Master Thesis

Development of a discrete event simulation tool with genetic algorithms for the design and optimization of automated warehouses

Marco Torlaschi

Supervisor

Franco Lombardi

Co-Supervisor

Emiliano Traini

Final Report for the Thesis

Master in Computer Engineering

DIGEP -Dipartimento di Ingegneria Gestionale e della Produzione

Politecnico di Torino

Italy, Turin

December 2020

Abstract

The automatization and optimization of production systems is central in the new industrial development, and a big part of production efficiency reside in logistics. Automated warehouse can aid in creating smarter and more flexible storage systems. while using less space and energy then classic systems. They provides a fully digital controllable environment where it is easier to apply responsive policy to enhance the functionality not only of logistics, but of the entire production system. However, due to the number of variable and the interdependence of operations,it is not trivial to asses the best combination of technologies, infrastructures and strategy of operation for a given situation. The goal of this Master Thesis is to develop a proof of concept for a tool which able to check the different options and than select the optimal for the specific application. For the evaluation of the performance of a warehouse i used a discrete event simulation, capable of simulating the operation of a specific situation given the parameters about the warehouse and the task it has to perform, and output a set of statistics about operations. The simulator support different technologies, different size, infrastructures setup and strategies of operation. The simulator is written in Python using the SymPy framework. The simulator is then used to optimize the warehouse parameter against a fitness computed on the output statistics. The optimizer ,also written in Python, skims different set of parameters, running the simulations and evaluating the result the find the optimal solution based on throughput, energy consumption, cycle time, infrastructure usage and other metrics, weighted at user will. The optimization uses a genetic algorithm, a optimization method which is inspired by biological evolution. Some case study are utilized and discussed to present the software functionality.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The automated warehouse market, worth 18 billion U.S. dollars in 2019, is estimated to grow to 30 billion in 2025 [1]. Given the rate of growth of the system new technologies and applications are bound to be created, creating a ever more variegated environment around the simple core idea of automating the storage of goods. Applications range from bulk material in a manufactory intake to single small perishable products of food retail industry. This ecosystem of applications need to be represented by an equally vast landscape of technological solutions , to fit the different use case.

All of this is should to be viewed in the framework of Industry 4.0, where an automated warehouse is more than a storage systems without human interaction. A modern production chain can deploy a range of method to increase efficiency, dynamically responding to exogenous events and adapt to prediction. Inside this mindset a automated warehouse can keep up with the complexity and speed of change of the rest of the production and distribution system. It can deploy smart and dynamic strategy of operation to achieve better performance, efficiency, and energy saving than a tradition human operated storage, where is harder to implement complex and fast changing policy[8].

Considering the number of possibility that the present, and future, of storage present designer and customers need a way to make informed decision. We can split the process of deploying a warehouse in two:

- Design. The design of the warehouse before construction. Selecting the best technology and hardware for the specific scenario. Application have constraint, like dimension, and specific need, like throughput.

- Operative. The strategy of storage to be used after construction, during operation, like where to put a specific type of item.

This two aspect are linked, a design can better suit a operative strategy and vice versa. Given the high complexity of the resulting problem, and the non-human information heavy environment, some sort of model is needed.One approach is to create a mathematical model of the system. This has been done but the discrete and parallel nature of operation present a problem that is hard to fully represent with only statistical methods.

The goal of this thesis is to present a different way, that of process based discrete event simulation. This is done by creating a software that simulated, step by step, every operation of the warehouse, thus attacking the chaotic nature of the system

a priori, not trying to tame the complexity but by reproducing it. I implemented a simulator that can than be used to connect the scenario as input, i.e. the parameter of the system, with statistics about the operation as output. This can be used as is, to evaluate a design, or can be used as a function for an optimization, given goals for the design. In this thesis i present the results of the simulation with changing parameter as a way to show its functionality and evaluate the results.

In this thesis i also utilized the simulation to find the optimal warehouse in same scenario. The optimization used is a genetic algorithm, a method inspired by biological evolution. I choose it because it perform well when the problem present different approaches to the optimal solution and where sub-solution are meaningful. The scenario results are shown and discussed, with a focus on energy efficiency strategy.

Chapter 2

State of the Art

2.1 Autonomous storage technologies

One can easily think about different way to implement an autonomous storage systems, and the transition between technologies isn't trivial. This is the problem that lie under the development of methods and tools for evaluating warehouse design.

The most widespread technology is the Automated Storage and Retrieval System (AS/RS). This system is based on transelevator that can horizontally move along warehouse corridor and vertical to reach all the warehouse floor. The advantage of this technology, on precedent systems based on conveyor belt, is the added flexibility in implementation and easier maintenance of infrastructures. But this systems have a hight initial investment cost.

Figure 2.1: Transelevator

From this technologies two different solution have been developed:

- CBAS/RS (Crane-Based Automated Storage / Retrieval System)
- AVS/RS (Autonomous Vehicle System / Retrieval System)

The former is based on conveyor belt and cranes. The latter, arose from the development in autonomous vehicle technology, where cranes are substituted by free moving vehicles. They give a better flexibility in the warehouse design, the number of vehicle is not fixed and they are able to reach deeper into rack, opening up the possibility on warehouse layouts. A subset of these technology, the one most analyzed in this thesis, are SBS/RS (Shuttle Based Storage / Retrieval System). It works with shuttles that can be moved between floors by the lift, allowing for great flexibility in operation and warehouse design.

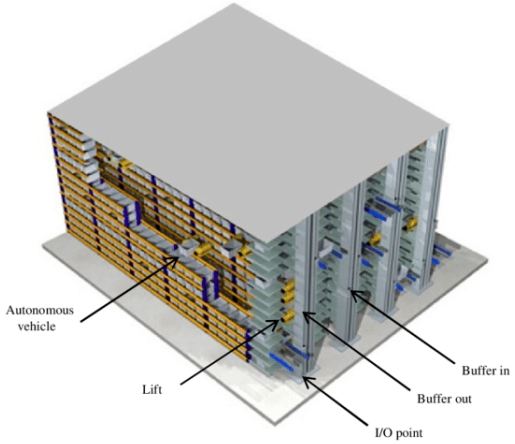


Figure 2.2: AVS-RS digram

Figure 2.3: Shuttle system

Given the increase in flexibility that these new technologies permit, a grater emphasis can and need to be put in design, thus the emergent need for smarter tools of modeling, analysis and optimization of autonomous warehouses.

2.2 Simulation

The complex nature of warehouse operation has lead to the need of modeling the system, with the purpose of crafting a tool to connect the design of a warehouse to it's performance. This is a useful aid in the design and decision process behind the development of new technologies and aid to costumer in installation choice. There are two main path that can be taken to achieve modeling, the first being analytical model and the second a simulation.

The analytical model is used to model a particular mode of operation [10][11][26][9]. This is useful to asses the performance of a single technology and has the has advantage, over simulation, of being interpretable and and analytically optimized. The concurrency nature of operation pose a problem, given that the behavior of the system can be chaotic, to the analytical modeling. Model can be very complex and a stochastic approach need to be taken to represent the nonlinear nature of the system, thus lowering the fidelity of the model to the real system.

The other approach is the simulation, taken by many [12][4][13][5]. The simulation approach gives up in the interpretability of the analytical model to achieve a closer adherence to the real system, by reproducing steps by steps the operation, thus reproducing the concurrency and non-linearity as is. The first clear drawback of simulation is that the can require significant processing power. This is a problem in case of realtime application, but not when the model has to be used for design. Most simulation are written to simulate only one technology or mode of operation, but this method have the capability of taken as parameter not only the parameter of warehouse construction but alse the technology and strategy of operation (the goal of this thesis).

2.3 Optimization

The other need, after modeling the system, is the optimization of operations. Optimization can work to optimize a single aspect, like a single parameter of a technology [25] or the warehouse infrastructure [15], or the whole warehouse design, like in this thesis.

Various methods of optimization are used, each model and goal present different need [19][16][23][22]. In this thesis, the genetic method is applied, for reasons explained in the methodology chapter.

The optimization can be used with different fitness. There are many metric that can be chosen as parameter to optimize. One of the more used is cycle time, which is widely used as performance metric for warehouse but doesn't really represent well all the possible feature of the system. The fitness can be defined as a combination of more metric, using a multi-objective optimization. Many research used a combination of operation time and energy efficiency [21][24][3]. The tool proposed in this thesis allows for the definition of the fitness as a combination of many metrics, and the scenarios presented mainly use construction cost and energy consumption.

Chapter 3

Metodology and Software description

3.1 Simulation

3.1.1 Discrete events simulation

The simulation is implemented as a **Discrete events simulation**. This type of simulation models systems by events that happen at a single point in time, modifying the state of system. The simulation can jump to next event time because between states nothing can happen. Discrete events simulation are widely used for modeling logistic systems [7].

3.1.2 Simulator implementation principle

The implementation of the simulator is done encapsulating most of the operation logic inside the methods of the Strategy class. It is thus possible to tailor the simulation to a specific case of study. The technology and strategy of allocation can be selected by parameters,so that different possibility can be directly tested against each others by an optimization algorithm that can consider technology as a numerical parameter.

As of now, the implemented technologies are:

- AS/RS
- AWS/RS(telescopic fork)
- AWS/RS(shuttle-satellite)

These are chosen because they represent a good spectrum from mostly linear operation, AS/RS, to very parallel, AWS/RS(shuttle-satellite). This can be seen in operation flowcharts above. Using this simulator we can evaluate if the increase in concurrency is worth the added overhead, and possibly infrastructure cost, in more complex technologies.

3.2 Optimization

3.2.1 Optimization interface

The optimization interface is written with the goal of making any existing method implementation easy to fit to work with the simulator. To achieve this one can define a set of the simulation parameter to be a range of valid value (using the classes **OptimizationParameter** and **OptRange**). A solution is a simple float array, from 0 to 1, of length equals to the number of parameter that are defined as a range. This array is then mapped to the ranges. The optimization algorithm doesn't need know specific about the meaning of values, or their valid values.

The fitness is computed as a weighted sum of the fields in **Monitor.Results**, so that is possible to optimize for a single metric or create a specific complex of factor to fit a specific situation.

3.2.2 Genetic algorithm

Genetic algorithms are a class of optimization method inspired by natural selection [18]. They are metaheuristics, they aren't guaranteed to find the best solution, but are useful in problem where the solution space is too big and exact method need much run-time to be useful. A heuristic method only check solutions based on algorithm that provide better results then random search.

A genetic method create a set of solutions (called population), then solutions are removed from the population or used to produce new solution based on a evaluation test, this value is often called fitness. Others natural phenomenas can be emulated to improve the methods, like bottleneck effect, founder effect, population separation and others.

The steps used in this implementation are:

1. A random population is created.

2. A set of solution are select by **Fitness proportionate selection** (also called roulette wheel selection)[17]. The probabilities for each solution to be selected is:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{3.1}$$

   Where f is the fitness and N the number of solutions. This method allow for 'bad' solution to be selected,albeit less likely , so to have good recombination.

3. New solution are generated by couple from the selected solution, using the **crossover** method. This is done by having each value of the new solution be one of the values for that fields in the parents, which parent selected randomly.

4. Each new solution is then **mutated**. Each field of the solution has a probability of changing, thus creating new alleles.

5. New solutions replace the worst one in the population.

6. Repeat from step 2.

The bottleneck effect is implemented. In natural science a bottle event happen when a population is reduced drastically by an outside cause (like a very cold winter) and is then able to return to to its previous size. This reduce the gene pool, selecting the best one at surviving the event, thus triggering a mutation of the general gene pool for the population [20]. This is emulated in the genetic algorithm by having a probabilities for each generation of substituting more than the usual faction of population. This is useful for, from time to time, eliminating bad sub-solutions and leaving space to new one to emerge.

This type of algorithm has synergies with the warehouse optimization problem. First of all sub-solutions , like the number of lift or the strategy parameter, have meaning. So having a population where many solution coexist and using the crossover method for generation new one enable the ability to aggregate sub-solutions. The solutions containing the good sub-solution, that are probably not optimal, can fuse to create a better solution. This effect is difficult to obtain using standard neighbourhood search (typical of other optimization methods) because a big jump, of many need to happen. The other big advantage of having a population is that vastly different approach, like using lots of lift and few shuttle against the contrary, are searched concurrently.

The problem now lies in tuning the parameter of the algorithm. The most important are:

- the population size. If the size of the population is too small it became unable to represent all approaches to the solution and to maintain a good poll of sub-solution. If the population is to big a lot of overhead is added for generating and testing an unnecessary amount of solutions.
- The fraction of the population to be substituted. If it is too small a lot of generation are necessary to obtain a change of the pool and bad solution remain in the pool polluting new solutions. If to big the effect is similar to that of a small population, good sub-solution are deleted and approach unexplored.
- The mutation rate. If too small it become slow to mutate bad alleles in good one. If too big it become difficult to fine tune alleles to optimize good solution.

So if time is not a problem a big population with small swap and mutation is the best choice. If instead time is a factor a balance need to be found.

3.3 Software description

The simulation software is written in python, using the SymPy framework, and contains the following packages:

- IdeaSim
- SimMain
- Resources
- Task
- Trace

In addiction there is a package called Opt which contains the modules for the optimization.

3.3.1 Simpy

Simpy is a opensource python framework for process-based discrete-event simulation [2].

A process in simpy is python generator function that *yield* event to be managed by the simpy environment. An event can be:

- process wait a period of time, **env.timeout()**
- another process to be started, **env.process(process_to_start)**
- a request for a resource

Resource models anything that is shared between processes. Simpy define three types of resource:

- **Resource**, the basic resource that can be requested by a limited number of processes.
- **Container**, it represent a resource that is consumed as an homogeneous good.
- **Store**, a set of python object.

The simulation run by jumping to the next event scheduled and handling the events. Simulation can be run in real-time by setting a delay for each simulation step.

3.3.2 IdeaSim package

IdeaSim is an extension of the SimPy framework. It adds support and logic for easier request of resources based of filter functions, event generation and handling, performer objects witch execute actions that are dependent from other actions or conditionally executed based of the state of the simulation.

It contains five modules:

- Actions
- Event
- Manager
- Resources
- Simulation

Simulation module

The Simulation module defines the Simulation class which extends of the SimPy **Environment** class. It encapsulate, as well as the simpy **environment**, the basics utility for this type of simulation:

- A logger class, under **Simulation.Logger**. The method to log a message is **Simulation.logger.log(self,msg,indent=0,type=Type.Normal)**. It takes:
  - The **msg** argument is the message to be logged.
  - The **indent** argument control how many space indentation are added in the output line. Useful for log readability
  - The **type** argument take one value from the **Simulation.Logger.Type** enumeration. It is used to differentiate log from normal activity from error and warnings

  The log can be enable and disabled via the method **Logger.enable(boolean)**. The logger add at the start of the line the simulation time at which the log was generated.

  In this implementation pretty much all events are logged, plus details about the event. Indentation is used to show hierarchy between different log sources.

- The management of resources. Resources are gonna be explained in details in their module section. The management of resource is critical for performance so method to access them are designed to reduce the computation complexity of search methods and size of the sets. Four attribute are used too store them:
  - **Simulation.free_res** which is a Resources object. This is a class that extend the SymPy **FilteredCointainer**. Sympy processes can ask for resources from this class.
  - **Simulation.free_map** which is a dictionary mapping the id of a resource to its free status (true if contained in **free_res**)
  - **Simulation.all_res** which is a dictionary mapping the id of all resources to the resource instance.
  - **Simulation.all_performer** is a list containing all performer. Performer are a subset of the resource which need to be accessed often but are few in comparison to all resources, so having a smaller list to search for them greatly help performance.

  Operation that can be performed on resources are:

  - Adding a new resource to simulation. The method used is **add_res(res)**.
  - Finding resources. This is the process used for retrieving an instance of a resource given the id (using the method **find_res_by_id(id,free=True)**) or a list of resources filtered by a function (using the method **find_res(func,free=True)**). The **func** argument is lambda that takes as argument a resource and return a boolean. The **free** argument if for searching for free resources only. For finding a performer the method **find_performed(func,free=True)** should be preferred for performance optimization.

  - Getting a resource. Getting mean waiting until the request resource is free and than blocking it, removing it from the free set.
    Similar to the find operation the getting can be done by id, using **get_res_by_id(id)**, or by a filter function using **get_res(func,sort_by=None)**. If no sort function is specified a random resource is selected. The **sort** argument take a lambda taking a resource as argument and returning a numerical value. The smallest value is selected.
  - Putting a resource. Putting mean freeing a resource that was previously requested via a get method. The method is **put_res(res)**.
  - The method **is_free(res)** to know to free status of a resource.

- The manager instance. The Manager class manage events (details in Manager module section).

- Other simple functionality:

  - A status that can be anything passed as argument in the instantiation of the Simulation class. Used to have some data globally readable for all the simulation object (event,,processes,resources). Method **get_status()** to read it and **modify_status()** to read and modify it. Event can have conditional trigger based on the status that are reevaluated upon modification.
  - The method **wait(delay)** used by simulation processes to wait delay simulation steps.
  - A global mutex that can be used to enforce a global sequence of operation in the simulation.

Resources module

The Resources module defines class used to manage infrastructures in the simulation. The class defined are:

- **Resource** that represents every object that can be required by a process in the simulation. It generates automatically an unique id for each resource instantiated. It also keep track of the total time it has been used for monitoring purpose.

- **Performer** extend the **Resource** class and represent a resource that can perform an activity. Using the method **Performer.add_mapping(action_type,func)** is possible to define a function to be called when an action is set to be performed by the performer. This is done via the **Performer.perform(action,takan_inf)** method where **action** is an **Action** instance and **taken_inf** a list of **Resources** already requested by the **Action_Graph** (details in the Actions Module). The function used has to yield values as yielded to the SimPy. It also define the **IllegalAction** exception.

- **Movable** interface used for resource that can be moved. It has a **position** argument and the **Move(position)** method which only logs the movement and should be overridden in implementation.

- **Resources** class that extend the SimPy **FilteredStore**. Precesses can ask for a resources and wait until it is available and put a resource when not more needed.

Event module

The Event module define the **Event** class. An **Event** represent something that is going to happen inside the simulation. Its argument are:

- **time**,at which simulation step the event should be triggered. If **None** is passed the event isn't scheduled and has to be launched manually via the **launch** method.
- **event_type** used by **Manager** to call a function mapped to the event
- **param** a dictionary for adding information to a specific instance

Manager module

The Manager module defines the **Manager** class. This class is used as a singleton inside the simulation object. It manages the triggering of events. Using the **add_mapping(event_type, func)** method is possible to add a function to be called when a event of a specific type is triggered. The event instance is passed as argument to this function. If the function return an **ActionsGraph** object the actions are executed. The function can raise the **Manager.RetryLater(*args, delay=None)** exception, to ask the manager to reschedule the event **delay** steps in the future.

Actions module

The Actions module defines classes used to represent and menage actions that can be performed by a **Performer**. The class defined are:

- **Action** is a single action that ha to be performed. It has an auto-generated its unique id and takes nine arguments:
  - **action_graph** is the **ActionGraph** of which the action part of.
  - **type** is the type used in the mapping of the **Performer**
  - **who** is the id of the performer that has to do the action. It can also be a filter function the select among the performer blocked by the **ActionGraph**.
  - **sort_by** is the function to be used to rank the performer in case a filter is used in the **who** argument. The lower return value is going to be selected.
  - **param** is a dictionary used to pass information about the action to the performer
  - **after** is a list of the actions id that this action has to wait for the completion. Actions selected have to be in the same **ActionGraph**

  - **condition** is a function evaluated before before execution of the action. If the return value false the action will be skipped (the function mapped on the **Performer** isn't called) but the action is considered completed (other action waiting for it are free to be executed). The function take as arguments the **Simulation** instance and a list of the resources taken by the **ActionGraph**
  - **on_false** function to be called if **condition** returned false
  - **branch** is the id of a **Branch** that ha to be true for the action to be executed. Branches are special action that are going to be explained later. Like for **condition** an action is considered completed even if skipped.

- **Block** is a class that extends **Action**. It is a special action that is not performed by a **Performer** but is for requesting a **Resource**. All arguments are the same but **who** is the resource to be requested.

- **Free** is a class that extends **Action**. It works like block but for releasing a resource.

- **GenerateEvent** is a class that extend **Action**. It is a special action used to launch an event. It only takes the **action_graph**, **after** and **branch** arguments plus the **Event** to be triggered

- **Block** is a class that extend **Action**. It is special action that only check the condition for the execution (given in the **condition** argument) and act as flag inside the **ActionsGraph**. Other action can be dependent upon the state of a branch. Used to account for different actions path if special cases are emerging after an **ActionsGraph** scheduling.

- **ActionsGraph** is a container for a set of **Actions** that are linked with dependency. If a function mapped on the **Manager** to an **Event** return an **ActionsGraph** its execution is scheduled. Care has to be taken when creating an **ActionsGraph** not to create deadlocks with dependency.

- **Executor** is the class that manage the execution of an **ActionsGraph**. It's instantiated by the **Manager** for every **ActionsGraph** returned by an event. It defines the **AbortExecution** exception that can be raised anywhere inside an execution to abort the whole **ActionsGraph** execution.

3.3.3 Task package

The Task package contains modules that define classes use to represent and schedule the handling of pellets. It contains 3 modules:

- **Item** module defines the **Item(item_type, weight)** class. The class represents a single pallet, and an unique id is assigned to it. The **Item** has an **item_type** which is a string id that tell the warehouse if two pallet can be placed in the same **Channel**. A channel is a LIFO container where only the first pallet can be retrieved, so only items of the same type can be stored in it. One **Item** also has a weight (in Kg) which is used in energy consumption calculation. Every pallet has the same size and can be put in any channel (if empty).

- **Task** module defines the **Task(item, order_type)** which represent a single handling order of a single pallet. It takes as argument an instance of **Channel**, which is the pallet to be handled, and a **order_type** that is one of **OrderType.DEPOSIT** and **OrderType.RETRIEVAL**. They tell the warehouse if the pallet should be taken from the bay to a channel or the order way around, they are value of an enumeration defined in the same module. On a retrieval order the item asked is one of that type, not a specific one. In practice items with the same type are indistinguishable.

- **TaskDispatcher** defines the **TaskDispatcher(simulation, tasks)** class. This class is used as a singleton inside the **Warehouse**. The **tasks** argument is a dictionary with time of scheduling as keys and **Task** objects as values. Upon creation it schedule events that will start the operation management of that task. More task can be added after using the **add_future_task(self, time, task)** method.

3.3.4 Trace package

The Trace package implements the creation of the a trace of orders for the simulation. The trace is a sequence of tasks with he time of arrival to the warehouse. This is done via the method **trace_generator(trace_par)** defined inside the Trace module. This method takes a **TraceParameter** object and returns a dictionary with time as keys and **Tasks** object as values. **TraceParameter** is class defined in the same module that contains the parameter of the trace creation, that are:

- **sim_time**, the time in simulation steps(seconds) until the trace has to be generated, and the simulation run time. The last seconds are left without task to allow the warehouse to perform all task assigned before the end of the simulation. This is useful for differentiate the case where the warehouse wasn't able to complete the task cause of low throughput from the case when a task was schedule in the last few seconds.

- **types**, a list of float representing the probability of each item to be of one type. The length of the list is the number of different type of items that exist. The sum of the probabilities must be one.

- **int_mean**, the average time between tasks. The trace is generate using a exponential distribution with **int_mean** as λ parameter

- **start_fullness**, fraction of the ubik of the warehouse that are filled at the start of the simulation. The same item type distribution and the same strategy (selected in the **SimulationParameter**) is used for the pre-filling, to ensure a realist starting state.

- **seed**, the seed used for the random generation, useful for testing different design on the same trace. Random if **None**. Retrieval and deposit are generated with the same probabilities (so that the fullness of the warehouse is maintained), with the exception that no retrieval will be generate for types not yet present in the warehouse.

3.3.5 Resources package

The Resources package defines the infrastructures of the warehouse, the modules are:

- **ActionType** defines an enumeration, **ActionType**, that defines the set of action that can be performed by infrastructures. The values are:
  - **MOVE**, movement action
  - **GET_FROM_BAY**, loading a pallet from the bay
  - **DROP_TO_BAY**, unloading a pallet to the bay
  - **PICKUP**, used when a machine (i.e. a lift) picks up another machine (i.e a shuttle)
  - **DROP**, use when a machine (i.e. a shuttle) drops up another machine (i.e a satellite)
  - **BLOCK**, special action used for requesting a resource
  - **FREE** , special action used for releasing a resource
- **Movement** defines some classes an methods to control position and movement of infrastructures
  - **Position(section, level, x, z)** identifies a point in the where house. The values are not absolute but relative to the dimension of infrastructures. A **level** value of 2 mean being in the second floor (all count start from zero). A **section** the portion of the warehouse reached by a lift. Every section has lift and every lift a section, in a one to one relation. It's impossible to move anything from one section to another so each section behaves almost as a separate warehouse. The **x** dimension is the row of channel depth wise, the **z** dimension is how deep the point i in the channel.
  - **MovableResource(sim, position, acc, max_v, par** extend the **Resource** class. The added arguments are:
    - **position**, an instance of **Position**
    - **acc**, the acceleration (ms^{-2}) of the machine. It also set the deceleration.
    - **max_v** max velocity reached by the machine (ms^{-1})
    - **par** instance of **SimulationParameter**. An object containing all the parameter of the warehouse.

    The method **Move(sim, position, parameter)** takes the position where the **Resource** has to move and return the time taken for the movement. The distance **d** is computed using the method **Distance**. Then the formulas used are:

$$t_{acc} = v_{max}/a \tag{3.2}$$

$$s_{acc} = v_{max} t_{acc}^2 \tag{3.3}$$

    Then if $d > s_{acc}$ (so that there is time to reach v_{max}):

$$t_{tot} = t_{acc}^2 + (d - s_{acc})/v_{max} \tag{3.4}$$

else:

$$t_{tot} = \sqrt{d/a} \tag{3.5}$$

Then the result is rounded to the second using a stochastic rounding. The probability of rounding up is $x - floor(x)$. This assure that the average of rounding on same value converge on the real value. This has to be done because rounding error from recurrent movement time (i.e. the time taken by the lift from bay to first level) are going to sum up, substituting the real value with the one the rounding method converge to. For example a movement of 10.6, using nearest integer, is going to become a effective movement of 11 . If this is a movement that appears often this would stray the simulation further and further from reality.

The **Move** method also updates the energy consumption of the resource, using the **EnergyModel** module. The other method of the class,__**drag**__ is used when the resource is loaded on another resource that move. It updates the position of the resource and call the same method on loaded resource if needed.

- **Distance(p1, p2, parameter)** method takes two position, and an instance **SimulationParameter**, and returns the manhattan distance in meter. If positions have different sections it return a infinite value.

- **Bay**, is the bay where pallet are taken and deposited. It only have position and it can perform no action(other than blocking and freeing)

- **Channel** is where pallet are put. The channel class extends the **Resource** class and the **LifoStore** class, defined in the same module, that is an extension of the SymPy **Store** class. The **_do_put** method is overridden to have the same functionality but making the Store LIFO instead of FIFO. This is done to emulate the real channel in which only the last pallet put is accessible. Only item of the same type can be put in a channel. Argument of the channel class, other then the one inherited from the resource class,are:

  - **capacity**, how many pallet it can contain.
  - **lift**, an instance of the lift of the section in which the channel is put.
  - **position**
  - **orientation**, in the same level and x position 2 channel are present (the shuttle run between two row on channels). The value is LEFT or RIGHT, representing the two position a channel can be relative to the shuttle space, part of the **Channel.Orientation** enumeration.

  The class exposes the method **first_item_z_position** that returns the z position of the first occupied ubik. Used for satellite (or fork) movement distance computation.

- **Lift**, the lift is the infrastructure that can move vertically. The class **Lift** is an extension of **MovableResource** and **Performer**. It can perform three action:

  - **MOVE**, this action type (mapped to the method **move_lift**) moves the lift to a position. Possible parameter are :

    - "level", the goal level of the movement
    - "resource", the id of a resource whose position level is the goal of movement
    - "auto", the lift will go to the level of a shuttle that is blocked by the **ActionGraph**, in case of more the one the first to have been blocked is select.
    - "auto_sat", the lift will go to the level of a satellite that is blocked by the **ActionGraph**
  - **PICKUP**, this action type (mapped to the method **pickup**) tell the lift to load a shuttle. It does not check that the positions coincide, so right positioning have to be assured by the action sequence. Possible parameter are :
    - "shuttle", the id of the shuttle to load
    - "auto", like for movement, it select a blocked shuttle
  - **DROP** this action type (mapped to the method **drop**) tells the lift to unload a shuttle. The shuttle is put on the same level as the lift on position $x = 0$. It take no parameter.

- **Shuttle**, the lift is the infrastructure that can move in the x dimension. The class **Shuttle** is an extension of **MovableResource** and **Performer**. It can perform three action:
  - **MOVE**, this action type (mapped to the method **move_shuttle**) move the shuttle to a position. Possible parameter are :
    - "x", the goal x position of the movement
    - "resource", the id of a resource whose position x is the goal of movement
    - "auto", the shuttle will go to the position of a satellite that is blocked by the **ActionGraph**
  - **PICKUP**, this action type (mapped to the method **pickup**) tells the shuttle to load a satellite. It does not check that the positions coincide, so right positioning have to be assured by the action sequence. Possible parameter are :
    - "satellite", the id of the satellite to load
    - "auto", like for movement, it select a blocked satellite
  - **DROP** this action type (mapped to the method **drop**) tell the shuttle to unload a satellite. The shuttle is put on the same x position as the shuttle on position $z = 0$. It take no parameter.

- **Satellite**, the satellite is the infrastructure able to move inside channels and load pallet. In this simulator it is also used to emulate a fork, if the technology select use it. It can perform five action:
  - **MOVE**, this action type (mapped to the method **move_satellite**) move the satellite to a position. Possible parameter are :
    - "z", the goal z position of the movement

* "resource", the id of a resource whose position z is the goal of movement

- **PICKUP**, this action type (mapped to the method **pickup**) tells the satellite to load a pallet. The parameter it needs is "channel_id", the id of the channel in which the satellite loading.
- **GET_FROM_BAY**, this action type (mapped to the method **get_from_bay**) tells the satellite to load a pallet from the bay. The parameter it needs is "item", an instance of the item taken.
- **DROP** this action type (mapped to the method **drop**) tell the satellite to unload a pallet in a channel. Like for PICKUP, it needs a "channel_id" parameter.
- **DROP_TO_BAY** this action type (mapped to the method **drop_to_bay**) tells the satellite to unload a pallet a the bay. It doesn't need any parameter.

- **EnergyModel module** defines a the **energy** method. This method takes two position and return the energy consumed. It derives the infrastructure that is moving from moving direction.

  The model use to compute consumption is a modification of the one defined by Ekren [14][6]. The specific powers of x axis movements are:

$$p_{x,a} = (a_x f_+ g c_{r,x}) \frac{v_{x,get}}{\eta_x} \tag{3.6}$$

$$p_{x,c} = (g c_{r,x}) \frac{v_{x,get}}{\eta_x} \tag{3.7}$$

$$p_{x,d} = (d_x f_r - g c_{r,x}) \frac{v_{x,get}}{\eta_x} \tag{3.8}$$

  Where a_x and d_x are acceleration and deceleration, f_r is the mass resistance factor, g is gravity acceleration, c the friction coefficient and η the transmission yield. This formulas are equal for the z axis, been both horizontal movement. The specific powers of y axis movements are:

$$p_{y,a,U} = (a_y f_r + g c_{r,y} + g) \frac{v_{x,get}}{\eta_x} \tag{3.9}$$

$$p_{y,c,U} = (g c_{r,y} + g) \frac{v_{x,get}}{\eta_x} \tag{3.10}$$

$$p_{y,d,U} = (-d_y f_r + g c_{r,y} + g) \frac{v_{x,get}}{\eta_x} \tag{3.11}$$

$$p_{y,a,D} = (a_y f_r - g c_{r,y} + g) \frac{v_{x,get}}{\eta_x} \tag{3.12}$$

$$p_{y,c,D} = (-g c_{r,y} + g) \frac{v_{x,get}}{\eta_x} \tag{3.13}$$

$$p_{y,d,D} = (d_y f_- g c_{r,y} + g) \frac{v_{x,get}}{\eta_x} \tag{3.14}$$

  Where U is for movements going up and D for movements going D. Energy is then computed using:

$$e_x = p_{x,a} t_{x,a} + p_{x,c} t_{x,c} + p_{x,d} t_{x,d} \tag{3.15}$$

$$e_z = p_{x,a}t_{x,a} + p_{z,c}t_{z,c} + p_{z,d}t_{z,d} \tag{3.16}$$

$$e_{y,U} = p_{y,a,U}t_{y,a} + p_{y,c,U}t_{y,c} + p_{y,d,U}t_{y,d} \tag{3.17}$$

$$e_{y,D} = p_{y,a,D}t_{y,a} + p_{y,c,D}t_{y,c} + p_{y,d,D}t_{y,d} \tag{3.18}$$

Where t is time spent accelerating (a),decelerating(d) or keeping constant (c). This times are computed as for movement time in Movement module.

3.3.6 SimMain Package

The SimMain Package contains modules used to launch a simulation and gather information about it. I will now describe the modules it contains

SimulationParmater module

The SimulationParmater module defines a class contain all the supported parameter of the Warehouse. The parameter are:

- **Nx**, number of floor
- **Ny**, number of channels layer per floor.
- **Nz**, width of the warehouse in number of ubik.
- **Lx**, floor height in meter
- **Ly**, dimension of channel in the x (shuttle movement) direction (meter).
- **Lz**, dimension of ubik in z (satellite or fork) movement direction.
- **Ax**, acceleration of the lift (ms^{-2})
- **Ay**, acceleration of the shuttle (ms^{-2})
- **Az**, acceleration of the satellite (ms^{-2})
- **Vx**, max velocity of the lift (ms^{-1})
- **Vy**, max velocity of the shuttle (ms^{-1})
- **Vz**, max velocity of the satellite (ms^{-1})
- **Cr**, friction coefficient
- **Fr**, mass resistance factor
- **rendiment**, rendiment of the engines
- **Wli**, weight of the lift (multiplied by 10 in case of AS/RS) in kg
- **Wsh**, weight of the shuttle in kg
- **Wsa**, weight of the lift in kg
- **Nli**, number of lifts

- **Nsh**, number of shuttles per lift, put in the same section of the lift. Total number of shuttles is $N_li * N_sh$
- **Nsa**, number of satellites per shuttle, put in the same section as the shuttle. Total number of satellites $N_li * N_sh * N_sat$
- **bay_level**, the level at which the bay is placed. Can be in between floor, if a decimal is passed
- **tech**, the technology used. Where 0 means AS/RS,1 means AWS/RS(telescopic fork) and 2 AWS/RS(shuttle-satellite).
- **strat**, The strategy used to select the channel where to put/take a pallet. A value of 0 means random, a value of 1 means nearest avaible channel with weighted manhattan distance and 2 emptier channel. More Strategy can be implemented.
- **strat_par_x** and **strat_par_y** are the weight for the weighted manhattan distance. The **strat_par_x** is for x distance and **strat_par_y** for y (vertical) distance. Double zero is equivalent to random, and equals value is equivalent to a true manhattan distance.

Warehouse module

The Warehouse module defines the class **Warehouse(sim, parameter, trace_parameter)**. This class constructs and contains all the **Resources** object used in the simulation, as well as calling the **trace_generator** and the **TaskDispatcher** for scheduling the tasks events. If takes as argument the **Simulation** object, a **SimulationParameter** object and a **TraceParameter** object. If the parameter Nz can't be evenly divide between channels, channel of capacity as close to equal as possible are created. Then, if the **start_fullness** parameter in the **TraceParameter** object isn't zero, channels are occupied using the same strategy selected for the simulation.

3.3.7 Strategy Module

The Strategy module contains the logic of operations of the warehouse. This is done in the **Strategy** via the **strategyn** and **Implementn** where the n is the strategy and the technology number. In this way is easy to add support for addition technology or strategy in the simulation, by adding the relative method. Strategy method take the **Task** to be done e return the id of the channel selected. Implemented strategy are:

- **strategy0**, randomly selects a viable channel. Viable for a deposit task mean empty or not full contain the same type of items as the task one, for a retrieval contain the right type of item.
- **strategy1**, selects the nearest viable channel using the weighted manhattan distance, using the formula:

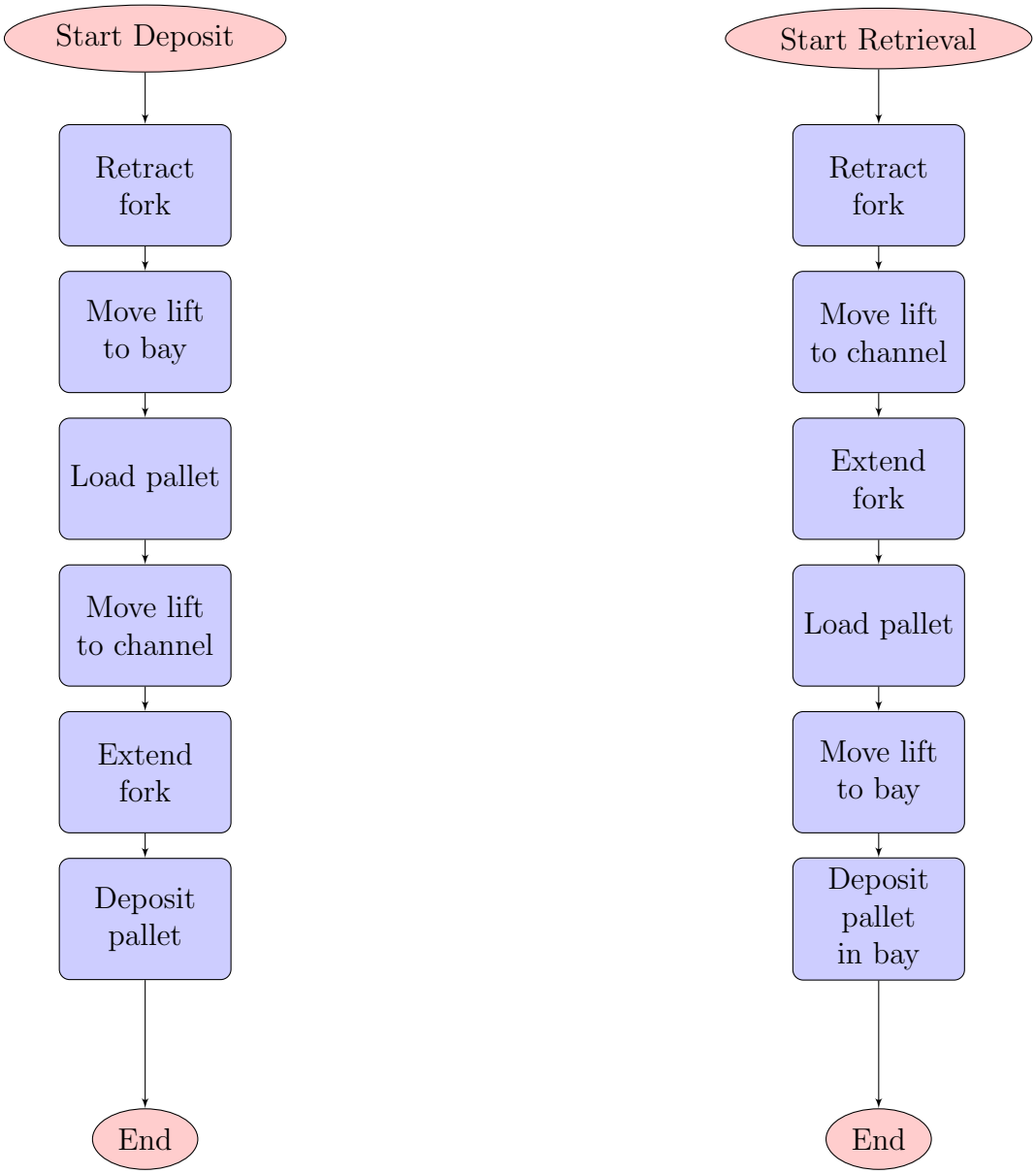
$$distance = w_x L_x channel_x + w_y L_y |channel_level - bay_level| \tag{3.19}$$

  Where w_x is the strat_par_x and w_y is strat_par_y. In case of more channels with the minimum distance one is randomly selected.

- **strategy2**, select one of the emptier channel

The **Implementn** method takes the **Task** and the channel id selected by the strategy, returning an **ActionGraph** build to achieve the task. Implemented technology are:

- AS/RS in **implement0**, this technology use a lift embedded in a moving column,so that the lift can move in 2 dimension. This is emulated in the simulator with a parallel movement of a **Lift** (on the y axis) and a **Shuttle** (on the x axis). The technology also fork for moving pallet inside channels. This is emulated with a satellite movement.
  Flowcharts for retrieval and deposit are:

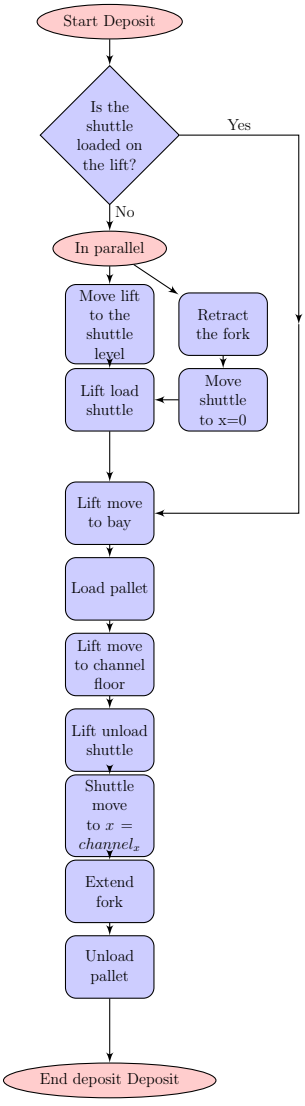


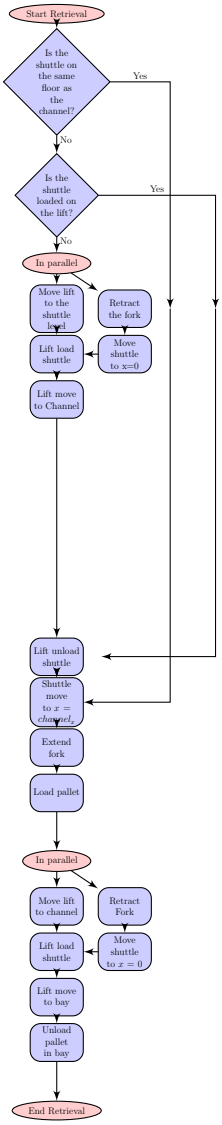
Movement of zero distance require no time, so no branches are required, in this tech, for favorable starting state. For example if the fork doesn't need to retracted at the start of a deposit, the action is executed anyway because the

movement take no time (and consume no energy) thus not invalidating the simulation. Branches are be used for other technologies, were extra movement aren't free or a different sequence has to be used.

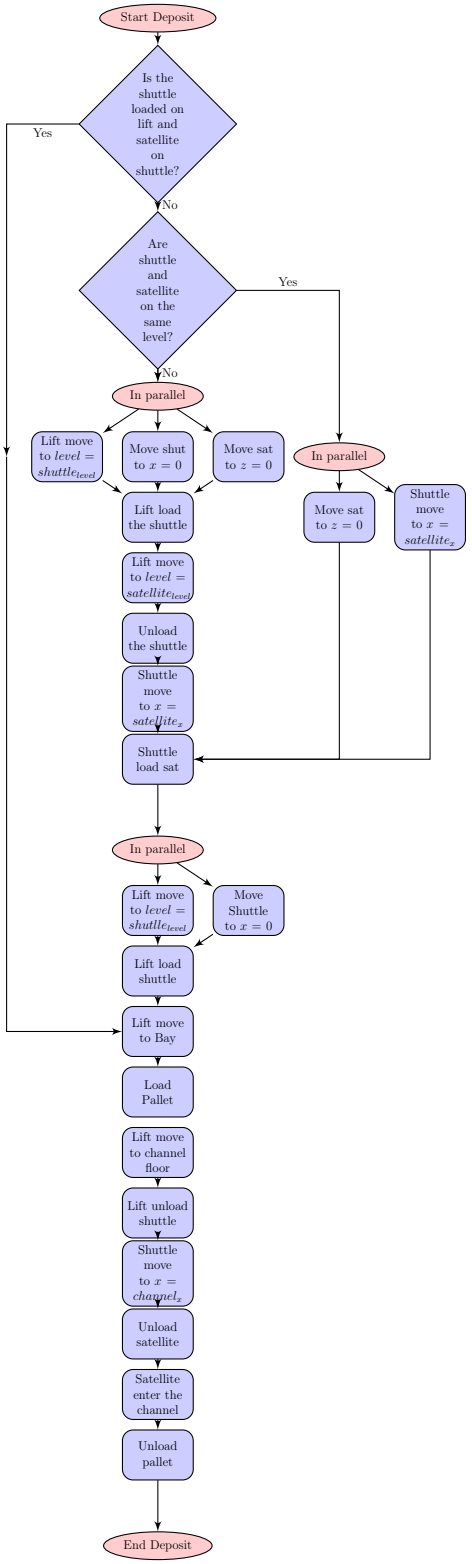
- AWS/RS(telescopic fork) in **implement1**, this technology uses a lift and a shuttle. The shuttle is moved between levels by the lift. The shuttle has a fork used form moving pallet in channels. Like for AS/RS the fork movement is emulated by a satellite movement. When a resource is not used for a portion of the action sequence it freed so that another execution can block it. The current execution will block it again (and possible wait for it to be freed) when the sequence need it again. For example, in a retrieval, when the shuttle has been dropped in the floor to load the pallet the lift is released, and then blocked again when the shuttle has finished is movement. In this way more tasks can be execute in parallel, and resource are used for most of the time, if task are available.

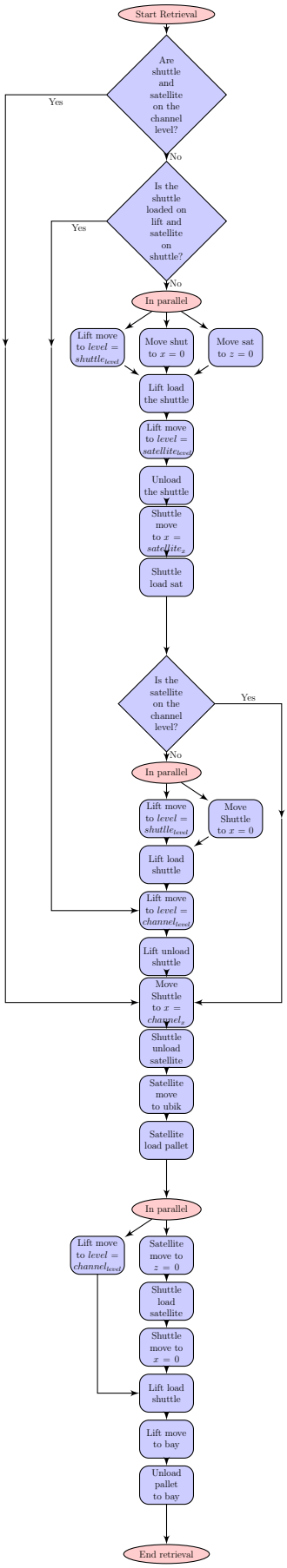
  Flowcharts are:





- AWS/RS(shuttle-satellite) in **implement2**, this technology use lifts (for y movement), shuttles (for x movement) and satellites (for z movement). Lifts can shuttles and shuttles can load satellites. Satellites load pallets. Like for AWS/RS(telescopic fork) resource are freed when not required by that portion of the sequence.




Start Retrieval
Are shuttle and satellite on the channel level?
Yes
No
Is the shuttle loaded on lift and satellite on shuttle?
Yes
No
In parallel
Lift move to $level = shuttle_{level}$
Move shut to $x = 0$
Move sat to $z = 0$
Lift load the shuttle
Lift move to $level = satellite_{level}$
Unload the shuttle
Shuttle move to $x = satellite_x$
Shuttle load sat
Is the satellite on the channel level?
Yes
No
In parallel
Lift move to $level = shuttle_{level}$
Move Shuttle to $x = 0$
Lift load shuttle
Lift move to $level = channel_{level}$
Lift unload shuttle
Move Shuttle to $x = channel_x$
Shuttle unload satellite
Satellite move to ubik
Satellite load pallet
In parallel
Lift move to $level = channel_{level}$
Satellite move to $z = 0$
Shuttle load satellite
Shuttle move to $x = 0$
Lift load shuttle
Lift move to bay
Unload pallet to bay
End retrieval

Monitor module

The Monitor Module is used to gather and read data about the execution. It define the **Monitor** class, that accumulates data during the simulation. The **get_result(cost_param)** method returns an instance of **Monitor.Results**, which contains the following field:

- **mean_task_wait**, average time a task has waited
- **mean_task_op_time**, average time for complementing a task based on active time, $working\_time/tasks\_done$ (sec)
- **mean_task_tot_time**, average time for complementing a task based on total simulation time, $tot\_time/tasks\_done$ (sec)
- **Th**, Throughput. Tasks per hour. In practice is the lowest value between task per hour of the trace ($3600/int\_mean$) and the trough theoretical value for the specific warehouse. To see the real throughput capability the trace λ should be pushed until the warehouse isn't able to complete all tasks ($completeness < 0$).Value is in ($\frac{task}{hour}$)

$$Th = 3600/mean\_task\_tot\_time \tag{3.20}$$

- **completeness**, fraction of task complement on tasks scheduled by the trace.
- **working_time**, total active time of the the warehouse. The warehouse is considered active when at list one resource is blocked. (sec)
- **tasks_done**, number of tasks completed.
- **time_per_task**, the average time a task time compute from the point of view of tasks. The time each task as taken is registered and then averaged. Task can be done in parallel so $time\_per\_task >= mean\_task\_op\_time$.
- **energy_consumed**, total energy consumed (Wh).
- **energy_consumed_per_task**, total energy consumed (Wh).
- **area**, energy consumed per task, $energy\_consumed/tasks\_done$ (Wh)
- **volume**, total volume of the warehouse, $N_z L_z N_y L_y N_x L_x$ (Wh)
- **Nx**, same as **SimulationParameter**
- **Ny**, same as **SimulationParameter**
- **Nz**, same as **SimulationParameter**
- **num_lift**, number of lifts
- **num_shuttle**, number of shuttles
- **num_sats**, number of satellites

- **lifts_util_proc**, average fraction of the time as lift was performing operation other then waiting on the active time.

$$lifts\_util\_proc = \frac{\sum_{n}^{Lifts} \frac{n_{util\_proc}}{working\_time}}{num\_lift} \tag{3.21}$$

- **shut_util_proc**, same as lifts_util_proc bu for shuttles.
- **sat_util_proc**, same as lifts_util_proc bu for satellites.
- **lifts_util**, same as lifts_util_proc but considering all the time the lift has spent blocked waiting.
- **shut_util**,same as lifts_util but for shuttles.
- **sat_util**, same as lifts_util but for satellites.
- **single_CT**, single cycle average time. A cycle is defined as two times the lift is at the bay. Single cycle is when it has done only a deposit or retrieval task. (sec)
- **double_CT**, double cycle average time. A double is a cycle where a deposit and a retrieval are done. (sec)
- **single_CT_V**, variance of single cycle time
- **double_CT_V**, variance of double cycle time
- **single_CT_E**, average energy used in a single cycle, (Wh).
- **double_CT_E**, average energy used in a single cycle, (wh).
- **strat_param**, -1 if strat_par_y = 0, else $\frac{strat\_par\_x}{strat\_par\_y}$
- **cost**, the cost of building the warehouse, computed using **ConstParam**

The class **CostParam** contains cost for infrastructure. It's parameter are:

- **lift**, price of a shuttle loading lift;
- **transelevator**
- **shuttle_fork**, cost of shuttle with fork, as used in AWS/RS(telescopic fork)
- **shuttle**, cost of a satellite loading shuttle, as used in AWS/RS(shuttle-satellite)
- **satellite**
- **scaffolding**, cost of one m^2 of floor

main module

The main module to launch a simulation. It define the **Test** class that contains the **test(parameter, trace_parameter, log=False)->Monitor.Results** method, which launch the simulation. It takes a **SimulationParameter** object ,a **TraceParameter** object and optionally a **Monitor.CostParam** object. The **log** argument is a boolean that control if the log is printed. The value returned is a **Monitor.Results** object containing statistics about the simulation (as described above).

3.3.8 Opt package

The opt package contains modules use for the optimization.

Optimization module

The optimization module defines classes that are used to run optimizations. Class defined are:

- **OptRange** class is used to define a range of values for a parameter. **low** is the minimum, **high** the maximum and **decimal** is a boolean to select if the range is discrete (only integer value) or real value in the interval are valid.
- **OptParameter** is an extension of **SimulationParameter** that can take **OptRange** as parameter. The method **map** takes a **Solution** object and return the **SimulationParameter** object with value set as the solution imply.
- **FitnessParameter** is an extension of **Monitor.Results**. Each parameter is the weight for the which every value in the Monitor.Results.
- **Solution(opt_par, t_par, fitness_par, array)** class is a List of float between 0 and 1. It represent a specific solution given the **OptParameter**. Every not fixed parameter, so **OptRange**, in **OptParameter** is mapped to a value of the array. To retrieve **SimulationParameter** from the solution each value is mapped from the array from the [0-1] interval to [low-hight] and rounded to nearest integer if float=FALSE. This assure that the optimization algorithm doesn't need to know anything about the specific of simulation or warehouse. The **array** parameter take a list of integer the will be used as seed for simulations. If the list contains more then one element more simulation are run and average results are taken. The **get_fitness** method return the fitness calculated as:
$$\sum_i^{Results} w_i value_i \tag{3.22}$$
- **Opt** exposes the method **optimization(opt_par, trace_par, fitness_par, generations, processes)** which is used to launch an optimization. Other then the three parameter class it takes a **generation** arguments which is the number of cycle of optimization that have to be run and a **processes** argument which is the number of process in the process pool used for execution of simulations. Each simulation run in a process.

Genetic module

The genetic module implements the genetic optimization algorithm. The method used to launch a cycle is **opt** that takes the following arguments:

- **opt_par**, OptimizationParameter object.
- **t_par**, TraceParameter object.
- **f_par**, FitnessParameter object.
- **pop_size**, the size of the population, how many solutions are kept alive.

- **pop_swap**, the fraction of population that is replaced each generation. The worst $pop\_size * pop\_swap$ solutions are deleted, and the same number are newly generated.
- **mut**, average mutation of a mutating value of the solutions. Mutation is computed as $newValue = oldValue + mut * r$, where r is a random value generated by a normal distribution.
- **mut_change**, increase of mut each generation, $mut_{new} = mut_{old} * mut\_change$.
- **mut_perc**, average fraction of value of the solutions that are mutated.
- **bottle_neck_prob**, probability for a generation to be bottleneck.
- **bottle_neck_swap**, population replaced in a bottle neck generation.
- **n_processes**, size of process pool.

Internally the method defines the **Chromosome** class, which is an extension of **Solution**. This class defines the method **Mutation**, that mutated the solution as described above, and the **Crossover(c1,c2)**, that takes two Chromosome and generate a new one. The new solution is generated by selecting each value randomly from one of the two Chromosome.

The other class defined is **Population**, the contains and manage the population of Chromosomes. It exposes the following methods:

- **get_best**, returns the Chromosome with minimum fitness
- **get_couple**, yield couples of Chromosome used to generate a new Chromosome. The algorithm to select the couple is:
  - Chromosome are sorted by fitness;
  - For each Chromosome the value following value is computed:

$$v_i = \sum_{j=0}^{j<i} \frac{1}{fitness_j} \tag{3.23}$$

  - then values are normalized, $v_i = v_i / v_max$, so that the max value is 1.
  - For each Chromosome that has to be selected a random float between 0 and 1 r is generated, the selected Chromosome is the one where $v_{i-1} < r \leq v_i$.
- **generation**, advance the population by a generation. This is done by choosing pop_swap couples, using them to generate pop_swap new chromosomes. Then the pop_swap worst chromosomes are deleted and replaced by the new one.

The initial population is randomly generated. Every cycle the opt method yield the best solution, update the mut parameter and call **generation()** on the pop.

Chapter 4

Verification, results and discussion

4.1 Introduction

The verification of the simulation is made hard by the fact that is not trivial to model the system mathematically. So to check the simulator results i run simulation variating only one parameter and checking that results are what we expect. All the following graphs are generate using the following parameter, and only changing the one studied:

rendiment = 0.9,
Fr = 1.15,
Cr = 0.02,
Nsa = 2,
Nsh = 3,
Nli = 3,
Vz = 1.2,
Az = 0.7,
Vy = 0.9,
Ay = 0.8,
Vx = 4,
Ax = 0.8,
Cy = 0,
Wli = 1850,
Wsh = 850,
Wsa = 350,
Lz = 1.2,
Ly = 1.5,
Lx = 1,
Nz = 50,
Ny = 10,
Nx = 50,
bay_level = 0,
tech = 0,
strategy = 1,
strategy_par_x = 1,
strategy_par_y = 1,
types = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1],

sim_time = 21600,
seed = 1,
start_fullness = 0.5,
int_mean = 25,

When variating dimension (Nx,Ny,Nz) the others are changed to maintain the same volume (and are in the case of Nx and Nz). Plots are made for the three implemented technology, where tech0 is transelevator + fork, tech1 is elevator + shuttle + fork and tech2 elevator + shuttle + satellite.

Interesting features of each results are then briefly discussed

4.2 Dimension

4.2.1 Depth

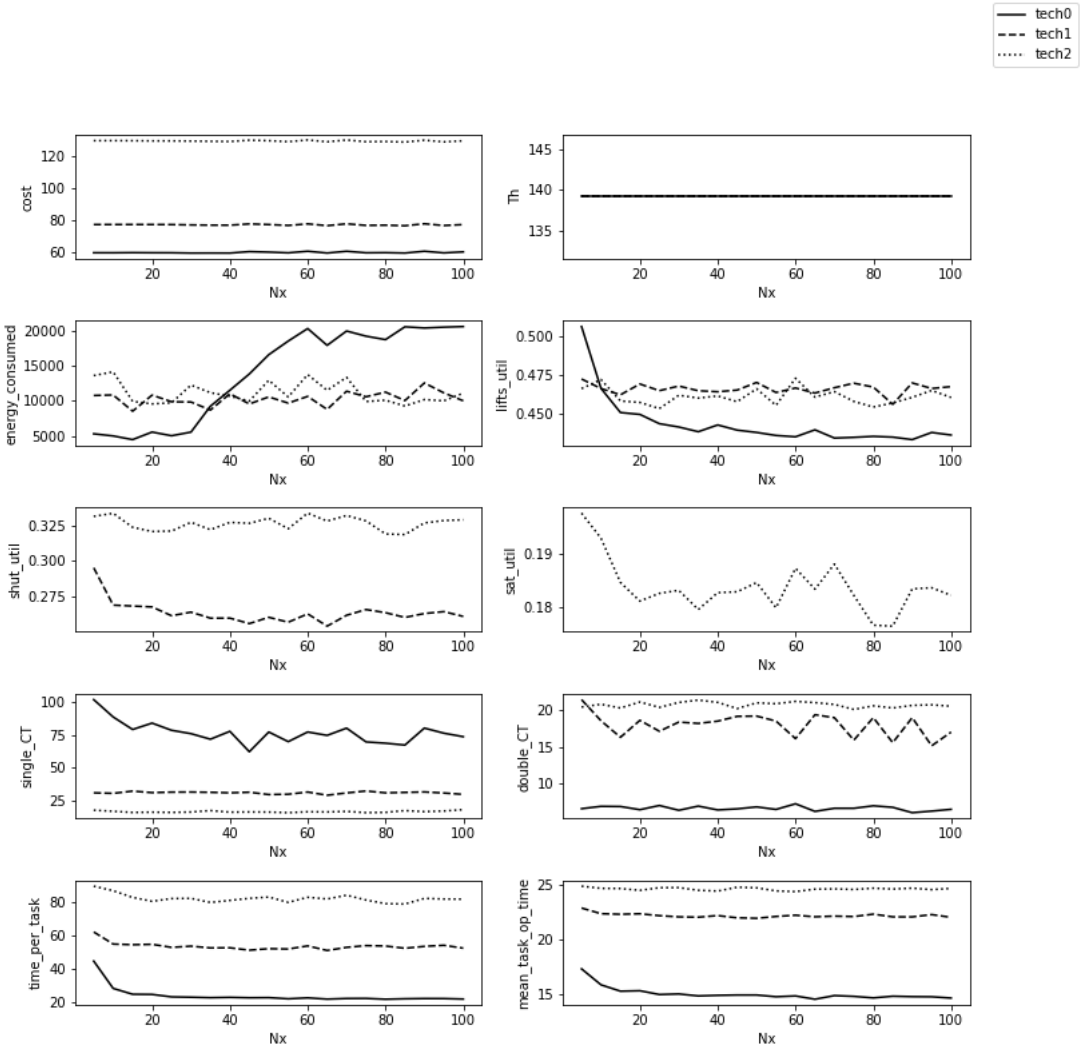


Figure 4.1: Variating Nx

When changing Nx, also Nz is changed to maintain the base area constant. So

in fact we are changing the base shape of the warehouse. We can see that this isn't changing most of the metrics for tech1 and 2, the only apparent effect is a initial decrease of satellite utilization, as we should expect given that small Nx means big Nz, so a big travel time for satellites. We don't sea a similar variation in shuttle utilization probably due to the fact that channels for operations a re selected using the manhattan distance (strategy_par_x=1 and strategy_par_y=1) so channels far away in the x axis are not very used.

For tech0 we can sea a surge in energy consumption. This is due to the fact that using a transelevator a lot of mass has to be moved in the x dimension. The plateau is, like for the shuttle utilization, due to the strategy stop selecting far away channels.

4.2.2 Width

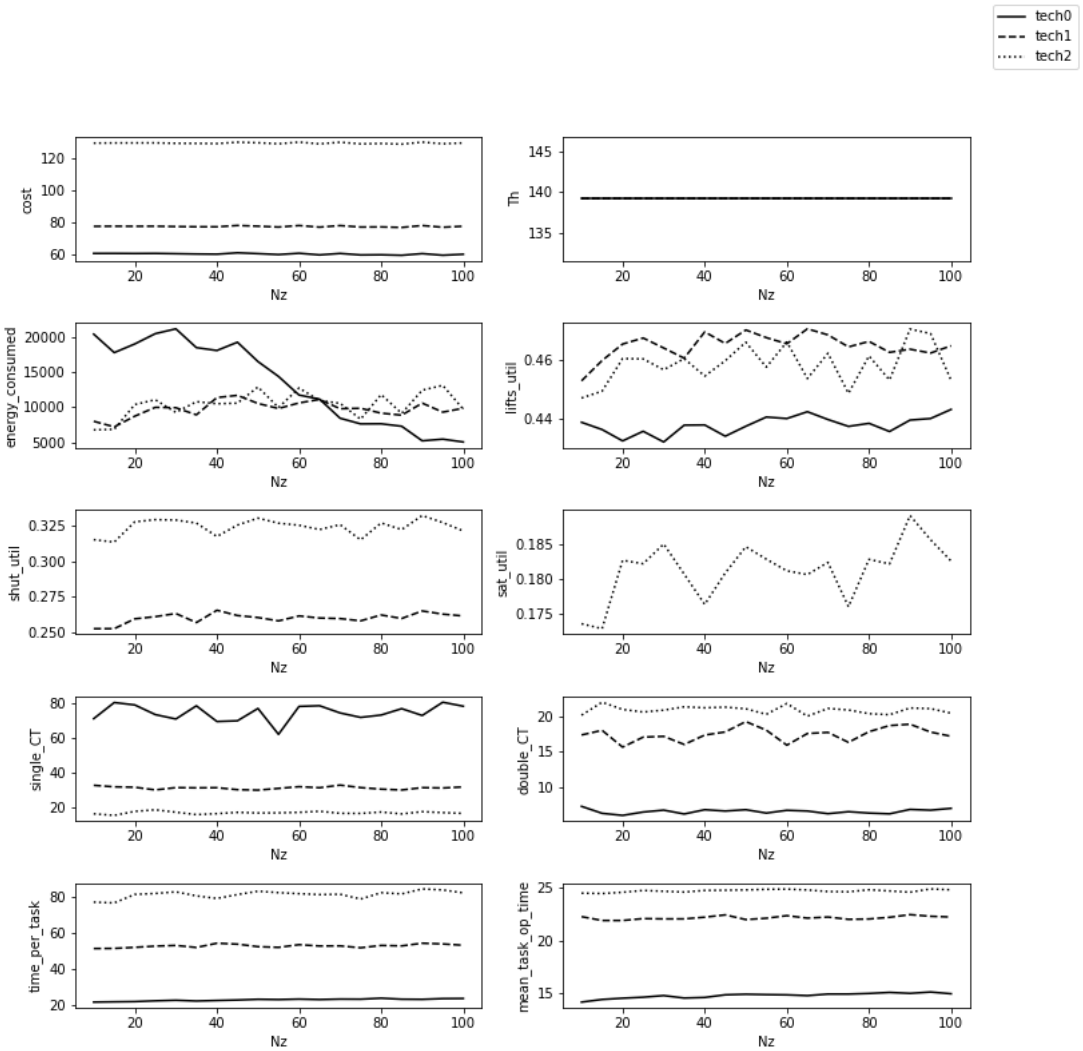


Figure 4.2: Variating Nz

As we expect variating Nz we the results are the same as Nz, but mirrored.

4.2.3 Hight

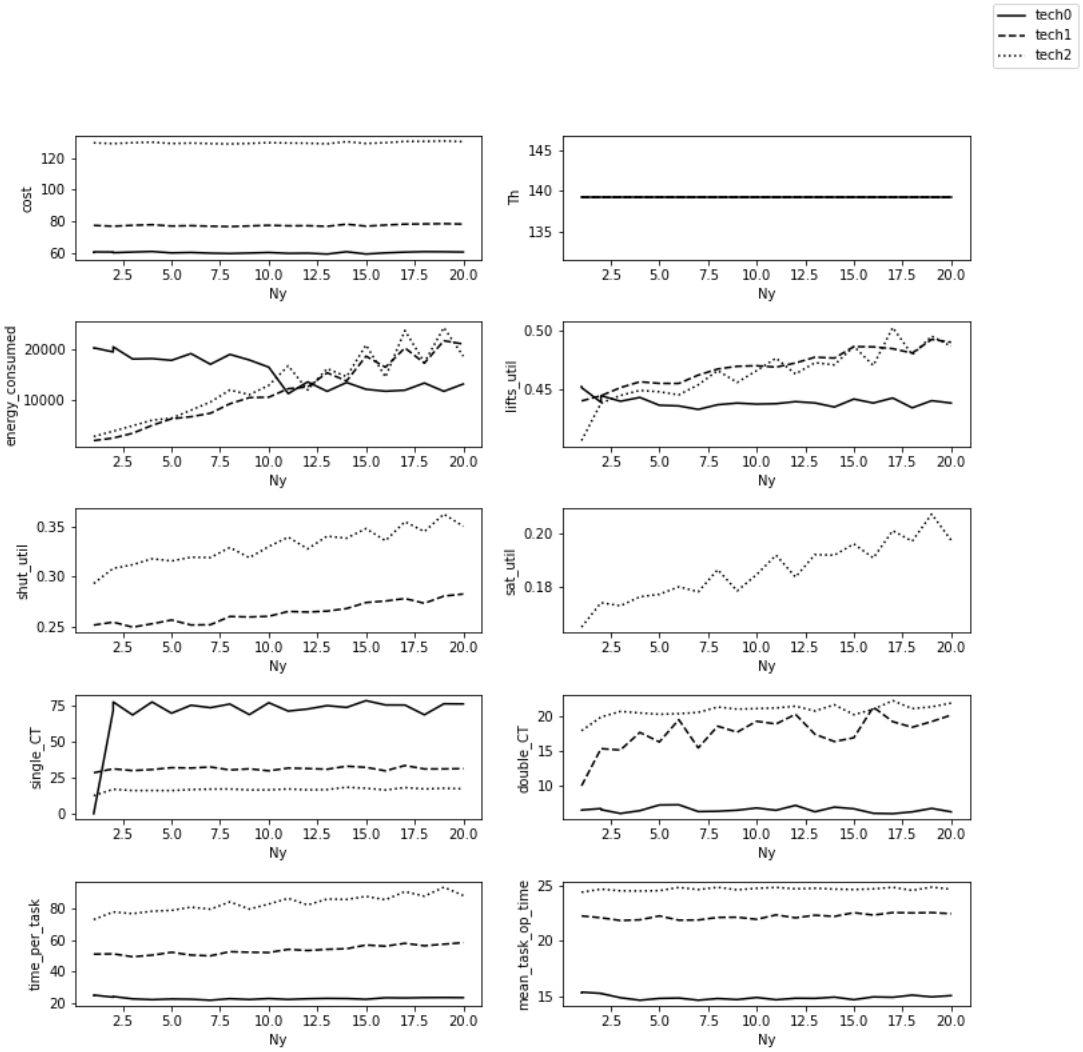


Figure 4.3: Variating Ny

Here we can see the effect of a tall warehouse(the base is keep as a square and the volume maintained constant). It interesting to see how it affect differently the energy consumption of different technologies. Tech0 better it's energy efficiency because horizontal movement are as costly as vertical one. In contrary tech 1 and 2 perform worse in a tall system due to vertical movement been the ones moving more mass. This can be mitigated be tuning the strategy parameter to consider the alter the weight of vertical movement in the selection of channels.

Using a strategy that only takes into account vertical distance (strategy_par_x=0), that effect disappear for tech 1 and 2. For the transelevator the right balance has to be found.

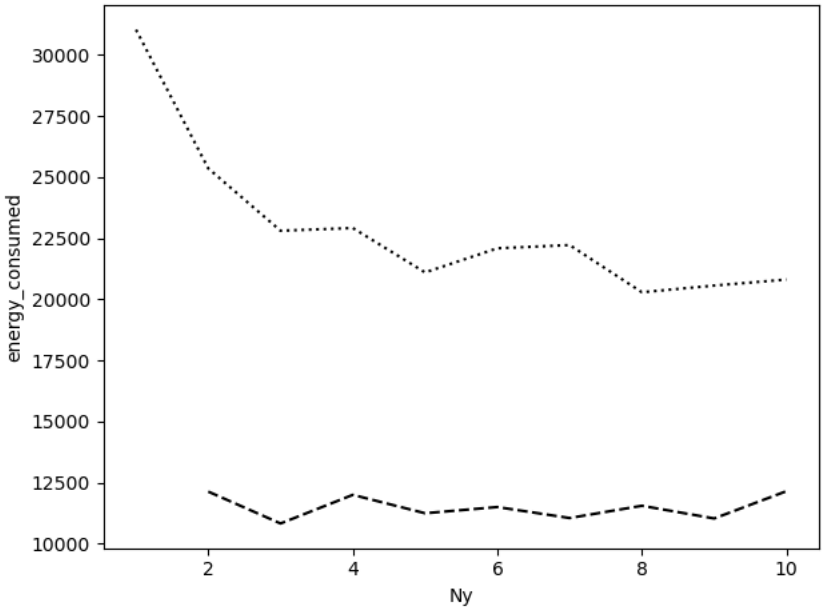
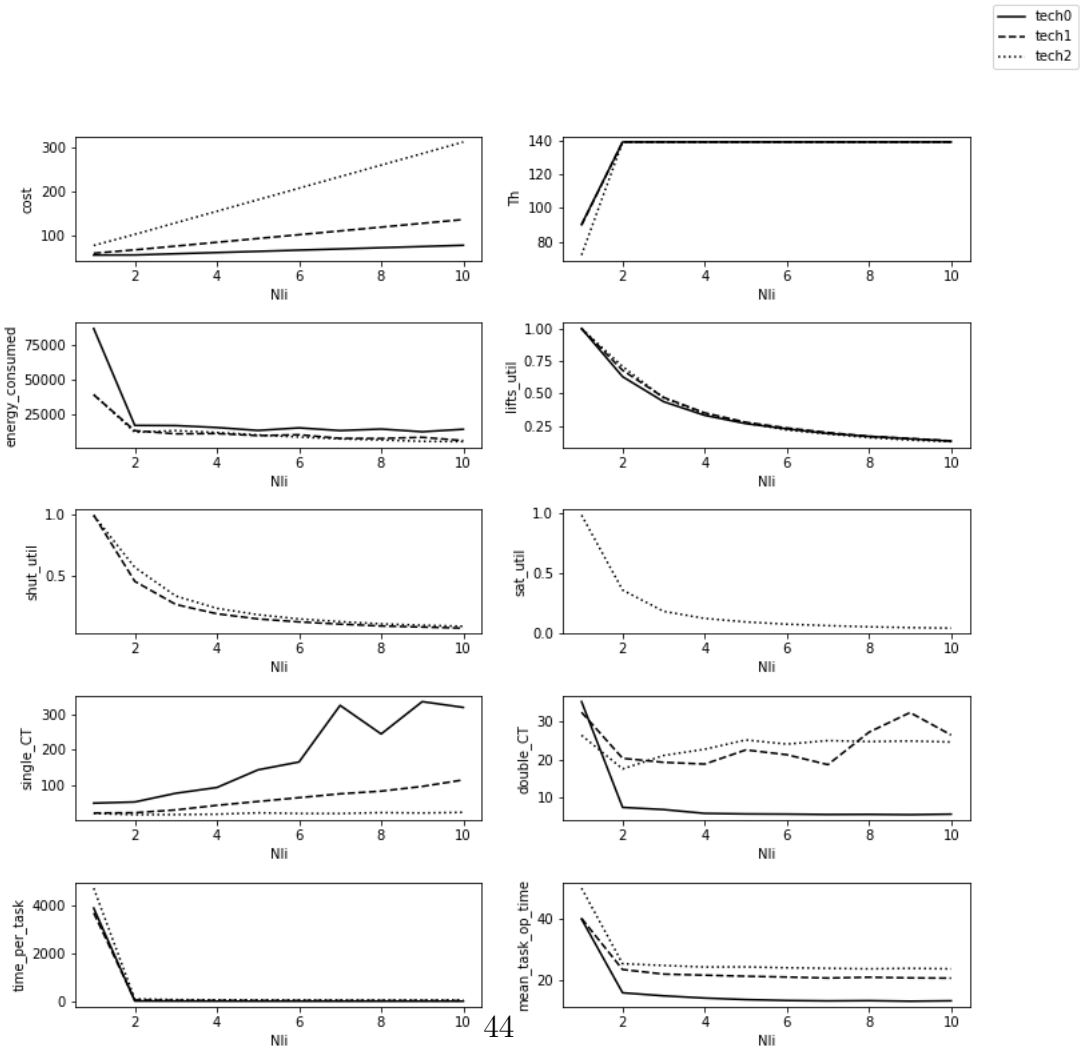


Figure 4.4: Strategy only y

4.3 Infrastructures

4.3.1 Lifts number



Form this figures we can see that the with only one lift the warehouse can keep up with the tasks (as denoted by th Th value). The strategy, as implemented, break up when a long backlog of operation is present, thus greatly worsening performance and energy efficiency.

Increasing the number of lift also increase the number of shuttle for tech1 and shuttle + satellite for tech2, thus the steeper increase in cost in respect to tech0 and the descending resource utilization.

4.3.2 Shuttles number

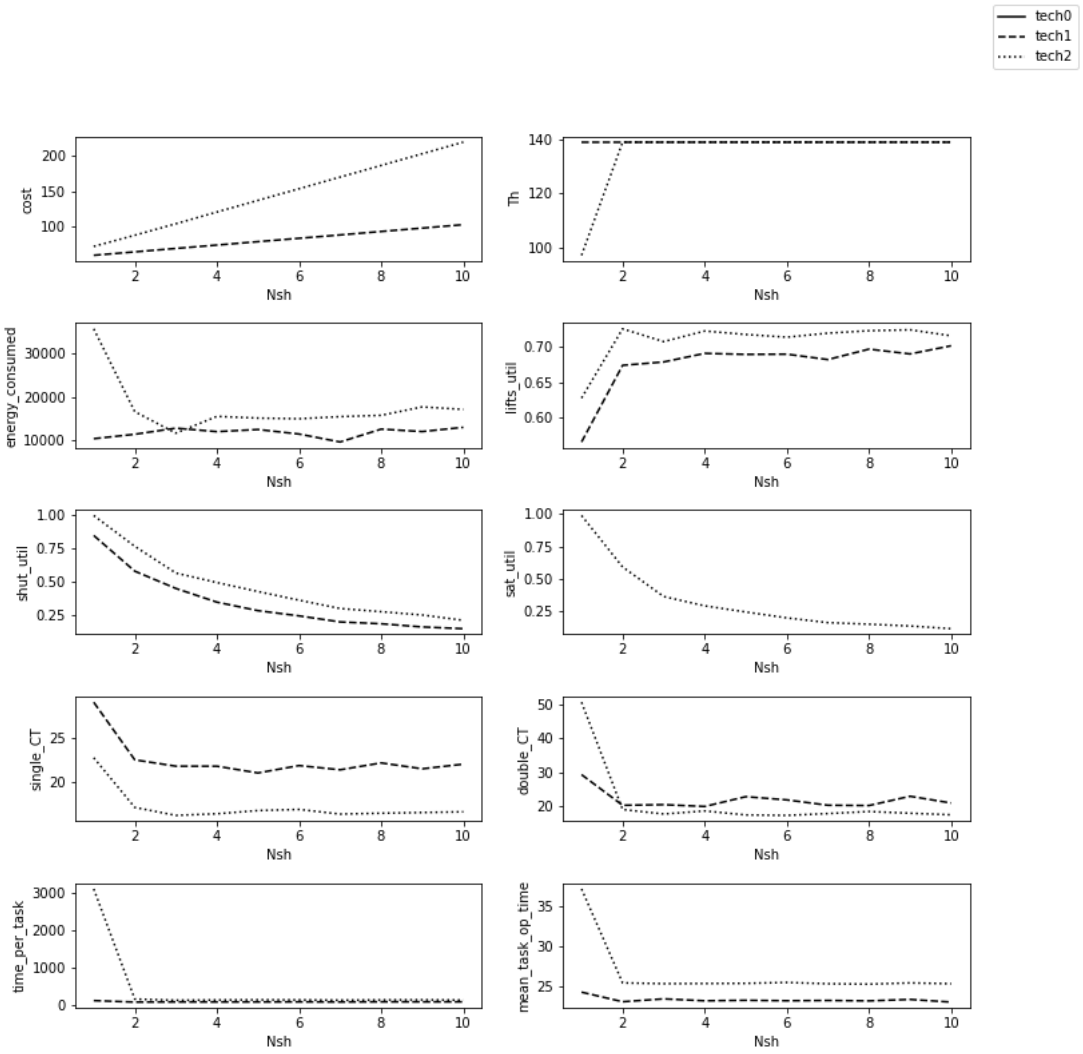


Figure 4.6: Variating Nsh with Nli=2

For this plot the number of lift was set to 2 (nstead of 3), too better show the shuttle number effect. We can see that tech2 isn't able too keep up with only one shuttle. Looking at the lifts utilization plot we can see that, for the 2 tech, the shuttle (and not the lift) become the bottleneck of the system, even if tech1 is able to keep up

with the tasks. The plateau in the same plot since 2 shuttle show that increasing further the shuttle number is not very useful, due to the lifts being the bottleneck.

4.3.3 Satellites number

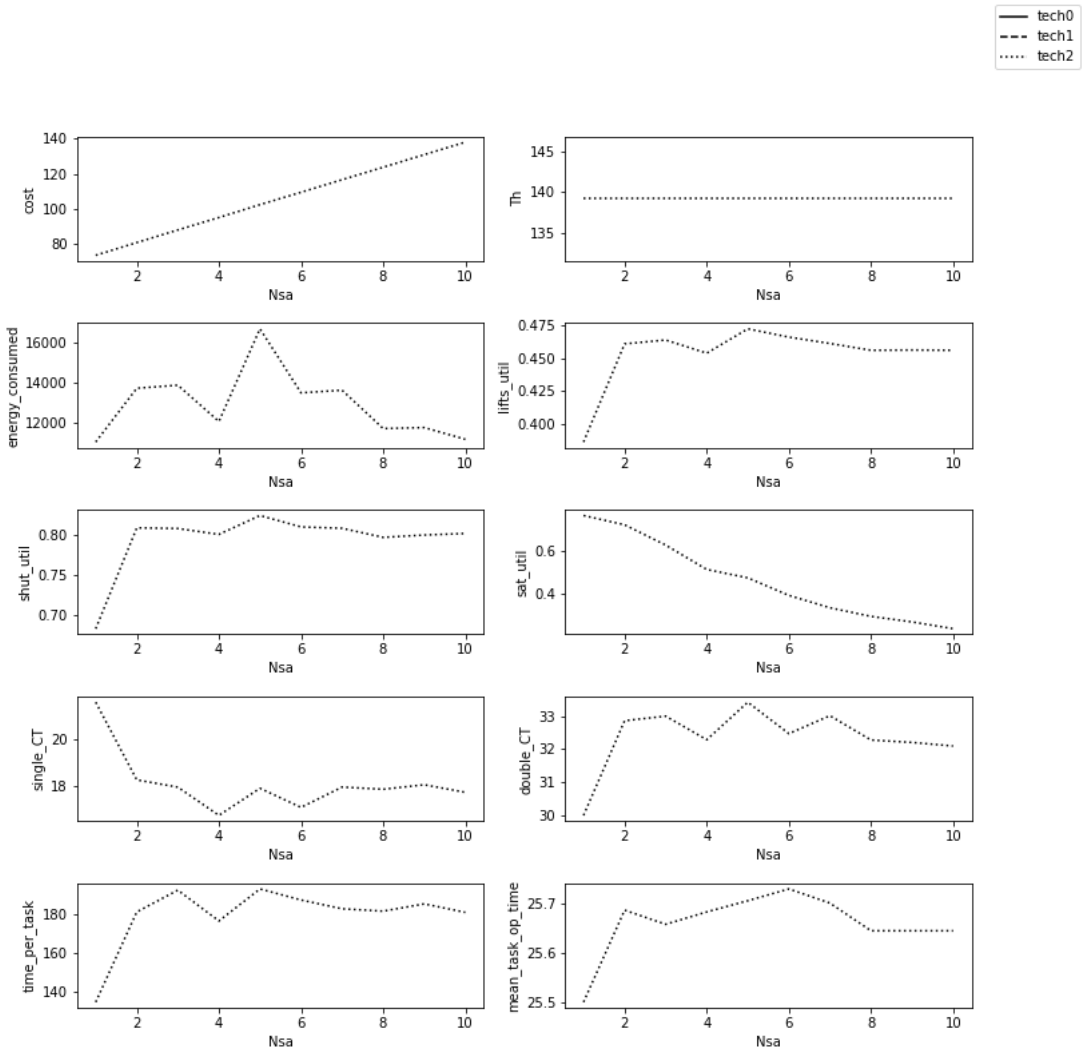


Figure 4.7: Variating Nsa with Nsh=1

Like for Nsh for this plot the number of shuttle per lift was set to 1. We can see from the lifts and shuttles utilization that with 1 satellite the z dimension is the bottleneck.

4.4 Operation strategy

4.4.1 Strategy parameter

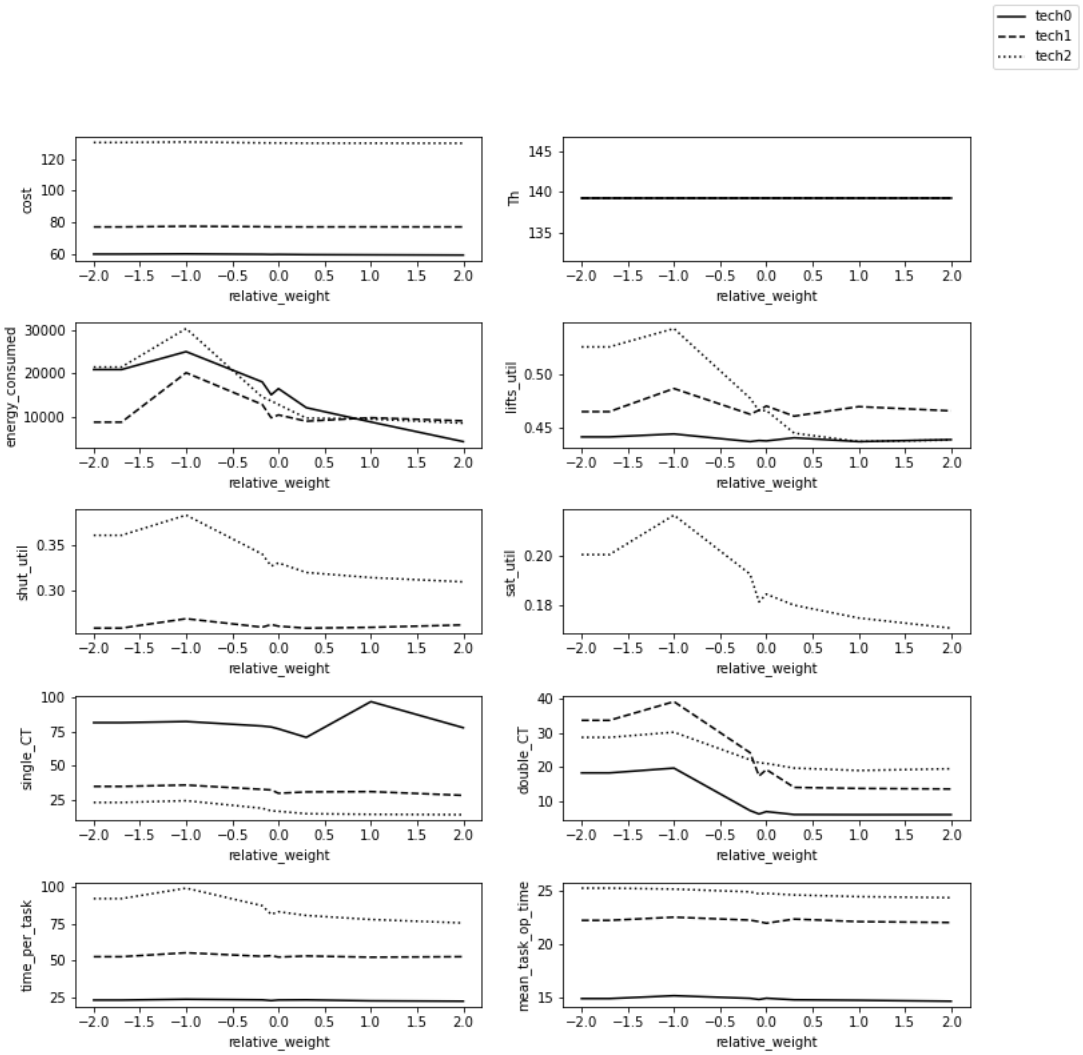


Figure 4.8: Variating $Log(strategy\_par\_y/strategy\_par\_x)$

In this plot we an see the effect of the axis weight on selection of the channels. On the we have only horizontal and on the right only vertical.

4.5 Trace parameter

4.5.1 Warehouse space occupied

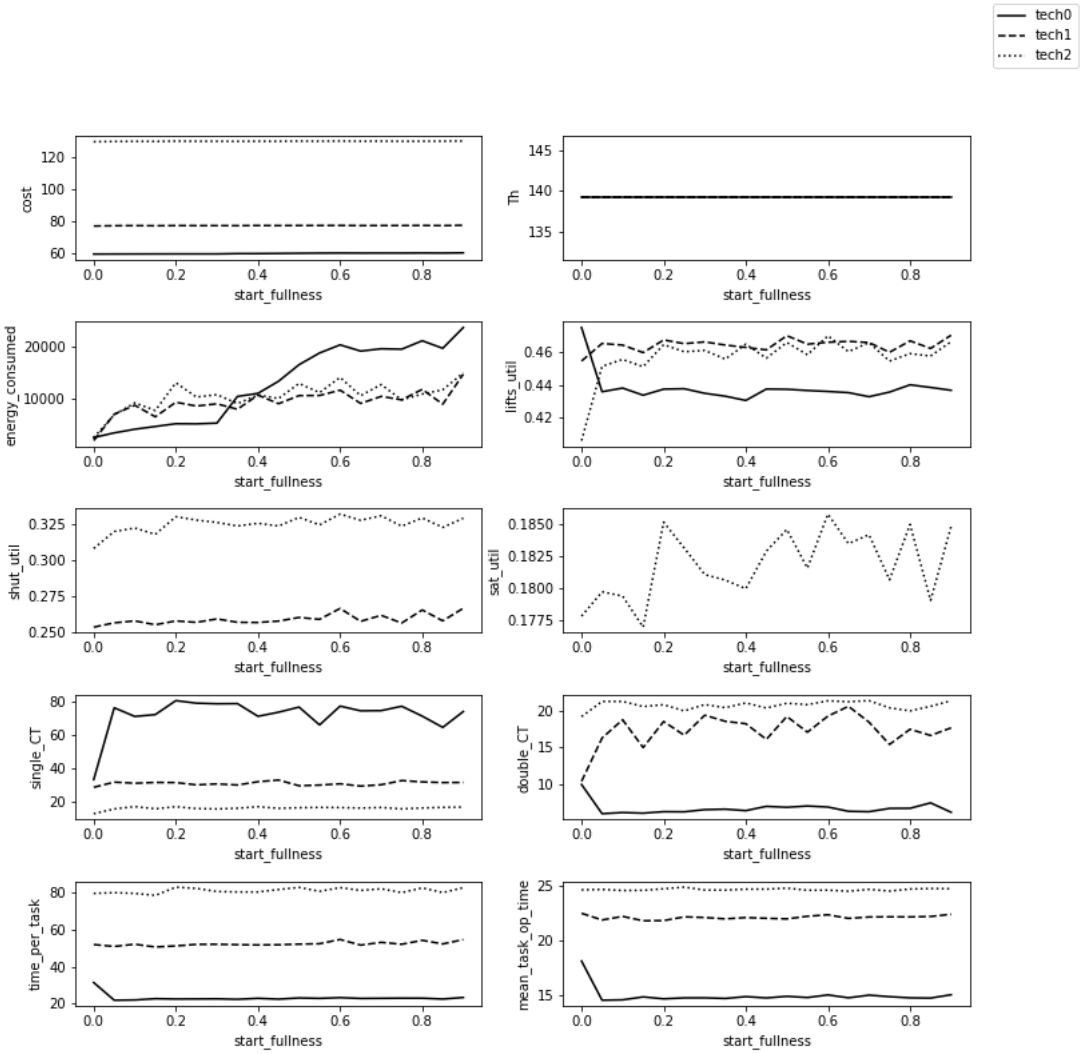


Figure 4.9: Variating start_fullness

The start fullness in the fraction of the ubik that are occupied at the start of the simulation and, due to the way traces are generated, is maintained for in average for the whole simulation. The biggest effect is on energy consumption, specially for tech0, where as seen before an increase in distance travel represent a significant hit to energy performance.

4.5.2 Interarrival time

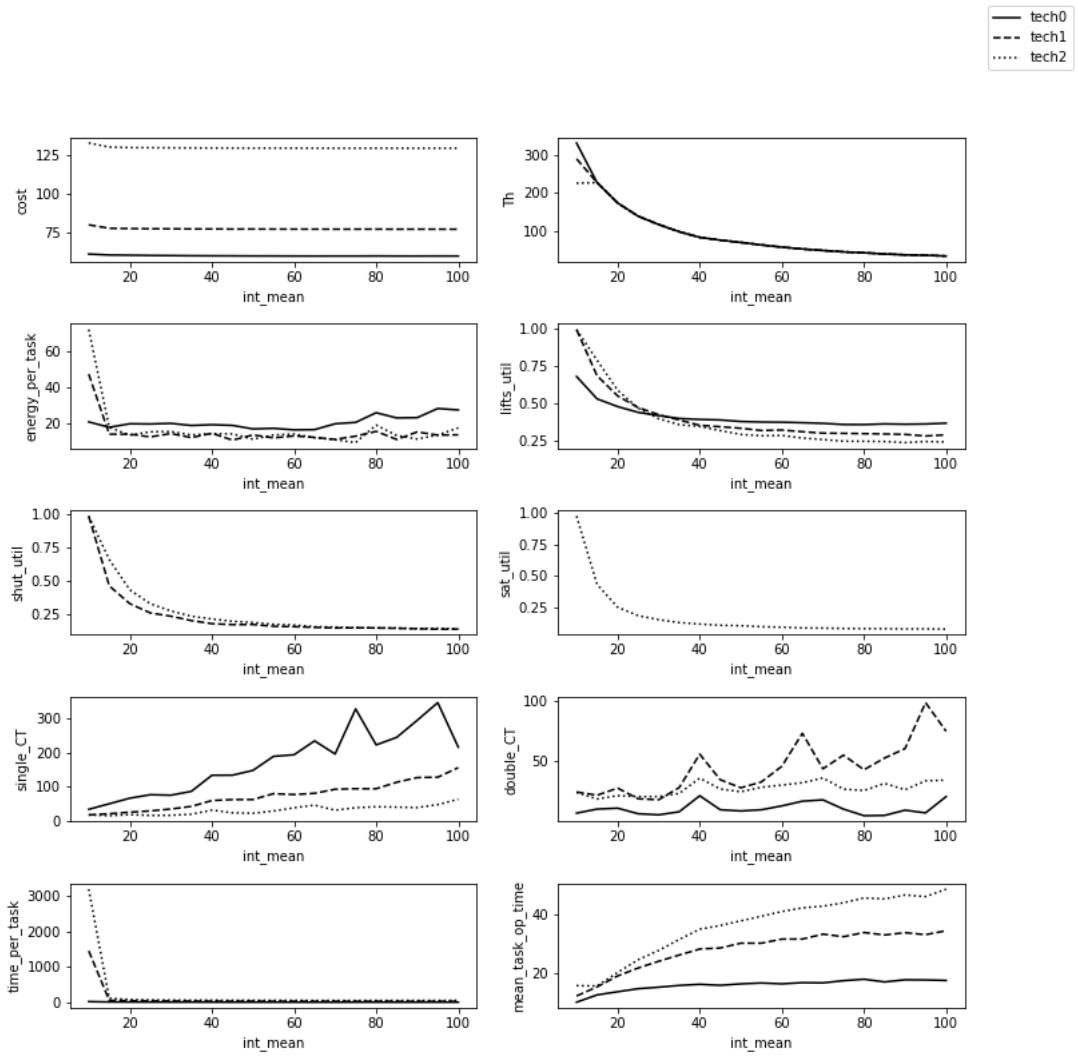


Figure 4.10: Variating int_mean

This is the parameter controlling the mean time between tasks. With to fast arrival the warehouse break down as seen for too few lifts case. The energy consumed plot has been substituted by the energy per task plot, too better display the performance independently by the tasks amount.

Chapter 5

Scenarios discussion

5.1 Introduction

In this section i'm going to run the optimization for different hipotetical cases of warehouses destination. The cases are have different costraint or traces and reppresent different real-world scenarios of aplliction. For each scenerio the optimization try to minimize respectivly the cost and the energy consumption of operation. Optimization for the tree supported technologies are separeted. The cost is computed cosidering the costs of consturction with an amortisation pariod of ten years, plus the cost of the energy consumed. All scenarion start from the same basic parameter,with modificatin specific for the cases. Ranges rappresent minimum and maximum values that can be checked by the optimizer. The duration of simulation is 3 hour.

Table 5.1: Optimization inputs by scenario

	Type	Few Items	Many Items	Few Tasks	Many tasks	Small Area	Reduced acc
rendiment	Fixed	0.9	0.01	0.01	0.01	0.01	0.01
Fr	Fixed	0.05	0.05	0.05	0.05	0.05	0.05
Cr	Fixed	0.00	0.00	0.00	0.00	0.00	0.00
Nsa	To optimize	[1,3]	[1,3]	[1,3]	[1,3]	[1,3]	[1,3]
Nsh	To optimize	[1,3]	[1,3]	[1,3]	[1,3]	[1,3]	[1,3]
Nli	To optimize	[1,5]	[1,5]	[1,5]	[1,5]	[1,5]	[1,5]
Vz	Fixed	1.2	1.2	1.2	1.2	1.2	1.2
Az	Fixed	0.7	0.7	0.7	0.7	0.7	0.2
Vy	Fixed	0.9	0.9	0.9	0.9	0.9	0.9
Ay	Fixed	0.8	0.8	0.8	0.8	0.8	0.2
Vx	Fixed	4.00	4.00	4.00	4.00	4.00	4.00
Ax	Fixed	0.8	0.8	0.8	0.8	0.8	0.2
Cy	Fixed	0.00	0.00	0.00	0.00	0.00	0.00
Wli	Fixed	1850.00	1850.00	1850.00	1850.00	1850.00	1850.00
Wsh	Fixed	850.00	850.00	850.00	850.00	850.00	850.00
Wsa	Fixed	350.00	350.00	350.00	350.00	350.00	350.00
Lz	Fixed	1.2	1.2	1.2	1.2	1.2	1.2
Ly	Fixed	1.5	1.5	1.5	1.5	1.5	1.5
Lx	Fixed	1.00	1.00	1.00	1.00	1.00	1.00
Nz	To optimize	[10,50]	[10,50]	[10,50]	[10,50]	10.00	[10,50]
Ny	To optimize	[3,20]	[3,20]	[3,20]	[3,20]	[3,20]	[3,20]
Nx	To optimize	[10,50]	[10,50]	[10,50]	[10,50]	10.00	[10,50]
bay level	Fixed	0.00	0.00	0.00	0.00	0.00	0.00
strategy	Fixed	1.00	1.00	1.00	1.00	1.00	1.00
strategy par x	To optimize	[0,10]	[0,10]	[0,10]	[0,10]	[0,10]	[0,10]
strategy par y	To optimize	[0,1]	[0,1]	[0,1]	[0,1]	[0,1]	[0,1]
Mean time between tasks	Fixed	25.00	25.00	100.00	20.00	25.00	25.00
Ubik occupied fraction	Fixed	0.5	0.5	0.5	0.5	0.5	0.5

The optimization is done via the Genetic method using the following parameter:

- 1000 generations
- population size of 30
- mutation probability for a character of 0.3
- mutation rate of 0.2
- population swapping of 0.2
- 0.1 bottleneck probability
- 0.8 bottleneck population swap

5.2 Few items types

This scenario rappresent an eviroment where few types of different items are present. This could be the input warehouses for a manifactur using simple input material. This is simulated by only having two types of of items.

Table 5.2: Few items types optimization results

		Transelevatore		Lift, shuttle, fork		Lift, shuttle, satellite	
	Type	Cost	Energy	Cost	Energy	Cost	Energy
Nz	Optimized	13.00	41.00	19.00	24.00	24.00	42.00
Ny	Optimized	4.00	7.00	9.00	4.00	5.00	5.00
Nx	Optimized	36.00	13.00	19.00	24.00	38.00	12.00
Average task op time (sec)	Performance	15.49	13.64	22.40	21.71	25.17	24.10
Average task tot time (sec)	Performance	25.41	25.41	25.41	25.41	25.41	25.41
Th (tasks/hour)	Performance	141.66	141.66	141.66	141.66	141.66	141.66
Completeness	Performance	1.00	1.00	1.00	1.00	1.00	1.00
task done	Performance	425.00	425.00	425.00	425.00	425.00	425.00
Energy consumed (KWh)	Performance	7.28	0.89	20.09	0.23	22.41	1.84
Energy per tasks (Wh)	Performance	17.15	2.10	47.27	0.56	52.73	4.33
Working time (sec)	Performance	6584.00	5798.00	9521.00	9228.00	10699.00	10242.00
Time per task (sec)	Performance	23.26	18.71	54.71	46.16	185.02	504.00
Area	Optimized	468.00	533.00	361.00	576.00	912.00	2520.00
Volume	Optimized	1872.00	3731.00	3249.00	2304.00	4560.00	4.00
Lifts	Optimized	3.00	5.00	4.00	3.00	2.00	8.00
Shuttles	Optimized	0.00	0.00	4.00	6.00	4.00	24.00
Satellites	Optimized	0.00	0.00	0.00	0.00	4.00	0.32
Lifts util	Performance	0.43	0.25	0.33	0.44	0.82	0.32
Shuttles util	Performance	0.00	0.00	0.45	0.34	0.82	0.32
Sats util	Performance	0.00	0.00	0.00	0.00	0.87	0.12
Single cycle (sec)	Performance	85.00	162.08	63.31	29.72	20.81	17.18
Double cycle (sec)	Performance	46.00	5.71	133.78	10.97	31.51	22.15
Single cycle energy (Wh)	Performance	27.25	67.10	28.52	0.11	21.79	2.57
Double cycle energy (Wh)	Performance	15.32	0.05	91.30	0.27	49.31	0.48
Strat par	Optimized	10.00	1.94	10.00	9.50	10.00	1.50
Cost (euro)	Performance	6.29	10.71	11.58	11.62	18.49	47.85

5.2.1 Transelevator

Here we can se that when only energy efficiency is searched, a larger (in the z dimension, and smaller x) warehouse is better. This is probably due to two factor:

- the x movement are the costly (energy wise) for a transelevtor system.

- the z axis can be split in more sections by adding lift, that are broghut from 3 to 5 (the maximum allowed by the optimization parameter). This reduce also the z distance of movement.

The strategy is changed to take more into account the vertical distance. This can be seen in almost every scenerio present here, beign the vertical movements the more energy hungry for all technology. The energy saving is almost 90

5.2.2 Lift,shuttle,fork

Cost

Here we can see that a very tall warehouse is chosen. This is probably due to shuttle beign more costly than lift. So to avoid needing long x movement (which would requiere more shuttle) a tall shape is chosen.

Energy

We observe here the opposite that the transelevator case. When the shuttle high cost is not considered x and z movement are to be preferred, beign the shuttles more energy efficient than a lift. The energy saved is considerable with a minimun increase in the cost.

5.2.3 Lift,shuttle,satellite

The difference between cost and energy optimization is the base shape, heavely preffering z lenght in the latter. The dimension is split using more lift and pararrelized adding a swarm of satellite. This greatly increase the cost, as count of shuttle double and 20 more satellites are needed

5.3 Many items types

This scenario ,the opposite of the last one, present a situation where lots of different goods need to be stored. In this case 40 types are handled.

Table 5.3: Many items types optimization results

		Transelevatore		Lift, shuttle, fork		Lift, shuttle, satellite	
	Type	Cost	Energy	Cost	Energy	Cost	Energy
Nz	Optimized	17.00	17.00	15.00	48.00	17.00	15.00
Ny	Optimized	5.00	19.00	5.00	4.00	6.00	5.00
Nx	Optimized	29.00	11.00	17.00	28.00	44.00	29.00
Average task op time	Performance	25.38	17.77	23.35	21.75	25.36	24.97
Average task tot time	Performance	25.41	25.41	25.41	25.41	25.41	25.41
Th	Performance	141.66	141.66	141.66	141.66	141.66	141.66
Completeness	Performance	1.00	1.00	1.00	1.00	1.00	1.00
task done	Performance	425.00	425.00	425.00	425.00	425.00	425.00
Energy consumed	Performance	16.96	14.22	9.95	2.04	29.06	4.42
Energy consumed per tasks	Performance	39.90	33.45	23.41	4.80	68.38	10.39
Working time	Performance	10787.00	7555.00	9924.00	9245.00	10779.00	10230.00
Time per task	Performance	459.92	37.97	68.24	48.59	210.40	73.20
Area	Optimized	493.00	187.00	255.00	1344.00	748.00	435.00
Volume	Optimized	2465.00	3553.00	1020.00	5376.00	4488.00	2175.00
Lifts	Optimized	1.00	2.00	2.00	5.00	2.00	5.00
Shuttles	Optimized	0.00	0.00	4.00	5.00	4.00	10.00
Satellites	Optimized	0.00	0.00	0.00	0.00	4.00	10.00
Lifts util	Performance	1.00	0.67	0.68	0.24	0.81	0.28
Shuttles util	Performance	0.00	0.00	0.59	0.37	0.81	0.26
Sats util	Performance	0.00	0.00	0.00	0.00	0.86	0.28
Single cycle	Performance	28.76	55.01	27.63	63.02	25.24	31.54
Double cycle	Performance	21.35	16.14	35.42	26.19	31.71	29.96
Single cycle energy	Performance	53.64	58.71	12.91	2.56	33.10	4.61
Double cycle energy	Performance	38.02	16.13	23.77	4.21	56.11	5.37
Strat par	Optimized	8.89	0.74	2.99	0.06	8.14	0.79
Cost	Performance	4.50	6.89	7.56	15.01	18.65	34.92

5.3.1 Transelevator

Like with the few items case a bigger area is preferred for energy efficiency but without the same success, in terms of energy saved, as before.

5.3.2 Lift,shuttle,satellite

Cost

The number of items types seem not to effect this technology as much as the others. Almost the same warehouse parameters are choosen.

Energy

In this case long channels cannot be exploited so the basic strategy of splitting the warehouse in more section, using more lift, is deployed.

5.4 Few Tasks

This scenario present a very low throughput requirement. The mean time between tasks is 100 seconds instead of 25.

Table 5.4: Few tasks optimization results

		Transelevatore		Lift, shuttle, fork		Lift, shuttle, satellite	
	Type	Cost	Energy	Cost	Energy	Cost	Energy
Nz	Optimized	11.00	46.00	45.00	16.00	16.00	34.00
Ny	Optimized	3.00	6.00	9.00	5.00	3.00	6.00
Nx	Optimized	41.00	17.00	13.00	44.00	14.00	29.00
Average task op time	Performance	20.38	16.57	49.73	30.42	50.42	41.11
Average task tot time	Performance	89.26	89.25	89.26	89.25	89.25	89.25
Th	Performance	40.33	40.33	40.33	40,33	40.33	40.33
Completeness	Performance	1.00	1.00	1.00	1.00	1.00	1.00
task done	Performance	121.00	121.00	121.00	121.00	121.00	121.00
Energy consumed	Performance	5.29	0.49	4.31	0.57	1.35	0.11
Energy consumed per tasks	Performance	43.72	4.06	35.60	4.69	11.15	0.93
Working time	Performance	2467.00	2006.00	4929.00	3681.00	6101.00	4974.00
Time per task	Performance	25.04	19.33	64.28	40.18	84.37	58.43
Area	Optimized	462.00	799.00	585.00	704.00	224.00	986.00
Volume	Optimized	1386.00	4794.00	5265.00	3520.00	672.00	5916.00
Lifts	Optimized	1.00	5.00	1.00	4.00	1.00	0.13
Shuttles	Optimized	0.00	0.00	3.00	4.00	2.00	0.24
Satellites	Optimized	0.00	0.00	0.00	0.00	4.00	0.14
Lifts util	Performance	0.99	0.22	0.83	0.21	0.72	10.41
Shuttles util	Performance	0.00	0.00	0.50	0.30	0.67	24.97
Sats util	Performance	0.00	0.00	0.00	0.00	0.41	0.14
Single cycle	Performance	72.83	26.00	39.79	201.88	25.64	10.41
Double cycle	Performance	35.48	5.82	79.00	204.39	35.43	24.97
Single cycle energy	Performance	67.79	49.99	23.07	3.17	4.07	0.31
Double cycle energy	Performance	21.12	1.92	38.36	5.16	9.63	0.60
Strat par	Optimized	10.00	1.35	5.72	4.48	8.02	1.39
Cost	Performance	2.98	11.79	9.84	11.16	9.64	30.05

5.4.1 Transelevator

When optimizing by cost infrastructure are reduced to the minimal (only one lift), given that tasks are slow.

When optimization by energy the same strategy as few items type (emphasis on the z dimension and lots of lifts) is deployed. This is the strategy used by all three technologies, probably due to the fact that concurrent operation cannot be used (the delay between tasks is bigger than the time taken a single task)

5.5 Many Tasks

This scenario present a higher than normal throughput requirement. The mean time between tasks is 20 seconds instead of 25.

Table 5.5: Many tasks optimization results

		Transelevatore		Lift, shuttle, fork		Lift, shuttle, satellite	
	Type	Cost	Energy	Cost	Energy	Cost	Energy
Nz	Optimized	29.00	29.00	29.00	30.00	14.00	12.00
Ny	Optimized	3.00	7.00	7.00	9.00	10.00	7.00
Nx	Optimized	16.00	41.00	41.00	30.00	45.00	30.00
Average task op time	Performance	15.28	12.19	19.53	18.35	20.17	19.64
Average task tot time	Performance	20.18	20.18	20.19	20.18	20.19	20.19
Th	Performance	178.33	178.33	178.33	178.33	178.33	178.33
Completeness	Performance	1.00	1.00	1.00	1.00	1.00	1.00
task done	Performance	535.00	535.00	535.00	535.00	535.00	535.00
Energy consumed	Performance	6.87	3.68	9.13	4.59	6.71	4.44
Energy consumed per tasks	Performance	12.83	6.87	17.06	8.58	12.54	8.29
Working time	Performance	8175.00	3054.00	10453.00	9816.00	10789.00	10505.00
Time per task	Performance	30.20	21.33	67.61	46.62	261.81	75.82
Area	Optimized	464.00	1392.00	1189.00	1200.00	630.00	360.00
Volume	Optimized	1392.00	22272.00	8323.00	10800.00	6300.00	2520.00
Lifts	Optimized	2.00	4.00	3.00	5.00	3.00	5.00
Shuttles	Optimized	0.00	0.00	3.00	10.00	3.00	10.00
Satellites	Optimized	0.00	0.00	0.00	0.00	3.00	10.00
Lifts util	Performance	0.67	0.36	0.46	0.32	0.49	0.35
Shuttles util	Performance	0.00	0.00	0.68	0.25	0.85	0.33
Sats util	Performance	0.00	0.00	0.00	0.00	0.95	0.36
Single cycle	Performance	43.32	84.51	37.91	44.09	19.66	21.88
Double cycle	Performance	23.05	12.41	47.65	23.33	29.43	27.88
Single cycle energy	Performance	21.66	20.13	8.94	3.34	10.53	4.40
Double cycle energy	Performance	8.93	1.63	26.41	6.53	0.95	3.34
Strat par	Optimized	10.00	4.25	9.73	3.31	0.70	5.57
Cost	Performance	4.41	28.49	14.52	26.67	17.50	35.28

5.5.1 Transelevator

Like before, splitting in section adding lifts.

5.5.2 Lift,shuttle,satellite

Technologies using shuttles can exploit parallel movement, in a situation where inter arrival time is shorter than the cycle time, to save energy. But having a slower cycle times and lifts being still a bottleneck they have for to use more of them to keep up with tasks, even when optimizing by cost, than the transelevator technology.

5.6 Small Area

This scenario analyze the problem of having a small warehouses area. This is done by locking the area parameter(Nx and Nz) of the warehouse to 10.

Table 5.6: Fixed area optimization results

		Transelevatore		Lift, shuttle, fork		Lift, shuttle, satellite	
	Type	Cost	Energy	Cost	Energy	Cost	Energy
Nz	Optimized	10.00	10.00	10.00	10.00	10.00	10.00
Ny	Optimized	4.00	6.00	8.00	5.00	18.00	8.00
Nx	Optimized	10.00	10.00	10.00	10.00	10.00	10.00
Average task op time	Performance	14.86	15.62	19.52	18.00	19.97	19.81
Average task tot time	Performance	20.19	20.19	20.19	20.18	20.19	20.19
Th	Performance	178.33	178.33	178.33	178.33	178.33	179.33
Completeness	Performance	1.00	1.00	1.00	1.00	1.00	1.00
task done	Performance	535.00	535.00	535.00	535.00	535.00	535.00
Energy consumed	Performance	6.22	1.28	23.37	1.17	15.38	1.97
Energy consumed per tasks	Performance	11.63	2.39	43.69	2.19	28.76	3.69
Working time	Performance	7952.00	6513.00	10444.00	6928.00	10688.00	10600.00
Time per task	Performance	27.78	18.24	79.53	43.92	108.84	31.96
Area	Optimized	100.00	100.00	100.00	100.00	100.00	100.00
Volume	Optimized	400.00	600.00	800.00	500.00	1800.00	800.00
Lifts	Optimized	2.00	5.00	3.00	5.00	3.00	5.00
Shuttles	Optimized	0.00	0.00	3.00	10.00	6.00	10.00
Satellites	Optimized	0.00	0.00	0.00	0.00	6.00	10.00
Lifts util	Performance	0.67	0.28	0.52	0.31	0.63	0.33
Shuttles util	Performance	0.00	0.00	0.71	0.24	0.61	0.31
Sats util	Performance	0.00	0.00	0.00	0.00	0.66	0.34
Single cycle	Performance	40.57	138.35	43.28	48.18	20.04	19.71
Double cycle	Performance	14.69	0.29	70.44	26.23	24.60	23.53
Single cycle energy	Performance	21.27	3.77	26.30	0.98	13.28	1.60
Double cycle energy	Performance	5.73	0.29	81.50	1.59	18.19	1.87
Strat par	Optimized	6.97	3.77	10.00	8.03	9.20	1.97
Cost	Performance	3.37	7.51	7.30	15.96	21.91	33.42

5.7 Fragile items (reduced acceleration)

This scenerio present the problem of having a goods that have to be handled with care, thus limiting the acceleraion in movement. This is done by locking the acceleration to $0.2m/sec^2$.

Table 5.7: Reduced acceleration optimization results

		Transelevatore		Lift, shuttle, fork		Lift, shuttle, satellite	
	Type	Cost	Energy	Cost	Energy	Cost	Energy
Nz	Optimized	11.00	30.00	18.00	44.00	10.00	43.00
Ny	Optimized	5.00	3.00	14.00	9.00	11.00	5.00
Nx	Optimized	20.00	49.00	16.00	42.00	27.00	17.00
Average task op time	Performance	18.00	15.97	24.33	22.98	25.33	24.61
Average task tot time	Performance	25.41	25.41	25.41	25.41	25.41	25.41
Th	Performance	141.66	141.66	141.66	141.66	141.66	141.66
Completeness	Performance	1.00	1.00	1.00	1.00	1.00	1.00
task done	Performance	425.00	425.00	425.00	425.00	425.00	425.00
Energy consumed	Performance	1.79	0.55	5.87	1.18	4.07	2.29
Energy consumed per tasks	Performance	4.21	1.30	13.80	2.78	9.58	5.38
Working time	Performance	7635.00	6788.00	10341.00	9382.00	10766.00	10548.00
Time per task	Performance	30.91	24.28	119.53	50.53	192.73	83.90
Area	Optimized	220.00	980.00	288.00	1848.00	270.00	731.00
Volume	Optimized	1100.00	2940.00	4032.00	16632.00	2970.00	3655.00
Lifts	Optimized	2.00	4.00	2.00	4.00	3.00	4.00
Shuttles	Optimized	0.00	0.00	2.00	8.00	3.00	8.00
Satellites	Optimized	0.00	0.00	0.00	0.00	6.00	24.00
Lifts util	Performance	0.64	0.34	0.59	0.34	0.53	0.38
Shuttles util	Performance	0.00	0.00	0.88	0.27	0.85	0.36
Sats util	Performance	0.00	0.00	0.00	0.00	0.76	0.14
Single cycle	Performance	57.06	121.14	32.78	46.85	20.99	25.99
Double cycle	Performance	13.36	7.05	39.57	25.95	42.14	30.04
Single cycle energy	Performance	8.90	11.23	7.60	1.20	3.68	2.24
Double cycle energy	Performance	1.46	0.50	19.81	2.24	6.50	3.43
Strat par	Optimized	3.44	0.57	2.56	2.41	8.74	3.16
Cost	Performance	3.93	8.52	8.12	29.45	17.58	49.03

Chapter 6

Conclusions

The discrete simulation approach for modeling warehouse allows for great flexibility, especially if the simulation is written as a framework where business logic can be defined in plain python. Using this method is possible, and easy with basic programming skills and language knowledge, to adapt the model to a specific case. An autonomous warehouse producer can implement in the simulation its set of technology and logic of operation, and thus have a functional model of fitted for its need without starting from scratch.

The tool can also be used for theoretical research, for example for testing logics of operation developed outside of the simulation scope. This is made possible by the general structure of the simulator.

The other use for the simulator can be the integration with others models, to achieve a wider representation of the industry process. This can be very useful in the process of building a digital double of a particular production or distribution system. This is by the fat that the simulator is written in Python, and not using a particular simulation tool, thus allowing for easy manipulation of input and output of the model.

The simple results of simulation discussed above, show how even only variating an input parameter its possible to identify some technology specific behaviour. This quirks can be non trivial to discover a priori, due to concurrent systems intrinsic complexity.

Energy saving is becoming a central concern in industry, an is probably going to grow in importance in the future. A simulator have to be capable of computing the energy efficiency of the warehouse, being it a performance parameter that is joining cycle time and deployment cost in the core evaluation metrics. Optimizations thus need to be multi objective to be used generally, each user has its own constraint and goals.

The optimizators runs presented is this thesis shown that the space for big energy saving exist, without greatly increasing building cost. The energy saving strategy is not general for all scenarios but need to be tailored to the specific design of warehouse, but this method could be used to identify a set recurrent directions.

References

[1] URL: https://www.statista.com/statistics/1094202/global-warehouse-automation-market-size/.

[2] URL: https://simpy.readthedocs.io/en/latest/contents.html.

[3] R. Accorsi et al. "Time and energy based assignment strategy for unit-load AS/RS warehouses". In: cited By 0. 2015. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84949679507&partnerID=40&md5=766cfb60e931e5eb6a6e934c6c67f970.

[4] T. Bartkowiak et al. "Novel approach to semi-automated warehouse for manufacturing: Design and simulation". In: vol. 591. 1. cited By 0. 2019. DOI: 10.1088/1757-899X/591/1/012040. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85072109861&doi=10.1088%2f1757-899X%2f591%2f1%2f012040&partnerID=40&md5=8ddb84e1dcedf95d10005dcecb40a1aa.

[5] A.C. Brito and J.A. Basto. "Automated warehouse design using visual interactive simulation". In: cited By 0. 2006, pp. 593–598. DOI: 10.7148/2006-0593. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84857774381&doi=10.7148%2f2006-0593&partnerID=40&md5=521ac1bd4d94610f96f2f12c4f3a5913.

[6] Ekren BY. "Graph-based solution for performance evalua- tion of Shuttle-Based Storage and Retrieval System. Int J Prod Res 55(21):6516–6526". In: (2017). DOI: http://dx.doi.org/10.1080/00207543.2016.1203076.

[7] Sprock T. Bock C. "SysML models for discrete event logistics systems". In: (2020). DOI: http://dx.doi.org/10.6028/JRES.125.023.

[8] L.S. Duncan. "SYSTEM DESIGN TRENDS IN AUTOMATED WAREHOUSING." In: cited By 0. 1985, pp. 23–34. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0022314228&partnerID=40&md5=15664ec19033550dcdc09bf8b7ab8e32.

[9] M. Eder. "An analytical approach for a performance calculation of shuttle-based storage and retrieval systems". In: *Production and Manufacturing Research* 7.1 (2019). cited By 5, pp. 255–270. DOI: 10.1080/21693277.2019.1619102. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85066146716&doi=10.1080%2f21693277.2019.1619102&partnerID=40&md5=cab538fff2bbf8bde724f6762386a514.

[10] M. Eder. "An approach for a performance calculation of shuttle-based storage and retrieval systems with multiple-deep storage". In: *International Journal of Advanced Manufacturing Technology* 107.1-2 (2020). cited By 2, pp. 859–873. DOI: 10.1007/s00170-019-04831-7. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85079778212&doi=10.1007%2fs00170-019-04831-7&partnerID=40&md5=cbabdff2e7470a9e0f0ca1a9dd6d1915.

[11] M. Eder. "An approach for performance evaluation of SBS/RS with shuttle vehicles serving multiple tiers of multiple-deep storage rack". In: *International Journal of Advanced Manufacturing Technology* 110.11-12 (2020). cited By 0, pp. 3241–3256. DOI: 10.1007/s00170-020-06033-y. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85091165258&doi=10.1007%2fs00170-020-06033-y&partnerID=40&md5=70d6105339874ebe418bff018ed48f69.

[12] B.Y. Ekren. "A simulation-based experimental design for SBS/RS warehouse design by considering energy related performance metrics". In: *Simulation Modelling Practice and Theory* 98 (2020). cited By 5. DOI: 10.1016/j.simpat.2019.101991. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85072840199&doi=10.1016%2fj.simpat.2019.101991&partnerID=40&md5=9ad6eecae756c71c7d7a10523ecb8d33.

[13] B.Y. Ekren, Z. Sari, and T. Lerher. "Warehouse design under class-based storage policy of shuttle-based storage and retrieval system". In: *IFAC-PapersOnLine* 28.3 (2015). cited By 16, pp. 1152–1154. DOI: 10.1016/j.ifacol.2015.06.239. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84953878654&doi=10.1016%2fj.ifacol.2015.06.239&partnerID=40&md5=5157607c597bb94e662e20958a5f7add.

[14] Sari Z Lerher T Ekren BY Akpunar A. "A tool for time, variance and energy related performance estimations in a shuttle- based storage and retrieval system. Appl Math Model". In: (2018). DOI: http://dx.doi.org/10.1016/j.apm.2018.06.037.

[15] L. He, L. Zhao, and X. Ma. "Optimization design of automated warehouse storage area based on nonlinear programming". In: *Dongnan Daxue Xuebao (Ziran Kexue Ban)/Journal of Southeast University (Natural Science Edition)* 37.SUPPL. 2 (2007). cited By 3, pp. 309–315. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-41949134063&partnerID=40&md5=a8089b1517273e9c3e6804b786ee5004.

[16] K. Liu and Z. Cao. "Energy-optimized task scheduling of automated warehouse based on improved grey wolf optimizer". In: *Jisuanji Jicheng Zhizao Xitong/Computer Integrated Manufacturing Systems, CIMS* 26.2 (2020). cited By 1, pp. 376–383. DOI: 10.13196/j.cims.2020.02.010. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85082474966&doi=10.13196%2fj.cims.2020.02.010&partnerID=40&md5=e671b0d022b19502557edde1889c2809.

[17] Bäck Thomas Thiele Lothar. "A Comparison of Selection Schemes Used in Evolutionary Algorithms". In: (1996). DOI: https://doi.org/10.1162/evco.1996.4.4.361.

[18] Mitchell Melanie. *An Introduction to Genetic Algorithms*. MIT Press, 1996. ISBN: 9780585030944.

[19] T. Nishi et al. “Cell-Based Local Search Heuristics for Guide Path Design of Automated Guided Vehicle Systems with Dynamic Multicommodity Flow”. In: *IEEE Transactions on Automation Science and Engineering* 17.2 (2020). cited By 0, pp. 966–980. DOI: 10.1109/TASE.2019.2952920. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85083193027&doi=10.1109%2fTASE.2019.2952920&partnerID=40&md5=4af0e8ca4043fc877eae8b7c8f34a747.

[20] Lande R. “Genetics and demography in biological conservation”. In: (1988). DOI: https://doi.org/10.1126/science.3420403.

[21] M. Rajković et al. “A Multi-objective optimization model for minimizing cost, travel time and CO¡inf¿2¡/inf¿ emission in an AS/RS”. In: *FME Transactions* 45.4 (2017). cited By 5, pp. 620–629. DOI: 10.5937/fmet1704620R. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85021253157&doi=10.5937%2ffmet1704620R&partnerID=40&md5=c621e7ae13fb5a6d18ebcc7ebeef45e9.

[22] N.C. Truong, T.G. Dang, and D.A. Nguyen. “Optimizing automated storage and retrieval algorithm in cold warehouse by combining dynamic routing and continuous cluster method”. In: *Lecture Notes in Electrical Engineering* 554 (2020). cited By 0, pp. 283–293. DOI: 10.1007/978-3-030-14907-9_28. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85066297740&doi=10.1007%2f978-3-030-14907-9_28&partnerID=40&md5=e65fff446767380412e58315a6adf85f.

[23] L. Xing et al. “A novel tabu search algorithm for multi-AGV routing problem”. In: *Mathematics* 8.2 (2020). cited By 3. DOI: 10.3390/math8020279. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85080112878&doi=10.3390%2fmath8020279&partnerID=40&md5=a03f46264c01ed0ee87f86654ec4b2f7.

[24] B. Yetkin Ekren. “A multi-objective optimisation study for the design of an AVS/RS warehouse”. In: *International Journal of Production Research* (2020). cited By 1. DOI: 10.1080/00207543.2020.1720927. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85078928052&doi=10.1080%2f00207543.2020.1720927&partnerID=40&md5=e3c256925adaee7894374dcd927a9fa7.

[25] F. Zhang et al. “Design Optimization of Redundantly Actuated Cable-Driven Parallel Robots for Automated Warehouse System”. In: *IEEE Access* 8 (2020). cited By 0, pp. 56867–56879. DOI: 10.1109/ACCESS.2020.2981546. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85082933575&doi=10.1109%2fACCESS.2020.2981546&partnerID=40&md5=93ee46f90619166e9f11e6e86c1

[26] X. Zhao et al. “Analysis of the Shuttle-Based Storage and Retrieval System”. In: *IEEE Access* 8 (2020). cited By 0, pp. 146154–146165. DOI: 10.1109/ACCESS.2020.3014102. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85090295945&doi=10.1109%2fACCESS.2020.3014102&partnerID=40&md5=b72f50a7c0192dc24355e665386cb3b4.