UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Degree course:

Master's Degree in Management Engineering

Student:

Daniel Lijia Hu

TFM title:

Study for the design of a management system for AGV networks

TFM Director:

Antoni Guasch Petit

Expedition date:

June 2019

Declaration of Honor

I declare that, the work in this Master is completely my own work, no part of this Master Thesis is taken from other people's work without giving them credit, and all references have been clearly cited.

I understand that an infringement of this declaration leaves me subject to the foreseen disciplinary actions by *The Universitat Politècnica de Catalunya - BarcelonaTECH.*

Abstract

Automated Guided Vehicles are a vital part of the future intelligent manufacturing processes. In order to make the better profit, it is important to study if deadlocks can occur and how to tackle them. In this project we demonstrate how Petri Net models, which are perfect for representing deadlocks, can be mapped in the simulation software FlexSim. Eventually, we are using this software in order to evaluate different study cases with deadlocks.

Acknowledgments

I want to thank the director of this thesis, Toni Guasch, for giving me this idea in which I could develop this subject. This work could not been possible without his valuable advices, teachings and guides. It was a really interesting insight in a fascinating new field.

I am also grateful to the Automatic Control department, for the confidence you have expressed in me by using little pieces of my work for the Winter Simulation Multiconference'19.

Thanks all for trusting me.

Table of Contents

List of figures:

List of tables:

1. Structure

The motivation for the existence of this project is addressed in this section, marking the main points and structure of the memory, with the intention of fixing a clear, concise and ordered system to pursue its goals and expectations.

1.1. Objectives

Deadlocks are an important issue to be addressed in complex applications as flexible manufacturing systems, since they might result in system failures or blocking. The solution for deadlocks is far from being general, as each case has different optimal solutions. In order to study these situations, Petri Nets and simulation software are used together to analyze and try to find solutions to this problem.

The main goal of this project is to analyze and propose solutions to the deadlock problem using the FlexSim simulation software using AGV networks as the target system of the analysis.

Additionally, one secondary objective that is addressed in this document is the observation of parallelisms between FlexSim and Arena (other simulation software that is currently used in ESEIAAT), with the use of Petri Nets.

1.2. Scope

The scope of this master thesis is established in this section:

- Studying the FlexSim simulation software by the study of simple AGV networks which present deadlocks.
- Proposal of a solution for deadlocks which does not need to be the optimal one for each presented AGV network.
- Mapping the typical characteristics exhibited by the activities in a dynamic event-driven system.

1.3. Requirements

The requirements of the master thesis are:

- The work effort for this project is 300 man-hours (12 ECTS, being 25 h for each ECTS)
- Used simulation software version are:
  - FlexSim 2018 for Education
  - PIPE v.4.3
- The proposed solution for AGV networks must be applicable within the limits of FlexSim capabilities.
- All activities in a dynamic event-driven system must be described for FlexSim.

1.4. Usefulness

The main utility of this project is the final overview of the capacity for FlexSim to solve deadlocks in simple AGV networks.

Other outcome for this project is to validate the use of FlexSim as an alternative for Arena simulation software and assess its substitutability as a teaching software for ESEIAAT.

2. Automated Guided Vehicles

AGV corresponds to the acronym for automated guided vehicle, which is a portable robot that moves automatically with no need of having a driver. These systems help automate material handling even if the throughput does not warrant fixed path conveyors. These systems are suitable for short or medium distances moves, improving response time for material movement. They are efficient, dependable, and versatile material handling solution. Nowadays they are considered autonomous intelligent robots, so they are a vital part of the future Industry 4.0 processes.

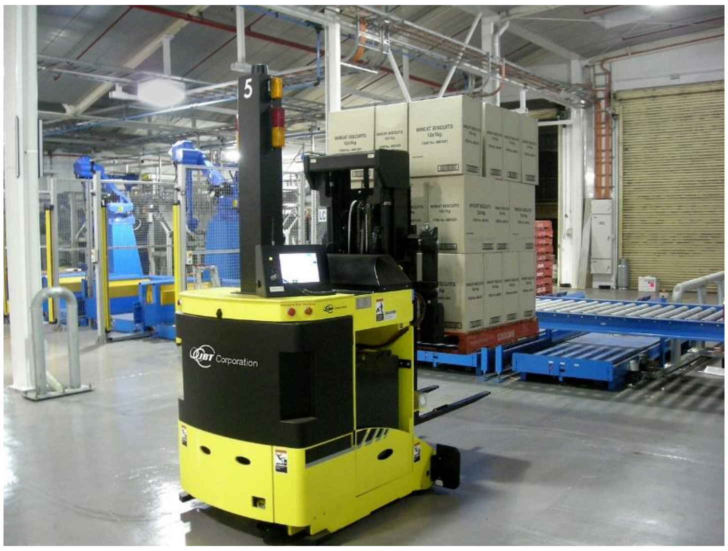


Figure 1. Automated Guided Vehicle handling material (source: http://ca.wikipedia.org)

Traditional AGVs use defined and previously programmed movements inside the installations. These carry out certain degree of difficulty if the route of a vehicle wants to be changed as the infrastructure is fixed. Recently, more flexible and intelligent vehicles have been created along algorithms that make decisions in non-familiar situations to avoid, for example, deadlocks.

In fact, this new generation of intelligent AGVs can solve one of the main problems: their response to unexpected situations. They can have a PLC incorporated to accomplish that function. This independent navigation systems might also imply that the plant manager might not need to change the environment of the plant [1].

These vehicles are flexible, since they can be relocated and their route can easily be remapped; accurate, as each detail of material handling can be tracked and efficient; safe, because they can operate in environments that might be hazardous for people and productive, as they can work 24 hours a day.

Much has been made lately about AGVs replacing people in the workplace and what that means for society and the economy. This is true in some cases, but it is also true that these vehicles will do work that people cannot or will not do. In these cases they are a win-win solution for everyone, because their work must be monitored, adjusted or completed by human workers who are employed to do it. This is just the start of seeing what AGVs are capable of doing in the workplace [2].

3. Deadlocks

A deadlock can be defined as a situation where one or more concurrent processes in a system are blocked forever because the requests for resources by the processes can never be satisfied. Deadlocks result from decentralized planning, which is the only realistic mode to govern large-scale logistic systems with highly dynamic interactions and no-detailed knowledge about future events [3]. Deadlocks have no global solution, so they are studied for each specific case.

An easy example is the classical problems of an intersection of two 2-way roads, with a car arriving from each lane, as it is seen in Figure 2. If the rule applied is the conventional “right car has priority over anyone”, a deadlock occurs. None of the cars can pass, unless at least one of them is removed or moves backwards.

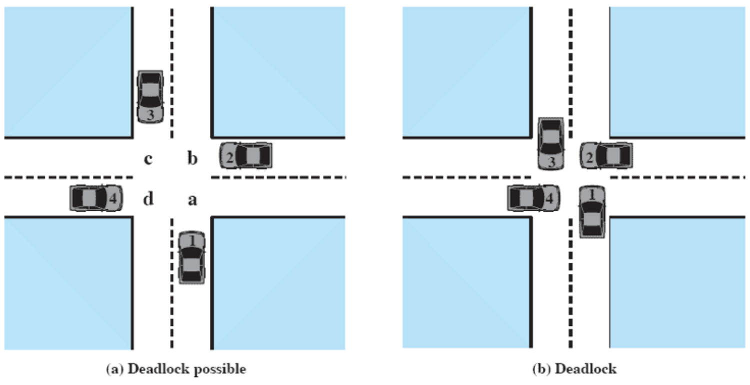


Figure 2. Deadlock of a road intersection

In a typical Flexible Manufacturing Process, raw parts of various types enter the system at discrete points of time and are processed concurrently, sharing a limited number of resources such as machines, robots, material handling system, fixtures and buffers. In such resource-sharing systems, deadlocks constitute a major issue to be addressed at the design and operation phases. This is a highly undesirable situation in which each of a set of two or more jobs keeps waiting indefinitely for the other jobs in the set to release resources. Both the lost production and the labor cost in resetting the system, meanwhile clearing the deadlock manually, can be avoided by proper design and careful operation [4].

3.1. Deadlock characterization

As it has been seen in the previous section, deadlocks problems must be dealt, and therefore it is important to characterize them, looking more closely to its features.

A deadlock situation can arise if the following four conditions take place simultaneously in a system. In fact, all deadlocks must present the existence of these conditions [5]:

1. **Mutual exclusion.** At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released. *For example:* People do queues in the supermarket at the shop clerk, since the supermarket’s worker cannot attend several people at once. It is not a sharable resource.

2. **Hold and wait.** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
   For example: The people in the supermarket's queue are still using their shopping carts and trolleys. They will not release this resource until they can "use" the shop clerk.

3. **No preemption.** Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.

4. **Circular wait.** A set {P_0, P_1, ..., P_n} of waiting processes must exist such that P_0 is waiting for a resource held by P_1, P_1 is waiting for a resource held by P_2, ..., P_{n-1} is waiting for a resource held by P_n, and P_n is waiting for a resource held by P_0.

3.2. Deadlocks in AGV systems

Deadlock problems have been attracting intensive research efforts in the design of automated manufacturing systems in which AGVs are used in the material handling systems and recent years have witnessed increasing research attention on solving deadlock problems for the AGV systems at Automated Container Terminals [6].

Figure 3. Automated operations at the Port of Hamburg (source: http://www.joc.com)

Firstly, it is important to characterize specifically the deadlocks for these systems. Note that the first three conditions mentioned in section 3.1 are always true for an AGV system. Resources are the zones of the path and the processes are the AGVs movement:

1. Mutual exclusion: A zone cannot have two vehicles in it at the same time (collision)
2. Hold and wait: Each vehicle has to be located within a determined time and waiting to move into a next zone.
3. No preemption: The vehicle cannot disappear suddenly from the system.

Therefore, the only avoidable condition is the **circular wait**. This situation can occur in this AGVs environment, depending on the control logic of the system, as various vehicles tend to share lanes along this system. Because of this condition, deadlock prediction must focus on this fact in order to prevent an imminent problem [7].

The most common kinds of deadlock in AGV systems are explained with some examples in section 6. However, before seeing the examples it is highly recommended to see how Petri Nets work in order to understand the development and solution.

3.3. Deadlock handling

Now that the basics of deadlocks have been exposed, they must be controlled anyhow in order to avoid them. Deadlocks can be handled in three ways [8]:

- Deadlock prevention: Ensures that at least one of the four necessary conditions characteristics explained in section 3.1 does not occur. Most approaches work by preventing the last condition of circular wait, and in fact, it is the only preventable condition in AGV systems.

- Deadlock avoidance: Examines the system's date dynamically and avoids deadlock states

- Deadlock detection and recovery: Allows the system to enter into deadlock state and then recovering from deadlock

3.3.1. Deadlock prevention

The simplest way to avoid deadlocks is to use systems where any of the characteristics explained in section 3.1. Using this limits ensure that deadlocks will never happen. Nevertheless, possible side effects of preventing deadlocks by this method are low device utilization or reduced system throughput.

As it has been mentioned, only the fourth and final condition for deadlocks can be preventable. This one is the circular-wait condition. One way to ensure that this condition never holds is to impose a total ordering of all resources types and to require that each process requests resources in an increasing order of enumeration [9].

In order to see if the system deadlocks are prevented, a reachability graph must be used. This representation corresponds to a finite graph that represent all states that can be reached from the initial state.

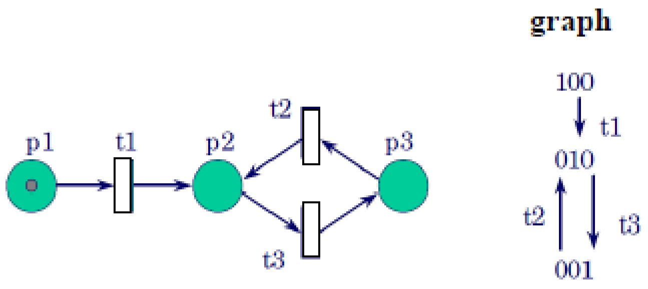


Figure 4. Example of a reachability graph

However, this clearly harms the flexible manufacturing process as the system becomes static because of the policies needed to prevent this deadlock. This is known to result in poor resource utilization. Additionally, the reachability analysis technique to arrive at deadlock prevention policies can become infeasible if the state space is very large. [4]

3.3.2. Deadlock avoidance

As a consequence of the previous sections, the usage of alternative methods is preferred for AGV systems. In deadlock avoidance, falsifying one or more of the necessary conditions dynamically is attempted by current state and possible future situations tracking. The idea is to let the necessary conditions prevail as long as they do not cause a deadlock but falsify them as soon as a deadlock becomes a possibility in the immediate feature. This leads to better resource utilization and that is vital for industrial processes [4].

AGV are expensive resources and because of that, the total downtime for the system must be minimized yet optimizing the resource utilization. Hence, a prediction-avoidance strategy is better, since the strategy makes sure that no deadlocks occur, resulting in better operations and higher throughput.

In order to avoid deadlocks, with the knowledge of the complete sequence of requests and releases for each process, the system can decide for each request whether or not the process should wait in order to avoid a possible future deadlock. The various algorithms that use this approach differ in the amount and the type of required information [9].

3.3.3. Deadlock detection and recovery

If a systems does not employ either a deadlock-prevention or a deadlock-avoidance algorithm then a deadlock situation may occur. In this environment, the system may provide and algorithm that examines the state of the system to determine whether a deadlock has occurred and other algorithm to recover from the deadlock.

At this point, however, we note that this scheme requires overhead that includes not only the run-time costs of performing the maintenance of the necessary information and executing the detection algorithm, but also potential losses inherent in recovering from a deadlock. [9]

4. Petri Nets

Petri Nets have established themselves as a powerful modeling formalism in computer science, system engineering and many other disciplines. They are a combination of well-defined mathematical theory with a graphical representation dynamism of the behavior of systems. The theoretic aspect of Petri nets models precisely the system behavior so as to be analyzed while the graphical representation enables the visualization of changes in the modeled system and simplifies the visualization of these complex systems. This combination is the main reason for the huge spread of the usage of Petri Nets. [10]

4.1. Basics definition

A Petri net is a particular kind of graphs compounded by three kinds of objects. These are *places*, represented by circles; *transitions* represented by bars; and *directed arcs*, which connect the first two ones.

The dynamic nature of the system is represented by the movement of *entities*, represented as dots, through the net. They can be temporal, in that it moves through the system, or permanent in that it serves other entities (normally called resources).

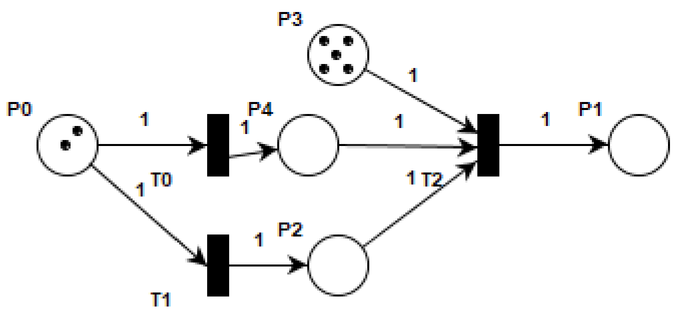


Figure 5. Example of a basic Petri Net

A Petri net is formally defined as a 5-tuple N = (P, T, I, O, M_0), where:

- P = {P_1,P_2,P_3,.....,P_{np}} is a finite set of places;
- T = {T_1,T_2,T_3,.....,T_{ne}} is a finite set of transitions;
- A = {A_1,A_2,A_3,.....,A_{na}} is a finite set of arcs that connect places to events and vice versa;
- W: $A_i \rightarrow$ {1,2,3,....} $\forall$ A_i is the weight associated with each arc;
- M0: $P_i \rightarrow$ {1,2,3,....} $\forall$ P_i is the initial number of entities in each place (initial marking).

The current location and distribution of entities in a Petri Net is called a marking of the system.

A transition can be fired if each transition input has the required number of entities specified by the weight associated with the arc from the place to the transition. Firing the event removes entities from the input places and adds entities to the output places. The number of entities removed or added equals the weight of the associated arc. [11]

4.2. Timed Petri Nets

Time is a crucial aspect when dealing with dynamic logistics, manufacturing or transportation processes, such as AGV systems. Therefore, it is needed to include the notion of time on the Petri Nets. The most used model shows the associated time of delay for enabling a transition to be fired.

The firing of a transition in a Petri Net corresponds to an event that changes the state of the system, it might be result of a verification of a logical condition in the system, as it happens in the previous section (immediate transitions) or can be induced by the completion of an activity, which naturally takes some amount of time (timed transitions).

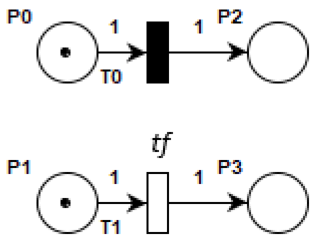


Figure 6. Graphical representation of an immediate transition (up) and a timed one (down)

As a convention, this document will use the PIPE software representation. Black rectangles for immediate transitions and white rectangles for timed ones. For sure, this white rectangle needs a time function (*tf*), which specifies the duration of that transition. [11]

4.3. Colored Timed Petri Nets

In order to increase the simplicity of the Petri Nets system, entities may acquire a value, often referred as "color". By means of this variable, too-big-to-handle nets get simplified very much. Additionally, these entities may be object or resources in the modelled systems and have attributes that might be not easily represented by a simple point. [12]

For example, in a manufacturing a process, a machine might process both pieces A and pieces B, but each piece have their own processing time. If we assign label (or color) "A" and "B", the Petri Net gets much simplified, as it is seen in Figure 7. Now the different processing times are a time function, depending on the color of the entity.

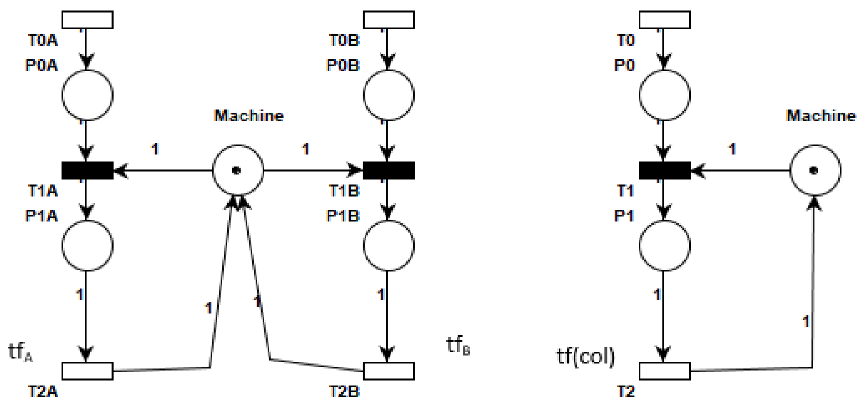


Figure 7. Simplification of a timed Petri Net (left) adding labels (colors) to it, resulting in a colored time Petri Net (right)

5. Mapping Petri Nets to an FlexSim Model

Once a Petri net conceptual model has been created and validated with the problem owners, it is significant to carry the validity and relevance of the model to the executable model. This is achieved by the mapping approaches explained below, and finally confirmed by the verification.

This state-space and causal or flow process logic expressed in the Petri Net must be mapped into a model that uses the FlexSim library blocks. It is convenient to use a table to map the Petri Net process model and the transition specifications to a FlexSim Model [13].

These Petri Net models can be mapped into equivalent FlexSim model code:

5.1. Sequential execution

In the Petri Net shown in Figure 8, transition T1 can only be fired after the firing of T0. This Petri net construct models the sequential relationship between activities. The place/timed transition pair can be coded in FlexSim using the **Delay** activity.

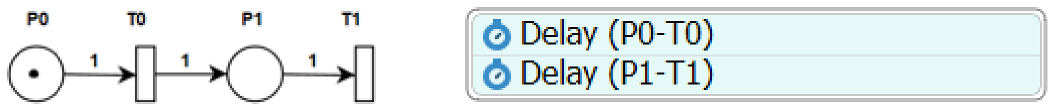


Figure 8. Sequential execution in Petri Net (left) and FlexSim (right)

5.2. Conflict

Transitions T0 and T1 are in conflict in this Petri Net: both are enabled, but the firing of either one disables the other. This situation will arise, for example, when an AGV must choose in an intersection between two different routes. The resulting conflict may be resolved in a non-deterministic way or in a probabilistic way. This situation where the entity must choose between two different transitions can be coded in FlexSim using the **Decide** activity. And it can be resolved in both mentioned ways.

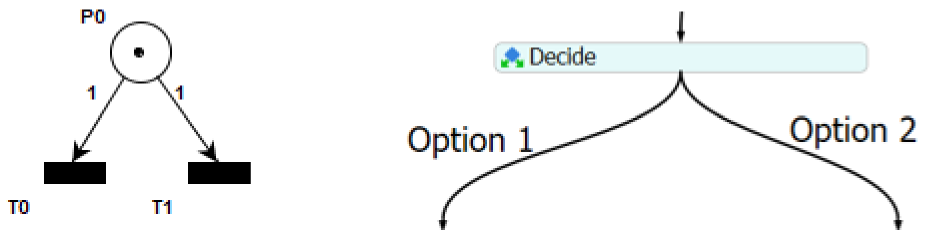
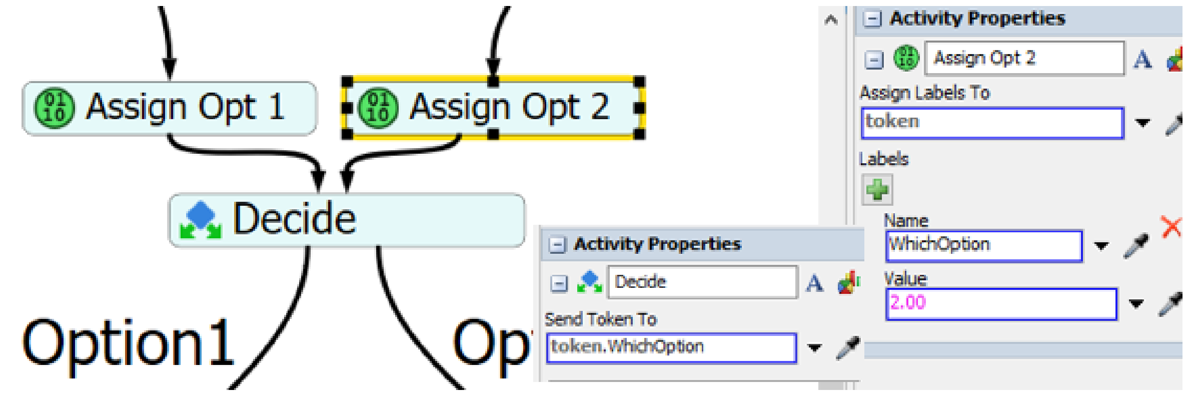


Figure 9. Conflict in Petri Net (left) and FlexSim (right)

Decide activity can have one or multiple inputs as well as one or multiple outputs. Each output has assigned a positive integer number (first option is number 1, second option is number 2… etc.) All inputs enter in the activity “Decide” and exits to the assigned output.

Therefore, in order to solve the conflict either in a deterministic or a probabilistic way will depend in the number that is “assigned” to each token, as the number assigned for each output is not changeable.

To solve it with a deterministic decision, labels (equivalent to colors in a colored Petri Net) can be used. This labels can be assigned before the **Decide** activity or they might exist from previous processes.



Send Token To: token.WhichOption

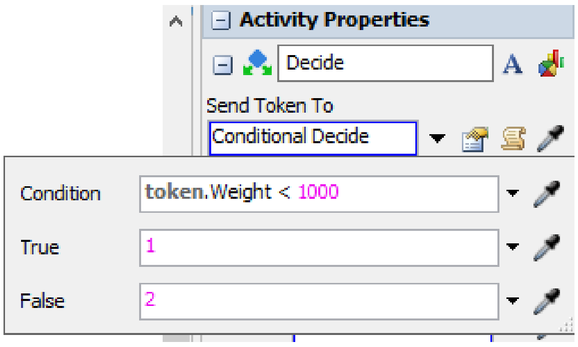
Assign Labels To: token

Name: WhichOption

Value: 2.00

Figure 10. Example of deterministic decision with FlexSim (label version)

For example, let's imagine that depending on the weight of the cargo we will choose one option or another (deterministic decision). Concretely, if the cargo weights less than 1000 u, it will exit by option 1, otherwise if will exit by option 2. There is an option in FlexSim that allows the user to do that, if previously those labels were assigned.



Send Token To: Conditional Decide

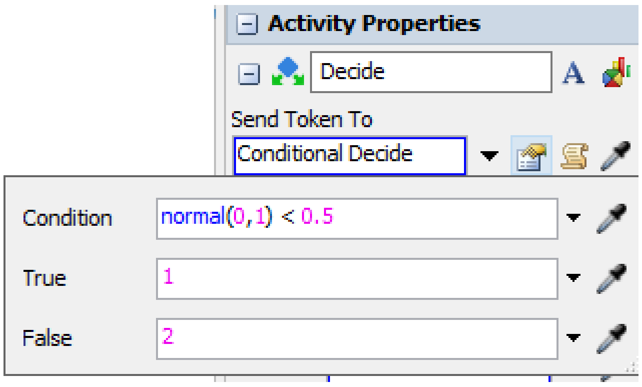
Condition: token.Weight < 1000

True: 1

False: 2

Figure 11. Example of deterministic decision with FlexSim (conditional version)

Conflict can also be solved probabilistically by means of this system, but adding a statistical expression. For instance, in Figure 12, the condition is selected by a normal distribution. There are several different statistical distributions as Bernoulli, uniform, Poisson, etc.



Send Token To: Conditional Decide

Condition: normal(0,1) < 0.5

True: 1

False: 2

Figure 12. Example of probabilistic decision with FlexSim

5.3. Concurrency with temporal entities

In Figure 13, both transitions T1 and T2 are concurrent between them. Concurrency is an important attribute of system interactions. So as to create concurrent transitions, there must exist a forking transition that deposits a temporal entity in two or more output places.

This might represent, for example, an AGV that transports two packages in its rack and splits the cargo anyhow between two different conveyor belts. The **Split** activity found in FlexSim deposits temporal entities at two (or more) output places.

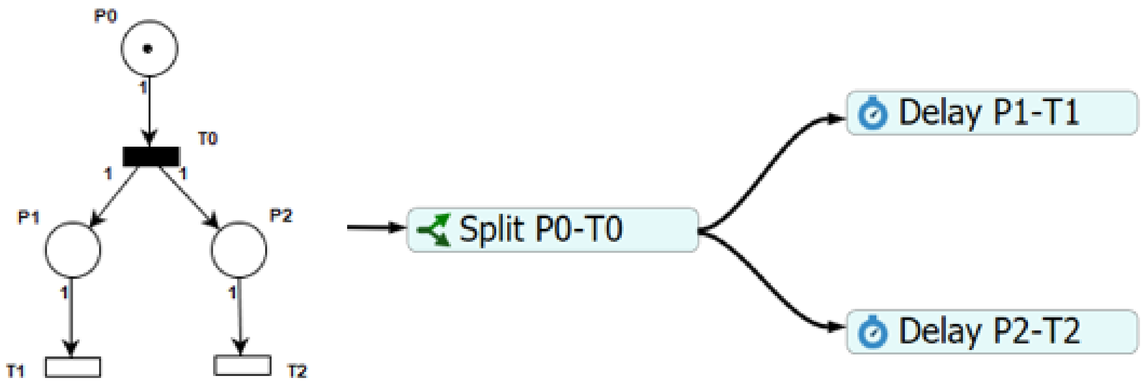


Figure 13. Concurrency with temporal entities in FlexSim

5.4. Synchronization with temporal entities

In Figure 14, both places P0 and P1 need to receive an entity in order to T0 to be enabled. Synchronization is quite usual in a dynamic system for an event.

This can be represented, for example, by a situation where two pieces need to be assembled before being transported. The existence of two different parts makes no sense from the point where they have joined into a sole entity.

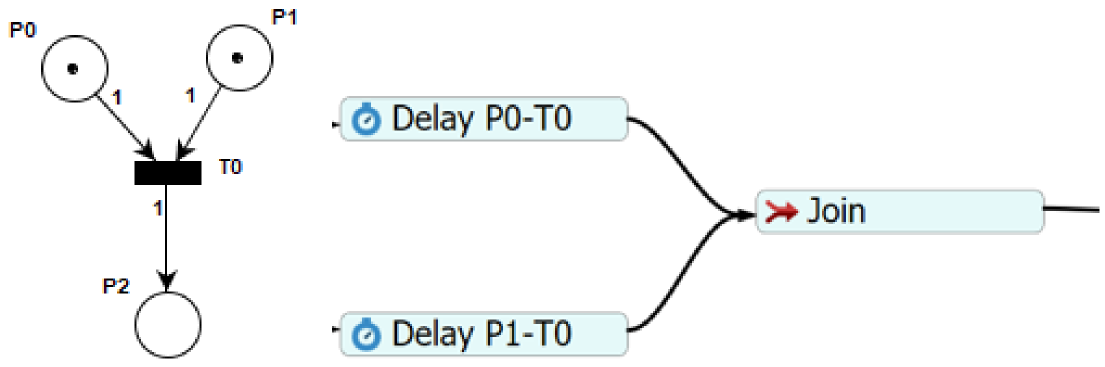


Figure 14. Synchronization with temporal entities in FlexSim

5.5. Concurrency and synchronization with resources

The Timed Petri Net that is shown in Figure 15 models a single-queue single-server process. This is a classic queuing model where arriving entities (T0) wait at a queue (P0) for the resource. When that resource is available (T1) it starts to process the entity. It remains at P1 until T2 is fired. Resource comes back to P2 and final pieces go to P3. As it can be seen, we have a synchronization with the resource (P0-P2-T1) and a concurrency (T2-P2-P3)

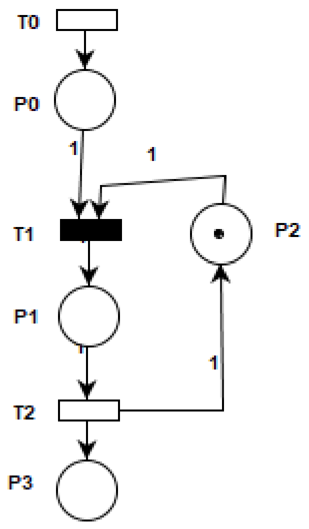


Figure 15. Queue-server model Petri Net

This whole Petri Net can be all simulated using **Join** and **Split** activities, as it has been explained above. However, because resources are so common in Petri Nets and in order to facilitate a rapid and comprehensive view of the scheme (imagine using the same resource for several different Petri nets), there are specific activities for them.

Therefore, Figure 15 can be represented in FlexSim software as it is shown in Figure 16.

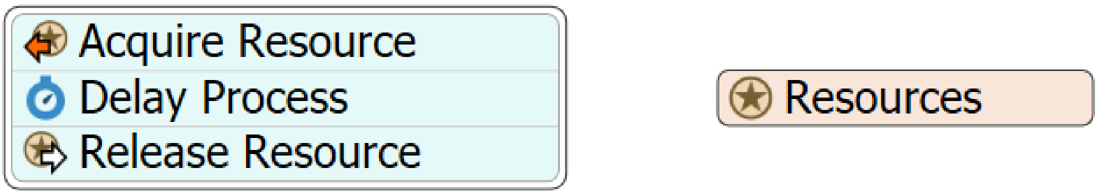


Figure 16. Resources activities in FlexSim

This easy scheme is the basic of using a resource in our manufacturing systems. For example, it might signify the usage of an AGV (the resource), and the delay is the time of transportation from the source to the destination of the product.

6. Deadlock cases

In this section, a number of deadlock cases where AGVs are involved is presented as examples of problems that these vehicle systems may encounter during its usage. These problems are used as examples for validating FlexSim as a useful tool to study this situations and to come across with a solution for these.

Each problem will have the same structure: Description of the problem, Petri Net and deadlock study, possible solutions, and finally, if needed so as to compare different solutions, the FlexSim simulation process flow with the correspondent results and discussion.

6.1. Case I: An AGV and a Machine

A simple case can be seen in Figure 17. A lone AGV is in charge of carrying raw pieces from the load station to the machine which processes them, and also from the machine (when eventually processed) to the unload station. If the AGV carries the raw piece while the machine still holds the processed piece, there will be a deadlock, as both processes cannot happen at the same time:

1. AGV wants to put the pieces into the machine to process them
2. Machine wants to put the final pieces in the AGV

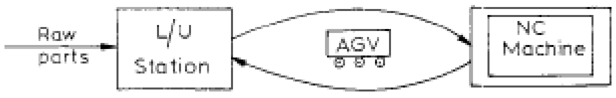


Figure 17. Case I: Simple manufacturing system composed by an AGV and a Machine

6.1.1. Place Transition Petri Net Model

Case shown in Figure 17 can be transformed into a Petri Net. Note that each token has the variable “loc (location)” assigned. This is because time for transitions T1 and T5, which correspond to the part waiting for the AGV, is variable. It depends on where is the AGV in each moment (next to the warehouse, loc = 0; or by the machine, loc = 1).

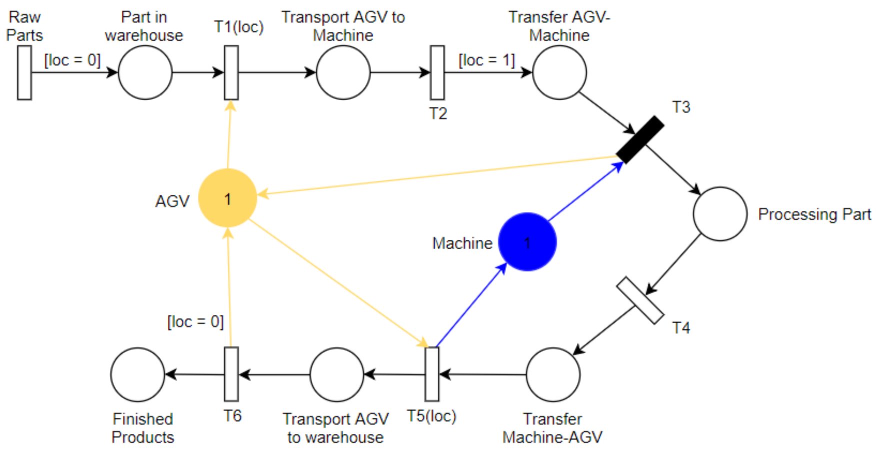


Figure 18. Case I: Initial marking of Petri Net

So as to input this system into the PIPE Petri Net Simulator, in order identify the possible deadlocks, it needs to be bounded. Infinite states are reachable due to the constant input of parts in the system. One easy way to bind the system is to connect the "Finished Products" with the input of the raw parts. This can be translated to: whenever a part has been finished and processed, automatically means that a new part comes into the system. We will assume that 2 pieces are entering the system waiting for being processed.

The obtained result is shown in Figure 19, where it is pointed that "S10" is a terminal state (deadlock).

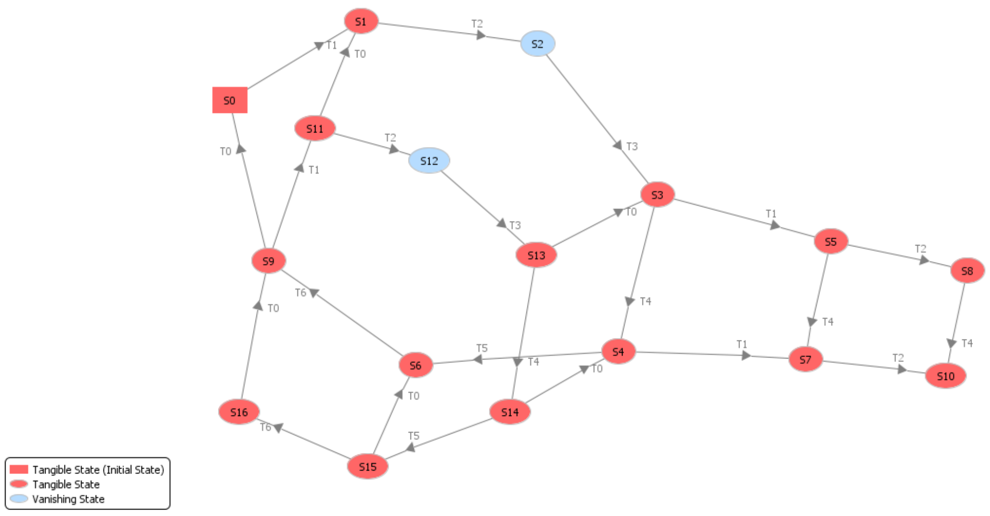


Figure 19. Case I: Reachability/Coverability Graph obtained by PIPE simulator

S10 represents the following system state: At some point, a new part arrives into the warehouse while the machine is processing another part. The AGV picks up this new part and transports it to the machine, where a deadlock happens clearly, because the machine is full and cannot put the finished piece on the AGV, and AGV is unable to unload its charge into the machine, causing the whole manufacturing process to stop.

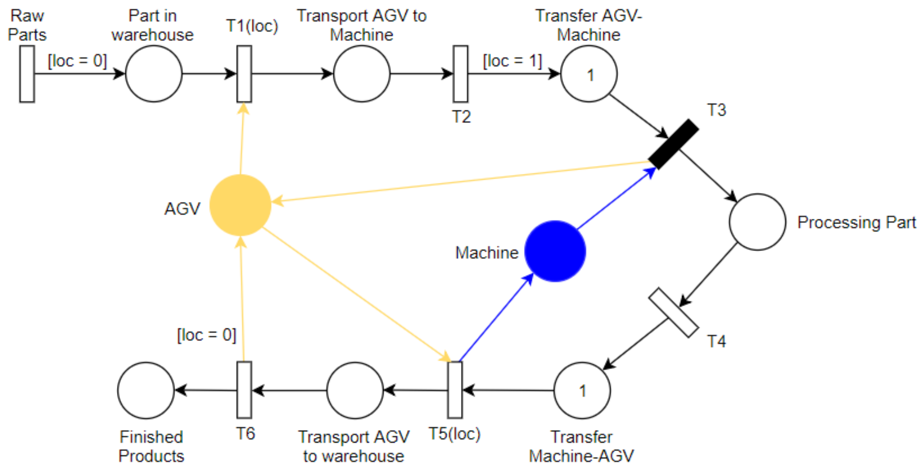


Figure 20. Case I: Deadlock situation represented in Petri Net

6.1.2. Possible solutions

There is a clear problem with an independent usage of the AGV and the machine. They must be dependable between them in order the system not to get deadlocked.

This is a very simple case where, if no willing to add more resources (more AGVs or machines), there is only a feasible solution.

The easiest solution is to **block** the AGV next to the machine until the piece has finished. It makes no sense that the AGV searches another piece if the machine is already full. Note that this fact simplifies the whole Petri Net, as it is shown in Figure 21. No longer is distinction for the AGV location needed.

There are other possibilities, like a buffer zone next to the machine where the AGV can drop temporarily parts, but this means change in resources and it would change the premises of the problem.

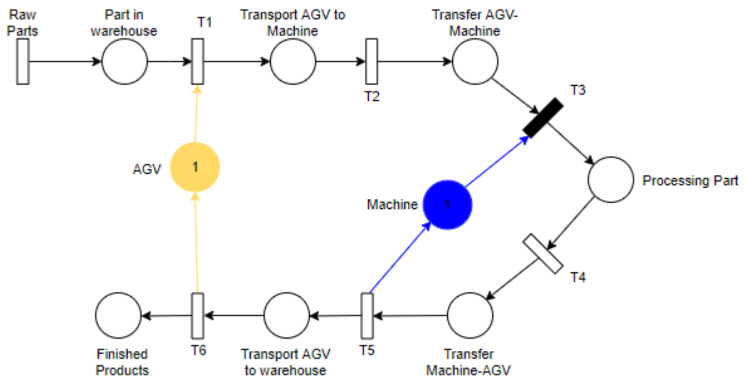


Figure 21. Case I: Possible solution represented in Petri Net

In order to check that this new system does not originate deadlocks, we can plot again the reachability graphic. As it can be seen in Figure 22, there are no end states and is totally cyclic, so no deadlock is possible.

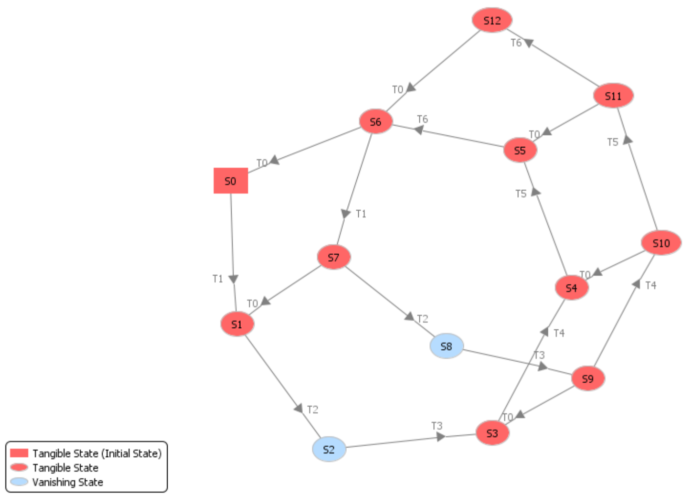


Figure 22. Case I: Reachability/Coverability Graph obtained by PIPE simulator for proposed solution

6.2. Case II: Intersection deadlock

Another kind of deadlock is produced when both AGVs during their path want to use the same intersection at the same time.

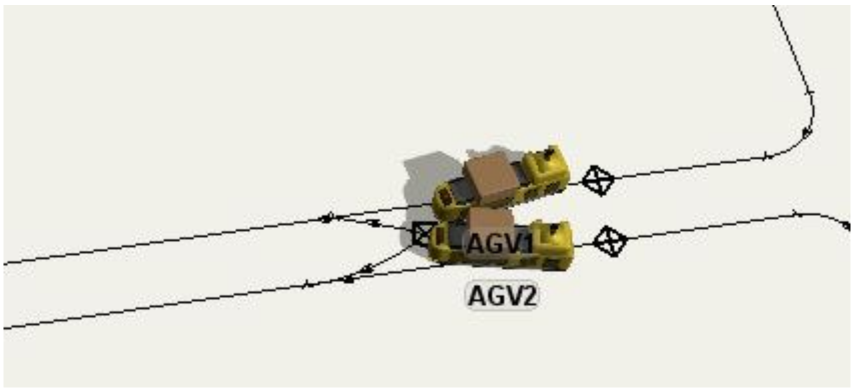


Figure 23. Case II: Screenshot of a Cross Lane Deadlock example, represented with FlexSim

In the example shown in Figure 23, there is a situation where both AGVs meet in an intersection, and want to use it at the same time. This situation is tricky, since both want to use the same resource at the same time, and if they do they might collide. The navigation system on both vehicles will tell themselves to stop, which results in a system where two vehicles are waiting for each other to move, but no one can.

As it can be seen in the picture, it is not as easy as to declare the only point of intersection of the roads as the resource, since in the situation shown in the image, the system is completely deadlocked. Even that one of the AGVs gets the right to pass, it will hit and probably damage the other vehicle. This possible scenario is reflected schematically in the following figure, where it is easy to see that vehicles aren't a single point and, therefore, the resource usage must be declared taking AGVs dimensions into account.

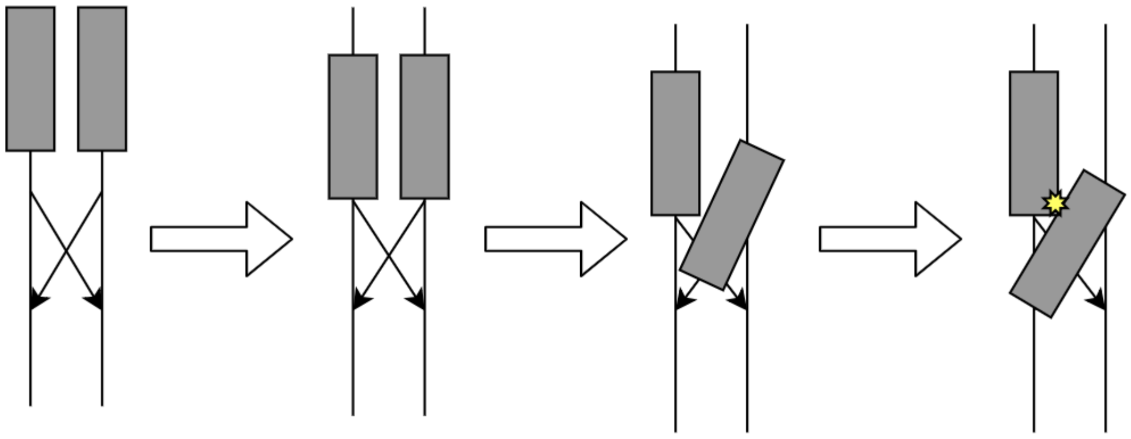


Figure 24. Case II: Intersection deadlock and possible collision

6.2.1. Place Transition Petri Net Model

There are several kinds of intersections, yet only one of them are represented in Figure 23. However, they all can be generalized in one Petri Net, where the intersection areas are resources that must be assigned to one of the AGVs.

First of all, an example of a deadlocked situation in the Petri Net. The road must be sectored in order to become a discrete simulation. Just for putting an example, this will be the different sectors:

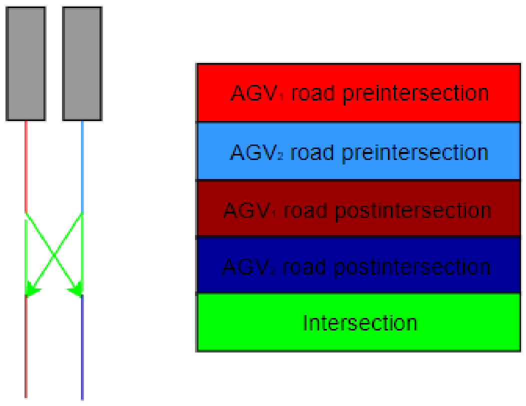

AGV1 road preintersection
AGV2 road preintersection
AGV1 road postintersection
AGV2 road postintersection
Intersection

Figure 25. Case II: Different sectors for the AGV network

The resulting Petri Net is the following one:

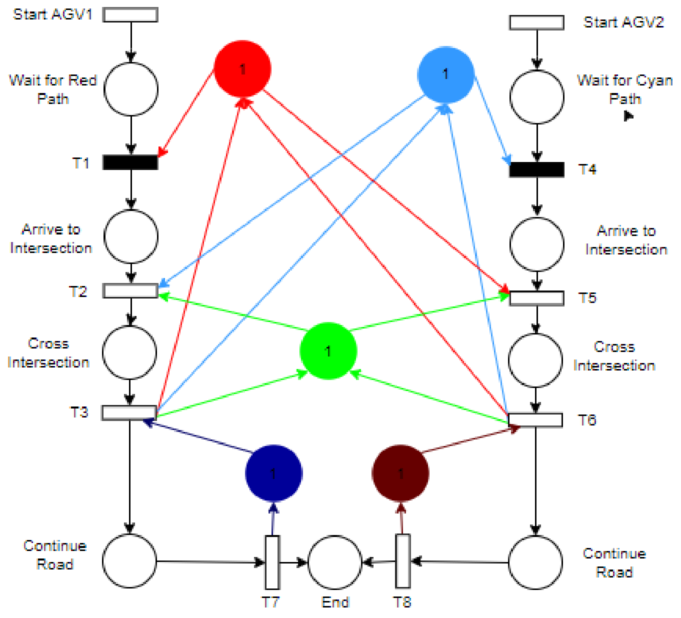


Figure 26. Case II: Initial marking of Petri Net

In order to identify the possible deadlocks, it needs to be bounded so as to input in PIPE Simulator. Infinite states are reachable due to the constant input of parts in the system. One easy way to bind the system is to delete T7 and T8 transitions and the End State, and supposing that after the intersection it enters again into the system, similar to a loop road. It will be assumed that only two AGVs are in the system.

The obtained result is shown in Figure 27, where it is pointed that "S4" is a terminal state (deadlock). This state corresponds to the situation represented in Figure 28, which is the previous state to the AGV crashing.

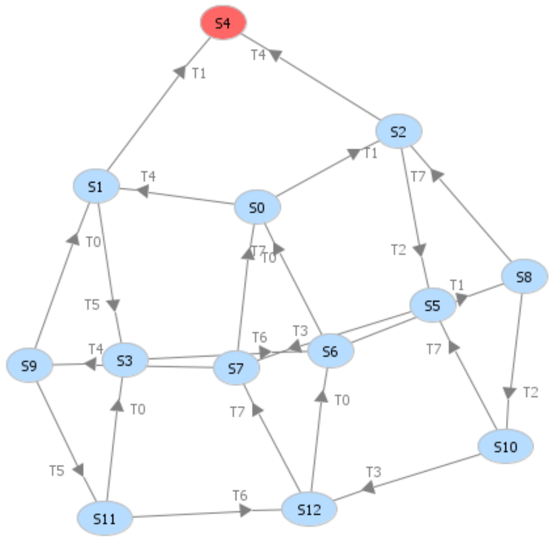


Figure 27. Case II: Reachability/Coverability Graph obtained by PIPE simulator

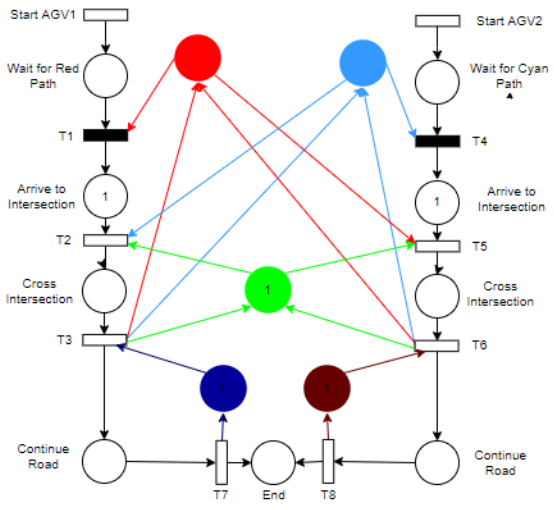


Figure 28. Case II: Deadlock represented in Petri Net

6.2.1. Possible solutions

The main problem here is the resource allocation, or to be clearer, the different assigned sectors in Figure 25. There must be a "safe space" before arriving to the intersection so as the vehicles cannot collide, and this is obtainable with an adequate road division. Of course, there are other structural solutions that may be not feasible, as widening the separation between the roads or changing the intersection system.

Despite that, the easiest solution is to widen the "Intersection" resource, ensuring that the AGV along all its movement will not invade the other vehicle space while waiting. In this way, the vehicles shall enter the new zone one by one, and they will not collide between themselves.

Note that this solution just subtly changes the Petri Net, but this little change allows the system to continue and not to get deadlocked. The fact that the pre-intersection roads are now rideable by the specific AGV avoids the possible collisions. New road division and Petri Net are displayed below:

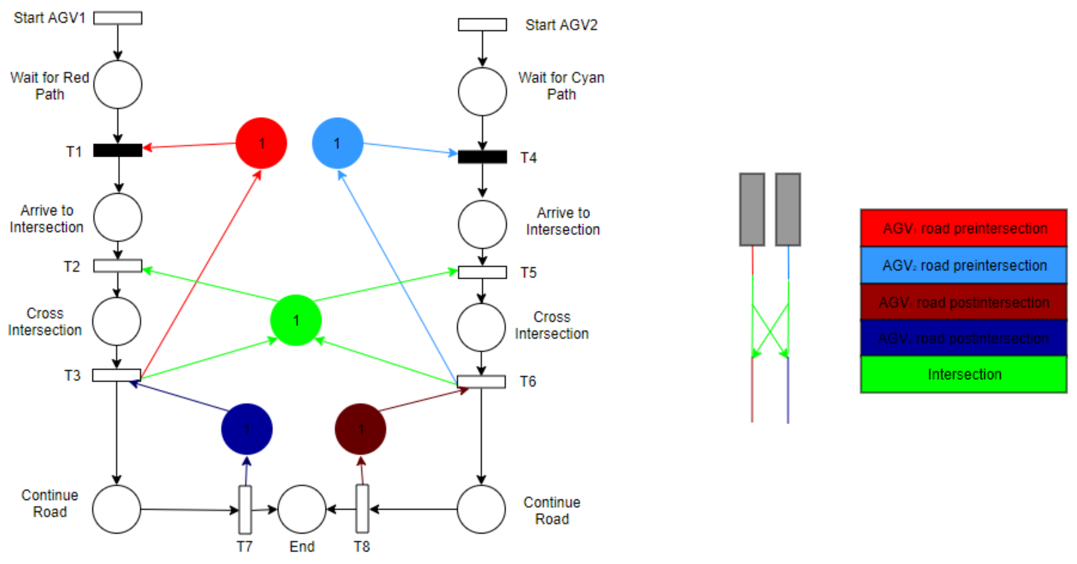


Figure 29. Case II: Possible solution, represented in Petri Net, changing the road division to solve deadlocks

So as to check that this new system does not originate deadlocks, we can plot again the reachability graphic. As it can be seen in Figure 30, there are no end states and is totally cyclic, so no deadlock is possible.

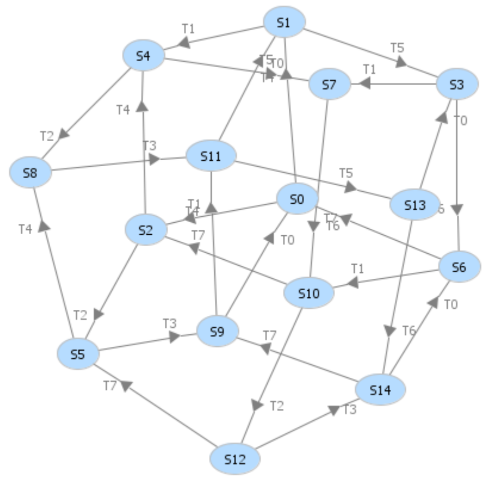


Figure 30. Case II: Reachability/Coverability Graph obtained by PIPE simulator for proposed solution

This case demonstrates the importance of planning the whole road network when planning AGV transportation. Not only the structure itself, but also the vehicle dimensions (keep in mind that the whole ride must be thought with the vehicle and cargo shape, not just a single point), the safe space declaration and the correct road division in order to avoid deadlocks and collisions.

6.3. Case III: Cyclic deadlock

The last deadlock, which is the most generic one, is the cyclic deadlock shown with a car example in Figure 2. This type of situation happens when there is a chain of vehicles requesting for resources (in this case, areas) in such a way that these form a cyclic requesting of them. This is very common in AGV systems and probably the most difficult situation to solve taking into account the whole system, because cyclic deadlocks occur very frequently.

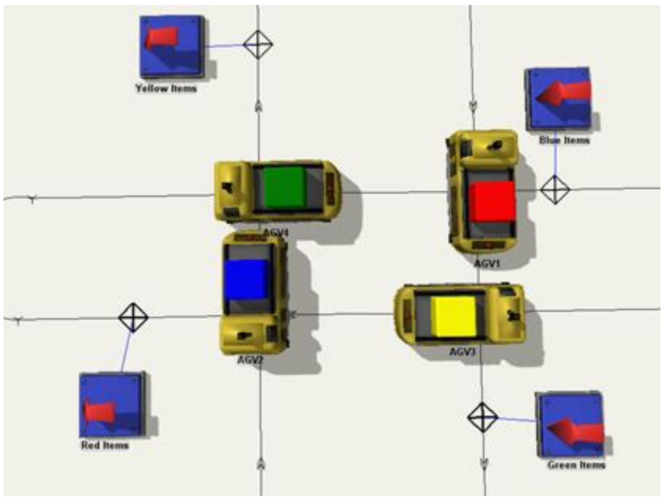


Figure 31. Case III: Screenshot of an example of cyclic Deadlock, represented with FlexSim

As it can be seen in Figure 31, all 4 AGVs want to move forward to the destination. For example, AGV2 wants to transport the blue box to the output that is called “Blue Items”. In order to do that, it needs to use AGV4 area, but AGV4 wants to transport green box to the analogous output and that area is occupied by AGV1, and this successively till having a cyclic deadlock.

Each cyclic deadlock might be solved as the previous case, the “intersection” deadlock. In fact, this is just a big intersection that gets collapsed if four vehicles enter the road knot. If any of the four vehicles would not have entered the system, this cycle deadlock would have not exist. Therefore, for this case, Petri Nets and solutions are analogous to case II.

However, this is the “easy” solution for this cyclic deadlock. Because these deadlocks usually originate from a much bigger problem and road network, this is a naïve solution for just this very concrete case. Normally, a complex road of intersections originate very different possible cyclic deadlocks, comprising a huge number of different possibilities (from two AGV’s involved to all the vehicles in the system). In these cases, deadlocks are usually not prevented because they are almost impossible to do so, if we want a profitable system. Normally, they enter into a deadlock and manual operators recover the system, or in the best cases, the managerial team try to avoid deadlocks.

An example from the automated port of Singapore is sketched in Figure 32, where a tremendous number of intersections can be seen in a simple glance [7]. This is where real cyclic deadlocks happen and they are a headache, and this proposed solution is not feasible anymore.

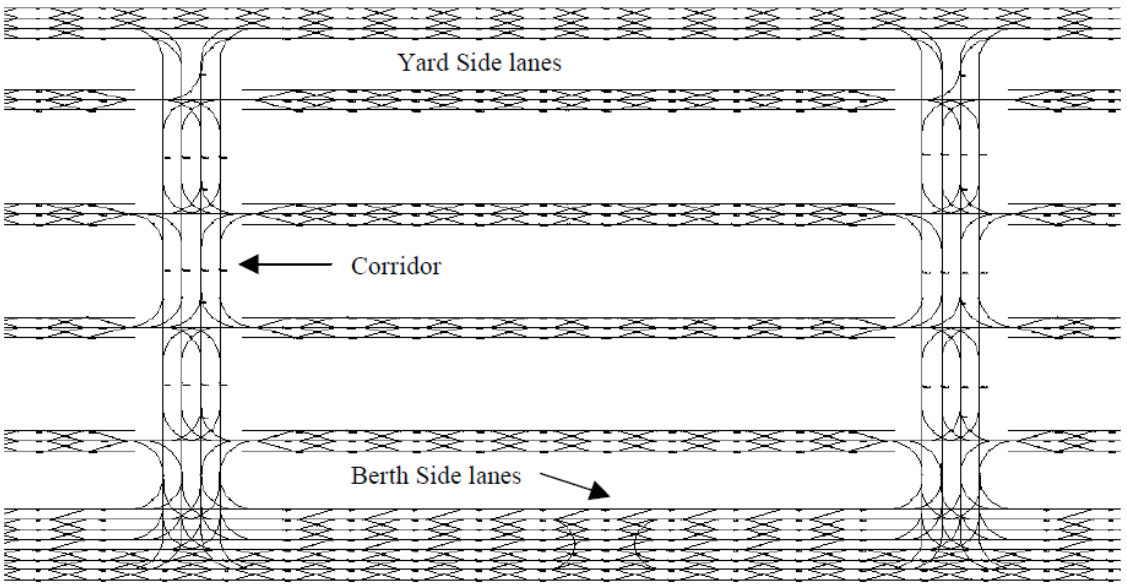


Figure 32. Sketch from a part from the Automated Port of Singapore. [7]

6.4. Case IV: Bridge crossing deadlock

The most basic deadlocks in AGV systems would be when two of the vehicles use the same bidirectional lane in opposite directions. An example on this is shown in Figure 33.

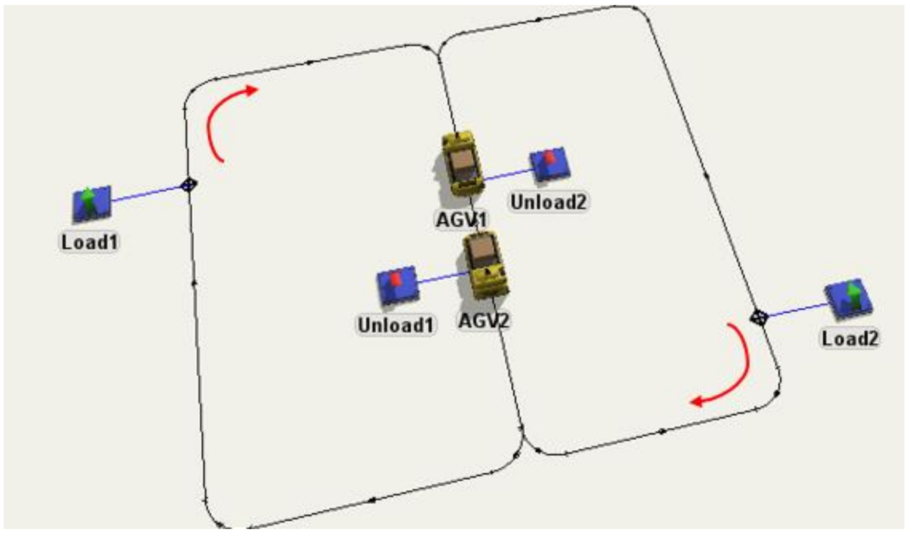


Figure 33. Case IV: Screenshot of an example of Bridge Crossing Deadlock, represented with FlexSim

In this example, AGV1 needs to move cargo from Load1 to Unload1 and analogously for AGV2. Eventually, a situation might occur where AGV1 and AGV2 would enter in the same lane from opposite sides and the whole system will be blocked. That is because AGV1 wants to access to Unload1 and AGV2 wants to access to Unload2, but they cannot also collide neither pass over other. Two vehicles that want to move and both vehicles have found that its path has been blocked.

6.4.1. Place Transition Petri Net Model

Case shown in Figure 33 can be transformed into a Petri Net, although it is not trivial to define. While an AGV covers certain part of the tracks, the other AGV cannot. Ideally every inch of the tracks should be a resource, but this is neither helpful nor able to be simulated.

Therefore, the sketch must be divided in different sections regarding the usage of the path by the AGVs. It has been decided to divide the different paths like displayed in Figure 34, where are represented with different colors.

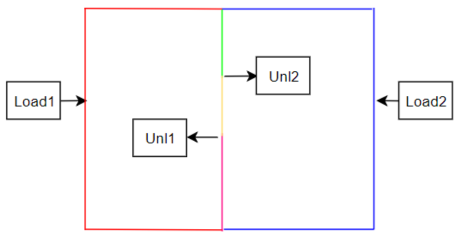


Figure 34. Case IV: Different sectors for the AGV network

Now that each path has been labeled, the Petri Net can be constructed as in Figure 35.

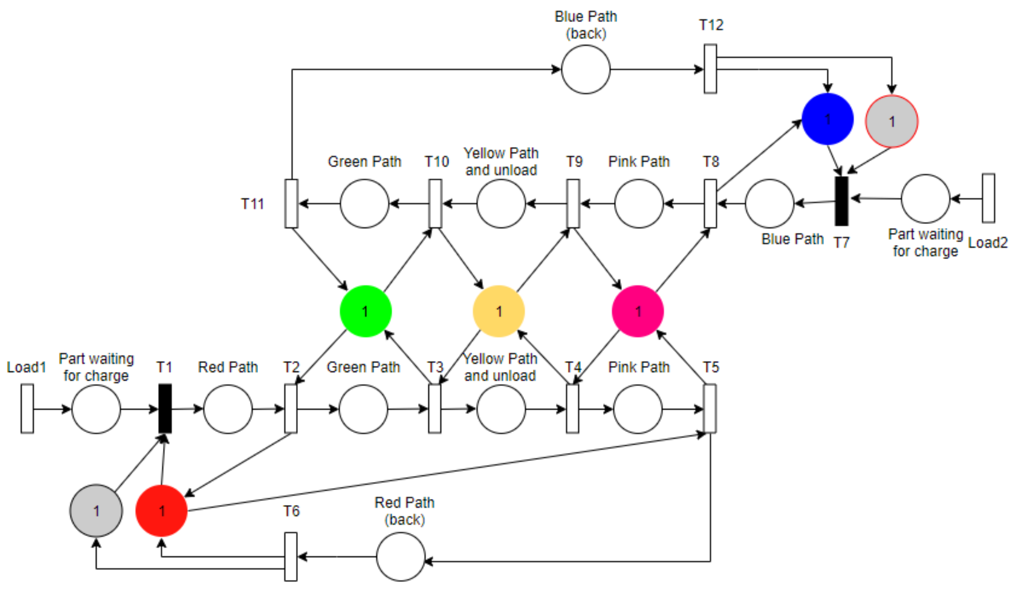


Figure 35. Case IV: Initial marking of Petri Net

In this case, we had to assume that there is just one part waiting for being charged in both load stations, and when coming back, is again waiting to be charged so as to represent it in PIPE simulator. Figure 36 shows the different possible states in which two different systems terminal states can be found (S10 and S11).

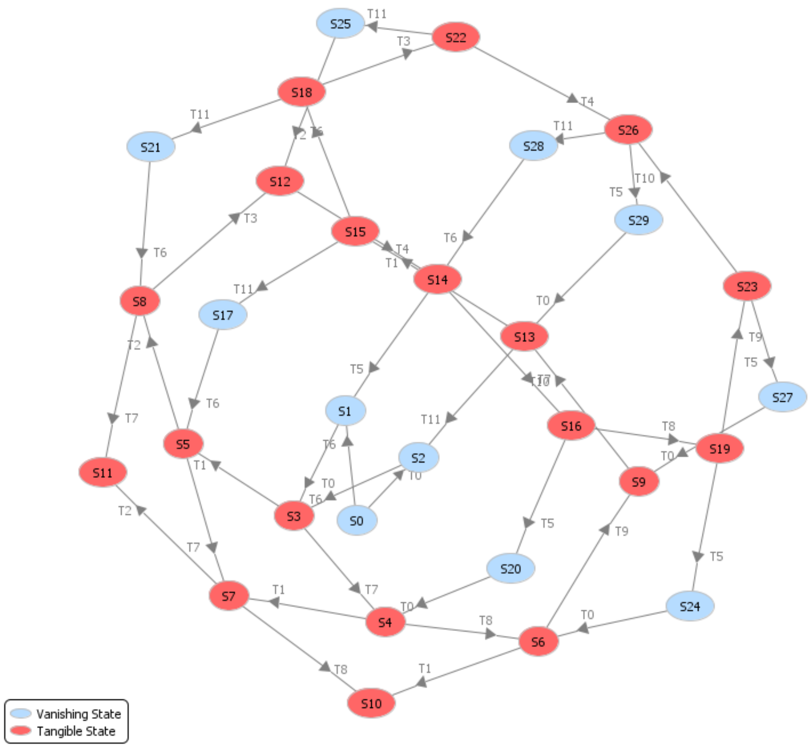


Figure 36. Case IV: Reachability/Coverability Graph obtained by PIPE simulator

Those deadlocks correspond to the following ones, represented in Figure 37. Note that it is really easy to understand imagining two vehicles in one lane bridge. They cannot use it at the same time because they are going through it in opposite ways.

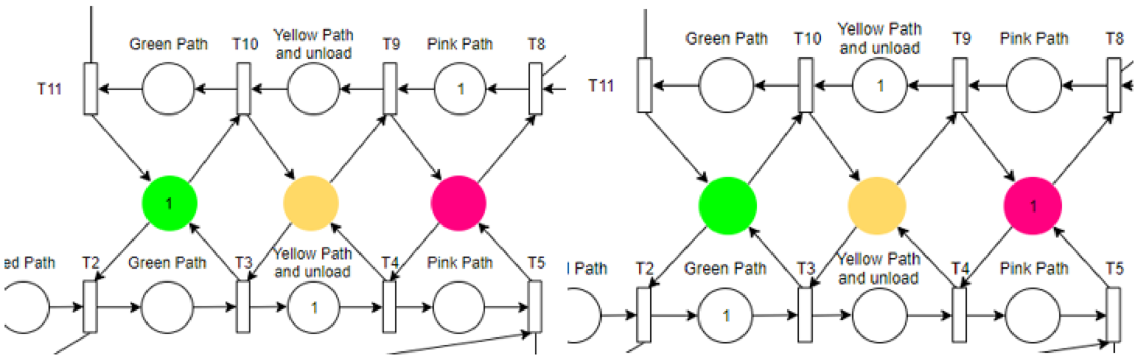


Figure 37. Case IV: Deadlocks represented in Petri Net (S10 at the left, S11 at the right)

6.4.2. Possible solutions

The system gets deadlocked if both AGVs are using the middle road at the same time by opposite sides. Therefore, this situation must be prevented anyhow.

Blocking the middle road by one AGV

One possible solution is that the first AGV that arrives to the middle road, traps all the resources until the vehicle gets off from it. The other AGV, if waiting, must wait until the first one releases the resource. New Petri Net is displayed in Figure 38.

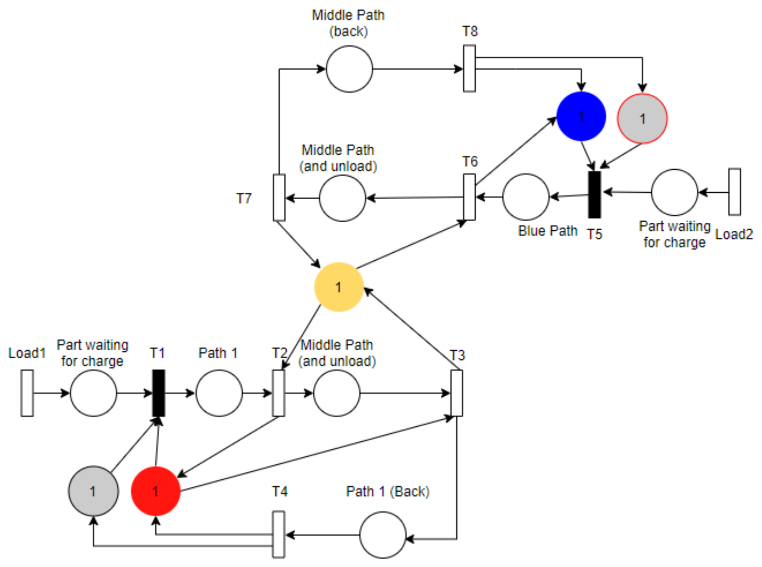


Figure 38. Case IV: First proposed solution (blocking the middle road)

Note that the different sub-paths that had been declared for the middle way do not longer exist. This is originated because for the new situation it is non-sense to continue describing the path as three different resources since one catches them all up at the same time. The obtained reachability graph is shown in Figure 39, which shows that the system is totally cyclic which means no deadlocks.

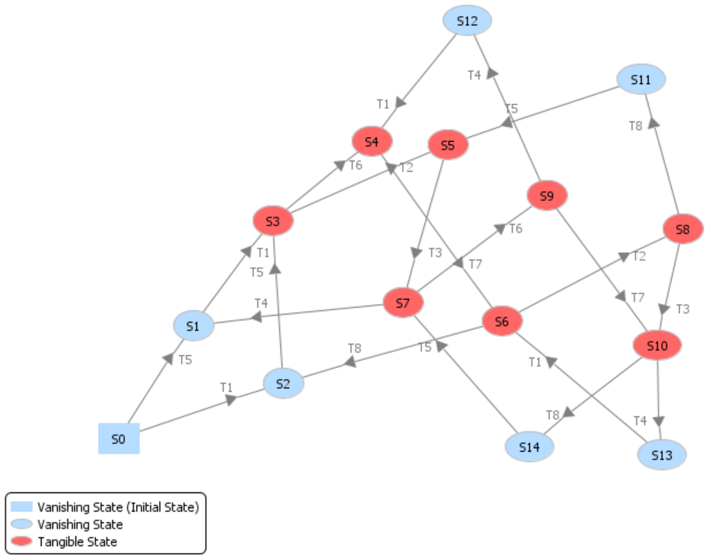


Figure 39. Case IV: Reachability/Coverability Graph obtained by PIPE simulator for first proposed solution

Switching directions of one AGV (road has one entrance and one exit)

Other smart solution is to switch directions of one of the AGVs, so that both enter the common path by the same entrance and use the same exit. This lets both AGVs use the bridge at the same time without deadlocking the system. This case might not apply for all situations in bigger systems, since it might be structurally difficult or inconvenient to do so.

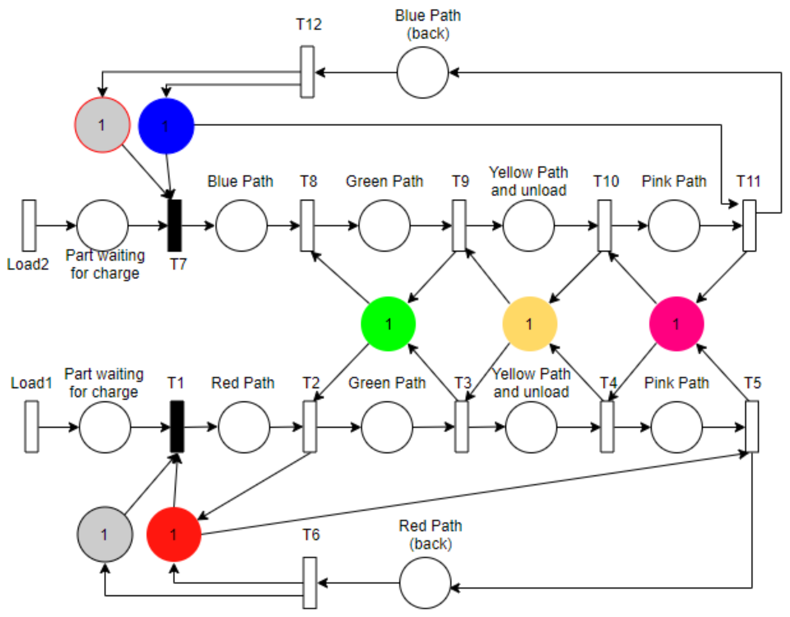


Figure 40. Case IV: Second proposed solution (same entrance and exit)

It is also remarkable that for this solution, there is also other extra one where AGVs are even more flexible and can use both routes. This happens because the red/blue paths are free when both AGVs are using the middle way. This might increase the productivity of the transport system, depending on the cases.

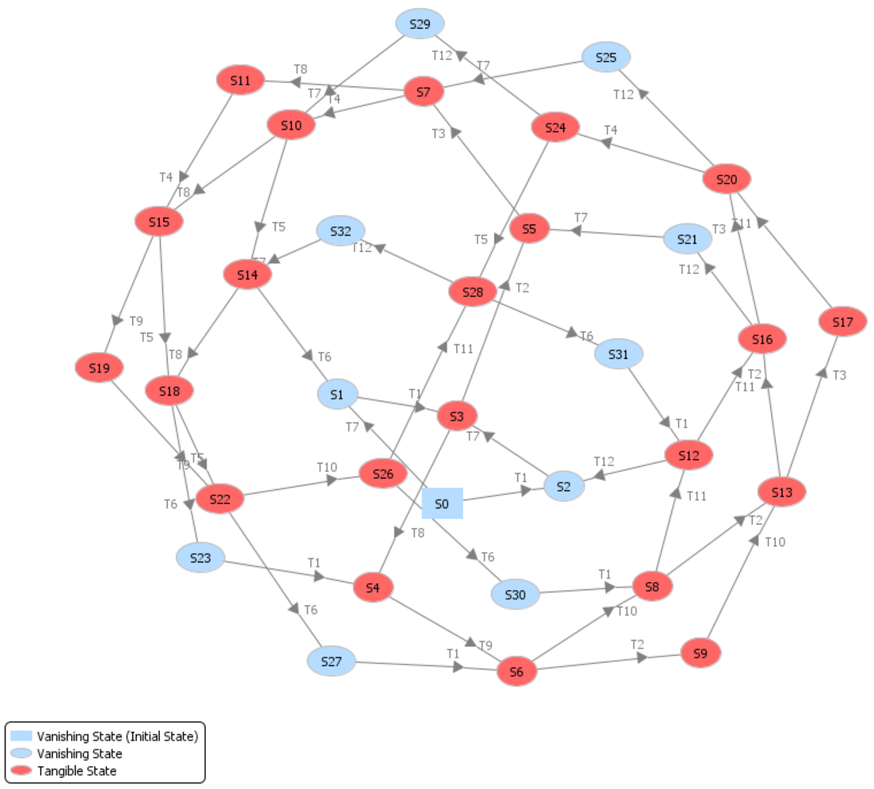


Figure 41. Case IV: Reachability/Coverability Graph obtained by PIPE simulator for second proposed solution

6.4.3. FlexSim process simulation flow

In order to compare both proposed solutions, they need to be simulated in FlexSim to get results for different state systems so as to compare them. This one is different from the previous cases, as it encounters two different alternatives to prevent deadlocks.

All previous information (Petri Nets) is used along with the explanation exposed in section 5. All four cases and solutions will be simulated. To recapitulate:

- Original case (possible deadlock)
- Solution 1 (blocking middle way by only one AGV)
- Solution 2 (both AGVs use the same entrance and exit, changing the direction of the original configuration)
- Solution 2b (same approach as solution 2, but with a possible improvement in which AGVs are flexible and after unloading, decides which cargo get)

The constructed diagrams are shown as it follows:

Original Case (possible deadlock)

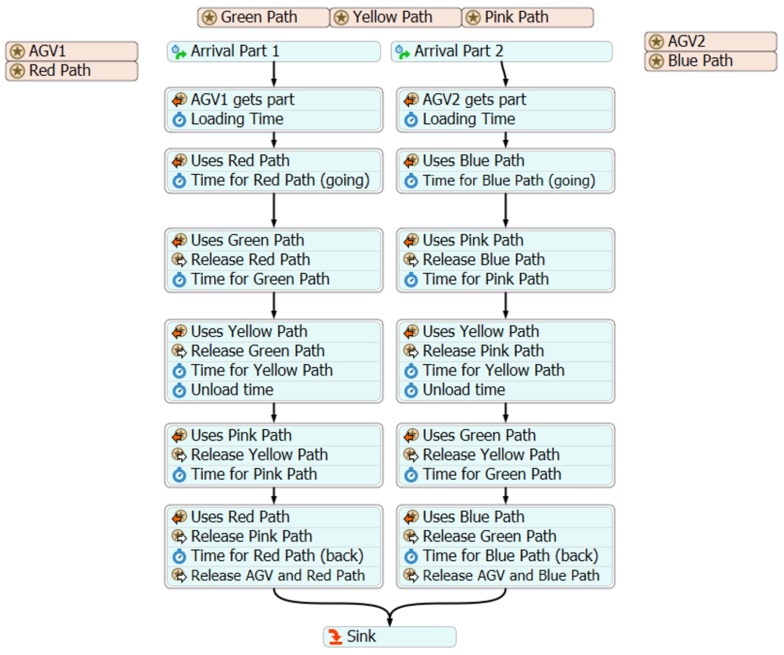


Figure 42. Case IV: Process created in FlexSim for the original problem

Solution 1 (blocking middle way by only one AGV)

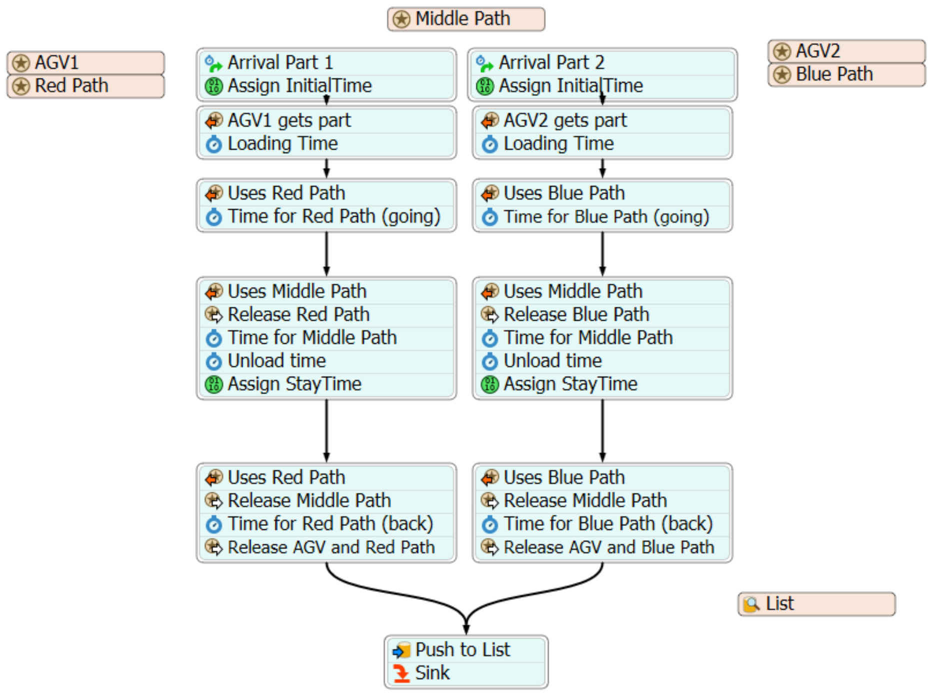


Figure 43. Case IV: Process created in FlexSim for the first proposed solution: blocking middle road

Solution 2 (both AGVs use the same entrance and exit)

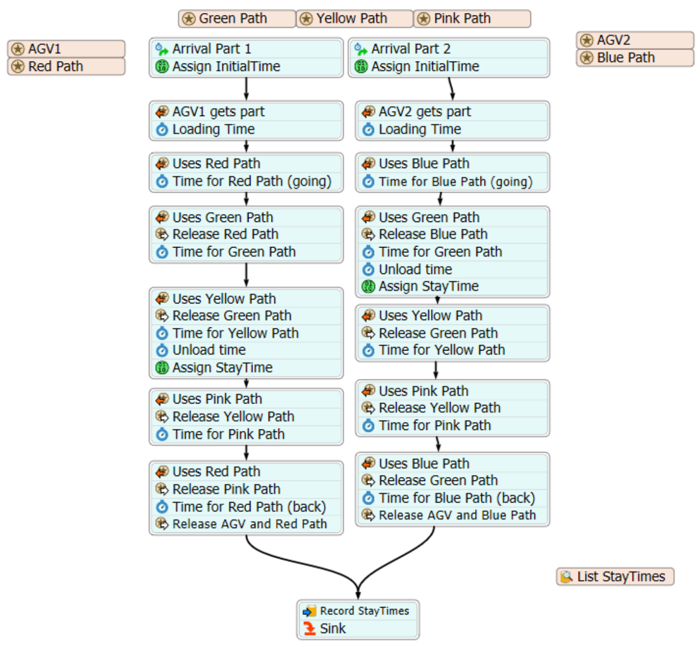


Figure 44. Case IV: Process created in FlexSim for the second proposed solution: same entrance and exit

Solution 2b (both AGVs use the same entrance and exit + flexible AGVs)

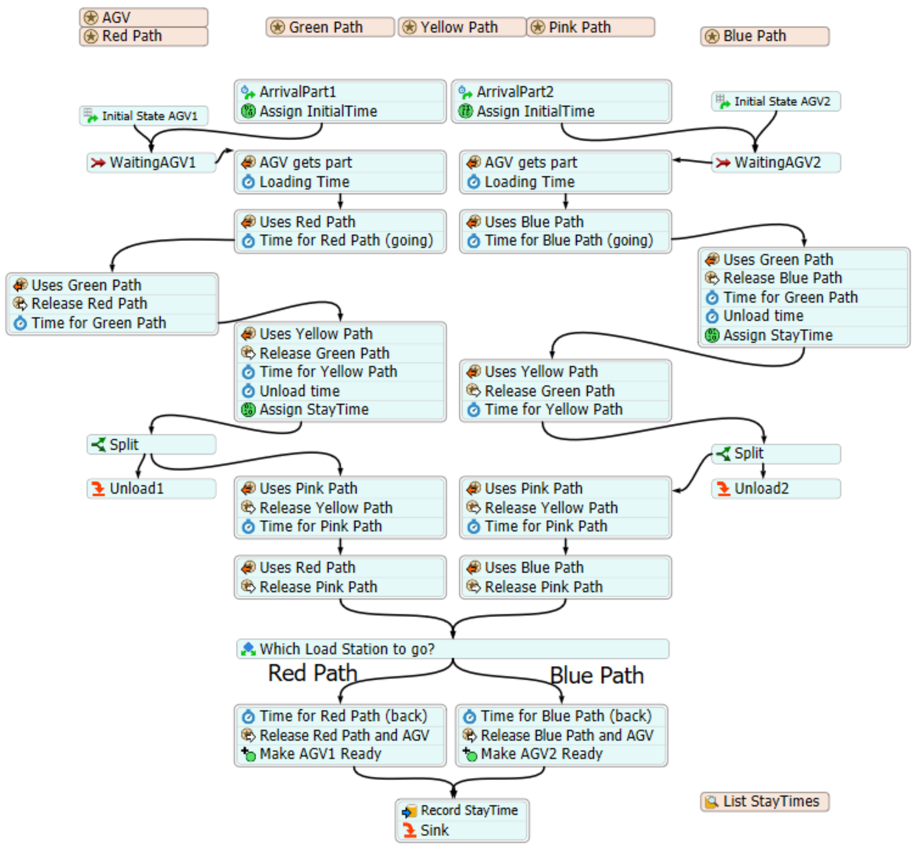


Figure 45. Case IV: Process created in FlexSim for the second proposed solution alternative: same entrance and exit along with flexible AGVs

The selected algorithm for the activity decide: “Which load station to go?” is based on the flow diagram that can be found in Figure 46, as well as the program coded below the mentioned diagram.

This procedure has been selected in order to get a higher usage of the AGVs, avoiding them to be stopped in the load stations if there is no cargo, or sending various AGVs if one station is overloaded.

Moreover, it uses a FIFO system in case of draw in both stations, which allows us to improve our Key Performance Indicator.

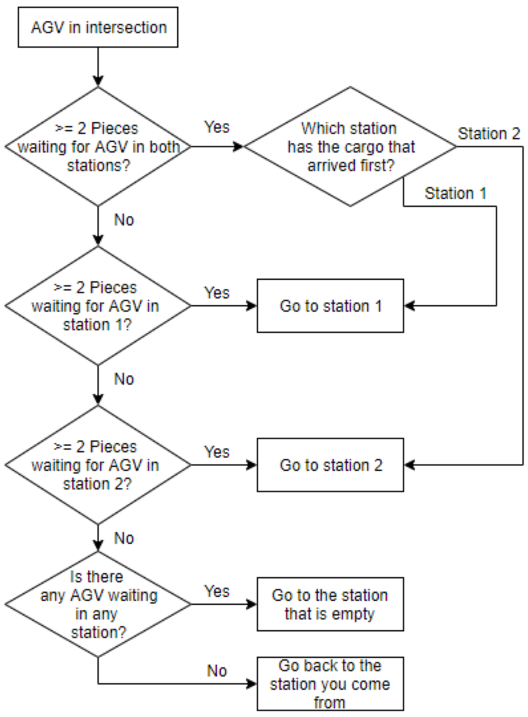

AGV in intersection
>= 2 Pieces waiting for AGV in both stations?
Yes
Which station has the cargo that arrived first?
Station 2
Station 1
No
>= 2 Pieces waiting for AGV in station 1?
Yes
Go to station 1
No
>= 2 Pieces waiting for AGV in station 2?
Yes
Go to station 2
No
Is there any AGV waiting in any station?
Yes
Go to the station that is empty
No
Go back to the station you come from

Figure 46. Case IV: Flow Diagram for the second alternative solution, in order to the AGV to decide where to go after the intersection

```
if (getstat(getactivity(processFlow, "WaitingAGV1"), "Content", STAT_CURRENT, current) >= 2 &&
getstat(getactivity(processFlow, "WaitingAGV2"), "Content", STAT_CURRENT, current) >= 2 &&
getlabel((gettoken(current, getactivity(processFlow, "WaitingAGV1"), 1)),"InitialTime") <=
getlabel((gettoken(current, getactivity(processFlow, "WaitingAGV2"), 1)),"InitialTime"))
        return 1;
else if (getstat(getactivity(processFlow, "WaitingAGV2"), "Content", STAT_CURRENT, current) >= 2 &&
getstat(getactivity(processFlow, "WaitingAGV1"), "Content", STAT_CURRENT, current) >= 2 &&
getlabel((gettoken(current, getactivity(processFlow, "WaitingAGV2"), 1)),"InitialTime") <=
getlabel((gettoken(current, getactivity(processFlow, "WaitingAGV1"), 1)),"InitialTime"))
        return 2;
else if (getstat(getactivity(processFlow, "WaitingAGV1"), "Content", STAT_CURRENT, current) >= 2)
        return 1;
else if (getstat(getactivity(processFlow, "WaitingAGV2"), "Content", STAT_CURRENT, current) >= 2)
        return 2;
else if (getstat(getactivity(processFlow, "Initial State AGV1"), "Input", STAT_CURRENT, current) >
getstat(getactivity(processFlow, "ArrivalPart1"), "Input", STAT_CURRENT, current))
        return 2;
else if (getstat(getactivity(processFlow, "Initial State AGV2"), "Input", STAT_CURRENT, current) >
getstat(getactivity(processFlow, "ArrivalPart2"), "Input", STAT_CURRENT, current))
        return 1;

return token.Way;
```

6.4.4. Results and discussion

Because this is a generic case with no values for each variable (mostly time arrivals and transport times), in order to compare some results, we find the need of present different situations to evaluate the different solutions. Results might vary differently depending on the different variables, and it is very likely that there is no global solution for all situations.

First of all, it is easy to demonstrate that it is very likely to originate a deadlock in the system. If both AGV's enter to the "bridge" at the same time, the problem occurs in FlexSim and the process cannot continue, as it is seen in Figure 47.

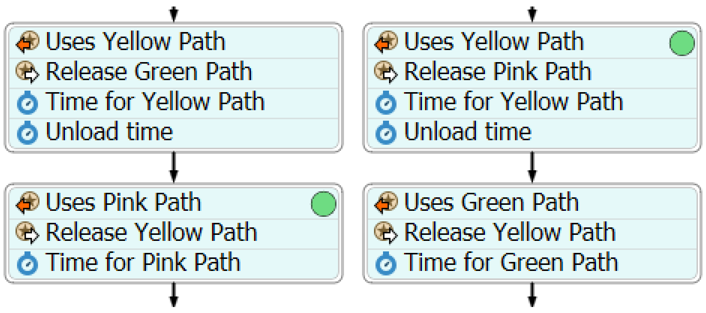


Figure 47. Case IV: Deadlock represented in FlexSim

The different times to declare are, according to Figure 34:

Table 1. Case IV: Different time concepts and its variable assignation

Concept	Variable	Concept	Variable	Concept	Variable
Arrival 1	$t_{arrival1}$	Green Path	t_{green}	Arrival 2	$t_{arrival2}$
Load 1	t_{load1}	Yellow Path	t_{yellow}	Load 2	t_{load2}
Unload 1	$t_{unload1}$	Pink Path	t_{pink}	Time Unload 2	$t_{unload2}$
Red Path	t_{red1}			Blue Path	t_{blue1}
Red Path (back)	t_{red2}			Blue Path (back)	t_{blue2}

Now, different cases can be presented and studied depending on the different solution systems. There are different ways to study how well the transportation system works. For example, the utilization of the AGV's, the time that lasts between the arrival of the piece and its unloading, the amount of pieces that have been unloaded, etc.

The main objective in this problem is to optimize the transportation between the arrival and the unloading. Therefore, the only key performance indicator would be the time between the piece arriving to the system and its exit, taking into account all possible idle times (AGV is stopped or unavailable).

Subsequently, different times are proposed in order to see which solution might fit the best.

Scenario 1: Underused system, where arrival times are significantly longer than the vehicle operational times.

$t_{arrival1} = t_{arrival2} = \exp(15)$ min

$t_{load1} = t_{unload1} = t_{load2} = t_{unload2} = 1$ min

$t_{red1} = t_{blue1} = t_{red2} = t_{blue2} = 1$ min

$t_{green} = t_{yellow} = t_{pink} = 0{,}33$ min

Table 2. Case IV: Results for Scenario 1, simulations for a horizon time = 1 day

	Blocking middle way by only one AGV	Both AGVs use the same entrance and exit	Both AGVs use the same entrance and exit + flexible AGVs
Min. Lead Time (min)	4,0	3,3	3,3
Max. Lead Time (min)	16,2	15,8	15,8
Avg. Lead Time (min)	5,8	5,2	5,2
Number of batches	191	191	191

As it can be driven from these results, if the system is underused, it does not really matter which of the three solutions are used, as there is not a high difference between all of them. We get better statistics for the flexible AGV system, as expected, but the cost of configuring this system may not be worth it economically.

Scenario 2: Slow-driving bridge, where the AGVs cannot drive through the middle way in a fast way.

$t_{arrival1} = t_{arrival2} = \exp(15)$ min

$t_{red1} = t_{blue1} = t_{red2} = t_{blue2} = 1$ min

$t_{load1} = t_{unload1} = t_{load2} = t_{unload2} = 1$ min

$t_{green} = t_{yellow} = t_{pink} = 2$ min

Table 3. Case IV: Results for Scenario 2, simulations for a horizon time = 1 day

	Blocking middle way by only one AGV	Both AGVs use the same entrance and exit	Both AGVs use the same entrance and exit + flexible AGVs
Min. Lead Time (min)	9,0	5,0	5,0
Max. Lead Time (min)	137,0	86,2	49,2
Avg. Lead Time (min)	35,8	16,8	13,2
Number of batches	186	188	189

Since the middle way is now a bottle-neck resource, it is important to be sharable as possible. The main difference from solution 1 to solutions 2 is the usage of the middle way. If the middle way is blocked by only one AGV, the resource is not as efficient as if both AGVs could use it.

This can be seen in the average lead times, where both solutions where the middle way is used by both AGVs are less than half than the first solution. Therefore, if the time that the AGV stays in the "bridge" is high, it is advisable to use just one entrance and one exit to improve the efficiency of the system.

Scenario 3: Overused system, where arrival times are shorter than the vehicle operational times.

$t_{arrival1} = t_{arrival2} = \exp(5)$ min

$t_{red1} = t_{blue1} = t_{red2} = t_{blue2} = 1$ min

$t_{load1} = t_{unload1} = t_{load2} = t_{unload2} = 1$ min

$t_{green} = t_{yellow} = t_{pink} = 0{,}33$ min

Table 4. Case IV: Results for Scenario 3, simulations for a horizon time = 1 day

	Blocking middle way by only one AGV	Both AGVs use the same entrance and exit	Both AGVs use the same entrance and exit + flexible AGVs
Min. Lead Time (min)	4,0	3,3	3,3
Max. Lead Time (min)	103,1	100,1	71,8
Avg. Lead Time (min)	37,3	36,0	34,9
Number of batches	551	552	553

In this case, we can start to see more differences between the three solutions. As it is expected, using flexible AGVs improves notoriously the system. This happens because the unload stations receive lots of cargo in comparison with the first scenario, and as they are random functions, one station might be hugely overloaded while the other one may be empty. The fact that both AGVs can work with both unload stations makes the cargo have lower lead times in the system.

Scenario 4: Unbalanced system, where parts “1” arrive faster than parts “2”

$t_{arrival1}$ = exp(5) min / $t_{arrival2}$ = exp(15) min

$t_{red1} = t_{blue1} = t_{red2} = t_{blue2}$ = 1 min

$t_{load1} = t_{unload1} = t_{load2} = t_{unload2}$ = 1 min

$t_{green} = t_{yellow} = t_{pink}$ = 0,33 min

Table 5. Case IV: Results for Scenario 4, simulations for a horizon time = 1 day

	Blocking middle way by only one AGV	Both AGVs use the same entrance and exit	Both AGVs use the same entrance and exit + flexible AGVs
Min. Lead Time (min)	4,0	3,3	3,3
Max. Lead Time (min)	69,8	63,1	38,2
Avg. Lead Time (min)	22,2	19,9	10,4
Number of batches	362	362	363

This scenario demonstrates even a more unbalanced situation, where one unload station receives much more cargo than the other. Flexible AGVs are the best solution in this case.

As it can be seen in Table 5, the average lead time is half than the other two solutions. This happens because both AGVs are going to the unload station 1 frequently, as it gets often overloaded. In the other situations, the AGV that is waiting in the other station remains idle for some time. Additionally, it is shown in Figure 48 than in the long run, the difference between the flexible solution and the other ones gets notorious, which partially explains the effort of the current industries into get more flexible plants.

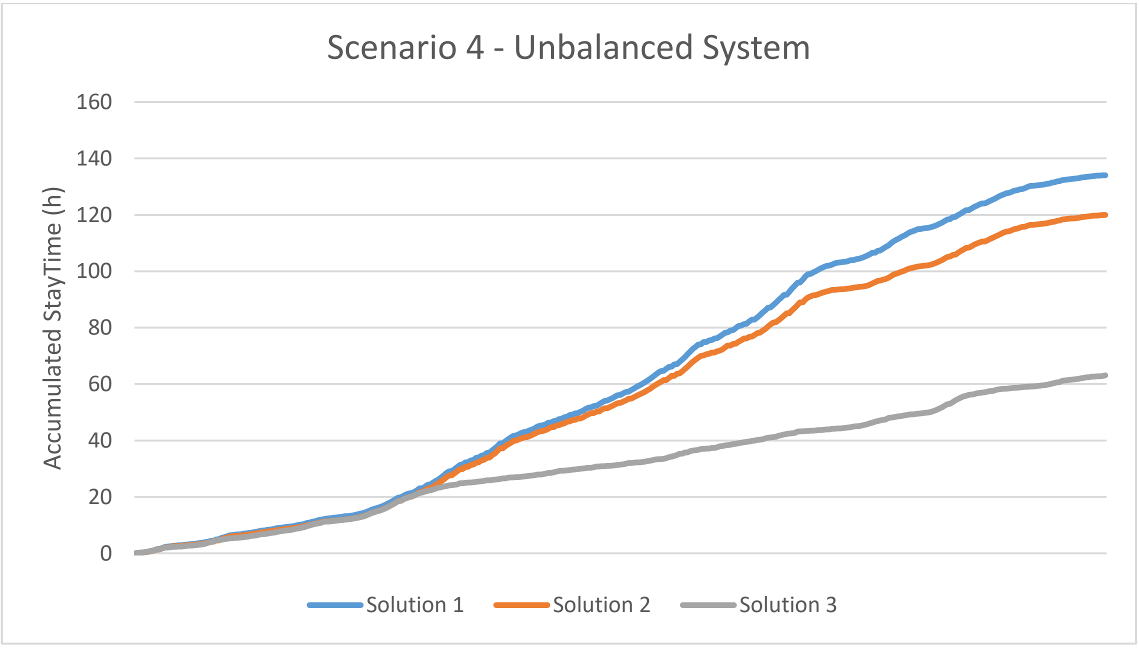


Figure 48. Case IV: Accumulated Lead time for the three proposed solutions in Scenario 4 – Unbalanced System

6.5. Case V: Shop deadlock

Shop deadlocks can occur when the vehicles are dispatched carelessly in an overloaded "shop".

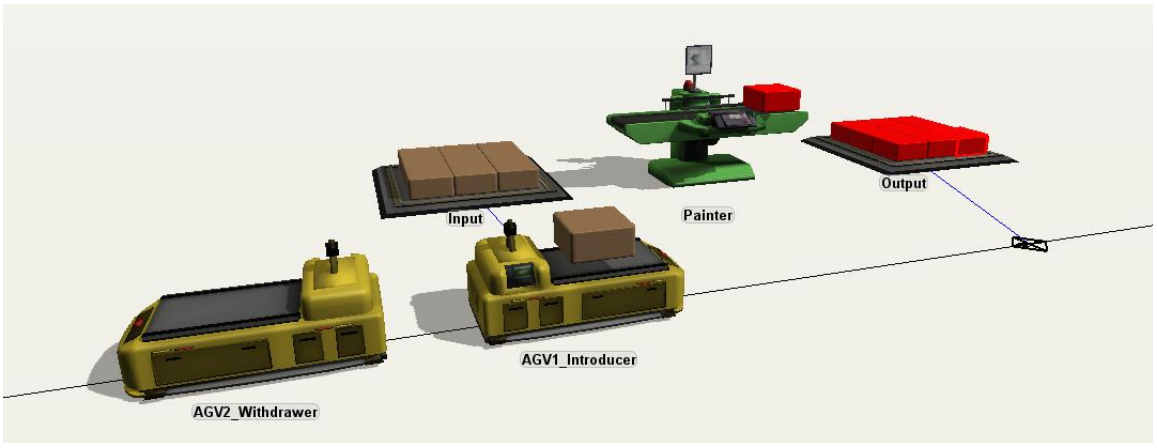


Figure 49. Case V: Screenshot of an example of Shop Deadlock, represented with FlexSim

In this area, illustrated in Figure 49, both storage areas ("input" and "output") have a limited amount of capacity of nine boxes. If all the storage space is used, AGV1 (Introducer) which introduces boxes into the input buffer is waiting for this to be liberated, and AGV2 (Withdrawer) cannot withdraw any box from the output place. The painter, which transforms the boxes from the input to the output, cannot also liberate any new box neither the box that has already been processed, keeping busy the machine. [7]

6.5.1. Place Transition Petri Net Model

As all previous cases where there are AGVs and roads involved, the road separation and assignation is important to simulate finitely the case. Again, every inch of the tracks should be a resource, but this is neither helpful nor able to be simulated.

Therefore, this case is no different and it must be divided in different sections regarding the usage of the path by the AGVs. It has been decided to keep it as simple as possible. As it is seen in Figure 50, the road can be divided in the "red path" which corresponds to the input area and the "blue path" to the output area.

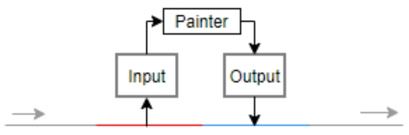


Figure 50. Case V: Possible different sectors for AGV network

Now that each path has been labeled, the Petri Net can be constructed. Even that the process seems easy, in the previous figure, if the system is analyzed with detail it can be seen that three different sub-processes are happening in this simple diagram. The Introducer AGV is in charge of loading cargo in the input station, the Withdraw AGW is in charge of unloading the output one, and finally the painter paints the cargo. This whole process gives as a result the diagram shown in Figure 51.

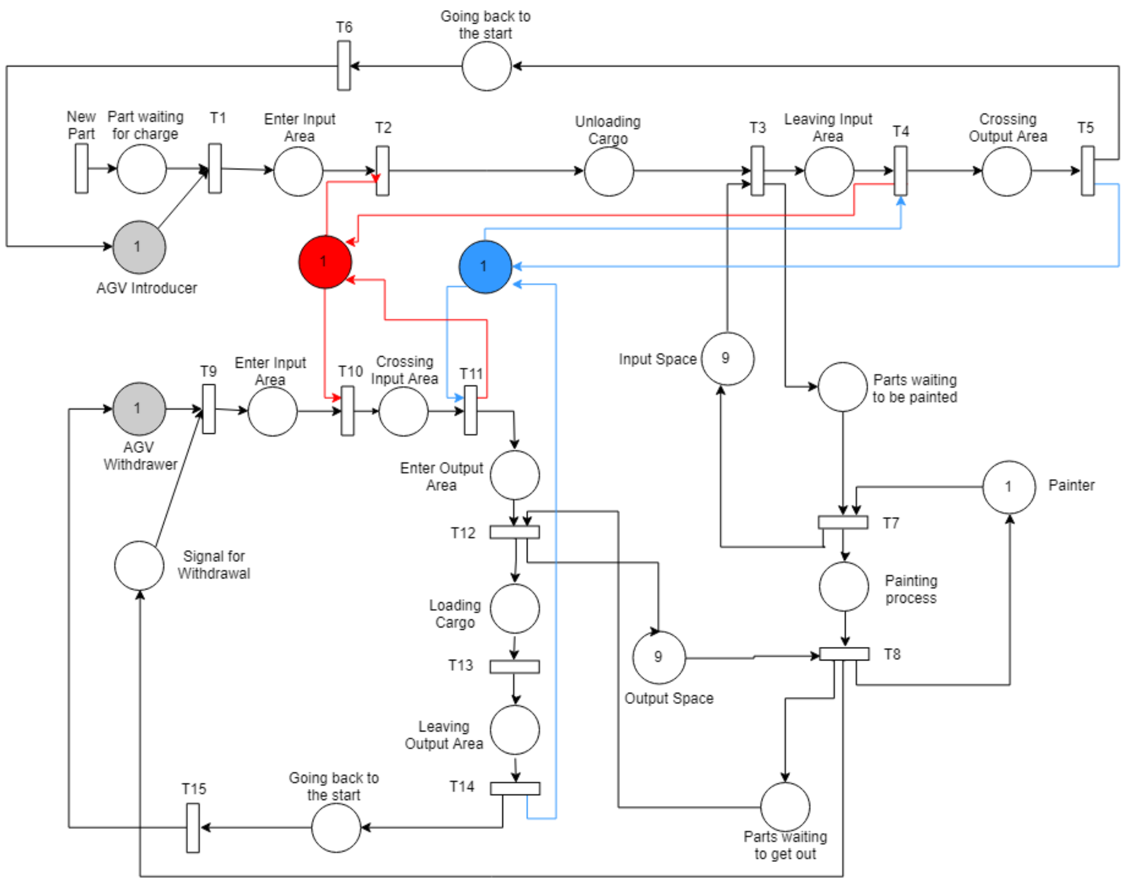


Figure 51. Case V: Initial marking of Petri Net

As usual, this system is not bounded because of the constant entry of cargo into the system. In Moreover, in order to reduce the number of states to get a more visible reachability graph, the capacity of the existing unload and load stations will be reduced to 1 each one.

In order to get the deadlock, will need to cover all possible places with cargo, so 4 pieces will be introduced at first (that corresponds to 1 piece for the introducer AGV, 1 piece for the load station, other one for the unload station, and finally one being painted). Additionally, when the AGV withdrawer gets the piece from the output station, a new piece will become available to be painted again. This way, the system gets bounded.

These changes are represented in Figure 52, marked with the pink color. Even using that reduction the PIPE software is not able to represent the whole reachability graph (There are 219 states with 468 arcs. The graph is too big to be displayed properly). More reduction has to be performed, and that is only achievable if we change the problem approach a little bit.

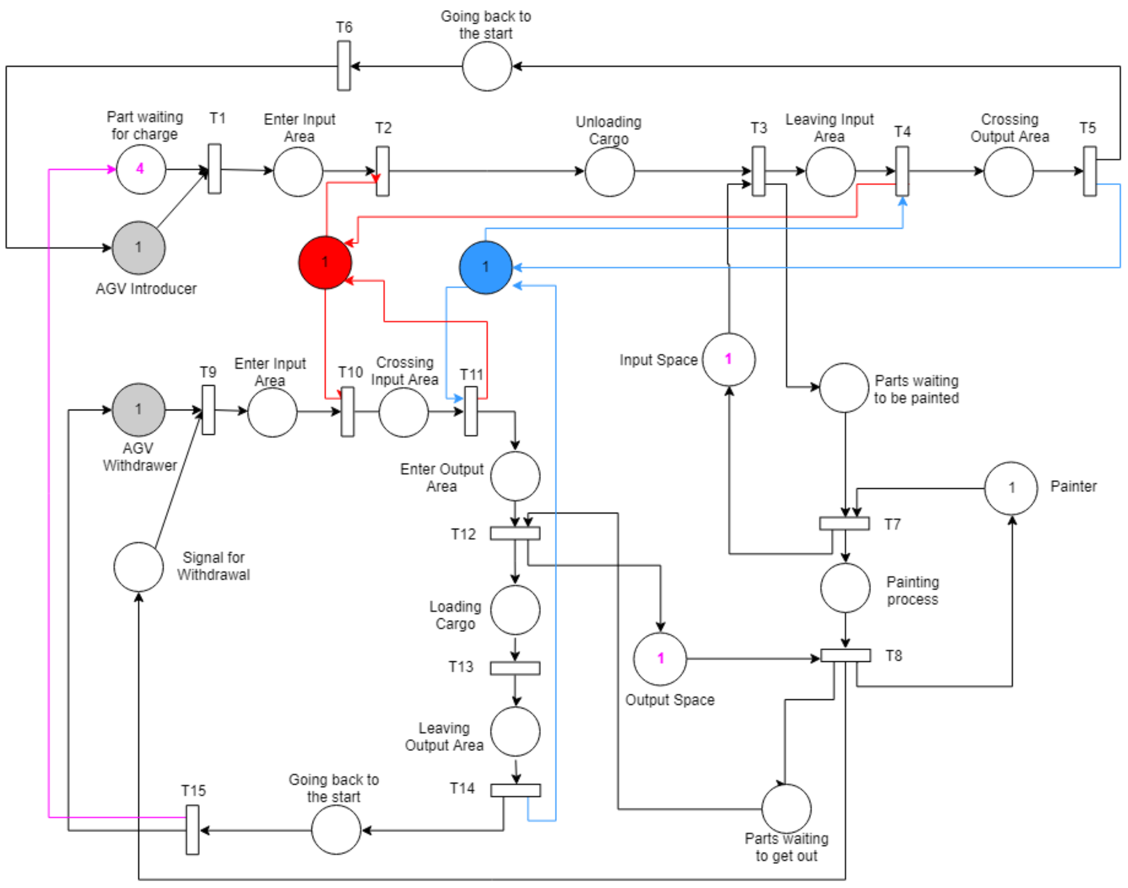


Figure 52. Case V: Initial marking of Petri Net, when the problem is bounded and reduced

The first step could be to reduce the problem grouping some activities in the diagram. Grey paths are no longer considered separate activities but just transition times. This makes more difficult to see in which state the whole process is in each time, yet it has the advantage of making an equivalent diagram that is smaller.

However, in this case, this equivalency is needed because of the PIPE software limitation in order to obtain the reachability graph. If the reduced system gets a deadlock, the original one will certainly have also the same phenomena.

The mentioned step reduces considerably the Petri Net from 16 states (+ 7 resources) to 9 + 7. The new studied diagram is located as it follows in Figure 53, along with the correspondent reachability graph represented in Figure 54.

In this last figure, S36 is clearly a deadlocked state, which confirms the suspected problem imagined for the initial situation drawn in Figure 49. As predicted, the problem comes when the AGV introducer is trying to load cargo for being painted, while the input area is full, the painter is being used and the output space is also full. AGV introducer waits the input area to be freed somehow, but the AGV withdrawer cannot reach the output area because the introducer is blocking it. Therefore, a deadlock is occurring.

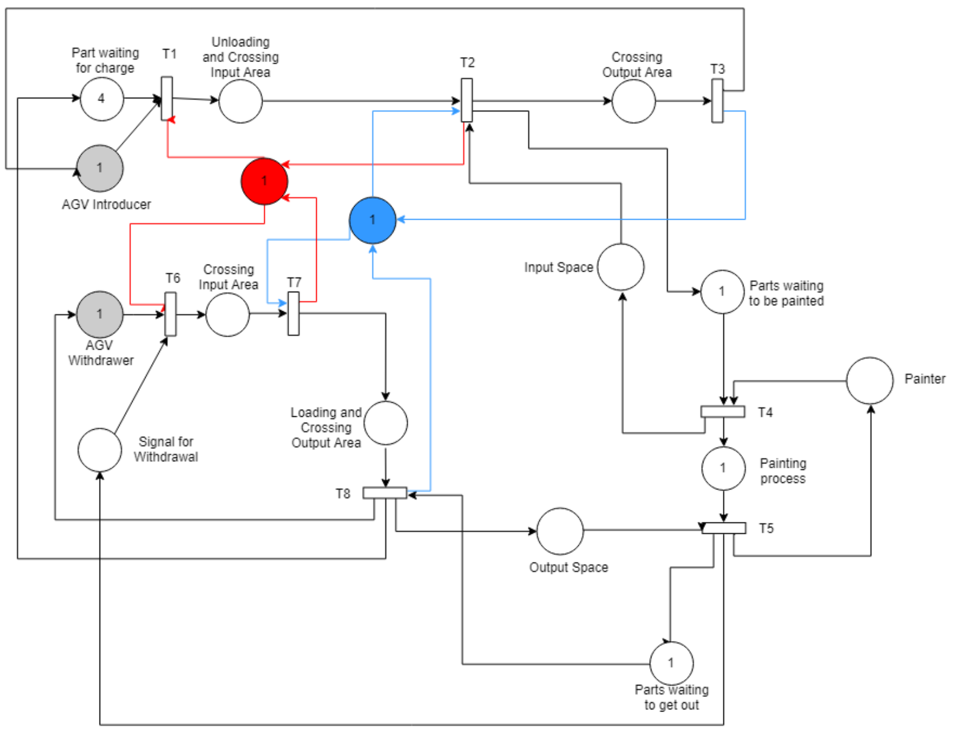


Figure 53. Case V: Initial marking of reduced Petri Net

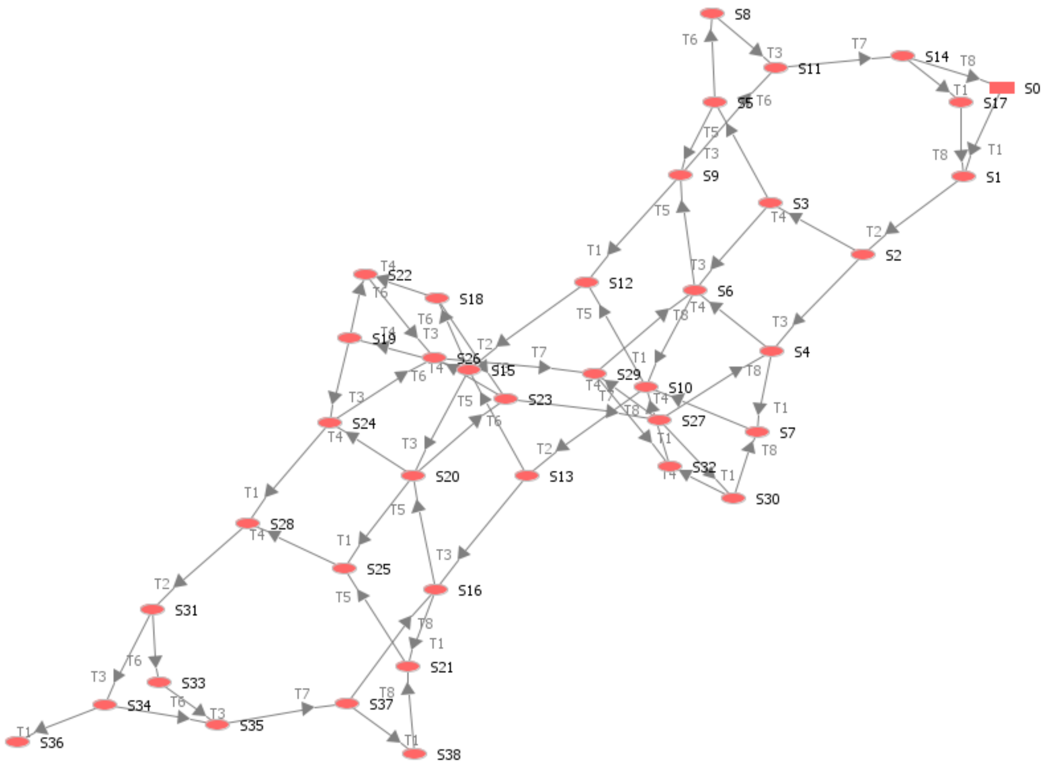


Figure 54. Case V: Reachability/Coverability Graph obtained by PIPE simulator

6.5.2. Possible solutions

The system gets deadlocked when the system is full (9 boxes in the load station, 9 boxes in the unload station and 1 box in the painter) and the first AGV that enters is the introducer. This is the only situation where the deadlock can happen, and this situation must therefore be avoided.

Note there are several structural solutions in this case (which might fit the best). For instance, an additional parallel lane for the AGV withdrawer to advance the stuck AGV introducer. Other possible solutions are changing the route in a way in which both AGVs do not interact between themselves. However, in the scope of this work, no structural solutions can be proposed.

Before proposing solutions, a little assumption must be made to concrete the problem. Figure 50 show the studied system in this case, but what exists out of it might influence in the capabilities of using one solution or not. All proposed solutions are based on the following premises that take place outside the system:

- There is a space where the AGV Introducer can remain still without blocking the Withdrawer
- There is a space where the AGV Withdrawer can remain still without blocking the Introducer

An example of this system could be this one:

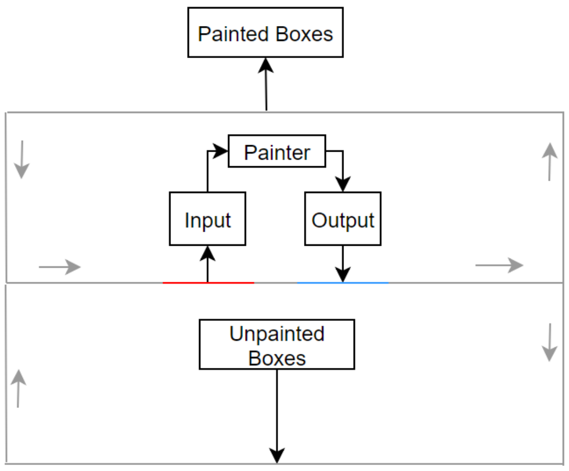


Figure 55. Case V: Example of system outside the painting boundaries

Here are some possible solutions:

Input Area Not full

Let the AGV introducer in if the input area is not full. This happens when the input area contains less than 9 boxes. If this is not the case, the AGV introducer shall wait outside until this conditional happens.

System not full

Let the AGV introducer in if the system has < 19 boxes. The vehicle might occasionally have to wait to the painter to finish its job and free one space in the input area, but for sure it will eventually happen because the system is not completely full.

Try while not blocking

This approach works the same as the previous one, but if the withdrawer arrives and the introducer is idle waiting for the cargo unload, the introducer should exit the system and try again later to let the withdrawer do its job.

Try-and-repeat

If the input area is full and the AGV introducer is trying to unload boxes, let the AGV introducer get out the system and try again later. This means that the vehicle might exit the system with the cargo that is needed to be painted and enter again with the same box.

6.5.3. FlexSim process simulation flow

So as to compare all proposed solutions FlexSim will be used to get results for different state systems as the previous case.

All previous information (Petri Nets) is used along with the explanation exposed in section 5. All five cases and solutions will be simulated. To recapitulate:

- Original case (possible deadlock)
- Solution 1 (Input Area not full)
- Solution 2 (System not full)
- Solution 3 (Try while not blocking)
- Solution 4 (Try-and-repeat)

The constructed diagrams are shown as it follows:

Original Case (possible deadlock)

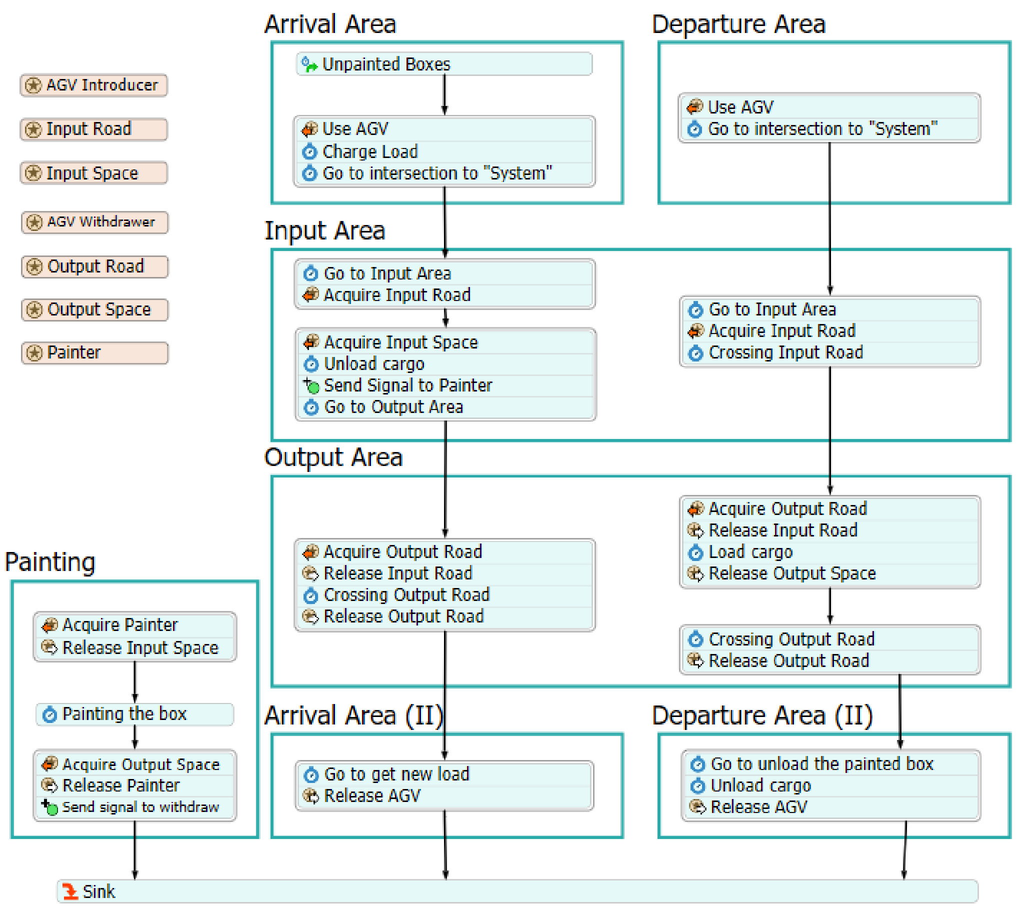


Figure 56. Case V: Process created in FlexSim for the original problem

Input Area not full

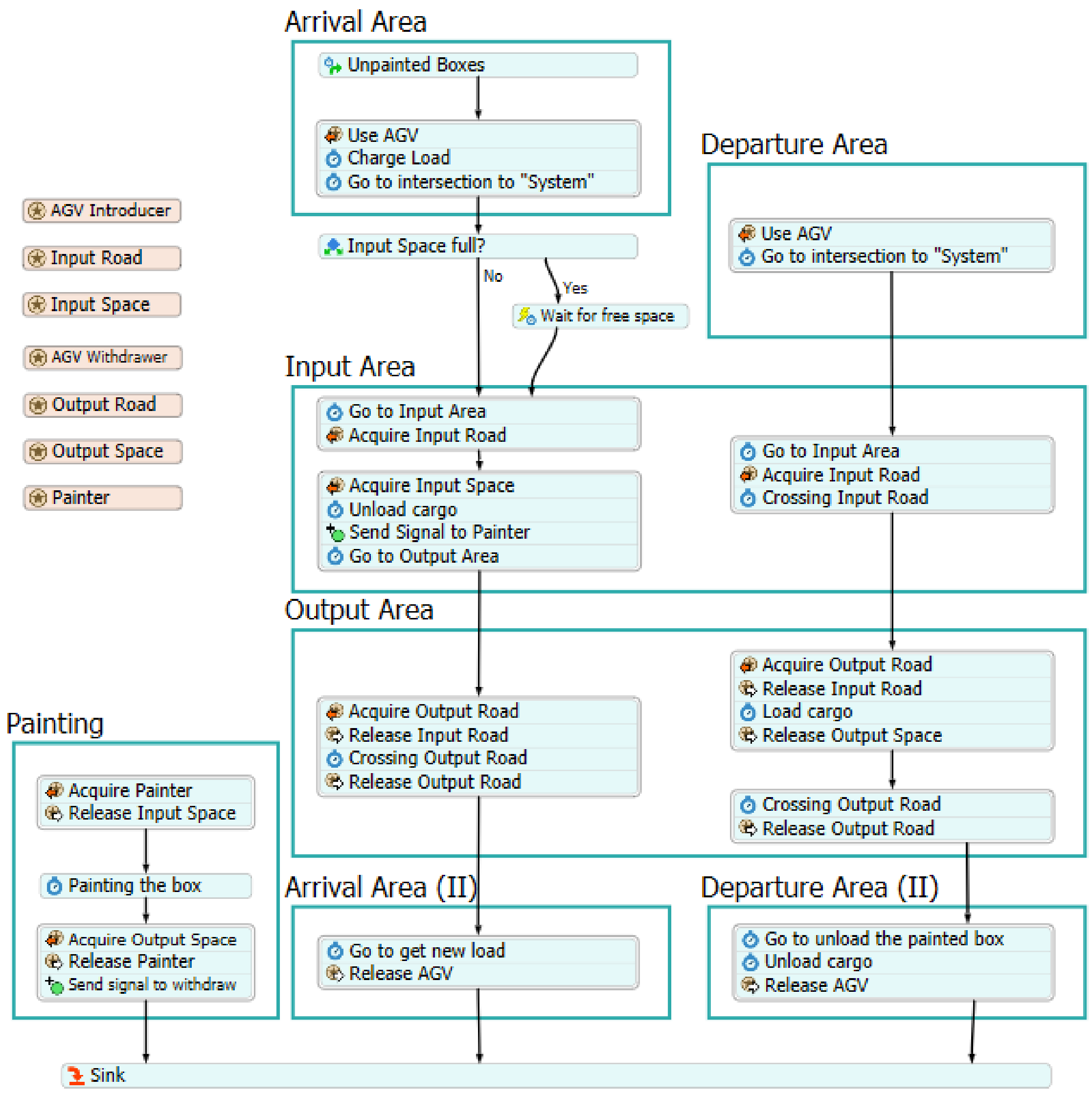


Figure 57. Case V: Process created in FlexSim for the first proposed solution: input area not full

System not full

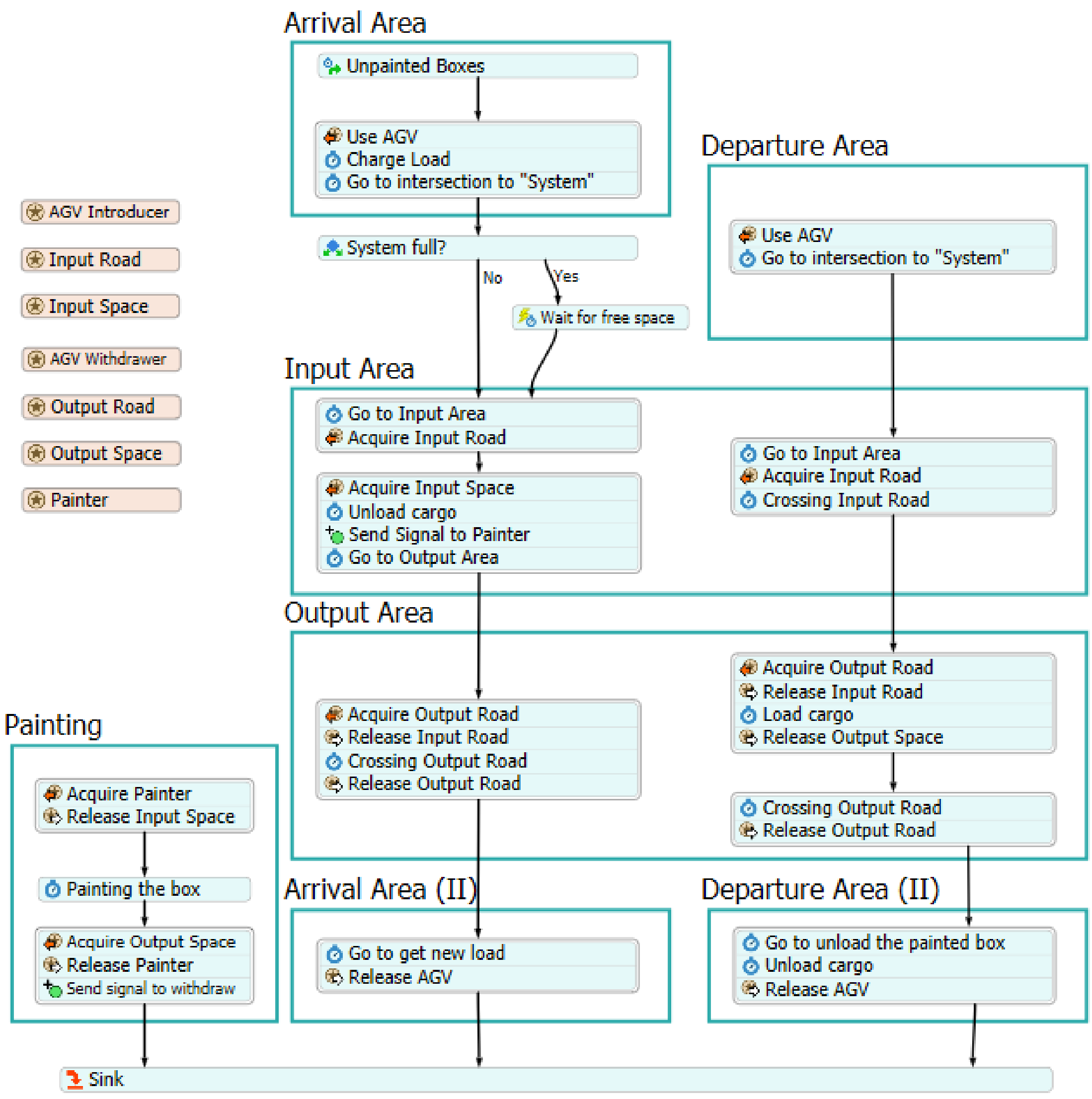


Figure 58. Case V: Process created in FlexSim for the first proposed solution: system not full

Try while not blocking

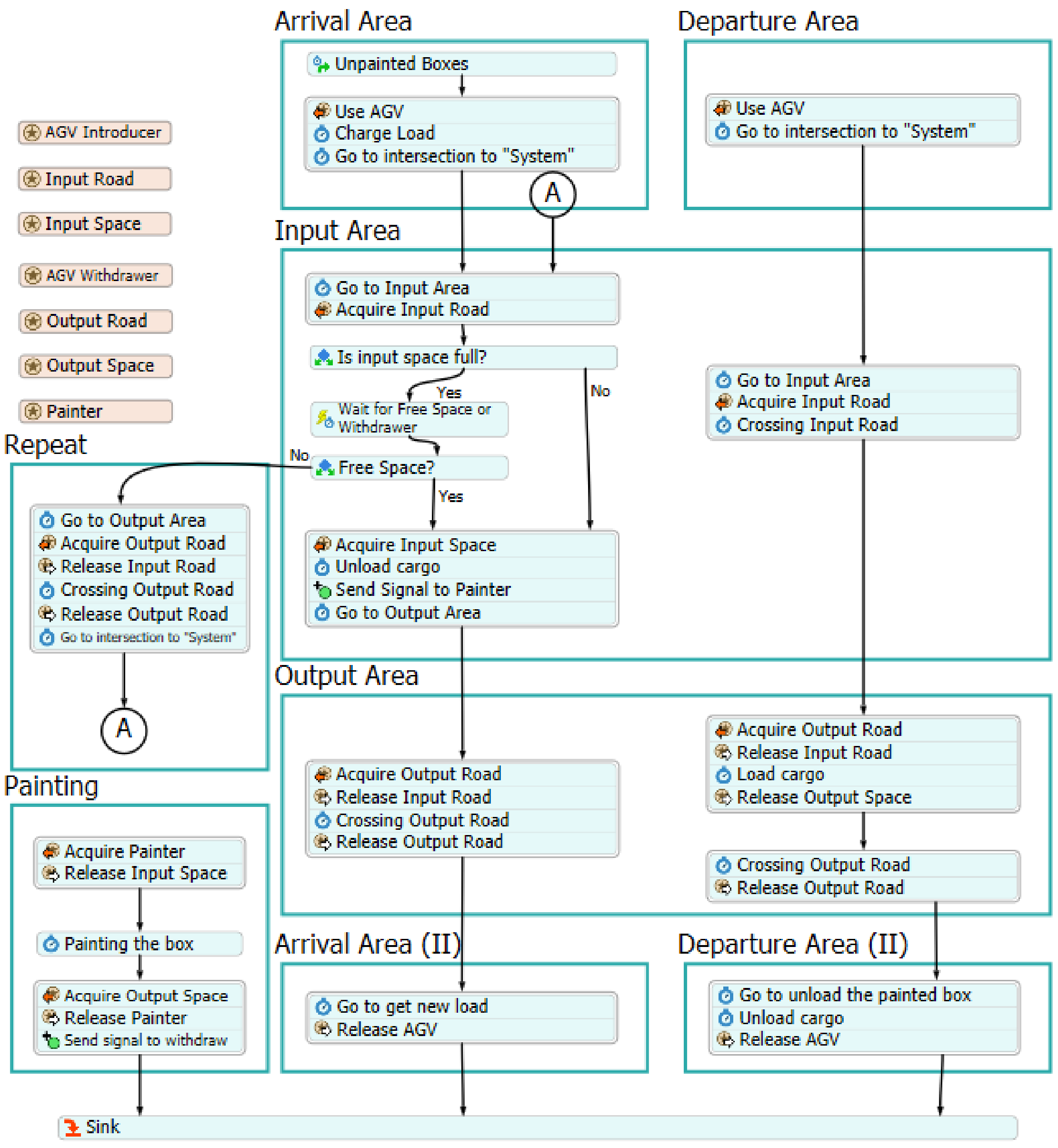


Figure 59. Case V: Process created in FlexSim for the first proposed solution: try while not blocking

Try-and-repeat

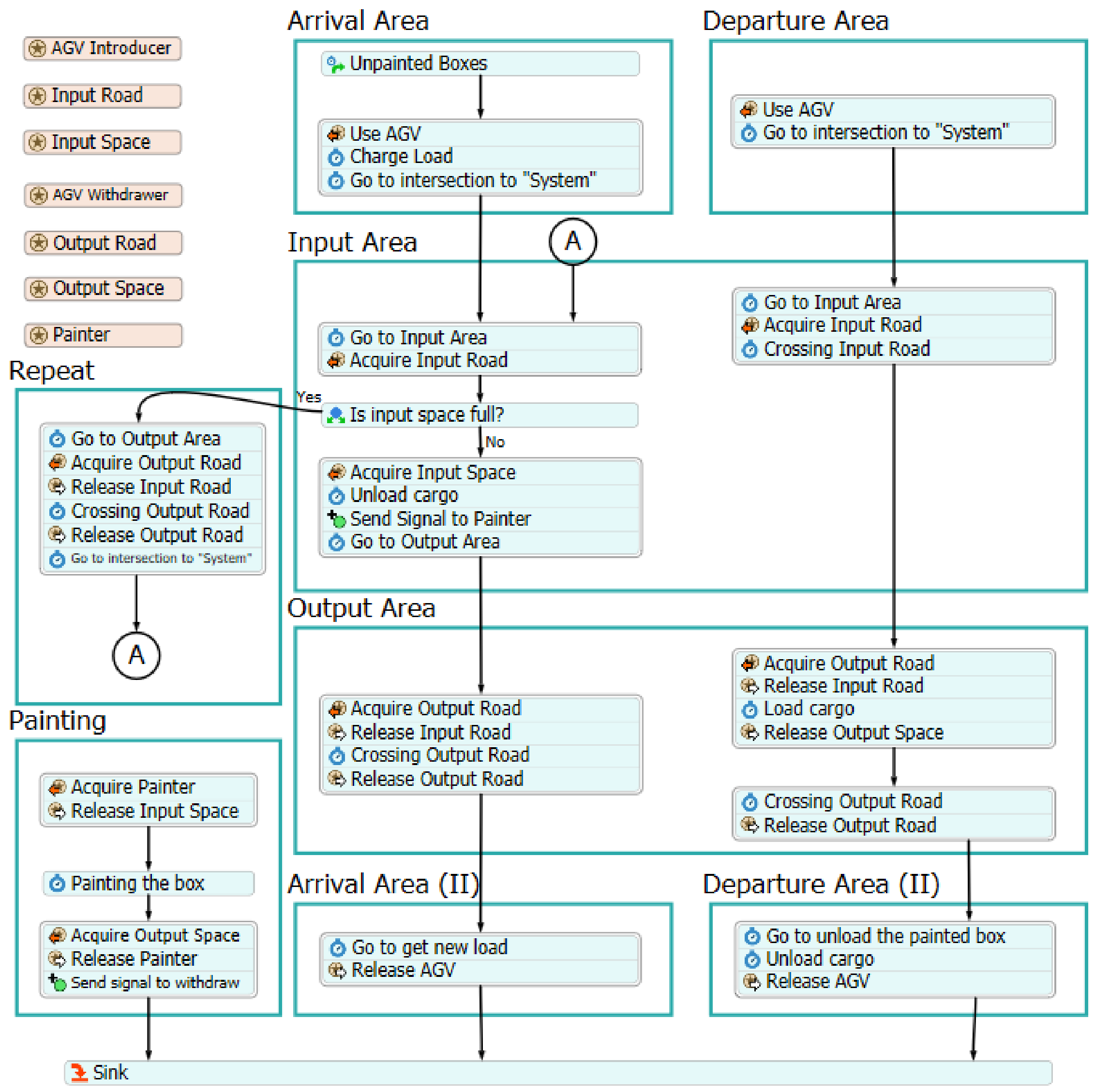


Figure 60. Case V: Process created in FlexSim for the first proposed solution: try and repeat

6.5.4. Results and discussion

As the previous case, this problem comes across the same "issue". As a generic case there are no stipulated time values. So, in order to get some results, various cases with different values must be studied.

At first, it is relatively easy to get a deadlock in our system. If the AGV withdrawer is significantly slower than the AGV introducer, and there is enough input of unpainted boxes, the deadlock can be encountered as it is shown in Figure 61. What this figure shows is the detected deadlock from before, where the AGV introducer cannot acquire input space because is full, but the AGV withdrawer cannot withdraw any box from the system since the previous vehicle is blocking the road.

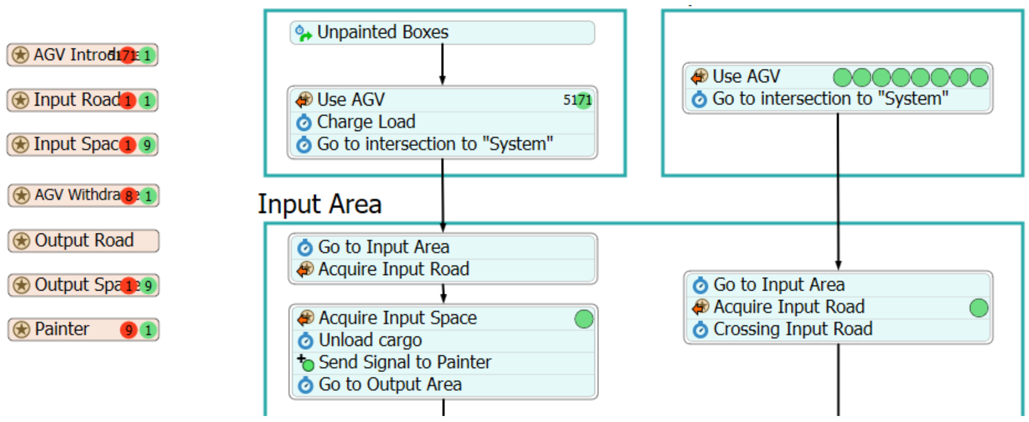


Figure 61. Case V: Deadlock represented in FlexSim

The different times to declare can be stipulated to be quite parallel between themselves, as it is shown in Figure 62 and summarized in Table 6. Note that the figure is not in scale and the shape of the factory is not even represented.

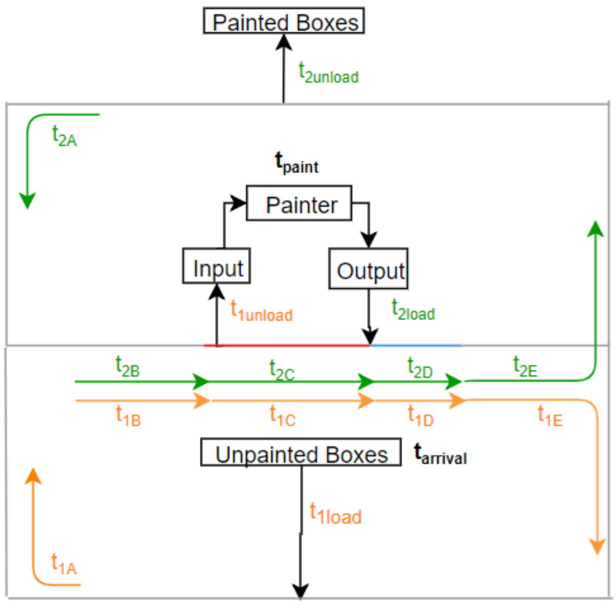


Figure 62. Case V: Different time concepts with the final sectors of the AGV network represented in a diagram

Table 6. Case V: Different time concepts and its variable assignation

AGV Introducer (1)		AGV Withdrawer (2)	
Concept	Variable	Concept	Variable
Load Unpainted Box	t_{1load}	Unload Painted Box	$t_{2unload}$
Unload Unpainted Box	$t_{1unload}$	Load Painted Box	t_{2load}
Go to intersection	t_{1A}	Go to intersection	t_{2A}
Go to Input Point	t_{1B}	Go to Input Point	t_{2B}
Go to Output Point (Cross Input Area)	t_{1C}	Go to Output Point (Cross Input Area)	t_{2C}
Cross Output Area	t_{1D}	Cross Output Area	t_{2D}
Go to starting point	t_{1E}	Go to starting point	t_{2E}
Painting the box	t_{paint}	Painting the box	$t_{arrival}$

As the Bridge crossing deadlock, different ways exist to see the good planning of the whole system. For example, the idle time of the AGV's, or the time that a box is not being used (neither painted nor transported). However, the same key performance indicator will be used as the previous case, since the better output of this system is to optimize the time that passes from when the unpainted box is picked by the AGV introducer and the exiting of the same box but painted.

Even that Figure 49 shows nine boxes in both input and output areas, the simulated scenarios will have just space for two boxes. This is done to reach the deadlock situation in a shorter time, and to not record unnecessary lead times.

Scenario 1: Underused system, where the arrival of boxes can be assumed by the system

$t_{arrival}$ = exp(15) min

$t_{2unload} = t_{1unload} = t_{1load} = t_{2load} = 1$ min

$t_{1X,\ 2X}$ = 1 min for X= A, B, C, D, E

t_{paint} = 1 min

Boxes in input area = Boxes in output area = 2

Table 7. Case V: Results for Scenario 1, simulations for a horizon time = 4 days

	Original case	Input Area not full	System not full	Try while not blocking	Try-and-repeat
Min. Lead Time (min)	12,00	12,00	12,00	12,00	12,00
Max. Lead Time (min)	12,00	12,00	12,00	12,00	12,00
Avg. Lead Time (min)	12,00	12,00	12,00	12,00	12,00
Number of batches	365	365	365	365	365

As it is expected, if the system is way underused, the system will not encounter any deadlock, and therefore all deadlock-solving systems are equivalent to the original problem. If this were the case, there would be no need to create a deadlock avoidance algorithm although this is not recommended since the system become susceptible to any deadlock if there is an unexpected alteration.

Scenario 2: AGV Withdrawer has a longer path than the Introducer outside the main process

$t_{arrival}$ = exp(15) min

t_{1X} = 1 min for X= A, B, C, D, E = $t_{2B,2C,2D}$

$t_{2A,2E}$ = 4 min 30 sec

$t_{2unload} = t_{1unload} = t_{1load} = t_{2load}$ =1 min

t_{paint} = 1 min

Boxes in input area = Boxes in output area = 2

Table 8. Case V: Results for Scenario 2, simulations for a horizon time = 4 days

	Original case	Input Area not full	System not full	Try while not blocking	Try-and-repeat
Min. Lead Time (min)	Deadlocked at box number 14	19,00	19,00	19,00	19,00
Max. Lead Time (min)		85,50	85,50	92,50	85,62
Avg. Lead Time (min)		56,33	56,33	63,49	55,23
Number of batches		364	364	356	364

In this scenario the first deadlock is encountered. This happens because the AGVs are no longer balanced, as the Introducer works at a higher rate than the withdrawer, which originates a bottle neck in the input area. This makes the AGV introducer to be blocked in the input road until some cargo is liberated, and as a consequence, the times for the “Try while not blocking” system are higher than the rest, since the AGV introducer remains still until the other vehicle arrives.

Even it has been taken into account that the same number of batches were delivered at the end of the period, it is bad for the system to have higher lead times for the boxes. Additional boxes arriving at higher rate will make the differences between algorithms grow.

It can also be noted than there are no notorious differences between the other three algorithms, so each solution will be appropriate for the system.

Scenario 3: Higher loading and unloading times for a painted box

$t_{arrival}$ = exp(15) min

$t_{1X,\ 2X}$ = 1 min for X= A, B, C, D, E

Boxes in input area = Boxes in output area = 2

$t_{1unload} = t_{1load}$ = 1 min

$t_{2unload} = t_{2load}$ = 4 min

t_{paint} = 1 min

Table 9. Case V: Results for Scenario 3, simulations for a horizon time = 4 days

	Original case	Input Area not full	System not full	Try while not blocking	Try-and-repeat
Min. Lead Time (min)	Deadlocked at box 177	18,00	18,00	18,00	18,00
Max. Lead Time (min)		79,00	79,00	86,00	80,00
Avg. Lead Time (min)		41,40	41,40	46,28	41,11
Number of batches		364	364	364	364

This is a similar scenario than before, but the higher times correspond to the unload/load process of the AGV withdrawer. For the same reason, the “Try while not blocking” is also punished to be the worst algorithm. The other solutions are also pretty similar between themselves, almost tied.

Scenario 4: First realistic case (random times for scenario 3 except for transport)

$t_{arrival}$ = exp(15) min

$t_{1X,\ 2X}$ = 1 min for X= A, B, C, D, E

Boxes in input area = Boxes in output area = 2

$t_{1unload} = t_{1load}$ = exp(1) min

$t_{2unload} = t_{2load}$ = exp(4) min

t_{paint} = exp(1) min

Table 10. Case V: Results for Scenario 4, simulations for a horizon time = 4 days

	Original case	Input Area not full	System not full	Try while not blocking	Try-and-repeat
Min. Lead Time (min)	Deadlocked at box 15	10,38	10,38	9,10	9,10
Max. Lead Time (min)		112,42	112,42	115,45	112,72
Avg. Lead Time (min)		46,61	46,61	49,22	47,51
Number of batches		363	363	363	363

In order to represent a more real case, exponential distributions have been applied to all times except for transport in scenario 3. Even though the times have changed considerably, there is no difference in the drawn conclusion. “Try while not blocking” is still the worst algorithm for the same reason as the previous scenario.

Scenario 5: Scenario 4 + High painting times

$t_{arrival}$ = exp(15) min

$t_{1X,\ 2X}$ = 1 min for X= A, B, C, D, E

Boxes in input area = Boxes in output area = 2

$t_{1unload} = t_{1load}$ = exp(1) min

$t_{2unload} = t_{2load}$ = exp(4) min

t_{paint} = exp(10) min

Table 11. Case V: Results for Scenario 5, simulations for a horizon time = 4 days

	Original case	Input Area not full	System not full	Try while not blocking	Try-and-repeat
Min. Lead Time (min)	Deadlocked at box 9	13,13	13,13	12,01	12,01
Max. Lead Time (min)		143,83	143,24	143,55	145,72
Avg. Lead Time (min)		63,45	67,70	65,98	64,17
Number of batches		362	362	359	360

Now that the painting process is high enough, differences can be spotted now from the “System not full” algorithm to the “Input Area not full”, being the first the worst of all of them, even that the “Try while not blocking”.

This is because of the high idle time of both AGVs waiting until the painter finishes its work. Even that this situation does not originate deadlock, it is not very optimized because the AGV Withdrawer could withdraw cargo while the Introducer is waiting for the painter to get a new box in order to unload the unpainted box.

Therefore, no matter what times are proposed for transportation and processes, but “Input Area not full” is a better algorithm than “System not full” since it avoids this situation.

Scenario 6: Scenario 5 + Longer AGV Paths

$t_{arrival}$ = exp(15) min	$t_{1unload} = t_{1load}$ = exp(1) min
$t_{1X, 2X}$ = 4 min for X= A, B, C, D, E	$t_{2unload} = t_{2load}$ = exp(4) min
Boxes in input area = Boxes in output area = 2	t_{paint} = exp(10) min

Now, with the complete scenario, where optimizing transportation is vital (now it is four minutes each track), an unlikely solution is driven. The "Try-and-repeat" algorithm is better than the "Input Area not full, regarding the average lead time (Note that, however, the "Input Area not full" achieves more number of batches. It happens even for the very long run, as it is seen in Table 12.

Table 12. Case V: Results for Scenario 6, simulations for a horizon time = 100 days

	Original case	Input Area not full	System not full	Try while not blocking	Try-and-repeat
Min. Lead Time (min)	Deadlocked at box 23	42,84	42,84	44,37	44,37
Max. Lead Time (min)		228,51	228,51	237,38	228,11
Avg. Lead Time (min)		160,62	163,11	166,15	151,61
Number of batches		5120	5099	5017	5055

The reasons for the better results are multiple: first of all, they are originated because of the definition of "Lead Time". For the "Input Area not full", the box remains idle more time in the system, whereas for the other algorithm the box remains idle more time in the arrival station (which does not count for the Lead Time). Other reasons are that AGVs sync better between themselves and the painter in some occasions (for instance, the AGV introducer arrives when the painter has just grabbed an unpainted box recently).

This is a clear example of the importance of simulating the processes and proposing different approaches in order to see which the best algorithm is. As it is plotted in Figure 63, lead times remain the same until the deadlock is reached (box 22), from where the "Try-and-repeat" algorithm gets better Lead Times for most boxes.

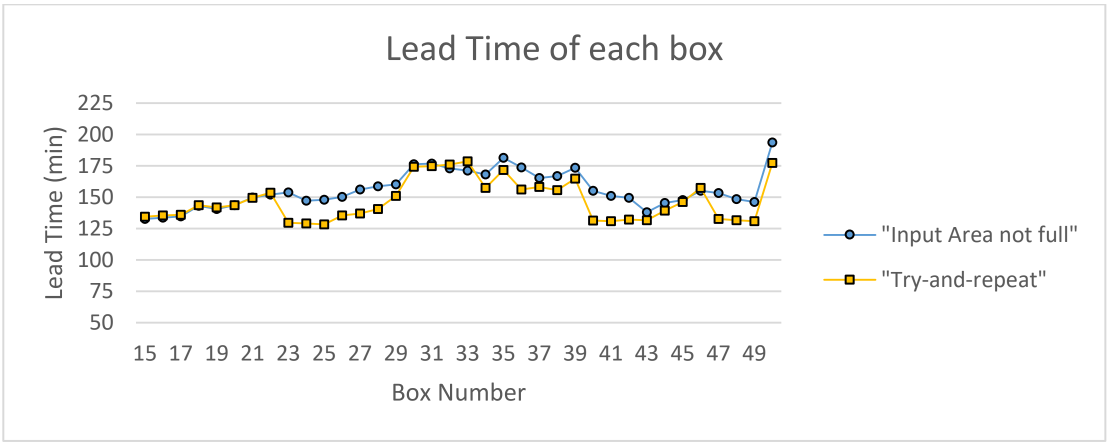


Figure 63. Case V: Lead time of each box, from the 15th to the 50th

6.6. Case VI: Manufacturing system

The system consists of three machines, an AGV and twelve local stations, as shown in Figure 64. Each load-unload station has a capacity of one part and also serves as a pallet storage location. We will assume that each part requires only one. We will also assume that the vehicle can carry only one pallet at a time. Thus, the main assumptions are as follows:

1. The vehicle has only one load carrying position
2. Each part requires only one operation
3. Operations can be performed by any machine
4. Machines do not break down

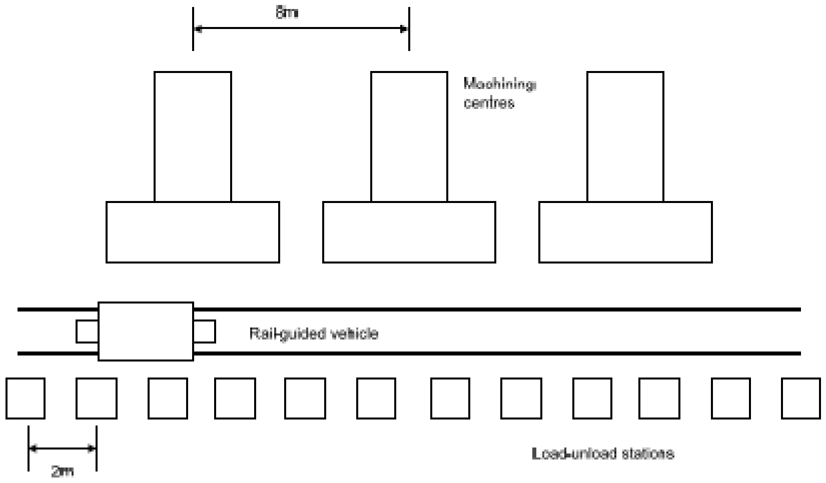


Figure 64. Case VI: Diagram of a Flexible Manufacturing System

The flow of actions is as follows:

- The operator loads the part on to the pallet placed in the corresponding load-unload station
- The vehicle moves the pallet with the part to any machine
- The machine processes the part
- The vehicle takes the pallet with the part back to the load-unload station
- The operator unloads the part

We assume that the system works 20 days a month, three shifts per day a 480 min for each shift. Table 1 shows each part number arrival rate for the 20 days period. The time between arrivals is an exponential distribution whose mean has been obtained from the data is in Table 1. This data is also used to compute the discrete probability for each part number.

Table 13. Case VI: Monthly forecast for arrivals

Part Number	Quantity per Month	Machining Time	Probability
1	320	18	0.19
2	270	60	0.16
3	260	36	0.15
4	180	40	0.11
5	140	36	0.08
6	120	24	0.07
7	100	32	0.06
8	80	28	0.05

9	72	20	0.04
10	64	20	0.04
11	48	30	0.03
12	40	32	0.02
TOTAL	1694	-	

The operator part loading or unloading time is 3 minutes. The vehicle transportation time is negligible but the vehicle part loading or unloading time is 30 seconds.

6.6.1. Place Transition Petri Net Model

This system can be modelled using the Place Transition Petri Net formalism. This system is necessary in order to have a more general and schematic view of the system, but more importantly, the detection of the existence of deadlocks.

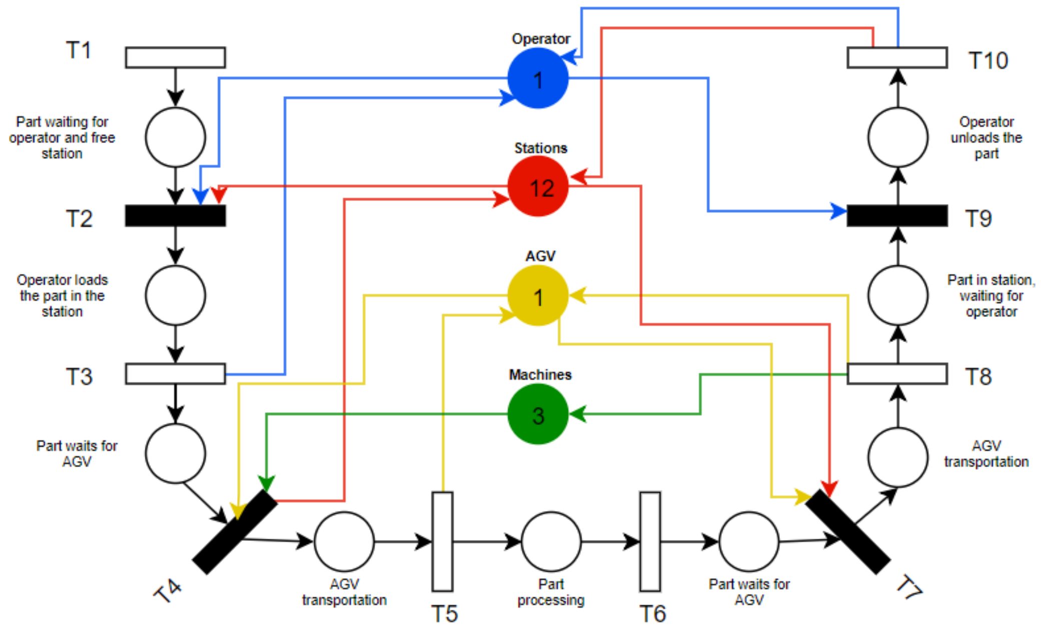

T1
Part waiting for operator and free station
T2
Operator loads the part in the station
T3
Part waits for AGV
T4
Operator
1
Stations
12
AGV
1
Machines
3
AGV transportation
T5
Part processing
T6
Part waits for AGV
T7
T10
Operator unloads the part
T9
Part in station, waiting for operator
T8
AGV transportation

Figure 65. Case VI: Initial marking of Petri Net

In order to model this system using PIPE Petri Net Simulator, so as to detect the possible deadlocks, it needs to be bounded. The system can reach infinite states because of the constant input of parts in the system. One easy way to bind the system is to connect T10 to T1 with a middle place between them. By means of this, whenever a part has been finished and processed, automatically means that a new part comes into the system. We will assume that 16 pieces are entering the system waiting for being processed. This can be shown in Figure 66.Figure 66.

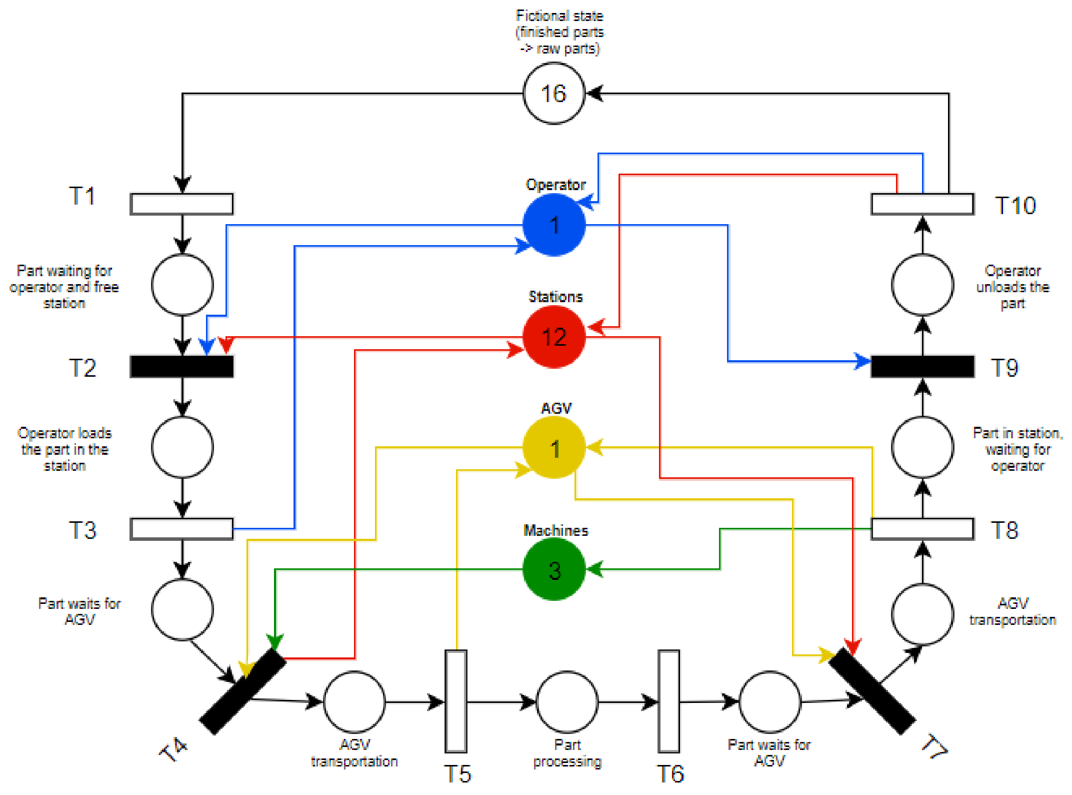

Fictional state (finished parts -> raw parts)
16
T1
Operator
1
T10
Part waiting for operator and free station
Stations
12
Operator unloads the part
T2
T9
AGV
1
Operator loads the part in the station
Part in station, waiting for operator
T3
Machines
3
T8
Part waits for AGV
AGV transportation
T4
AGV transportation
T5
Part processing
T6
Part waits for AGV
T7

Figure 66. Case VI: Initial marking of Bounded Petri Net

Even that now the system has limited states, according to the simulator, there are 23929 states with 46183 arcs. This graphic is impossible to be read by a human in order to find any possible deadlock.

The complexity of the net can be reducing lowering the number of resources to a minimum, in order to minimize the number of states. So as to respect the proportion, there needs to be more stations than machines, and a possible deadlock may occur when all parts are using one resource each one.

Therefore, a possible equivalent situation (deadlock-study-speaking) is the one presented in Figure 67. Using the reachability graphic, we are able to find the node that is shown in Figure 68, which corresponds to a deadlock state that corresponds to this situation:

- 2 pieces waiting for being loaded to the stations
- 2 pieces waiting in the stations for AGV
- 1 piece in the machine waiting for AGV

The AGV cannot unload the piece to the unload station because there are both occupied by pieces that are waiting to be loaded to the already busy machine. This deadlock is scalable to the original problem, since what this means is that if all machines and all stations are busy at the same time, a deadlock occurs.

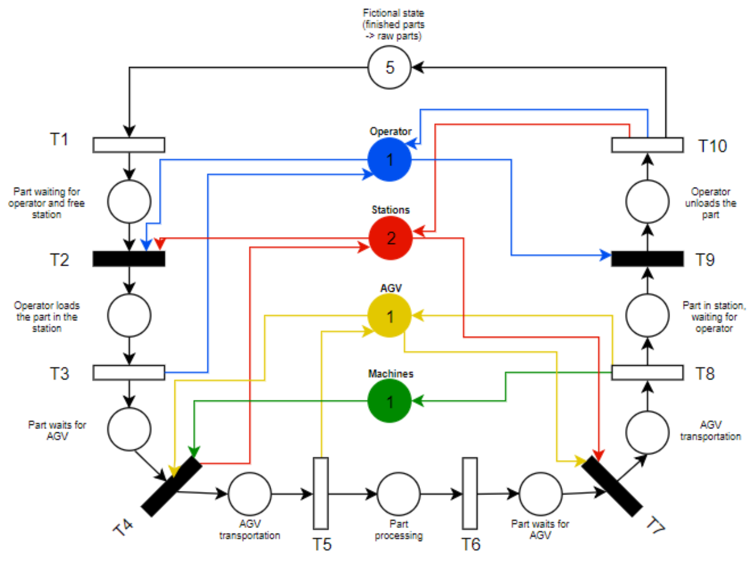

Fictional state (finished parts -> raw parts)
5
T1
Operator
1
T10
Part waiting for operator and free station
Operator unloads the part
Stations
2
T2
T9
AGV
1
Operator loads the part in the station
Part in station, waiting for operator
T3
T8
Machines
1
Part waits for AGV
AGV transportation
T4
AGV transportation
T5
Part processing
T6
Part waits for AGV
T7

Figure 67. Case VI: Initial marking of Bounded and Reduced Petri Net

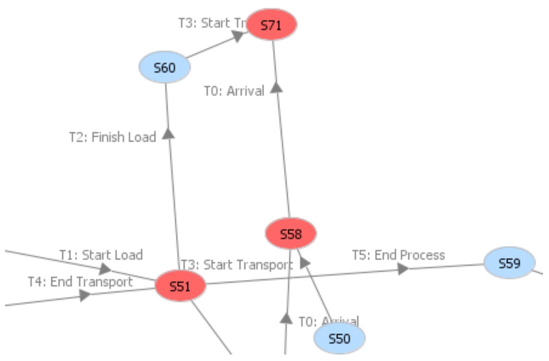


Figure 68. Case VI: Deadlock state in a section from the reachability graph

6.6.2. Possible solutions

There is a clear problem with the capacity of the system regarding the load/unload units. If these units get full while the three machines need to unload in the stations, the system gets completely blocked. Therefore, the way of solving this problem is to prevent deadlocks categorizing the stations, or avoiding deadlocks searching for conditions so as not to block the whole system.

Assigning part number per station

In order to prevent the formation of deadlocks, each station can be associated with a part number. That means: station 3 is blocked by part 3 during all its process (from the loading until the unloading, passing through the processing part).

Assigning part arrival per station

In a more general way, a station can be blocked during all its process by a part, regardless of the part number. This system adds more flexibility to the process as well as prevents the formation of deadlocks.

Assigning "n" load stations and "12 - n" unload stations

Separating the loading stations and the unloading stations is also a way to categorize these resources and preventing systems deadlocks. The operator loads all the parts from certain preassigned stations, and unloads them from the other ones.

Block a station if there are already 3 parts at the machines and 11 on the stations

This possible solution is different from the previous ones, since the system deadlocks are not prevented yet avoided. The deadlock occurs when a piece enters into the system when 11 stations have pending-to-load pieces and 3 parts are at the machines (future unload). An operator can still have all the stations loaded, but only if not all machines are in use. Because of this condition, the circular wait is eliminated dynamically with a condition.

6.6.3. FlexSim process simulation flow

In order to identify the different parts that arrive to the system, a label will be assigned corresponding to the "part number" that belongs to. As it is shown in Table 13, 1694 pieces arrive each month, which corresponds to a piece each 17 minutes.

All different situations correspond to different resources labeling, so all process flows are different among them. The different options are presented as it follows.

Original Case (possible deadlock)

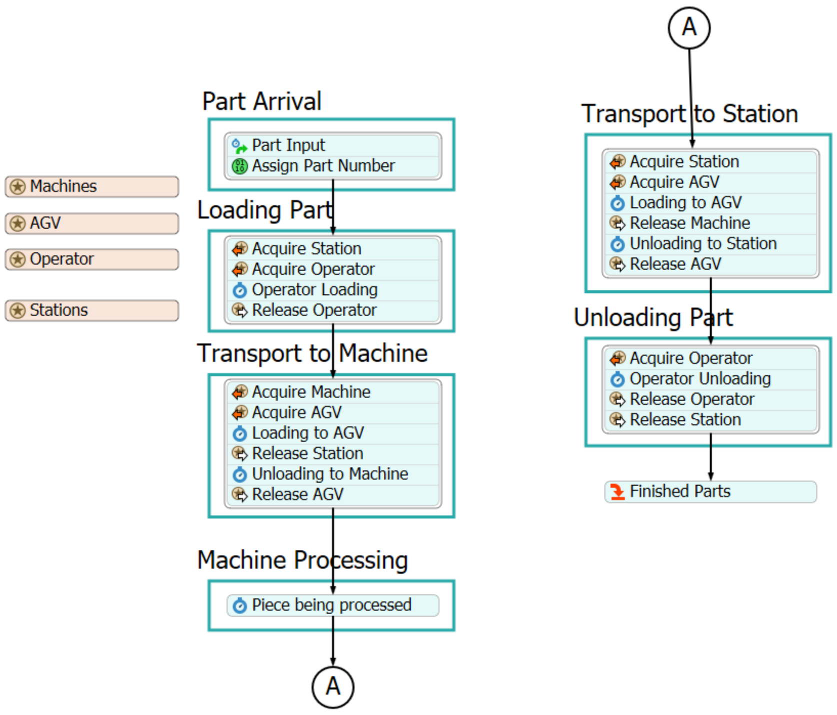


Figure 69. Case VI: Process created in FlexSim for the original problem

Assigning part number per station

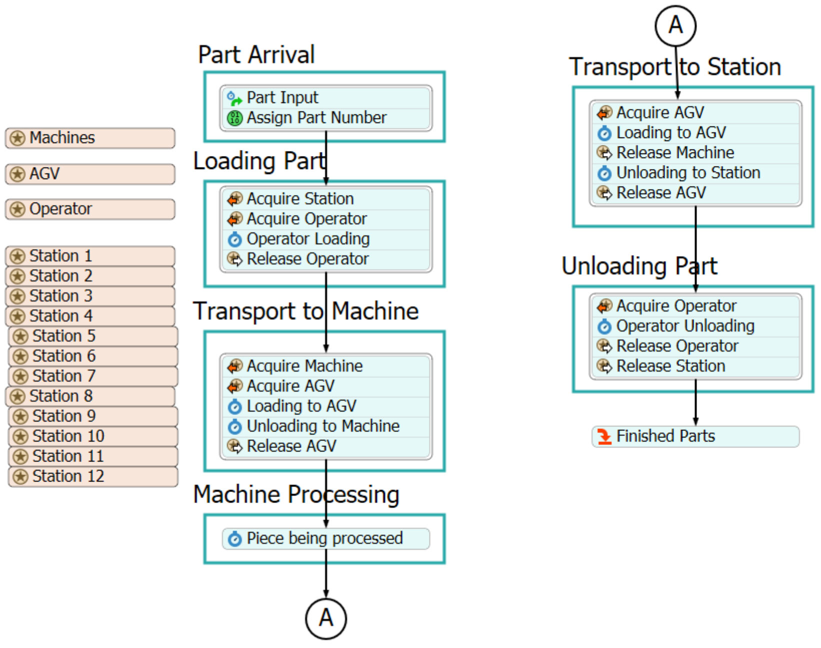


Figure 70. Case VI: Process created in FlexSim for the first proposed solution: assign part number per station

Assigning part arrival per station

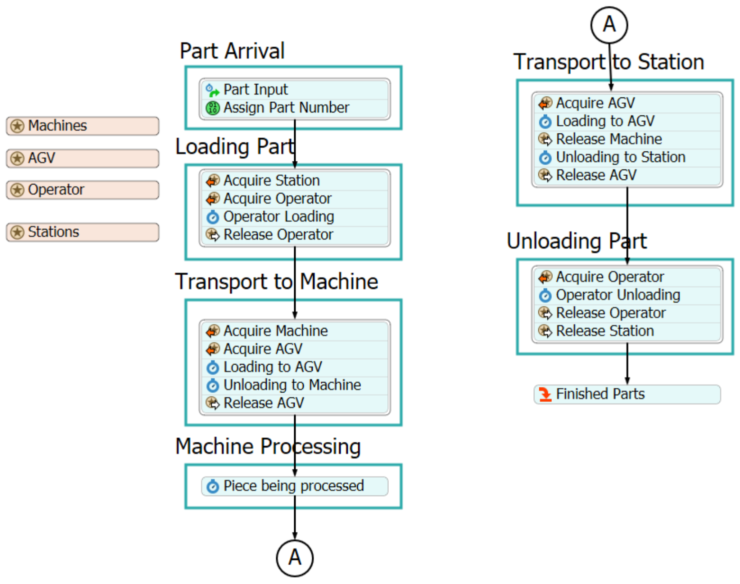


Figure 71. Case VI: Process created in FlexSim for the first proposed solution: assign part arrival per station

Assigning "n" load stations and "12 - n" unload stations

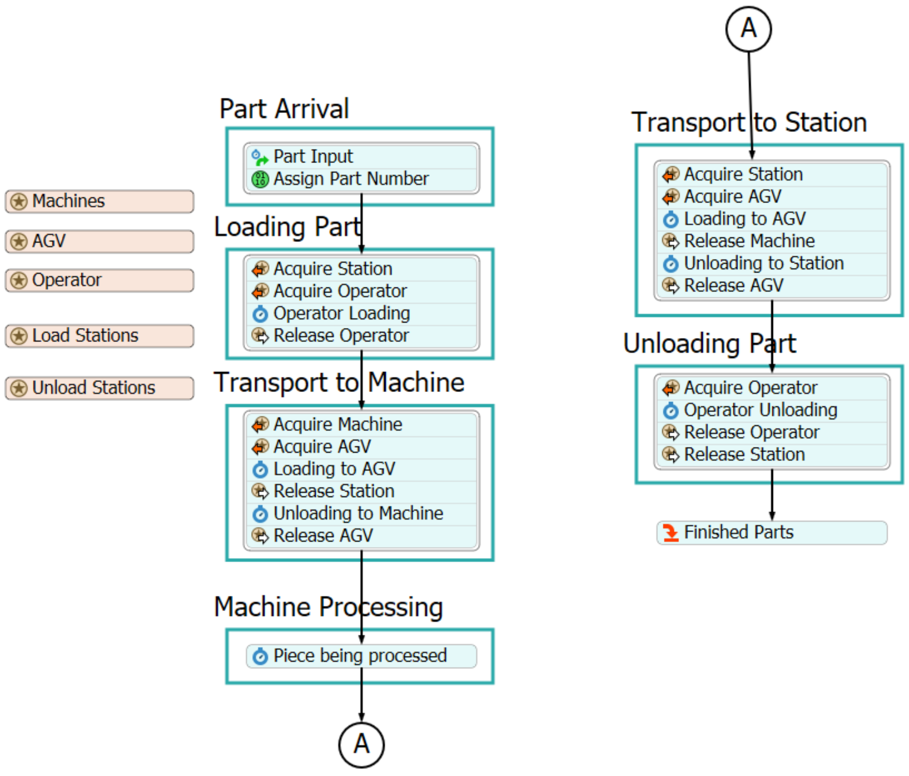


Figure 72. Case VI: Process created in FlexSim for the first proposed solution: assign "n" load stations and "12-n" unload stations

Block a station if there are already 3 parts at the machines and 11 on the stations

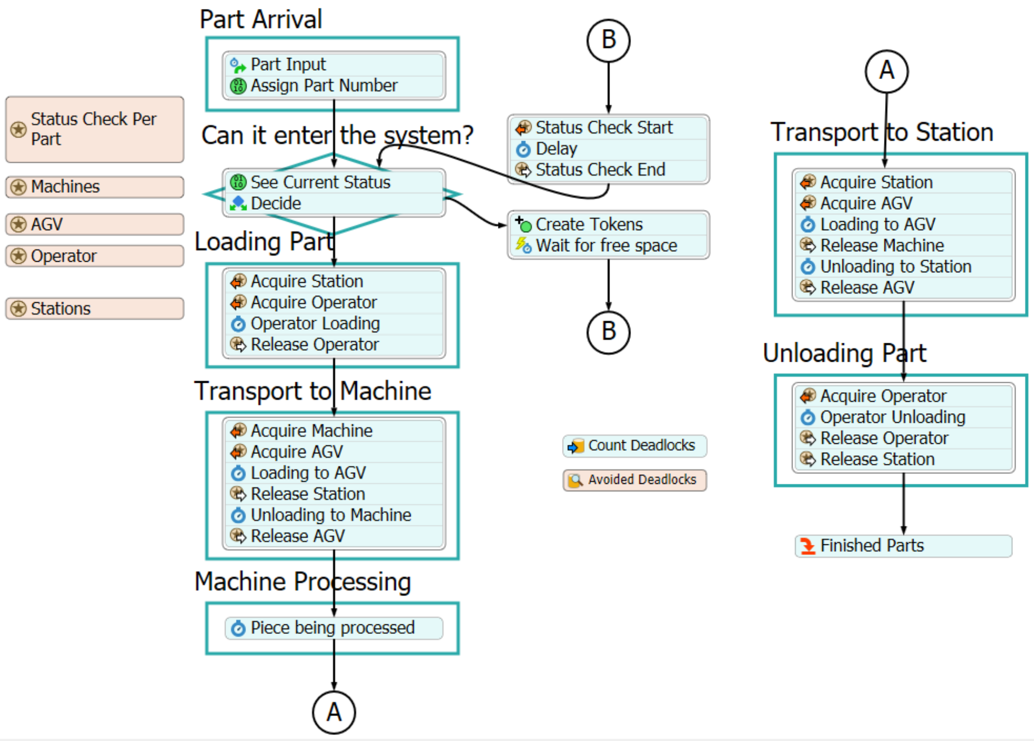


Figure 73. Case VI: Process created in FlexSim for the first proposed solution: block station if 3 parts in machine and 1 on the stations

6.6.4. Results and discussion

Once simulated all the processes mentioned in the previous section, some results can be obtained in order to see the performance of the different solution proposals.

But first of all, it can be demonstrated by FlexSim that Case VI originates a deadlock. If we simulate as the problem states (20 days), the system seems to flow correctly until certain point where a deadlock is found, which is unbreakable:

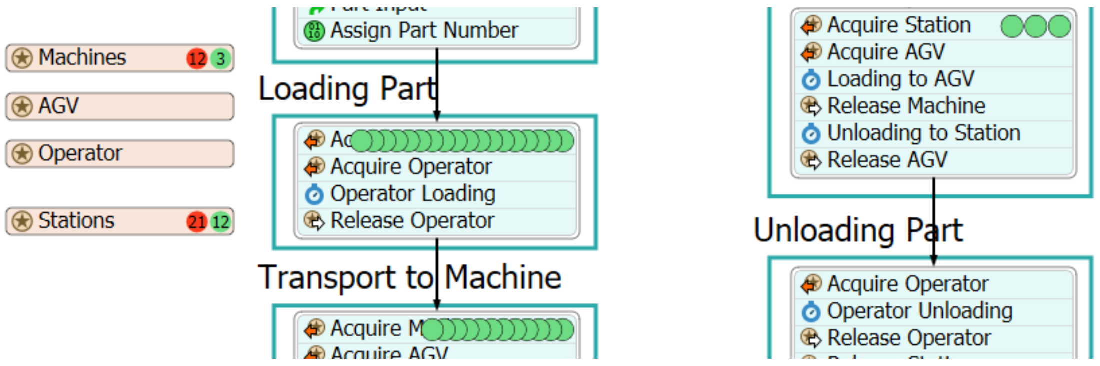


Figure 74. Case VI: Deadlock represented in FlexSim

Because of this, it makes sense to study different solutions for studying the performance of the system. There might be different ways to study the performance in this system, as the utilization of the different resources, or the waiting time for each part before being processed. However, for this system, the idle time for each part has been selected (Time that the piece is in the system and is not being processed, transported nor loaded/unloaded)

After performing the simulations, the obtained results are the ones showed in Table 14. As it can be seen, the Part Number per Station method is the worst solution, since the average lead time is significantly higher than the other ones. Ensuring always 1 unload station is also inefficient, if it is compared with having 2 unload stations. The other cases are practically equivalent, since the difference among them is just some minutes.

As it is said in 3.3.2 Deadlock avoidance, deadlock avoidance should have been better than the other systems since it signifies better resource. This is not seen in the first sight because the optimizable resource is just one station out of twelve and the system is pretty equilibrated and underused. In fact, most of the systems, in this case, will give similar outputs.

As a manner of fact, according to the simulations, this system has avoided 6 deadlock situations.

Table 14. Case VI: Results for simulations for a horizon time = 20 days

	Part Number Per Station	Part Arrival Per Station	11 Load and 1 Unload	10 Load and 2 Unload	9 Load and 3 Unload	8 Load and 4 Unload	Avoid Deadlocks
Total Idle Time (h)	748	391	423	391	391	391	391
Avg. Idle Time (min)	26,15	13,7	14,8	13,7	13,7	13,7	13,7

In order to see more statistical difference between the deadlock prevention and avoidance, it has been decided to change the number of resources. If we suppose that the system is only composed by 4 stations, and counting a horizon time of 200 days, scenarios are pretty different. This is because optimizing one resource out of four is more critical. The results can be seen in Table 15.

Table 15. Case VI: Results for simulations, modifying number of stations to 4, for a horizon time = 200 days

	Part Arrival Per Station	3 Load and 1 Unload	2 Load and 2 Unload	Avoid Deadlocks
Total Idle Time (h)	4653	4279	4071	4010
Avg. Idle Time (min)	16,5	15,1	14,4	14,2

With these simulations, it can be seen that in the long run, deadlock avoidance usually works better than deadlock prevention.

7. Conclusions

It is totally possible to map Petri Nets into FlexSim, so it is a very useful simulation software for manufacturing system processes. The software can detect deadlocks with an appropriate mapping of the real process by simulating the different cases; but more importantly, the whole process diagram can be easily changed and understand thanks to its user-friendly environment. It has shown several successful cases in which different detailed solutions have been obtained. All times and entities are trackable and the user can see where and when a problem rises, and study its causes and consequences.

It is a user friendly software similar to Arena, which is the currently used software in ESEIAAT. Inside the scope of work of this project it is demonstrated that both software's are equivalent because of the possibility of both to assume Petri Nets mapping.

However, FlexSim is quite complete and can also be synchronized with visual and location-based simulation. It is possible to model directly in 3D, which can be tremendously useful when translating a real plant into a simulation model.

It has also its own pseudocode which is not very difficult to interpret, which is highly useful for conditionals that can afterwards be applied in the program logic controller on the AGV systems, so as to improve the flexible manufacturing systems.

To sum up, it is possible to use FlexSim as a simulation tool in order to solve deadlocks with AGV systems. Moreover, it was demonstrated that mapping Petri Nets into this software was possible so that many parallelisms with Arena could be observed.

8. Future work

It is possible to use FlexSim for deadlocks associating the cases to a 3D modelling. The software also has a special AGV package module, so as to get an even more user friendly scenario. Using this modelling offers a visual representation and the user can see immediately what is happening, and is also able to determine if the process was correctly simulated seeking for its similarities to the real world process.

Other future work that can be done is using FlexSim for handling deadlocks by more cases of "deadlock avoidance". This field is really interesting but requires some expertise on programming.

Bibliographic references

[1] J. Wilkins, "Interempresas," 24 August 2017. [Online]. Available: https://www.interempresas.net/Electronica/Articulos/194384-Guiando-a-la-industria-con-robots-inteligentes-autonomos.html. [Accessed 11 August 2018].

[2] D. M. Products, 11 October 2017. [Online]. Available: https://www.dmmetalproducts.com/news/35-why-is-there-a-need-for-automated-guided-vehicles-agvs. [Accessed 11 August 2018].

[3] K. H. Kim and H.-O. Günther, Container Terminals and Cargo Systems: Design, Operations Management and Logistics Control Issues, Springer, 2007.

[4] N. Viswanadham, Y. Narahari and T. Johnson, "Deadlock Prevention and Deadlock Avoidance in Flexible Manufacturing Systems using Petri Net Models," *Transactions on Robotics and Automation,* vol. 6, no. 6, pp. 713-723, 1990.

[5] E. Coffman Jr, M. Elphick and A. Shoshani, "System Deadlocks," *Computing Surveys,* vol. 3, no. 2, pp. 67-78, 1971.

[6] Q. Li, J. T. Udding and A. Y. Pogromsky, "Modeling and control of the AGV system in an automated container terminal," *Proceedings of the AsiaMIC ,* 2010.

[7] K. M. R. L. Moorthy and W. H. Guan, "Deadlock Prediction and Avoidance in an AGV system," Singapore-MIT Alliance, 2000.

[8] R. Bhattacharya and S. Bandyopadhyay, "An improved strategy to solve shop deadlock problem based on the study of existing benchmark recovery strategies," *International Journal of Advanced Manufacturing Technology,* no. 47, pp. 351-364, 2010.

[9] A. Silberschatz, P. B. Galvin and G. Gagne, Operating System Concepts (9th edition), Kendallville: Wiley, 2013.

[10] J. Wang, "Petri Nets for Dynamic Event-Driven System Modeling," in *Handbook of Dynamic System Modeling*, Chapman & Hall/CRC, 2007.

[11] A. Guasch, J. Figueras and J. Casanovas, "Conceptual modeling using Petri Nets," in *Formal languages for computer simulation: transdisciplinary models and applications*, IGI Global, 2013, pp. 1-37.

[12] W. M. P. van der Aalst, "Timed coloured petri nets and their application to logistics," TNO Institute for Production and Logistics, Eindhoven, 1992.

[13] H. J. Pels and J. Goossenaerts, "A Conceptual Modeling Technique for Discrete Event Simulation of Operational Processes," *IFIP International Federation for Information Processing,* vol. 246, pp. 305-312, 2007.