596

Ryan, S. M., Baynat, B., & Choobineh, F. (2000). Determining inventory levels in a CONWIP controlled job shop. *IIE Transactions (Institute of Industrial Engineers)*, *32*(2), 105–114. https://doi.org/10.1080/07408170008963883

Sadeghi, F., & Levine, S. (2017). CAD2RL: Real single-image flight without a single real image. *Robotics: Science and Systems*, *13*. https://doi.org/10.15607/rss.2017.xiii.034

Salimans, T., Ho, J., Chen, X., Sidor, S., & Sutskever, I. (2017). *Evolution Strategies as a Scalable Alternative to Reinforcement Learning*. http://arxiv.org/abs/1703.03864

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal Policy Optimization Algorithms*. http://arxiv.org/abs/1707.06347

Shannon, C. E. (1948). A Mathematical Theory of Communication. *The Bell System Technical Journal*, *27*, 623–656.

Shin, M., Ryu, K., & Jung, M. (2012). Reinforcement learning approach to goal-regulation in a self-evolutionary manufacturing system. *Expert Systems with Applications*, *39*(10), 8736–8743. https://doi.org/10.1016/J.ESWA.2012.01.207

Shingo, S. (1985). *A Revolution in Manufacturing: The SMED System*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, *550*, 354. http://dx.doi.org/10.1038/nature24270

Spearman, M. L. (1991). Analytic congestion model for closed production systems with IFR processing times. *Management Science*, *37*(8), 1015–1029. https://doi.org/10.1287/mnsc.37.8.1015

Spearman, M. L., Woodruff, D. L., & Hopp, W. J. (1990). CONWIP: a pull alternative to kanban. *International Journal of Production Research*, *28*(5), 879–894. https://doi.org/10.1080/00207549008942761

Spearman, M. L., & Zazanis, M. A. (1992a). 1. *Operations Research*, *40*(3), 521–532. https://doi.org/10.1287/opre.40.3.521

Spearman, M. L., & Zazanis, M. A. (1992b). Push and pull production systems. Issues and comparisons. *Operations Research*, *40*(3), 521–532. https://doi.org/10.1287/opre.40.3.521

Sugimori, Y., Kusonoki, K., Cho, F., & Uchikawa, S. (1977). Toyota production system and Kanban system Materialization of just-in-time and respect-for-human system. *International Journal of Production Research*, *15*(6), 553–564. https://doi.org/10.1080/00207547708943149

Sundermeyer, M., Marton, Z. C., Durner, M., Brucker, M., & Triebel, R. (2018). Implicit 3D orientation learning for 6D object detection from RGB images. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *11210 LNCS*, 712–729. https://doi.org/10.1007/978-3-030-01231-1_43

Suri, R. (1998). Chapter Nine: POLCA -- The New Material Control and Replenishment System for QRM. In *Quick Response Manufacturing A Companywide Approach to Reducing Lead Times*. https://www.routledge.com/Quick-Response-Manufacturing-A-Companywide-Approach-to-Reducing-Lead-Times/Suri/p/book/9781563272011

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning : an introduction.* MIT PRESS. https://mitpress.mit.edu/books/reinforcement-learning-second-edition

Sutton, R. S., Mcallester, D., Singh, S., & Mansour, Y. (2000). *Policy Gradient Methods for Reinforcement Learning with Function Approximation*.

Takahashi, K., & Nakamura, N. (1998). Ordering alternatives in JIT production systems. *Production Planning and Control*, *9*(8), 784–794. https://doi.org/10.1080/095372898233551

Tardif, V., & Maaseidvaag, L. (2001). An adaptive approach to controlling kanban systems. *European Journal of Operational Research*, *132*(2), 411–424. https://doi.org/10.1016/S0377-2217(00)00119-3

Tatsiopoulos, I. P. (1983). *A microcomputer-based interactive system for managing production and marketing in small component manufacturing firms using a hierarchical backlog control and lead time management technology*. Lancaster University.

Team Simpy. (2023). *Overview — SimPy 4.1.1 documentation*. https://simpy.readthedocs.io/en/latest/

Thürer, M., Fernandes, N. O., & Stevenson, M. (2020). Material Flow Control in High-Variety Make-to-Order Shops: Combining COBACABANA and POLCA. *Production and Operations Management*, *29*(9), 2138–2152. https://doi.org/10.1111/poms.13218

Thürer, M., Silva, C., & Stevenson, M. (2011). Optimising workload norms: The influence of shop floor characteristics on setting workload norms for the workload control concept. *International Journal of Production Research*, *49*(4), 1151–1171. https://doi.org/10.1080/00207541003604836

Thürer, M., Stevenson, M., & Protzman, C. W. (2015). COBACABANA (Control of Balance by Card Based Navigation): An alternative to kanban in the pure flow shop? *International Journal of Production Economics*, *166*, 143–151. https://doi.org/10.1016/j.ijpe.2015.05.010

Thürer, M., Stevenson, M., Silva, C., Land, M. J., & Fredendall, L. D. (2012). Workload Control and Order Release: A Lean Solution for Make-to-Order Companies. *Production and Operations Management*, *21*(5), 939–953. https://doi.org/10.1111/j.1937-5956.2011.01307.x

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., & Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. *IEEE International Conference on Intelligent Robots and Systems*, *2017-September*, 23–30. https://doi.org/10.1109/IROS.2017.8202133

Tremblay, J., Prakash, A., Acuna, D., Brophy, M., Jampani, V., Anil, C., To, T., Cameracci, E., Boochoon, S., & Birchfield, S. (2018). Training deep networks with synthetic data: Bridging the reality gap by domain randomization. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, *2018-June*, 1082–1090. https://doi.org/10.1109/CVPRW.2018.00143

Tsitsiklis, J. N., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, *42*. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.15.3298

Van Baar, J., Sullivan, A., Cordorel, R., Jha, D., Romeres, D., & Nikovski, D. (2019). Sim-to-real transfer learning using robustified controllers in robotic tasks involving complex dynamics. *Proceedings - IEEE International Conference on Robotics and Automation*, *2019-May*, 6001–6007. https://doi.org/10.1109/ICRA.2019.8793561

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., … Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, *575*(7782), 350–354. https://doi.org/10.1038/s41586-019-1724-z

Wang, H., & Hsu-Pin (Ben) Wang. (1991). Optimum number of kanbans between two adjacent workstations in a JIT system. *International Journal of Production Economics*, *22*(3), 179–188. https://doi.org/10.1016/0925-5273(91)90093-9

Wang, J., Li, X., & Zhu, X. (2012). Intelligent dynamic control of stochastic economic lot scheduling by agent-based reinforcement learning. *International Journal of Production Research*, *50*(16), 4381–4395. https://doi.org/10.1080/00207543.2011.592158

Wang, L., & Prabhu, V. (2006). Parallel algorithm for setting WIP levels for multi-product CONWIP systems. *International Journal of Production Research*, *44*(21), 4681–4693. https://doi.org/10.1080/00207540500490970

Wang, T. C., Liu, M. Y., Zhu, J. Y., Tao, A., Kautz, J., & Catanzaro, B. (2018). High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. *European Conference on Computer Vision*, 8798–8807. https://doi.org/10.1109/CVPR.2018.00917

Watkins, C. J. C. H., & Dayan, P. (1992). Technical Note: Q-Learning. *Machine Learning*, *8*(3), 279–292. https://doi.org/10.1023/A:1022676722315

Wight, O. W. (1970). Input/output control: a real handle on lead times. *Production and Inventory Management*, *11*, 9–10.

Williams, E. J., & Ülgen, O. M. (2013). Simulation applications in the automotive industry. In *Use Cases of Discrete Event Simulation: Appliance and Research* (Vol. 9783642287770, pp. 45–58). Springer-Verlag Berlin Heidelberg. https://doi.org/10.1007/978-3-642-28777-0_3

Williams, R. J. (1987). Reinforcement learning in connectionist networks. *Technical Report ICS 8605*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3–4), 229–256. https://doi.org/10.1007/BF00992696

Xanthopoulos, A. S., Chnitidis, G., & Koulouriotis, D. E. (2019). Reinforcement learning-based adaptive production control of pull manufacturing systems. *Journal of Industrial and Production Engineering*, *36*(5), 313–323. https://doi.org/10.1080/21681015.2019.1647301

Yan, H., Lou, S., & Sethi, S. P. (2000). Robustness of Various Production Control Policies in Semiconductor Manufacturing. *Production and Operations Management*, *9*(2), 171–183. https://doi.org/10.1111/j.1937-5956.2000.tb00332.x

Yue, X., Zhang, Y., Zhao, S., Sangiovanni-Vincentelli, A., Keutzer, K., & Gong, B. (2019). Domain randomization and pyramid consistency: Simulation-to-real generalization without accessing target domain data. *Proceedings of the IEEE International Conference on Computer Vision*, *2019-October*, 2100–2110. https://doi.org/10.1109/ICCV.2019.00219

Zäpfel, G., & Missbauer, H. (1993a). Production Planning and Control (PPC) systems including load-oriented order release - Problems and research perspectives. *International Journal of Production Economics*, *30–31*(C), 107–122. https://doi.org/10.1016/0925-5273(93)90085-Y

Zhao, W., Queralta, J. P., & Westerlund, T. (2020). Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: A Survey. *2020 IEEE Symposium Series on Computational Intelligence, SSCI 2020*, 737–744. https://doi.org/10.1109/SSCI47803.2020.9308468

Appendix A - Order due dates for section 5.2 demonstration

[118.32960547688907,
128.5782503398625,
138.81673344703108,
149.04620736909615,
159.26762198957215,
169.4817711518165,
179.68932634876447,
189.89086160036388,
200.08687218223045,
210.27778896687965,
220.46398957194944,
230.64580714356867,
240.8235373605483,
250.9974440810001,
261.1677639397706,
271.3347101255548,
281.49847550979376,
291.6592352583502,
301.81714902677317,
311.972362817535,
322.12501056075945,
332.27521546715184,
342.4230911920138,
352.56874284161665,
362.7122678472672,
372.8537567277248,
382.9932937569204,
393.13095755096913,
403.26682158608764,
413.4009546571047,
423.5334212846861,
433.66428207811646,
443.79359405942574,
453.92141095377815,
464.04778345031946,
474.17275943707426,
484.2963842129807,
494.418700679725,
504.5397495156789,
514.6595693339398,
524.7781968262138,
534.8956668940622,
545.0120127688405,
555.1272661215002,
565.2414571632796,
575.3546147381924,
585.4667664081143,
595.5779385311815,
605.688156334131,
615.7974439791443,
625.9058246256997,
636.013320487878,
646.1199528875253,
656.2257423036331,
666.3307084182586,
676.4348701592786,
686.5382457402372,
696.640852697529,
706.7427079251275,
716.8438277070596,
726.9442277477966,
737.0439232007277,
747.1429286948585,
757.2412583598694,
767.3389258496562,
777.4359443644609,
787.5323266716974,
797.6280851255642,
807.7232316855283,
817.8177779337593,
827.9117350915872,
838.0051140350467,
848.0979253095724,
858.1901791438988,
868.2818854632183,
878.3730539016437,
888.4636938140221,
898.5538142871374,
908.6434241503428,
918.732531985657,
928.8211461373569,
938.9092747210966,
948.9969256325825,
959.0841065558288,
969.17082497102,
979.2570881620002,
989.3429032234139,
999.4282770675156,
1009.5132164306673,
1019.5977278795413,
1029.6818178170458,
1039.7654924879855,
1049.8487579844739,
1059.9316202511095,
1070.0140850899288,
1080.0961581651472,
1090.177845007699,
1100.2591510195875,
1110.3400814780537,
1120.4206415395736]

Appendix B - Order release schedule for section 5.2 demonstration

[0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1.7171490267731429,
11.872362817534963,
22.025010560759426,
32.175215467151816,
42.323091192013806,
52.46874284161663,
62.61226784726716,
72.75375672772475,
82.8932937569204,
93.0309575509691,
103.16682158608762,
113.30095465710468,
123.43342128468606,
133.56428207811643,
143.69359405942572,
153.82141095377813,
163.94778345031943,
174.07275943707424,
184.19638421298066,
194.318700679725,
204.4397495156789,
214.55956933393975,
224.67819682621382,
234.79566689406215,
244.91201276884044,
255.02726612150013,
265.1414571632796,
275.25461473819234,
285.36676640811424,
295.4779385311815,
305.588156334131,
315.6974439791443,
325.80582462569964,
335.91332048787797,
346.0199528875253,
356.12574230363305,
366.2307084182586,
376.3348701592786,
386.4382457402372,
396.540852697529,
406.6427079251275,
416.7438277070596,
426.8442277477966,
436.9439232007277,
447.04292869485846,
457.1412583598694,
467.2389258496562,
477.33594436446083,
487.4323266716974,
497.5280851255642,
507.62323168552825,
517.7177779337593,
527.8117350915871,
537.9051140350467,
547.9979253095723,
558.0901791438988,
568.1818854632182,
578.2730539016437,
588.3636938140221,
598.4538142871373,
608.5434241503427,
618.632531985657,
628.7211461373569,
638.8092747210966,
648.8969256325825,
658.9841065558288,
669.07082497102,
679.1570881620002,
689.2429032234139,
699.3282770675156,
709.4132164306673,
719.4977278795412,
729.5818178170458,
739.6654924879855,
749.7487579844739,
759.8316202511095,
769.9140850899288,
779.9961581651472,
790.077845007699,
800.1591510195875,
810.2400814780536,
820.3206415395736]

Appendix C – Code repositories for results reproduction

Chapter 6 code:

- https://github.com/TomeASilva/PPO_Production_System
- https://github.com/TomeASilva/PPO-Vs-PPO-Agent-learning-framework-to-control-Production-Flow

Chapter 7 code: https://github.com/TomeASilva/PPO_production_system_routes