Modeling Project Work 2

Optimization of the Picking Process in Inventories

Le Phuong Quynh

Supervisor: Dr. Jüttner Alpár
Péter Madarasi

December 8, 2020

1 Introduction

Order picking, the process of retrieving items from their storage locations to fill customer orders, is known as the most time consuming and laborious component of the warehouse activities. The project aim to simulate this process in the warehouse. More specifically, picker-to-parts systems (i.e, the order picker travels along the aisles to retrieve products) are considered, as these systems account for the large majority of all order picking systems in Western Europe. We initially use several assumptions in order to make the model simple and easy to construct. However, since we also aim to real-life applications, some of those assumptions can be relaxed later on.
This report is organized as follows. In the section 2, project overview and objectives, the author quickly remarks what has been done in the previous project work and states a scope of work for this project. In section 3, set up and block diagram of the programming part are presented. In addition, I also briefly introduce object oriented program which is the main improvement in this project work. After setting up code with sample data sets, some evaluations and virtual analysis will be presented in section 4.

2 Project Overview and Objectives

Previous work

In the last semester, I had introduced simple activities in the warehouse and an optimizing problem: to reduce working time inside warehouse by avoiding collisions in picking products process. I also implemented a programming work by Python language to simulate the simple picking process using "first come first serve" rule, together with a source code to generate random data sets for testing purpose.
In the simple set up inside warehouse, there is a number of locations where products are placed. A worker working inside warehouse reaches the locations in an order given by customers' requirements. A collision may arise when at a moment there are two or more workers wanting to get product at the same place. "First come first serve" rule states that a worker who comes first will have priority and all others need to wait until this man leaves. A result in my previous work shows that a raise in number of workers may reduce total processing time inside warehouse (with the same given customers' orders). It is not always true since the more people the more collisions they might have, in that cases we will encounter the problem of increasing delay time.
However, by practical testing, the old source code shows several bugs leading to a serious problem. In the case of big collisions, i.e the waiting time is required long, the code does not shift the delay time for each worker correctly then lead

to a wrong in order of taking tasks. Since each task is ordering by customers' requirements, pickers are desired to complete a whole task before move to the next one. Therefore, fixing bugs is an urgent mission in this following project. In addition, project also contains some updates in programming methodology and evaluations.

Scope of work

In this semester, my work is divided into three main parts including:
First, as mentioned above, fixing bugs is the most important task before having to any further improvement.
Second, cleaning code and implementing object oriented program (OOP).
Third, evaluating the results. Here we still use the same old source to generate random data sets, however, it is expected to have more values being used to evaluate and analyze.

3 Set up

Basic Conditions

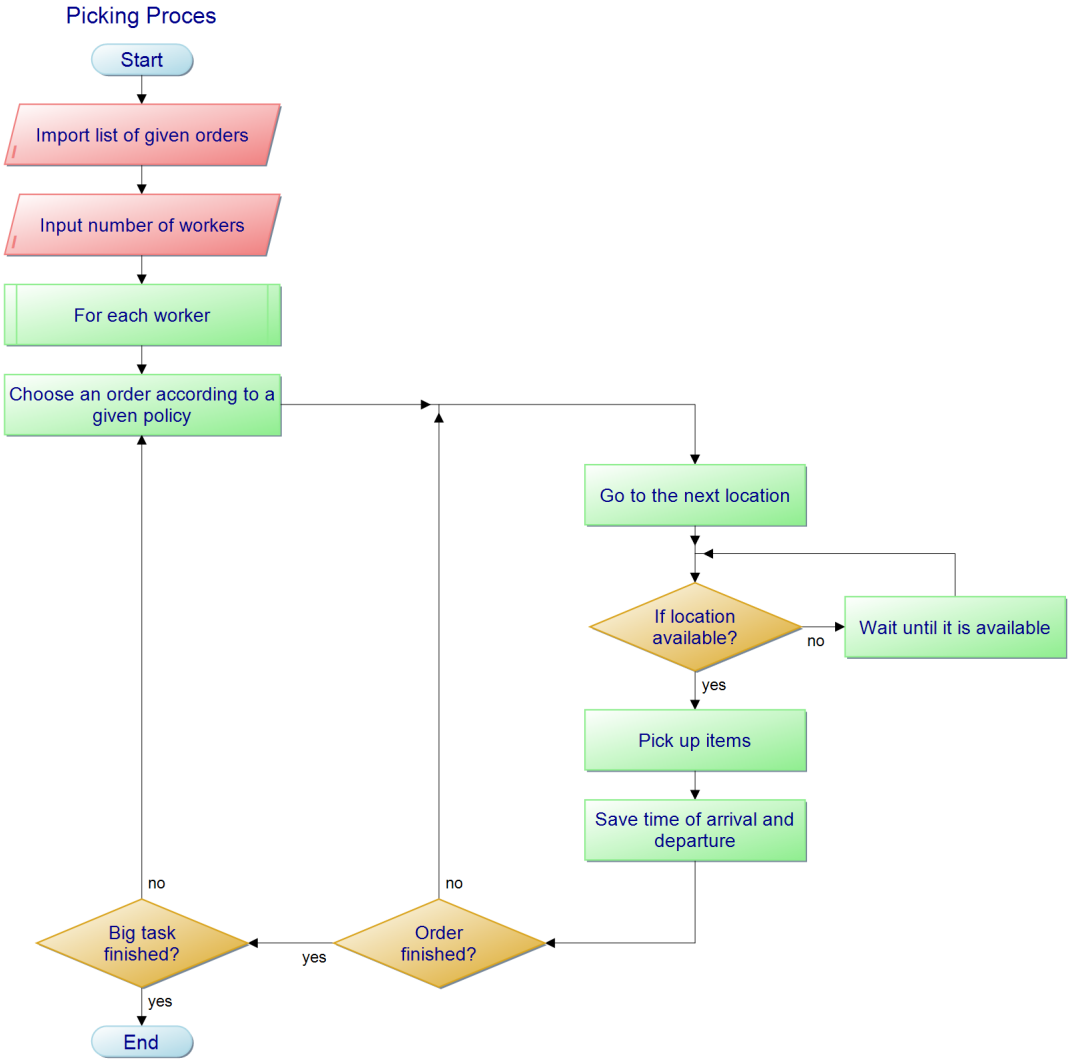
The system based on the following assumptions to simulate:
1. There are variety of goods in the system. Each location in the warehouse can have only one type of goods. The demand for a variety of goods is fixed and known, at this time we also assume that we need not to care about the amount of goods in stock, i.e, the remaining goods always exceeds demand.
2. To simplify the processing, here have only one speed for all workers, said a maximum speed. Moreover, except the collision mentioned at the beginning, we consider all other traffic jams during working time are neglectable. It helps us to have a fixed information of traveling time between any two locations.
3. **Picker to Order**: the picker is responsible for collecting items related to an order. With a detailed list of the items to be collected, the picker travels to the locations where the goods are stored, picks the needed amount and move the the next location. After finishing an order, the picker comes back to the starting point to leave items for packaging and delivery process then get a new order here.

Programming Set Up

Develop a Controller class to hold a dictionary of pickers and their picking processes in the warehouse
- A dictionary is initially empty.
- Picker takes an order from big task according to a policy, new order is taken only if the previous order is done.

- Picker can be looked up by key in the dictionary, which is numbered from 1.
- A value in dictionary is the process of that picker from the start to the end including locations, arriving and leaving time that locations and waiting time if collision happens.
- User can look up a location at a time to check if there is any collision.

Order picking simulation allows calculating and analyzing all activities performed by pickers. The algorithm calculates the status duration of pickers (e.g, traveling time, picking up time, waiting time). The given status time is used to estimate the total order picking time. Simplified form of algorithm for each picker is presented below:

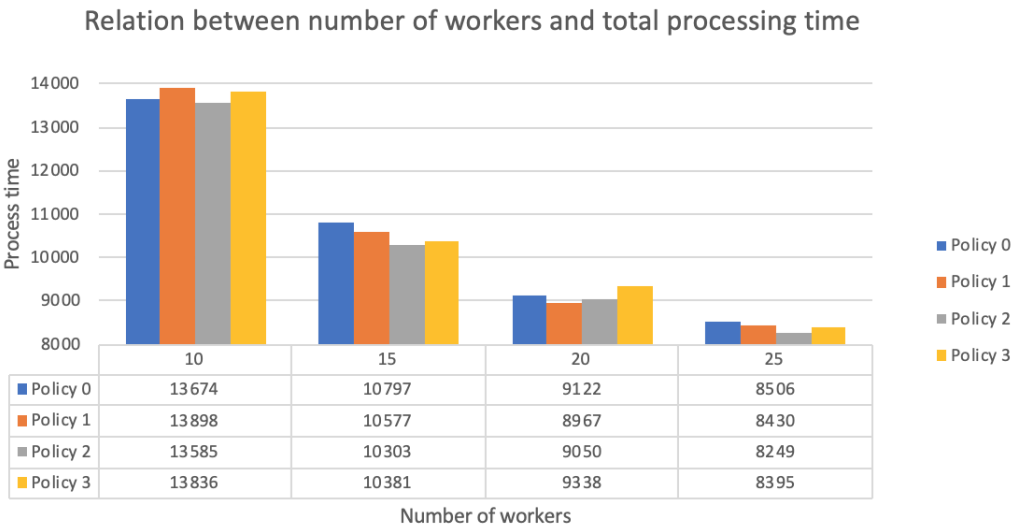


In this report, I implemented 3 new policies for choosing the next order for a picker. All of three policies will consider the time point where a picker has finished the previous task and already comeback to the starting location.
At that specific time point, we can check for every location that if it is occupying by other pickers or not. If yes, we can calculate the time needed till

that location would be free, which is call estimated queuing time. For each order, there is a set of given locations then we can compute sum of queuing time from those locations. For policy 1, chosen order is the one has this number minimum.
For the next case of choosing order, we choose the minimum average queuing time, i.e, we take sum of queuing time divide by number of locations contained in that order and choose the one has min result.
Moreover, it makes sense that we can compare these queuing time with the estimated processing time of an order (total time a worker will need if we get rid of all collisions). Thus, for the third policy, we choose the order has the smallest ratio.
The policy 0 is simply the case of choosing a random order from list of given orders.

4 Evaluations and Analysis

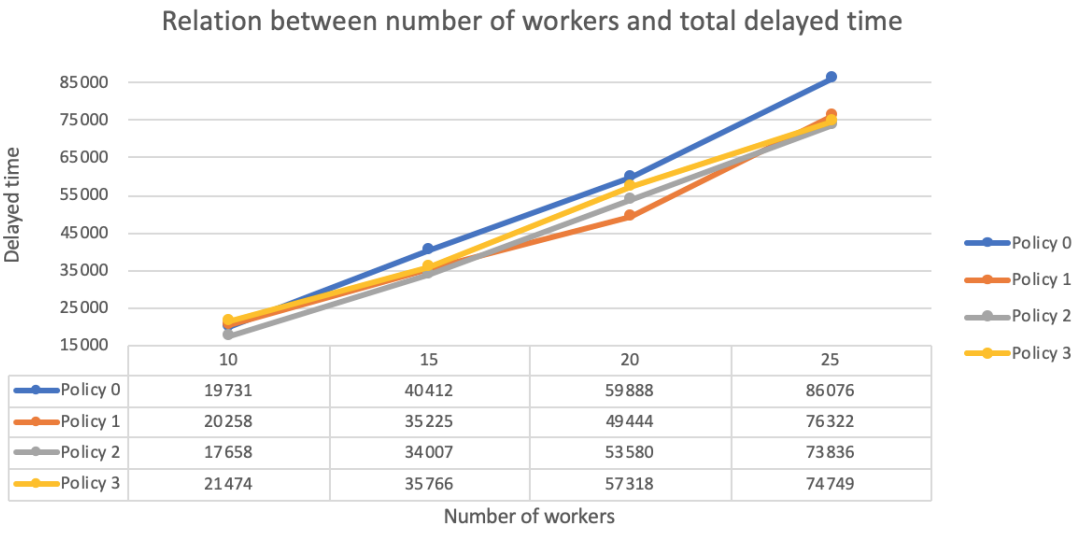
We use a sample including 185 orders, 20 locations applying to policies 1, 2 and 3. We also change pickers in range from 10 to 25 people to see the trend of picking process if we increase or decrease the number of workers. The result is sum up by tables below:



	10	15	20	25
Policy 0	13674	10797	9122	8506
Policy 1	13898	10577	8967	8430
Policy 2	13585	10303	9050	8249
Policy 3	13836	10381	9338	8395

Figure 1: Processing time of picking process with respect to number of workers

In Figure 1, we care about the processing time of picking process that means the time from beginning till the last worker finished his jobs. It is clearly to see that the process time will lessen if there are more workers since we are considering



	10	15	20	25
Policy 0	19731	40412	59888	86076
Policy 1	20258	35225	49444	76322
Policy 2	17658	34007	53580	73836
Policy 3	21474	35766	57318	74749

Figure 2: Total delayed time of all workers

one unchanged sample task. However, it is also pointing out that this value has no significant difference between different policies.
In Figure 2, we consider the total delayed time of workers, that is sum of waiting time that each picker needs to spend during their work. We can see the similar trend between all policies that if we increase number of pickers, there will be more collisions so increase the delayed time. In this simulation, picking processes according to policy 1, 2 and 3 show better performance in comparison with policy 1 when number of workers increases. We still have not seen an outstanding result getting from policy 1, 2 or 3. The problem may raise from inside the algorithm, we just consider estimated queuing time at one time point. However, in fact, when picker gets to those "crowded" locations, all collisions may already disappear. The algorithm now can not predict this reality situation.

5 Conclusions

Order picking is often considered as the most crucial warehouse activity. Any inefficiency in order picking can lead to unsatisfactory service and high operational cost for its warehouse.
The simulation system is considered in this project is a simple one. We can extend it in many directions. Planning for the next semester, I will keep continuing improve the algorithms for more realistic calculating collisions.