Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

User interface for pick path optimization for a Warehouse Management System

Bachelor thesis

Artem Hurbych

Faculty: Faculty of Electrical Engineering
Study programme: Otevřená informatika
Supervisor: RNDr. Ladislav Serédi

Prague, May 2021

Thesis Supervisor:
RNDr. Ladislav Serédi
Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague
Technická 2
160 00 Prague 6
Czech Republic

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hurbych** Jméno: **Artem** Osobní číslo: **477691**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Uživatelské rozhraní pro optimalizaci pohybu zboží v rámci systému pro správu skladu (WMS)

Název bakalářské práce anglicky:

User interface for pick path optimization for a Warehouse Management System

Pokyny pro vypracování:

Studujte aktuálně existující možnosti optimalizaci pohybů zboží ve skladových prostorech. Navrhněte a implementujte zjednodušenou "proof of the concept" aplikaci pro správu skladu. Specificky se zaměřte na problematiku fyzického uložení zboží v regálech a vytváření objednávek na uložené zboží. Navrhněte grafické rozhraní pro vytvoření mapy skladu a úložných prostor – regálů a možných tras pohybu zboží. Navrhněte a implementujte různé algoritmy pro optimalizaci pohybu zboží při kompletaci dané objednávky. V GUI implementujte vizualizaci algoritmu, eventuálně zobrazení výsledku algoritmu. Srovnejte a diskutujte různé optimalizační strategie pro různé typy objednávek. Proveďte testování GUI uživatelem a diskutujte přínos vaši aplikace oproti existujícím řešením.

Seznam doporučené literatury:

1. John J. BARTHOLDI III, Steven T. HACKMAN. WAREHOUSE & DISTRIBUTION SCIENCE, Release 0.98.1 [online]. © 2019 Georgia Institute of Technology, Atlanta [8.10.2020]. Available at: https://www.warehouse-science.com/book/editions/wh-sci-0.98.1.pdf
2. John J. Bartholdi III Visualize warehouse travel [online]. ©2019 Georgia Institute of Technology, Atlanta [8.10.2020]. Available at: https://www.warehouse-science.com/apps/pickpath/index.html
3. Ruthie Bowles. Warehouse Optimization - Algorithms For Picking Path Optimization [online]. ©2019 Logiwa [18.11.2020]. Available at: https://www.logiwa.com/blog/picking-path-optimization-algorithm

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Ladislav Serédi, kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **04.03.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **19.02.2023**

RNDr. Ladislav Serédi podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles for writing an academic thesis. Moreover, I state that this thesis has neither been submitted nor accepted for any other degree.

In Prague, May 2021

.....
Artem Hurbych

Abstract

This thesis researches the pick path optimization in a warehouse. The goal is to create an application with basic Warehouse Management System functions, and a GUI enabling to create a warehouse plan, as well as to calculate and show an optimized pick path on this plan. The research section of this thesis describes the basic warehouse processes and a selection of algorithms appropriate for pick path optimization. Open-source software with functionality similar to the proposed application is described also. The implementation section of the document deals with the architecture and functionality of the application with a special attention to the used pick path optimization algorithms, including visualization of their results in the GUI. The last part of the thesis provides description of the carried out usability testing and discusses resulted pick paths obtained by employing different algorithms.

Keywords: Warehouse Management System, The Shortest Path Problem, Dijkstra's algorithm, Floyd-Warshall algorithm, Johnson algorithm.

Abstrakt

Práce zkoumá optimalizaci trasy vychystávání zboží v skladových prostorech. Cílem je vytvoření aplikace s uživatelským rozhraním nabízející minimální funkcionalitou pro správu zboží ve skladu, s možností vytvoření plánu plochy skladu včetně rozmístění regálů, a s výpočtem a zobrazením optimální vychystávací trasy zboží. Teoretická část se zaměřuje na mapování základních skladových procesů a na popis stávajícího open-source software řešícího podobnou problematiku. Zabývá se taktéž popisem vybraných algoritmů pro optimalizaci trasy a diskusí o jejich použitelnosti pro daný účel. Praktická část popisuje proces implementace s důrazem na použité algoritmy a na vizualizaci jejich výsledků, tj. vychystávací trasy. Poslední část práce se zabývá testováním vytvořeného uživatelského rozhraní a použitých algoritmů.

Klíčová slova: Systém řízení skladu, Problém nejkratší cesty, Dijkstrův algoritmus, Floydův–Warshallův algoritmus, Johnsonův algoritmus.

Acknowledgements

I would like to thank RNDr. Ladislav Serédi for mentoring my bachelor thesis. His advice was useful and helped me a lot.

In addition, I am grateful to people who were testing the implementation part of the developed application.

List of Tables

List of Figures

Contents

Acronyms

- **WMS** - Warehouse Management System
- **GUI** - Graphical User Interface
- **SQL** - Structured Query Language
- **SKU** - Stock Keeping Unit
- **TSP** - The Travelling Salesman Problem
- **SPP** - The Shortest Path Problem
- **UX** - User Experience

Chapter 1

Introduction

In this chapter we will describe the first steps which should be done before designing the system architecture or the implementation phase. This chapter mostly will be about product picking in warehouses and different algorithms which can be used for a pick path creation.

1.1 Research

A Warehouse Management System is a software solution that provides visibility into a business' entire inventory and manages supply chain fulfillment operations from the distribution center to the store shelf.

WMS solutions additionally enable companies to maximize their labor, space utilization and equipment investments by coordinating and optimizing resource usage and material flows. Specifically, WMS systems are designed to support the needs of an entire global supply chain, including distribution, manufacturing, asset-intensive, and service businesses.[1]

The most important warehouse operations include:

1. Receiving – receiving may begin with advance notification of the arrival of goods. This allows the warehouse to schedule receipt and unloading to coordinate efficiently with other activities within the warehouse.

2. Put-away – before product can be put away, an appropriate storage location must be determined. This is very important because where you store the product determines to a large extent how quickly and at what cost you later retrieve it for a customer.

3. Order-picking – on receipt of a customer order the warehouse must perform checks such as verifying that inventory is available to ship. Then the warehouse must

produce pick lists to guide the order-picking. Finally, it must produce any necessary shipping documentation and schedule the order-picking and shipping.

4. Checking and packing – packing can be labor-intensive because each piece of a customer order must be handled; but there is little walking. And because each piece will be handled, this is a convenient time to check that the customer order is complete and accurate. Order accuracy is a key measure of service to the customer, which is, in turn, that on which most businesses compete.

5. Shipping – shipping generally handles larger units than picking, because packing has consolidated the items into fewer containers (cases, pallets). Consequently, there is still less labor here.

Order-picking typically accounts for about 55% of warehouse operating costs; and order-picking itself may be further broken like this[1]:

Activity	% Order-picking time
Traveling	55%
Searching	15%
Extracting	10%
Paperwork and other activities	20%

Table 1.1: Research: Activities time part in order-picking

As we see, traveling while order-picking is a significant part of the warehouse organization, so it is important to minimize time spent for this. So, for every warehouse it vastly increases the efficiency to have a system which shows the most effective way to pick products on a warehouse plan. There is a program named Warehouse Science: Pick Path Optimizer from authors John Bartholdy and Wendi Tang[2] showed in the figure 1.1.

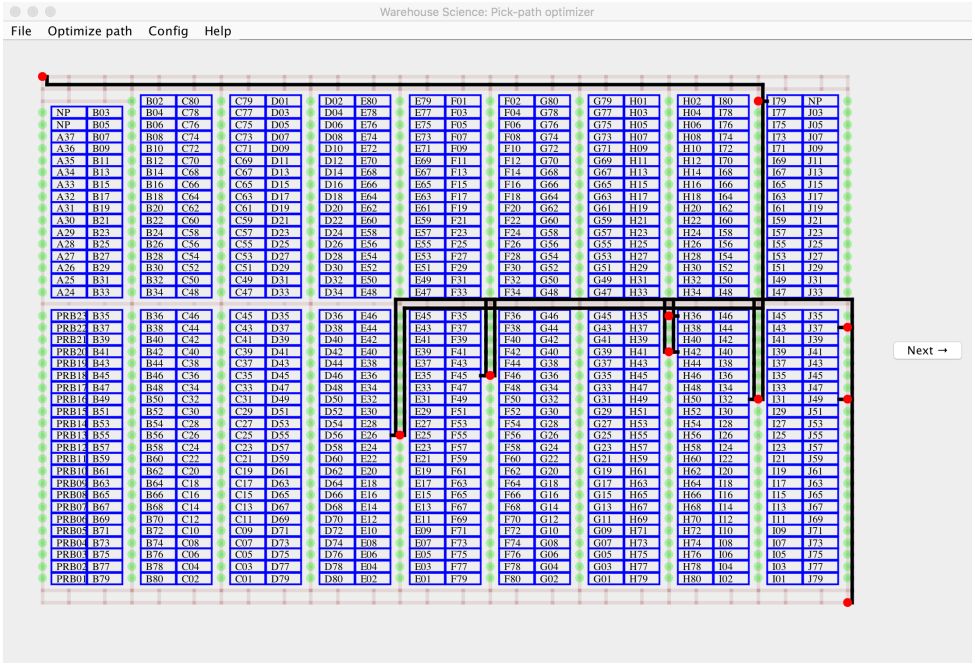

Figure 1.1: Research: "Warehouse Science: Pick Path Optimizer"

It implements our goals but has poor interface without ability to check information about products stored in some zone and has only the option to create a warehouse from .xlsx file and order list from .csv file. It may be inconvenient for users. Also, the program shows only simplified information about ways without real units and time. Our goal is implement something similar to screenshot from Warehouse Science: Pick Path but with a more interactive map and better WMS functions. The other mandatory goal is to make a built-in map creator.

To optimize travelling while order-picking in warehouses we need to try to solve problem which is in a certain way connected to another two different math problems. Those problems are The Travelling Salesman Problem and The Shortest Path Problem[3].

The Travelling Salesman Problem[4] asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?". Something similar to The Travelling Salesman Problem we need to solve to optimize the route of picker or picking-machine when it has order with different SKU, which are located in different parts of the warehouse. It saves time that might have been wasted on unnecessary or longer crossing. Typically the larger the order, the more time is saved. The difference between our problem and TSP is that we have slots not cities, we do not need to visit all slots(cities) but only certain slots and there is an option when we do not start, return or both to the start point.

The Shortest Path Problem[5] is the problem of finding a path between two nodes in a graph such that the sum of the weights of its constituent edges is minimized. In this

problem we need to find path only between two nodes, but in our situation we will need to find path between many pairs of nodes.

After some research it was decided to implement and compare those algorithms:

- Dijkstra's Algorithm[6]
- Floyd-Warshall Algorithm[7]
- Johnson's Algorithm[8]

In the following sections we will describe each algorithm. Letters E and V will be used to show algorithm complexity. E means number of edges and V means number of vertices.

1.1.1 Dijkstra's Algorithm

This algorithm finds the shortest path from one node to all other nodes. Complexity of this algorithm is $\Theta(E + V * log(V))$. It is possible to start it from starting point to first SKU location in order, then from every SKU location to the following location and then to finishing point.

However it does not guarantee that full found path is the shortest possible. To make our calculation fully accurate we need to do it for all permutations of SKU in order and get the shortest path, because sequence of SKU in order is not important for picker, but is important for finding the shortest path. So we will implement the classic Dijkstra's Algorithm and Dijkstra's Algorithm with permutation.

The three main values of the algorithms are described below:

- $dist$, an array of distances from the source node s to each node in the graph, initialized in the following way: $dist(s) = 0$; and for all other nodes v, $dist(v) = \infty$. This is done at the beginning because as the algorithm proceeds, the $dist$ from the source to each node v in the graph will be recalculated and finalized when the shortest distance to v is found.
- Q, a queue of all nodes in the graph. At the end of the algorithm's progress, Q will be empty.
- S, an empty set, to indicate which nodes the algorithm has visited. At the end of the algorithm's run, S will contain all the nodes of the graph.

The algorithm proceeds as follows:

- While Q is not empty, pop the node v, that is not already in S, from Q with the smallest $dist(v)$. In the first run, source node s will be chosen because $dist(s)$ was initialized to 0. In the next run, the next node with the smallest $dist$ value is chosen.

- Add node v to S, to indicate that v has been visited.

- Update $dist$ values of adjacent nodes of the current node v as follows: for each new adjacent node u,

  - if $dist(v) + weight(u, v) < dist(u)$, there is a new minimal distance found for u, so update $dist(u)$ to the new minimal distance value;
  - otherwise, no updates are made to $dist(u)$.[9]

1.1.2 Floyd-Warshall Algorithm

This algorithm finds optimal distance between all points. Complexity of this algorithm is $\Theta(V^3)$. Then we need to permute all sequences of SKU in order and compare path length for them to find the shortest one as in our permuted variant of Dijkstra's.

The realization of algorithm:

- We are taking a directed weighted graph $G(V, E)$ as an input. And first, we construct a graph matrix from the given graph. This matrix includes the edge weights in the graph.

- Next, we insert 0 in the diagonal positions in the matrix. The rest of the positions are filled with the respective edge weights from the input graph.

- Then, we need to find the distance between two vertices. While finding the distance, we also check if there's any intermediate vertex between two picked vertices. If there exists an intermediate vertex then we check the distance between the selected pair of vertices which goes through this intermediate vertex.

- If this distance when traversing through the intermediate vertex is less then the distance between two picked vertices without going through the intermediate vertex, we update the shortest distance value in the matrix.

- The number of iterations is equal to the cardinality of the vertex set. The algorithm returns the shortest distance from each vertex to another in the given graph.[10]

1.1.3 Johnson's Algorithm

This algorithm finds optimal distance between all points. Complexity of this algorithm is $O(V^2 * log(V) + V * E)$. Then use permutation of order items as in previous algorithm.

The three main parts of the algorithms are described below:

- The first part of the algorithm is fairly simple. Adding a new vertex, s, to the graph and connecting it to all other vertices with a zero weight edge is easy given any graph representation method.
- Reweighting is perhaps the most novel part of this algorithm. The basic concept is this: if all edges in a graph are non-negative, performing Dijkstra's algorithm on every vertex is the fastest way to solve the all-pairs shortest-path problem. If, however, some edges have negative weights, we can reweight them so that the following definition holds.
- Finally, Dijkstra's algorithm is run on all vertices to find the shortest path. This is possible because the weights have been transformed into non-negative weights. It is important, however, to transform these path weights back into the original path weights so that an accurate path weight is returned at the end of the algorithm. This is done at the end of the algorithm by simply reversing the reweighting process. Typically a data structure like a matrix is returned. At every cell i, j of this matrix is the shortest path from vertex i to vertex j.[11]

1.1.4 Algorithms conclusion

We will implement four algorithms. First of them — the Dijkstra's Algorithm without permutations just returns path, but there is no guarantee that this path is the shortest, it will be implemented to compare its results with other algorithms results.

Three other algorithms return exactly the shortest path. Those are Dijkstra's Algorithm with permutation, Floyd-Warshall Algorithm and Johnson's Algorithm. All those algorithms will be tested and their results will be compared in Chapter 4. Those algorithms were chosen because they are used to solve SPP[5] and the problem in the application is similar to SPP.

Chapter 2

Solution proposal

This chapter will describe system architecture, database model, and use case diagram to present application functionality.

2.1 Overall architecture

The proposal is to create a simplified WMS using the Java language. Java is decided to be used due to its being the second most popular programming language in the world. Unlike many other languages, Java is cross-platform and includes means to create a "complete" application - from GUI to backend.

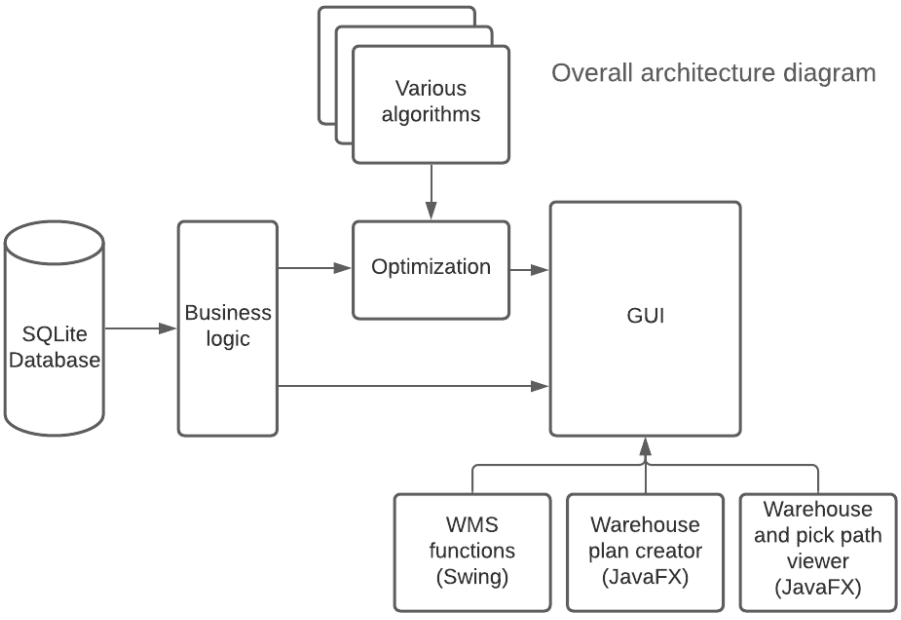

Figure 2.1: Solution proposal: Overall architecture diagram

2.1.1 GUI

In the beginning, I chose Swing library[13] to implement GUI because of some personal experience with this GUI library. In some short time, the work features of Swing have shown to be not as efficient as features of JavaFX[12]. Creating and displaying WMS plan is much easier with JavaFX. In JavaFX it is possible to change the size, position, and form of objects without recreating a full scene, unlike in Swing. So in this program, all menu parts were created using Swing. After that, when I realized that there was a necessity to draw everything related to a WMS plan using JavaFX, I needed to find some solution to integrate JavaFX with Swing.

After some research, I found a JFXPanel connector[14]. JFXPanel is a component to embed JavaFX content into Swing applications. The content to be displayed is specified with the setScene(javafx.scene.Scene) method that accepts an instance of JavaFX Scene. After the scene is assigned, it gets repainted automatically. Using the JFXPanel system creates another thread where JavaFX is running.

Code example:

```
fxPanel = new JFXPanel();
  Platform.runLater(() -> {
    Scene scene = createScene();
    fxPanel.setScene(scene);
  });
```

So after the manipulations mentioned above, Swing is used for the menu and all operations (add, delete, show all) with items. JavaFX is used for creating warehouse plans, creating orders, and showing optimized pick paths on the created warehouse plan.

2.1.2 Database layer

In addition, the application will use SQLite database[15] for saving all items stored in the warehouse and saving the warehouse's layout.
SQLite is used because it is simple and lightweight, has good performance, and does not need installation. Database classes are located in `db` package. There are six tables in the database. Their purpose is described in the table 2.1.

Table	Purpose
WMS	to keep the properties of the warehouse and start point location.
Rack	to keep location and sizes of rack. Also number of slots in this rack and information whether rack is two-lined or not are kept in this table. Two-lined rack means that slots where SKUs can be stored SKUs are on both sides of the rack.
Slot	to keep sizes and locations of slots which are located in the rack.
Node	to save locations of nodes. Nodes are always located in front of the slot and where the picker or pick path can turn.
Arc	to save arcs which connect nodes. TEach arc has an id of both nodes they connect and travel factor. Travel factor is the value which represents travelling complexity from one node to another.
SKU	to save SKU properties.

Table 2.1: Solution proposal: Database tables

Database diagram is shown in the figure 2.2.

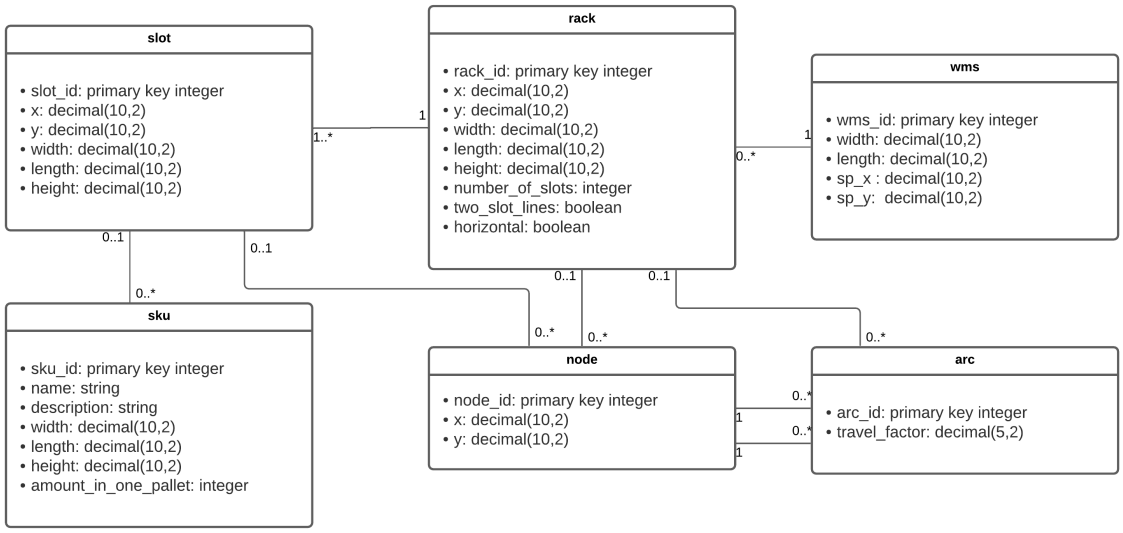

Figure 2.2: Solution proposal: Database diagram

The database class is separated into five classes to separate database functionality, each with its own type of functions: ConnectionDatabase, AddDatabase, DeleteDatabase, GetDatabase, UpdateDatabase. In the figure 2.2 functions of those classes are showed.

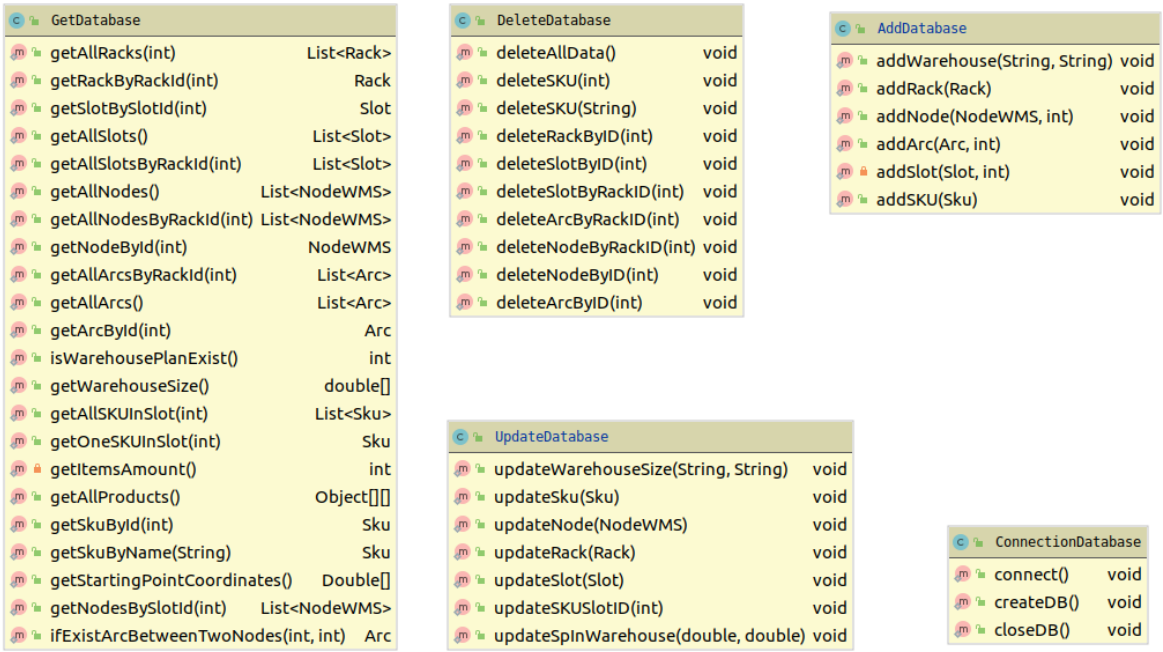

Figure 2.3: Solution proposal: Database functions

2.1.3 Use cases

In the figure 2.4 is showed use case diagram.

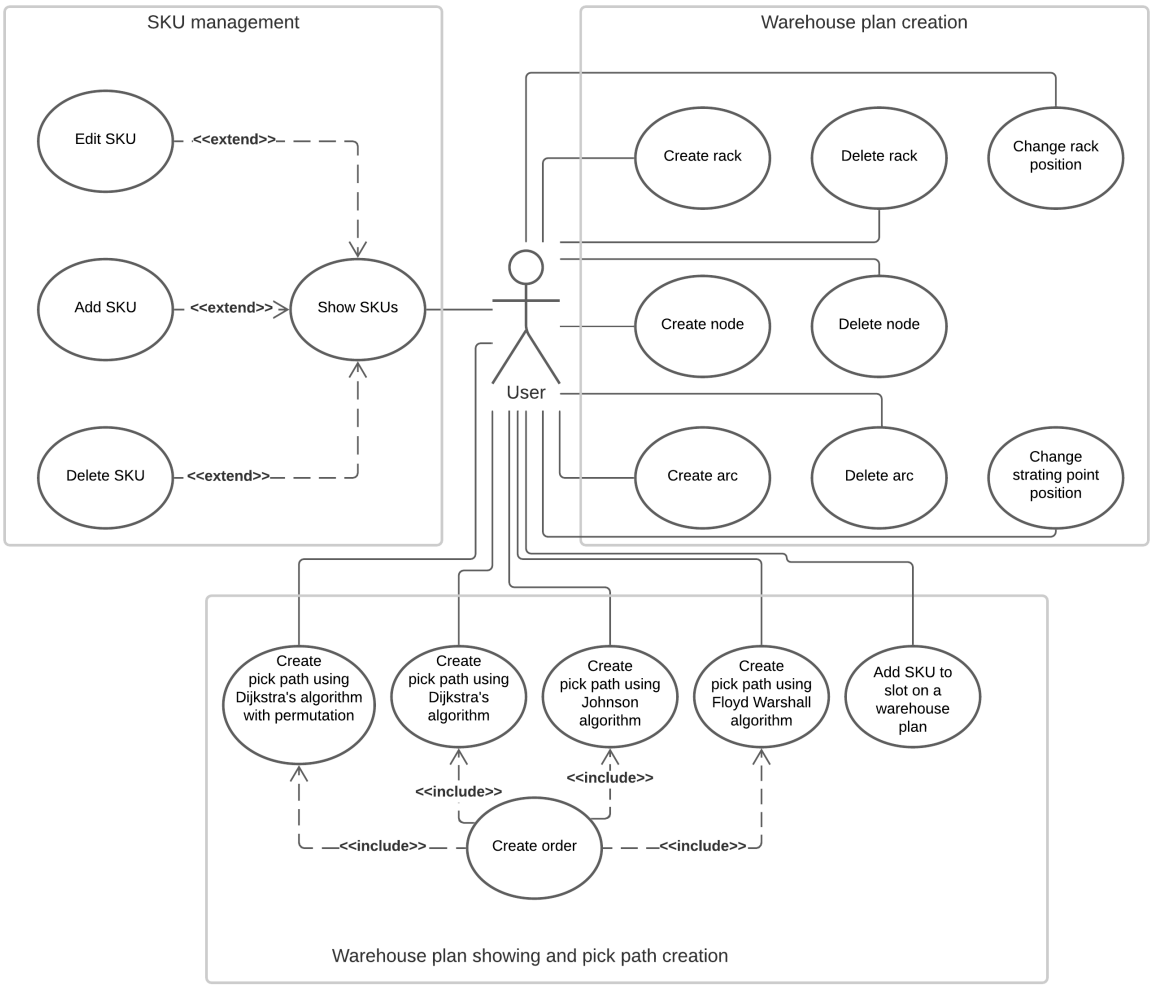

Figure 2.4: Solution proposal: Use case diagram

Chapter 3

Application

In this chapter we will describe the created application and its usage.

3.1 Application usage

The source code and a directory with bash script for application running is accessible in a GitLab repository dedicated to this project. Clickable QR code to GitLab is located in Appendix A. Before using the program, we advise to read the whole chapter, especially Program behavior 3.4.

3.2 UI Description

After starting the program main menu will be shown. In the menu there are abilities to show the product list, create warehouse plan, show warehouse plan and exit.

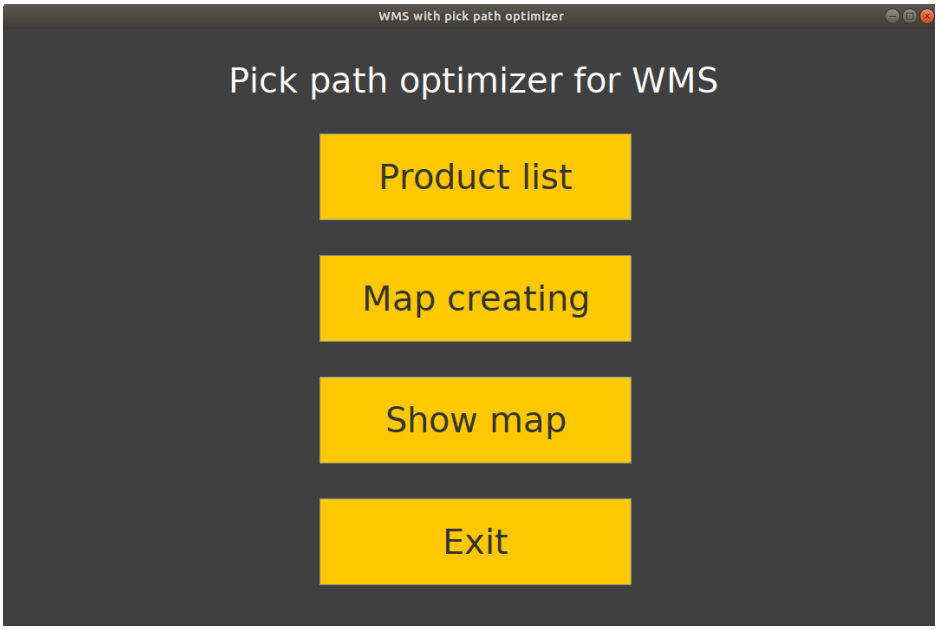

Figure 3.1: Program: Main Menu

3.2.1 Show product list

If a user chooses to show a product list, he will obtain a table with every product stored in the warehouse and fields with the add button. Adding some SKU to the warehouse is possible using those fields. To change information about some SKU, the user needs to click on the cell with information and enter a new value. To delete SKU from the database user need to push the DELETE button when this SKU is selected in the table.

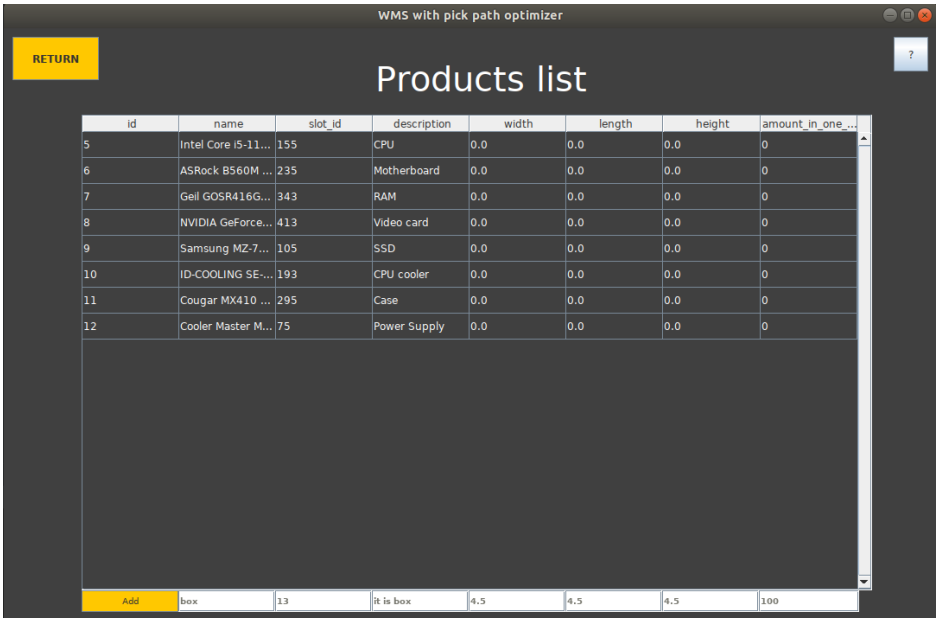

Figure 3.2: Program: Product list panel

3.2.2 Plan creating

After opening plan creating we will see an option to create a new warehouse plan with given dimensions. And to load the old warehouse plan from the database if one exists.

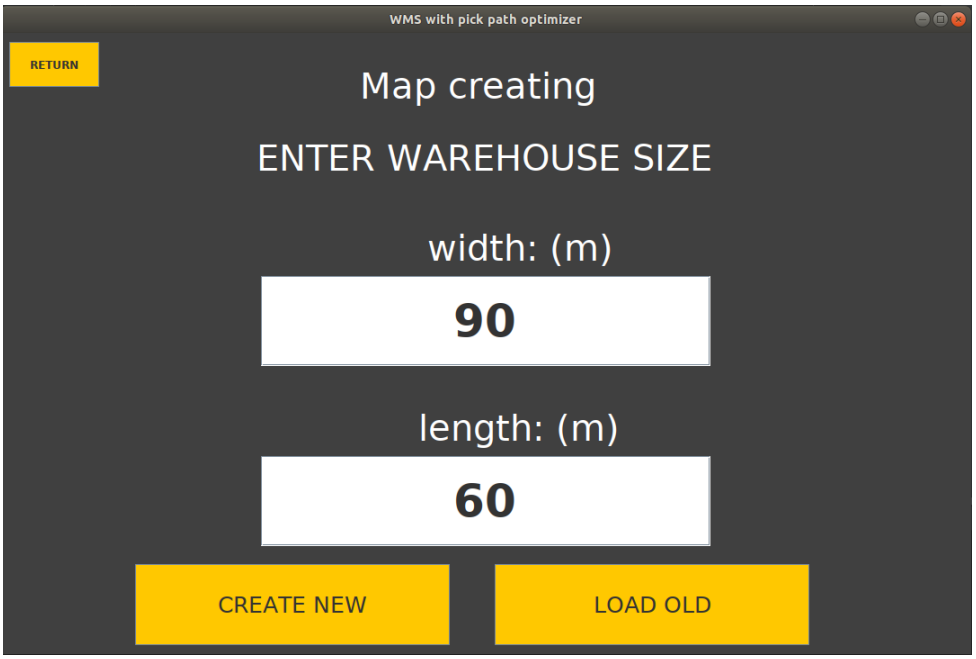

Figure 3.3: Program: Warehouse plan creating setting warehouse sizes

After clicking either of the buttons, a dialog will be shown with a short user guide (a list of keyboard shortcuts usable in the layout creation window) and - in its lower part – a form with parameters (dimensions, type, etc.) of the racks to be added to the plan.

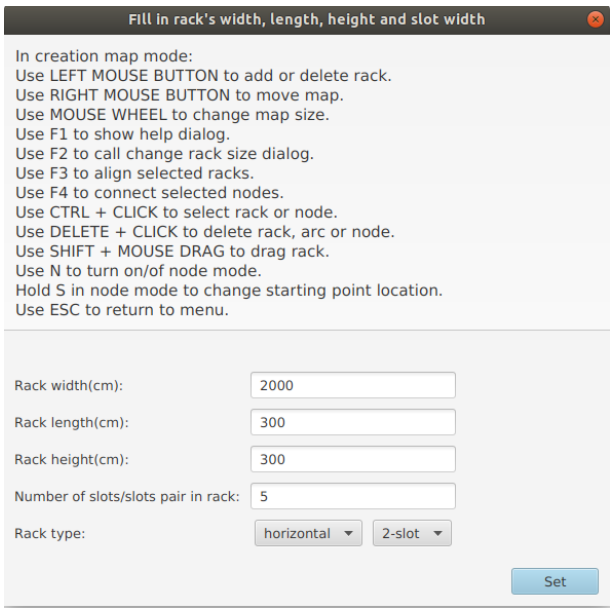

Figure 3.4: Program: Warehouse plan creating setting rack sizes dialog

After setting the rack parameters, the warehouse plan will be shown. If a user chooses to get the previous warehouse plan, it will load it from the database. Grey lines inside the racks separate slots. Using the left mouse button, it is possible to add a rack(if it is allowed in this position). User can use the right mouse button for moving warehouse plan. Also, it is possible to zoom using mouse wheel.

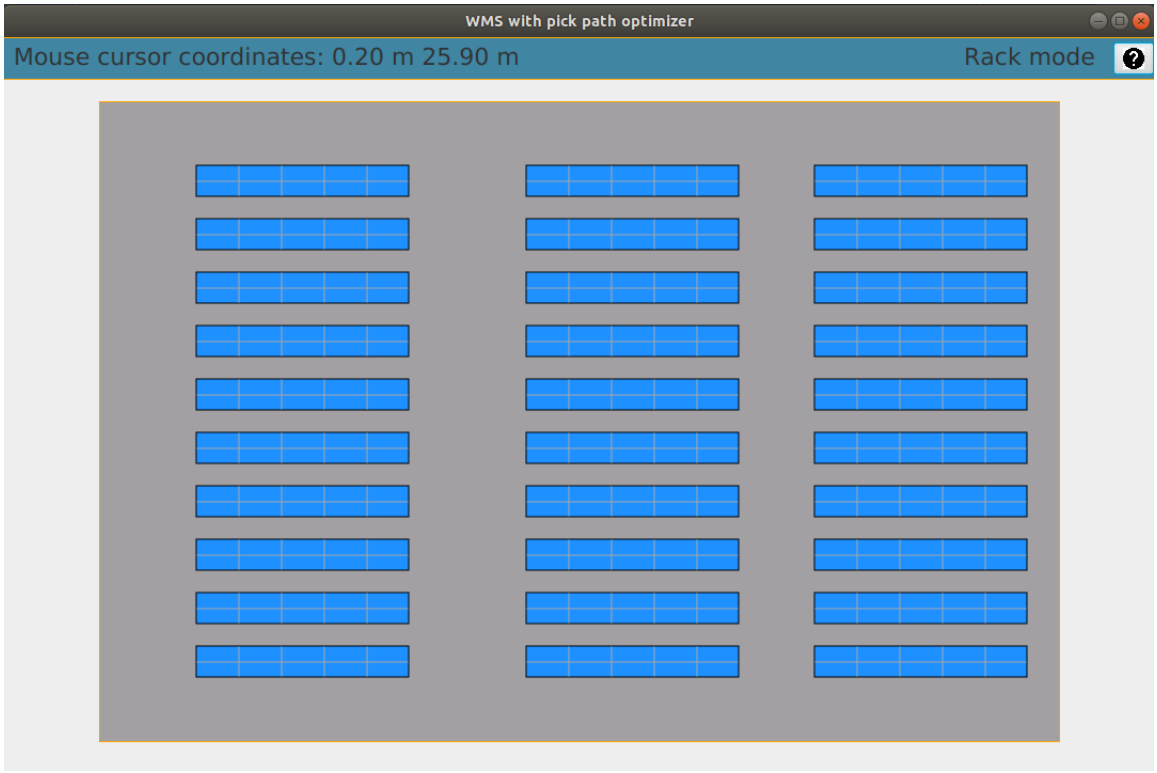

Figure 3.5: Program: Racks on warehouse plan creation panel

There is a help button on the upper blue bar. After using it or using the F1 button, the system will show a dialog with main functions (a small user guide). User guide text is similar to the guide in the dialog for setting rack parameters at the start.

The current coordinates of the mouse pointer relative to the upper left corner of the warehouse are shown in meters at the top left corner of the window. These coordinates are useful for adding the rack to its exact position. In addition, a label with mode status is next to the help button.

There are many different options to work with racks. Using the F2 button, it is possible to change rack properties if the user wants to add another type of rack. Using pressed CTRL and left-click, it is possible to select racks. Selected racks become yellow. After selecting a group of them, the user may correct their positions by pressing F3 and choosing the type of necessary correction. There are such opportunities as “align with X”, “align with Y”, “same vertical gap” and “same horizontal gap”.

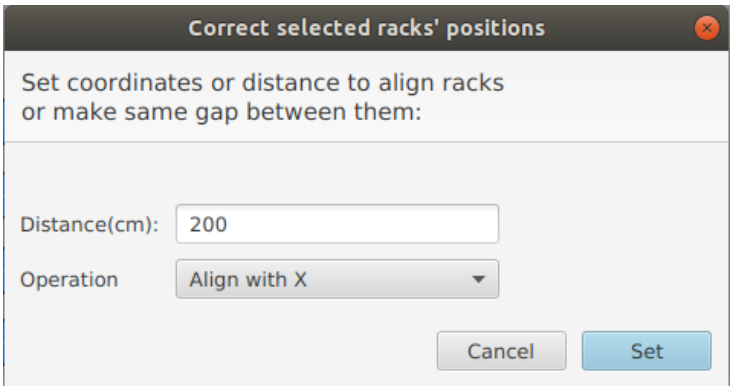

Figure 3.6: Program: Correct selected rack positions dialog

The racks are draggable by mouse. To drag rack, hold SHIFT, and using the left mouse button click, drag rack. Dragged rack is painted green. Also, when a user is dragging a rack, there are red lines that help to set the rack parallel to another rack. If some racks are parallel to the dragged rack, all of them turn green. If a rack would be dragged on another rack or outside of warehouse borders, it will be returned to the original location. Bottom right rack is dragged in the figure 3.7.

Figure 3.7: Program: Rack drag mode

By pressing the N key, it is possible to switch the program to nodes and arches mode. They are not shown with racks by default because when the user only creates racks, nodes and lines positions are not really important for him.

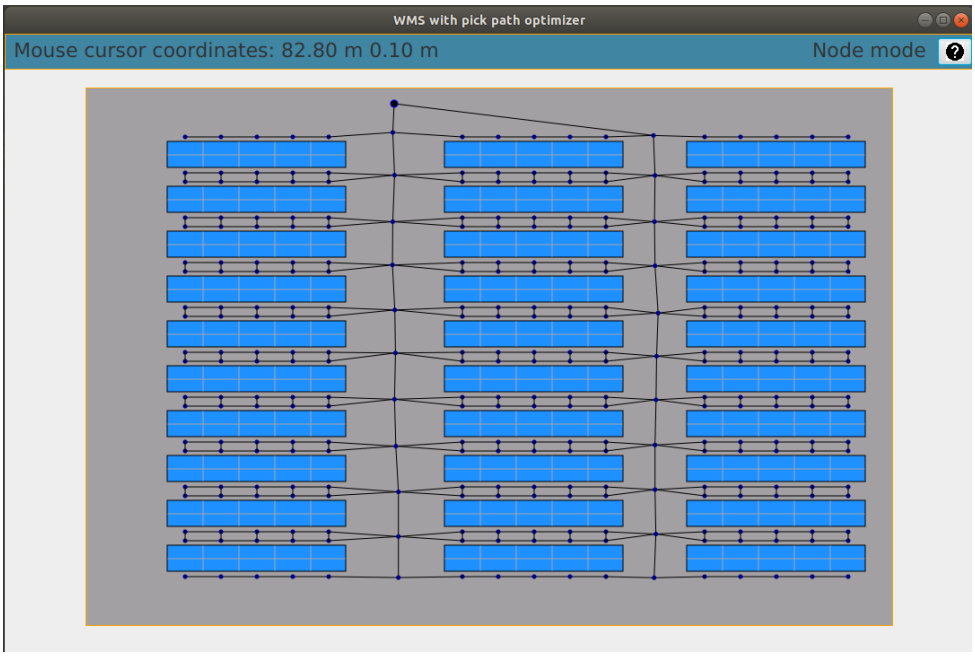

Figure 3.8: Program: Racks with nodes on warehouse plan

In this mode all operations are done only with nodes and lines, not racks. Nodes are needed to represent the position where SKU picker or picker-machine starts, turns, or has the ability to get SKU from the slot. Arches are needed to represent locations where the picker can go from one node to another.

For every slot in the rack automatically creates one node from one side. However, if the rack is created next to the warehouse wall nodes on wall side will not be created automatically. All nodes on one side of the rack are automatically connected with lines. By using the left mouse button, it is possible to add or delete nodes. If the node is deleted, all lines connected to this node will be deleted too.

Using a pressed CTRL and left mouse button, it is possible to select nodes. After selecting some nodes, it is possible to connect them with lines using the F4 button. It is possible to connect not only two nodes but any number of them. Nodes will be connected in the order they were selected.

If a mouse pointer hovers on a node for some short period of time the node location will be shown.

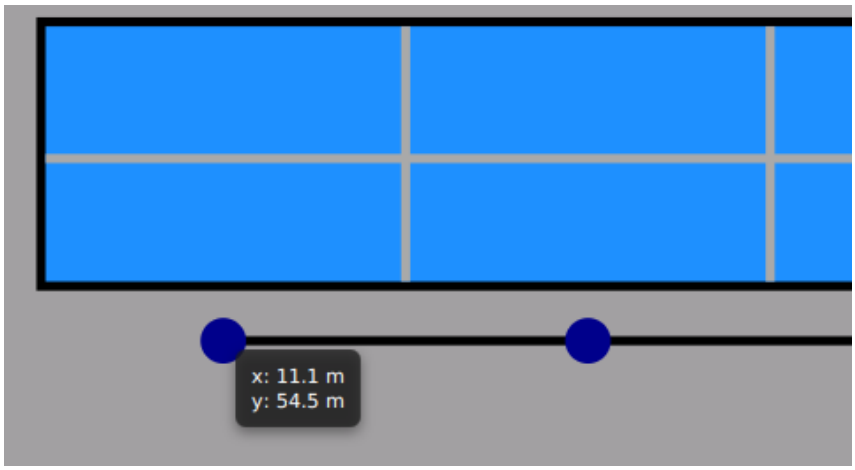

Figure 3.9: Program: Node coordinates pop up

If a mouse pointer hovers on an arc for some short period of time the travel factor of arc will be shown.

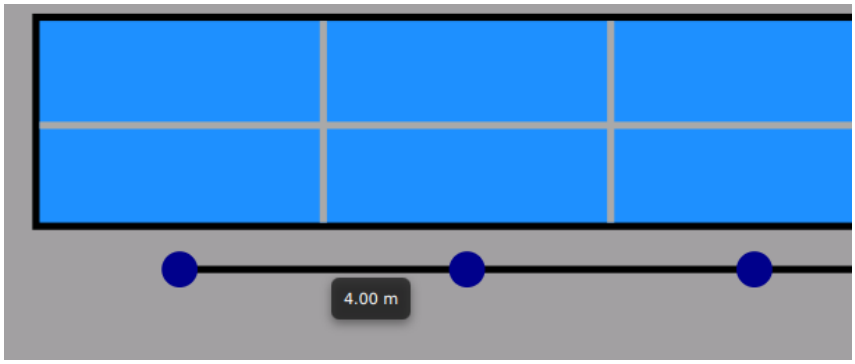

Figure 3.10: Program: Arc length pop up

One node looks different than the others and is created automatically in the (0,0) position. It is the starting point node. On this node order-picking can be started/finished or both. It is the location where a picker starts his path or location where a pick-machine stays and where it returns after picking. To move the starting point, the user needs to hold key S in node mode and just click on its new position on the plan using the left mouse button.

It is important not to forget to connect starting point to other nodes because if it is not connected, the user will not be able to use it during the order creation. In the figure 3.11 you can see Starting point node(black with a blue stroke) connected to a simple node(blue).

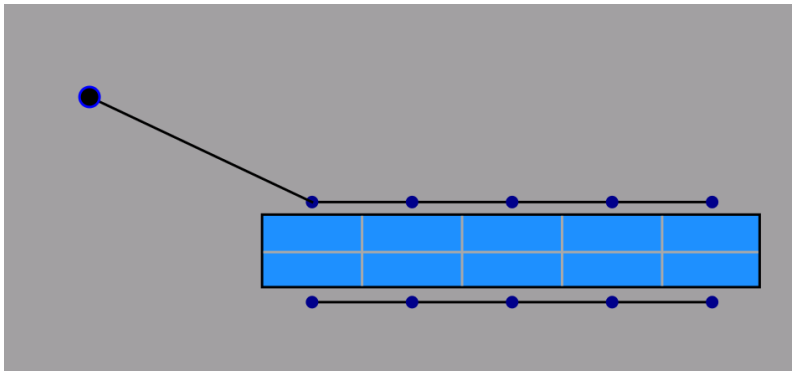

Figure 3.11: Program: Starting point node connected to simple node

To delete some element on the warehouse plan, the user needs to hold the DELETE button and use the left mouse button. If the user a node with connected arcs, those will be deleted too. Also, there is a possibility to delete selected racks or nodes by pressing the DELETE button after selecting.

If user tries to delete some rack with SKUs in the program will show a confirmation dialog.

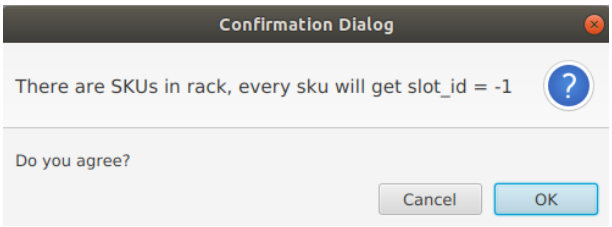

Figure 3.12: Program: Rack delete confirmation dialog

Warehouse plan creation is a time-consuming process, but as the user does it only once, this does not cause any significant issue.

3.2.3 Show plan

After opening the "Show plan" panel, the user will see the warehouse plan which was created in the Create plan panel. However, slots will be painted according to their content. Green means that the slot is empty. Yellow means that the slot contains one or more SKUs. In this panel it is possible to move the warehouse plan using right mouse click and to zoom as well.

Figure 3.13: Program: Show warehouse plan panel

When the cursor is on the slot, small information pop-up about this slot will be shown in the popup message. Information contains slot number(the first number is rack id and second is slot id) and a message that slot is empty if it is empty or information about SKU stored in the slot. Information about SKU contains name, description, how many SKUs fit in one pallet and sizes.

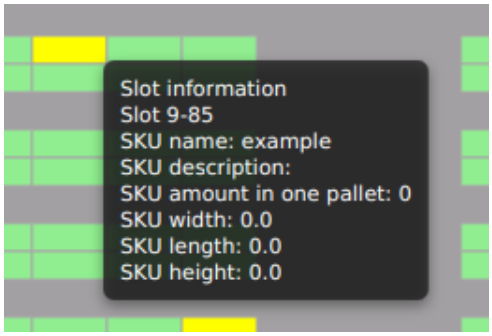

Figure 3.14: Program: Slot information pop up

Using the left mouse click on the slot, the user gets a dialog to add SKU to the slot. The slot where SKU is added will be dark green during adding. Only name field filling is mandatory. Only one type of SKU can be stored in one slot. If there is already SKU in this slot, the user will get a confirmation dialog. If the user confirms adding, the previous SKU will be deleted.

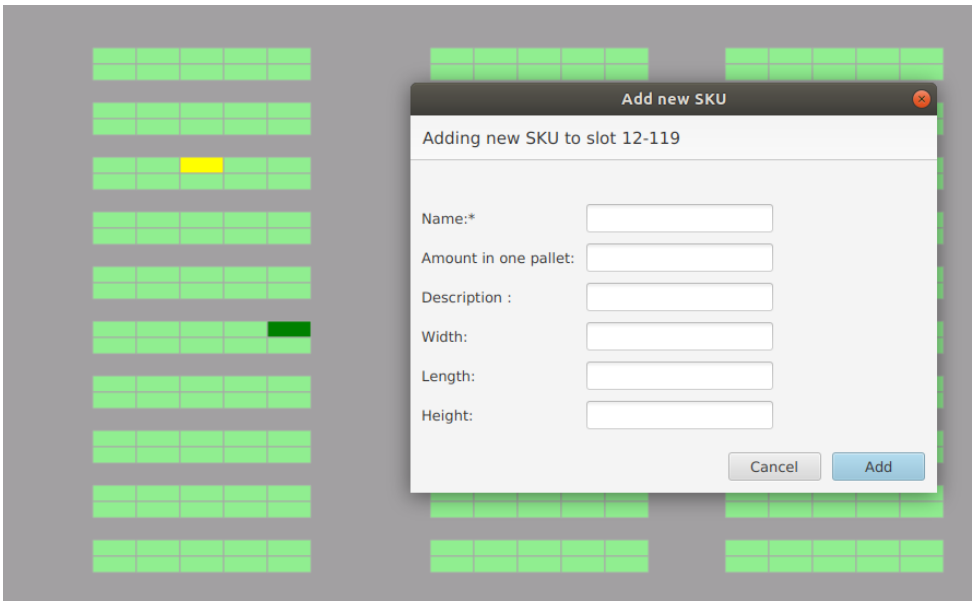

Figure 3.15: Program: Add SKU to slot dialog

On the upper blue bar on the right side is a help button similar as in a warehouse plan creation panel. After using it, the user guide dialog will be shown.

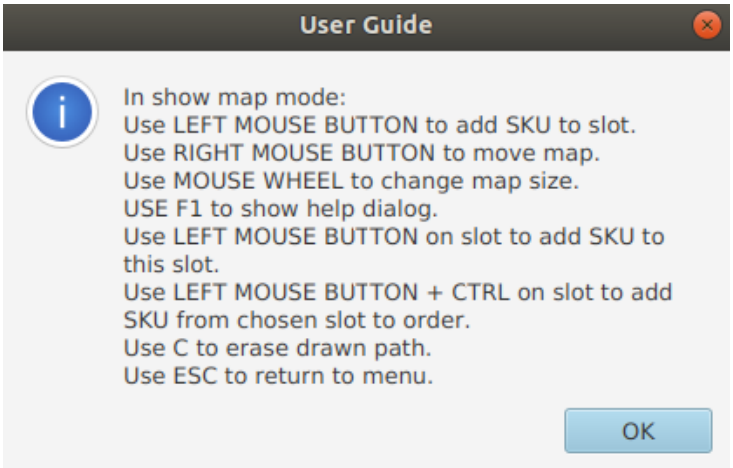

Figure 3.16: Program: Show plan panel user guide

In the center of the upper blue bar is located a button “Show SKUs in order”. It provides creating an order. After using it, the system will show a dialog where it is possible to add SKUs to order and create this order. It is possible to add SKUs with their ids or names. Ids can be filled in in different formats using a combination of numerals, - (dash) and , (comma) (for example, 1, 4-7 adds first and from forth to the seventh SKUs). Also, SKUs can be added to order using pressed CTRL and left mouse button click on the slot where it is stored. To delete SKU from order user can use the ”Delete” button in the dialog or the DELETE key when this SKU is selected. If the user needs to check the warehouse plan again, he can close the dialog. SKUs in the order list won’t be lost and will be shown again after pressing the “Show SKUs in order” button. The slots with SKU, which is in the order, will be painted beige. If the user selected some SKU in order creating dialog slot with it will be painted purple color. After adding SKUs to the order, the user can choose if he wants to start and/or to finish picking from starting point using checkboxes. Also, the user can choose an algorithm that will create a path.

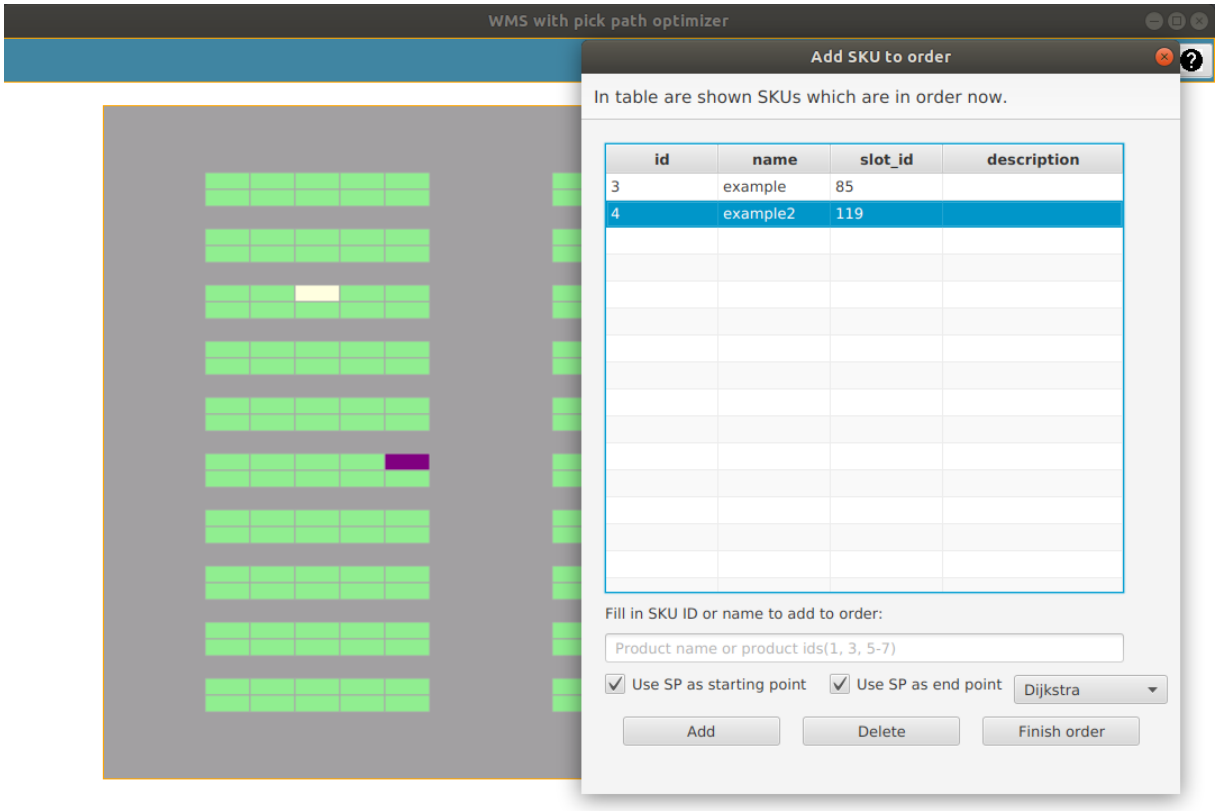

Figure 3.17: Program: Create order dialog

After clicking the ”Finish order” button the picking path will be calculated and drawn on the warehouse plan. Each part of the path is painted a different color. Using field and arrows in the top-right screen corner, it is possible to show different parts of a path. In this text input field, it is possible to select the segments of the pick path to display using a combination of numerals - (dash) and , (comma) in the same manner as in the

"Add SKU to Order" field described above. To show the whole path again, the user needs to write "all" or "All" to this field and press ENTER. Also, there is a label in the top left corner showing the length of the pick path and another label displaying the duration of the path calculation performed for the currently selected algorithm. Each calculation produces a log to the log file, which contains time, algorithm name, path length, and calculation duration. The log file is located in project directory and it is named `AlgorithmDuration.log`. It is possible to erase the path from the plan using the C button. Also, the path will be erased after new path construction.

Figure 3.18: Program: Full path example

Figure 3.19: Program: One path part example

Also, there are a lot of dialogs with warnings and confirmations to prevent a user from an unwanted action.

3.3 Algorithm implementation

Both versions of Dijkstra's algorithm were implemented by the author of this thesis. Implementation of Floyd-Warshall algorithm and Johnson's algorithm was taken from `psjava` library. It is possible to read more about library's algorithms usage on `psjava` official website[16][17]. Algorithms with permutation do not consider order of SKUs in the order, which gives an opportunity to find a shorter path. However, if the user wants to consider order, he can use Dijkstra's algorithm without permutation.

3.4 Program behavior

Additionally, I would like to say that the program described here was implemented and tested on Linux and Windows operating systems. So it must work correctly on those operating systems. However, it may have some issues on macOS due to different UI listeners behavior. Thus it is highly recommended to use this program on Windows and Linux but not macOS.

Chapter 4

Algorithms usage and comparison

In this chapter, some warehouse plans will be constructed in program described in Chapter 3. Then some products will be loaded to warehouse slots and pick paths will be created using different algorithms.

check algorithms, several SKUs (PC components) will be loaded to slots. All databases with warehouse plans will be available in project directory with names `"firstPlan"`, `"secondPlan"`, `"thirdPlan"`, so user may test algorithms without spending time on warehouse plan creation. To use one of these databases, write its name as the program argument. If argument name is not equal to mentioned above databases names, application will start with default database. We will also compare algorithms and their duration in this chapter. Those values cannot be used as an absolute value. They vary on different computers and depend on factors like computer hardware, the current load of the operating system, etc. Moreover, sometimes it can vary on the same computer. So they are used only to compare algorithms to each other, never to consider its absolute value duration.

4.1 First warehouse plan

In this section, a prototype of a small warehouse will be created to check if the algorithms are actually working. Here will be created the scenario 2 from website[18], which - according to the authors - may pose a challenge to certain path-finding algorithms. It is a good possibility to test our program and its algorithms. Firstly warehouse plan from the website was recreated in our program.

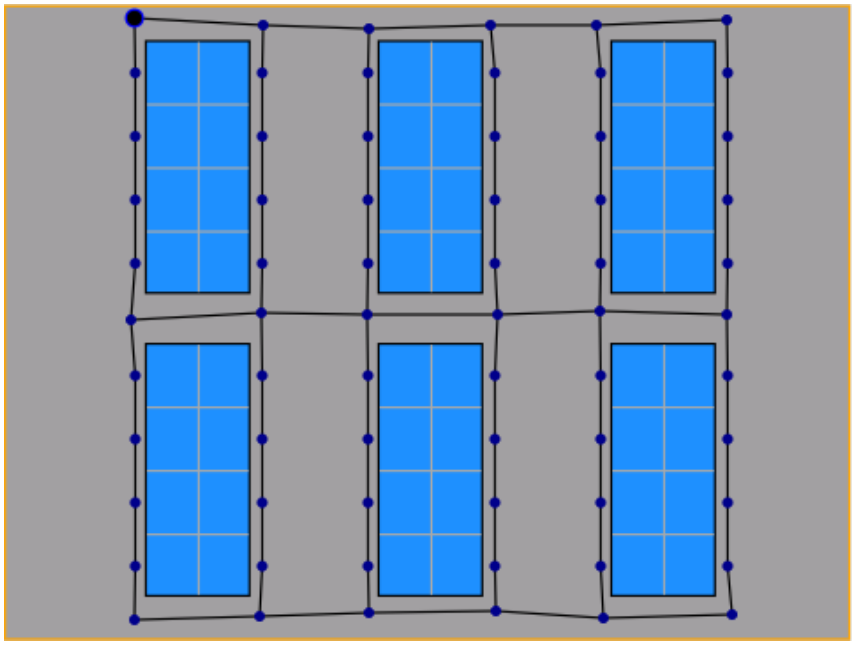

Figure 4.1: Tests: First warehouse plan

After this, some SKUs was added to slots and order was created.

Figure 4.2: Tests: First warehouse plan order

There are the most important numbers and values for the test:

- **Warehouse length:** 40 m
- **Warehouse width:** 30 m
- **Number of SKUs in order:** 4
- **Use SP as start:** true

- **Use SP as end:** true
- **Nodes amount:** 75
- **Arcs amount:** 75
- **Rack length:** 1200 cm
- **Rack width:** 500 cm
- **Number of slot pairs in rack:** 4

After finishing the order using algorithms with permutation, the program creates the path showed in figure 4.3.

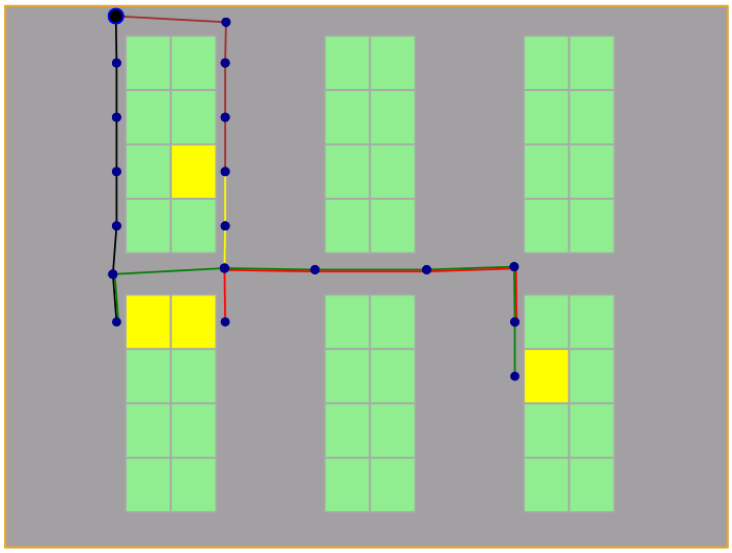

Figure 4.3: Tests: First warehouse plan test result

As we see, the path in the figure 4.3 is similar to the path on the website [18], so algorithms in the program work correctly in this case. Dijkstra's algorithm without permutation shows another path for SKU order 1, 2, 3, 4 which is longer as it was expected, because it considers SKU's order in the order. Founded path lengths and calculation duration are in the table 4.1.

Algorithm	Path length	Duration
Dijkstra's Algorithm	105.78 m	8 ms
Dijkstra's Algorithm(Perm)	95.44 m	8 ms
Floyd-Warshall Algorithm	95.44 m	130 ms
Johnson's Algorithm	95.44 m	92 ms

Table 4.1: First warehouse plan: Algorithms results

4.2 Second warehouse plan

In this section, a warehouse with a simple common rack layout will be created, which was shown in the program description in the figure 3.8.

There are some important numbers for our test.

- **Warehouse length:** 90 m
- **Warehouse width:** 60 m
- **Nodes amount:** 322
- **Arcs amount:** 477
- **Rack length:** 2000 cm
- **Rack width:** 300 cm
- **Number of slot pairs in rack:** 5

4.2.1 First test

To demonstrate how the algorithms work on this plan, the following simple order was created:

- **Number of SKUs in order:** 3
- **Use SP as start:** true
- **Use SP as end:** false

Algorithm	Path length	Duration
Dijkstra's Algorithm	188.33 m	15 ms
Dijkstra's Algorithm(Perm)	135.44 m	18 ms
Floyd-Warshall Algorithm	135.44 m	5500 ms
Johnson's Algorithm	135.44 m	832 ms

Table 4.2: Second plan, first test: Algorithms results

Here is the shortest path constructed by algorithms with permutation.

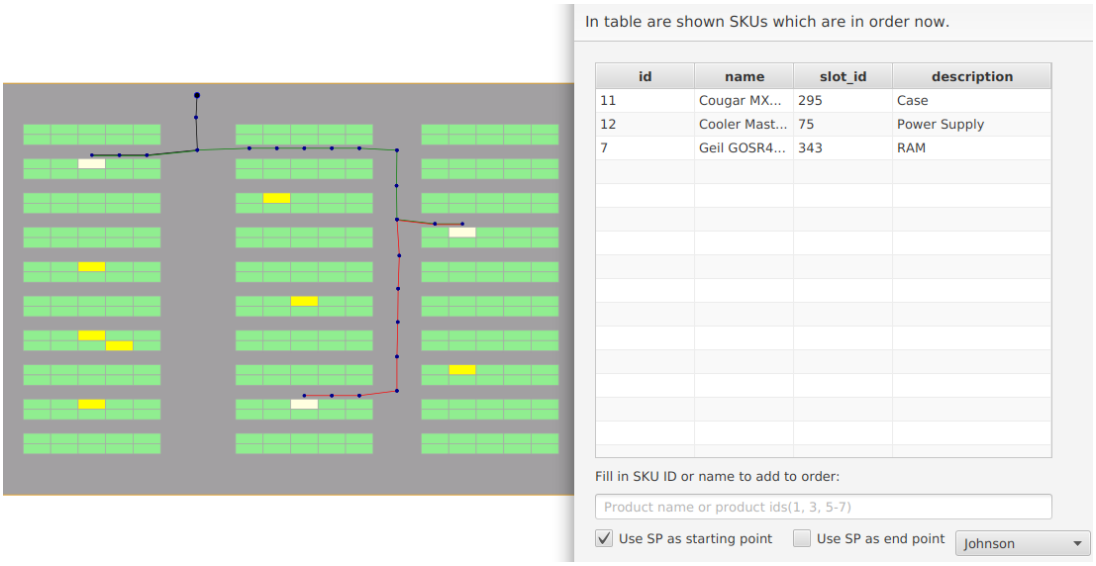

Figure 4.4: Second plan, first test: the shortest path with Johnson's algorithm

And here is the shortest path constructed by Dijkstra's algorithm. The difference in paths is caused by SKUs order in the order. Dijkstra's algorithm respects it, so firstly path goes to the bottom slot, then to slot on the left side and then to slot on the right side. It makes the path longer. To demonstrate SKUs order SKUs ids were added to slots in the figure 4.5.

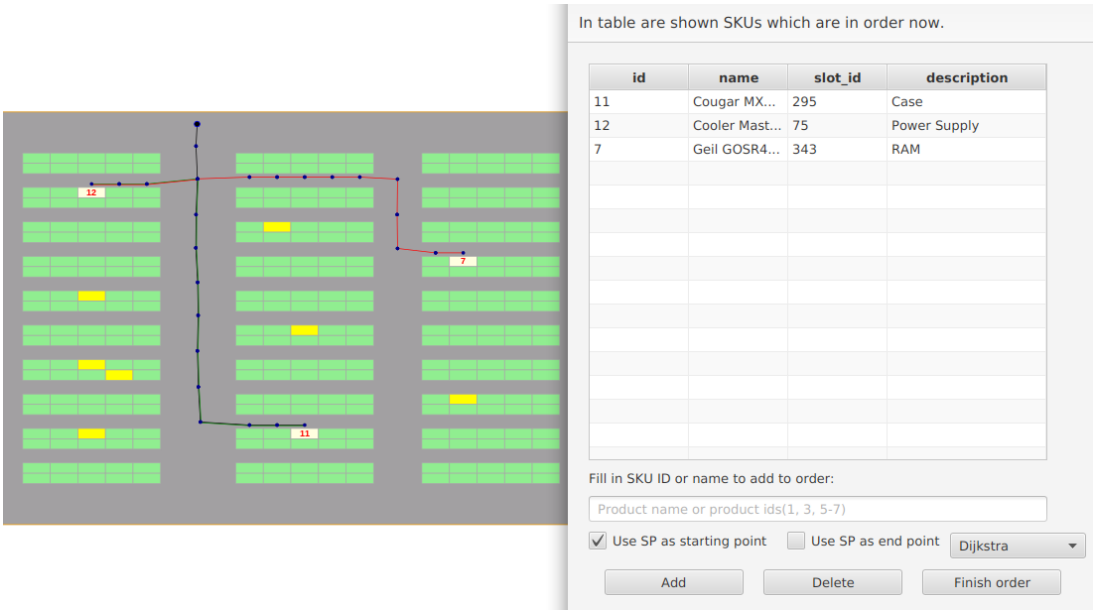

Figure 4.5: Second plan, first test: the shortest path with Dijkstra's algorithm

4.2.2 Second test

In the second test, SKUs amount will be more realistic. And the order will be similar to real-world PC order.

- **Number of SKUs in order:** 8
- **Use SP as start:** true
- **Use SP as end:** true

Here is the shortest path for this test.

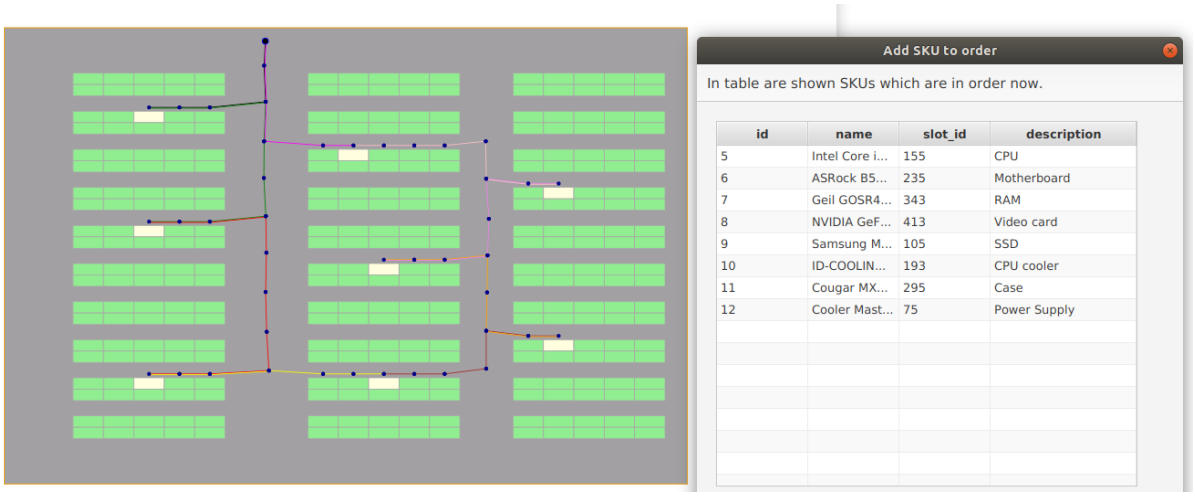

Figure 4.6: Second plan, second test: the shortest path

Algorithm	Path length	Duration
Dijkstra's Algorithm	430.47 m	83 ms
Dijkstra's Algorithm(Perm)	303.08 m	330 ms
Floyd-Warshall Algorithm	303.08 m	8755 ms
Johnson's Algorithm	303.08 m	2312 ms

Table 4.3: Second plan, second test: Algorithms results

4.2.3 Third test

In the third test, more SKUs will be added to the order.

- **Number of SKUs in order:** 10
- **Use SP as start:** true
- **Use SP as end:** true

Here is the shortest path for this test.

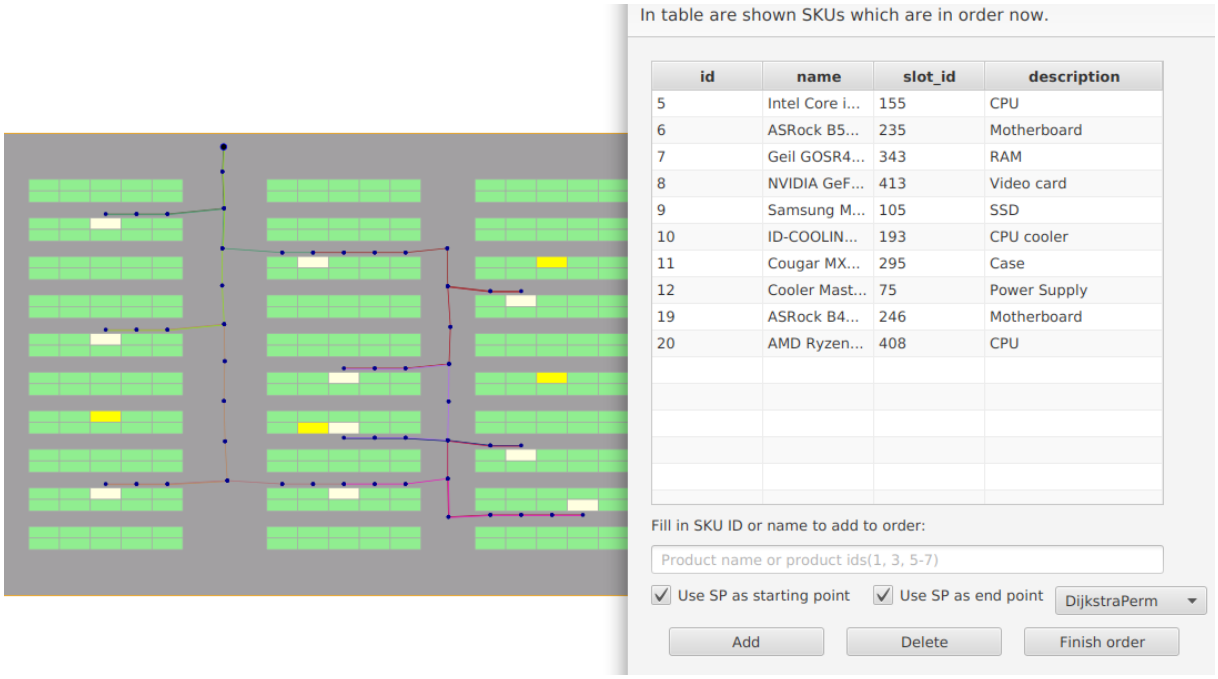

Figure 4.7: Second plan, third test: the shortest path

Algorithm	Path length	Duration
Dijkstra's Algorithm	600.51 m	51 ms
Dijkstra's Algorithm(Perm)	374.70 m	11265 ms
Floyd-Warshall Algorithm	374.70 m	77860 ms
Johnson's Algorithm	374.70 m	69395 ms

Table 4.4: Second plan, third test: Algorithms results

4.2.4 Fourth test

Unfortunately, algorithms with permutation cannot create a pick path for bigger SKUs amount in a reasonable time. It is caused by considering possible permutations, which has factorial time complexity. We always need permutation to try all SKUs order in the order to create actually the shortest pick path. So those algorithms are useful to small and middle-size warehouses, where orders have under 10 SKUs. However, Dijkstra's algorithm without permutation can calculate the path which will be the shortest with SKUs order given in the order. So it will be tried to construct a path for 20 SKUs with this algorithm.

- **Number of SKUs in order:** 20
- **Use SP as start:** true
- **Use SP as end:** true

Figure 4.8: Second plan, fourth test: the shortest path

Algorithm	Path length	Duration
Dijkstra's Algorithm	1129.02 m	178 ms

Table 4.5: Second plan, fourth test: Algorithms results

As we see in the table 4.5 algorithm duration is still short, so it is safe to assume that it can work with a larger amount of SKUs as well.

4.3 Third warehouse plan

In this section, a relatively complex warehouse layout form [19] will be re-created in the application. After filling several slots with SKUs, picking paths will be created using different algorithms. It is difficult to decide where is a starting point on this layout, so tests will not consider the start/end point for the pic path. The plan created in our application is shown in figure 4.9.

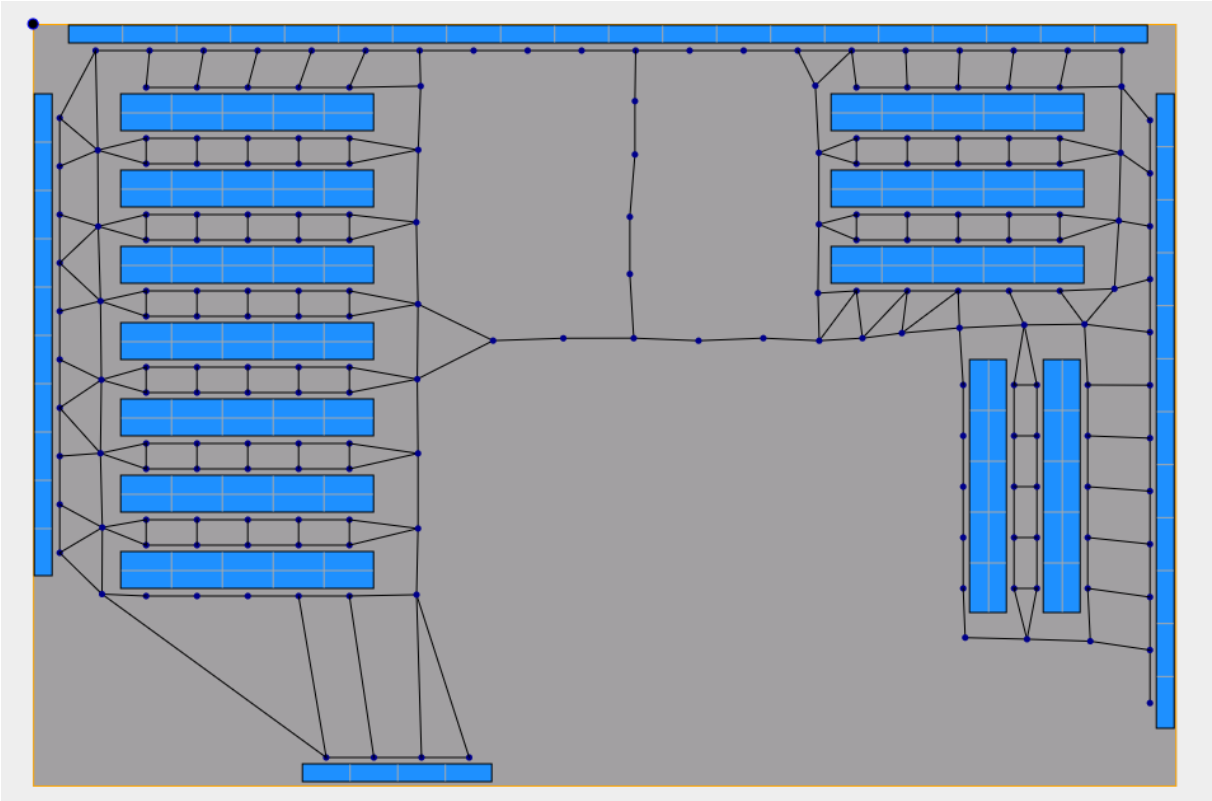

Figure 4.9: Third plan: warehouse plan

- **Warehouse length:** 90 m
- **Warehouse width:** 60 m
- **Nodes amount:** 207
- **Arcs amount:** 322
- **Rack length:** various
- **Rack width:** 300/150 cm
- **Number of slot pairs in rack:** 5

4.3.1 First test

- **Number of SKUs in order:** 5
- **Use SP as start:** false
- **Use SP as end:** false

Here is the shortest path for this test.

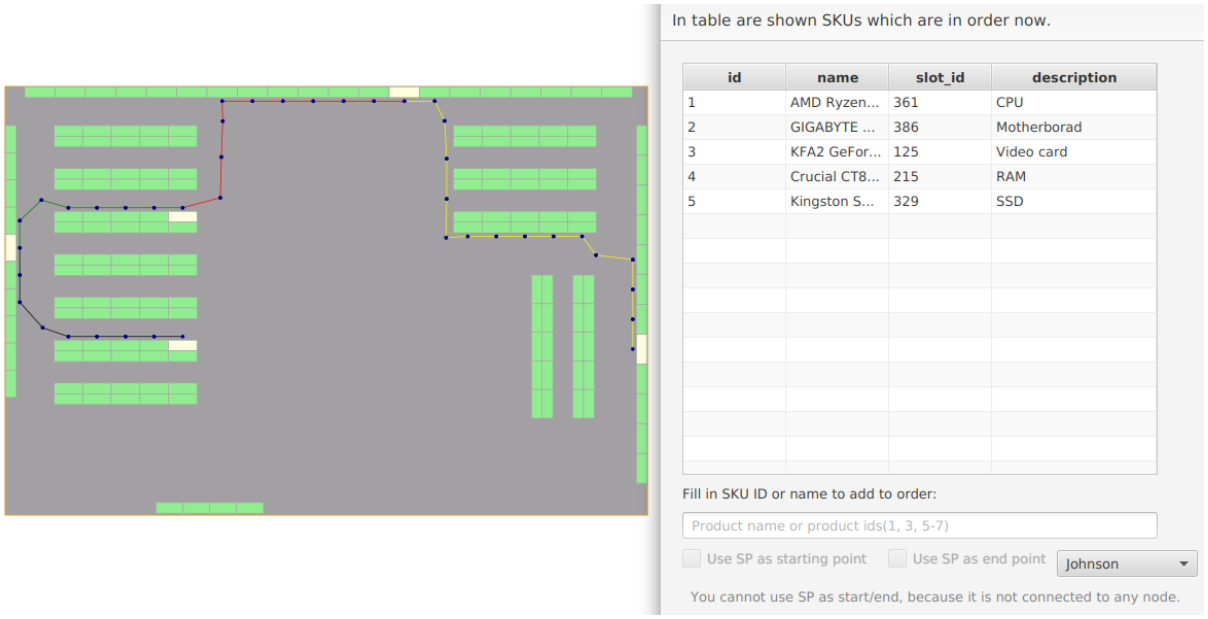

Figure 4.10: Third plan, first test: the shortest path

Algorithm	Path length	Duration
Dijkstra's Algorithm	271.57 m	33 ms
Dijkstra's Algorithm(Perm)	168.20 m	43 ms
Floyd-Warshall Algorithm	168.20 m	1952 ms
Johnson's Algorithm	168.20 m	468 ms

Table 4.6: Third plan, first test: Algorithms results

4.3.2 Second test

- **Number of SKUs in order:** 9
- **Use SP as start:** false
- **Use SP as end:** false

Here is the shortest path for this test.

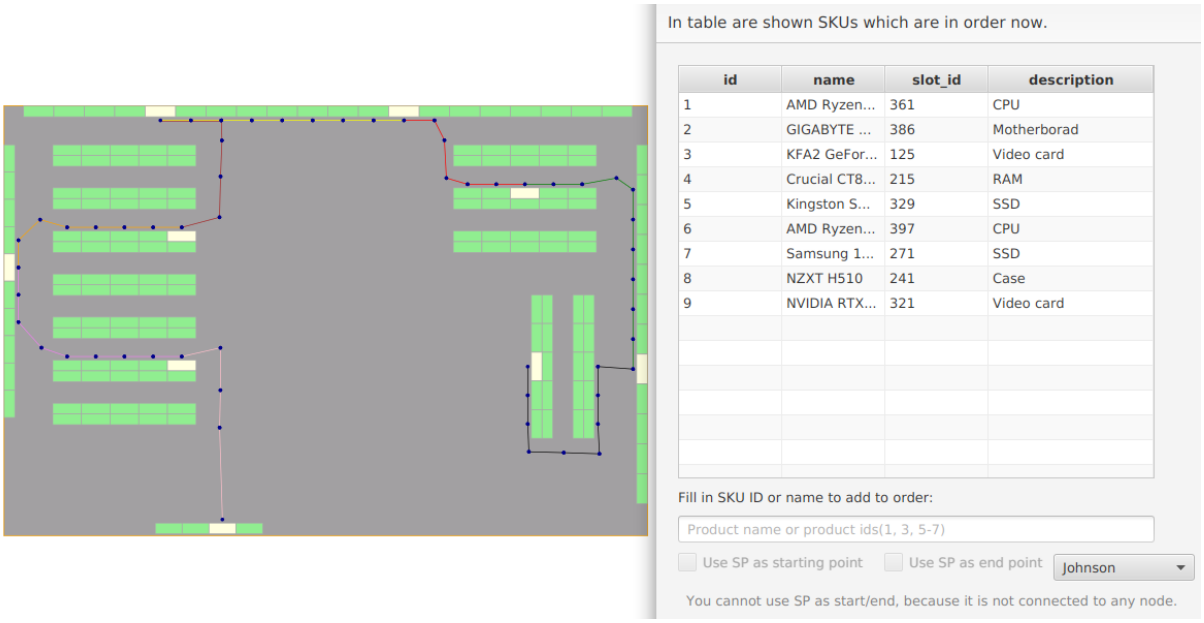

Figure 4.11: Third plan, second test: the shortest path

Algorithm	Path length	Duration
Dijkstra's Algorithm	542.24 m	34 ms
Dijkstra's Algorithm(Perm)	254.19 m	1376 ms
Floyd-Warshall Algorithm	254.19 m	8517 ms
Johnson's Algorithm	254.19 m	6263 ms

Table 4.7: Third plan, second test: Algorithms results

4.3.3 Third test

In this test, will be tried to load to warehouse big SKUs amount(30). As we know from previous tests, only Dijkstra's algorithm without permutation is useful in this case.

- **Number of SKUs in order:** 30
- **Use SP as start:** false
- **Use SP as end:** false

Here is the shortest path for this test. It is not easy to understand the whole path when it needs to visit so many slots, but the user can choose certain path parts using the path part field on this panel and it will be much easier to understand the path.

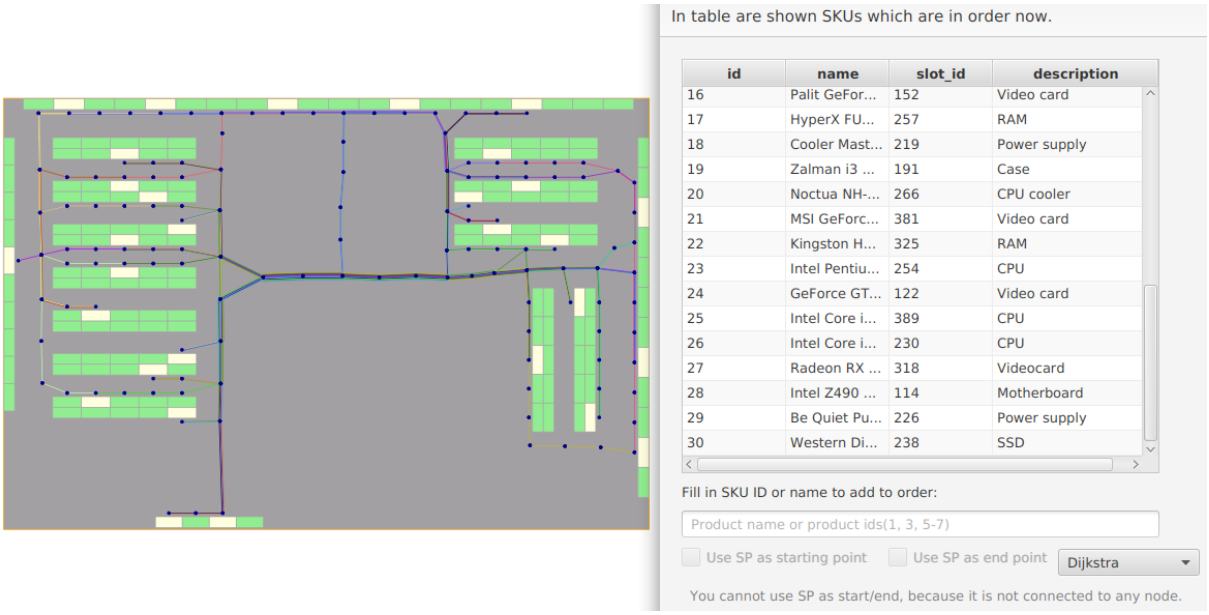

Figure 4.12: Third plan, third test: the shortest path

Algorithm	Path length	Duration
Dijkstra's Algorithm	1783.38 m	165 ms

Table 4.8: Third plan, third test: Algorithms results

As we see Dijkstra's algorithm can quickly process such SKUs amount as 30.

4.4 Algorithms results explanation

From our tests, it is seen that the fastest algorithm is Dijkstra's algorithm without permutations. It is expected because, as was said, permutation is a time-consuming operation.

Johnson's algorithm is always faster than the Floyd-Warshall algorithm. It could be explained by algorithms' complexity. Both algorithms calculate the shortest path between all node pairs and then these paths are used in the shortest pick path calculation. Johnson's algorithm complexity is $O(V^2 * log(V) + V * E)$ and Floyd-Warshall algorithm complexity is $\Theta(V^3)$, where is V - nodes amount and E - arches amount. So if the arches amount is not much higher than the nodes amount, Johnson's algorithm will be faster.

Dijkstra's algorithm complexity is $\Theta(V^2)$, so it can be expected that Dijkstra's algorithm with permutation is the fastest from algorithms that can change picking SKUs order to create a shorter path. However, Dijkstra's algorithm is used to find the shortest path between one nodes pair, so it is called every time when we need to find a path between important nodes(from those nodes is possible to pick SKUs from order).

Therefore with increasing SKUs amount in order, results of the Dijkstra's algorithm with permutation will be closer to Johnson's and Floyd-Warhsall algorithms. But all these algorithms are already time-limited by SKUs amount due to the complexity of permutation. So for small SKUs amount, Dijkstra's algorithm with permutation is the fastest one.

Chapter 5

User testing

During the implementation of the application, a considerable amount of usability testing was performed by the supervisor of this thesis as well as by the author's colleagues. Here you can see the testing program report, which was created in the middle of project implementation.

5.1 Report

5.1.1 Daria Dunina (CTU, FEE, OI 3rd year student)

Goal:

- Create WMS plan with racks, nodes and arcs.
- Add some different products to WMS.
- Delete one of them using appropriate panels.
- Add some products to the slot using the show plan panel.
- Create order.

Feedback:

- Product list panel
  - Consider putting all info about the product on double click in the table (currently only info in the cell shows up).
  - Consider making the menu smaller in Add product.
  - When putting incorrect data in Add product, nothing happens.

  - When deleting a product with a non-existent ID(name), no error message occurs.

- Warehouse plan creating panel
  - Not clear how to exit plan creation panel.

5.1.2 Anton Striapan (CTU, FEE, SIT 3rd year student)

Goal:

- Create WMS plan with racks, nodes and arcs.
- Add some different products to WMS.
- Delete one of them using appropriate panels.
- Add some products to the slot using the show plan panel.
- Create order.

Feedback:

- There were some problems with the validation of fields and displaying info about errors. It would be better if there were some visual explanation of which fields are mandatory and which are not, which type of data should be entered number or string because it might cause errors in future.
- Problems or issues while regular usage – were not found.

5.1.3 UX testing results

In many places, the input text validation is absent. It will be fixed in the subsequent versions of the application.

Chapter 6

Conclusion

In this chapter, we will sum up the algorithms results and experience with the application.

The main goal of this bachelor thesis was to create a user interface for pick path optimization for a WMS. A desktop application with main WMS functions, warehouse plan creation functions, and pick path creation functions was implemented in this bachelor thesis. The main WMS functions in this application are adding, editing, and deleting SKUs. Also, it is possible to show a table with all SKUs and information about them.

Warehouse plan creation functions are: adding to warehouse plan racks with different properties, adding nodes representing key positions in a warehouse, and connecting those nodes with arcs representing a space where picker or picker-machine can move. Furthermore, there are many different options to interact with those warehouse parts.

Pick path for orders creation functions are order creation, creating a pick path for the order using various algorithms, displaying this pick path or its parts. The functions mentioned above make this application unique. The research done in this thesis shows that there are no similar applications with such a wide range of functions. All found applications have very limited UI, less flexibility of warehouse layout creation and pick path calculating.

Also, there was a goal to implement various algorithms for finding the shortest path. Four algorithms were implemented. Three of them allow to obtain the shortest path: they use permutation to consider all SKUs orders possible. Those algorithms are Dijkstra's algorithm with permutation, Floyd-Warshall algorithm and Johnson's algorithm.

The fourth algorithm - Dijkstra's algorithm - does not use permutation. So it creates the shortest pick path, but only for exact order SKUs in the order. The permutation of SKUs in the order has factorial time complexity, so it is a highly time-consuming operation.

Therefore, algorithms with permutation are useful for warehouses where the number of SKUs in an order does not exceed 10. However, Dijkstra's algorithm without permutation

can be used for orders of arbitrary size. Additionally, I would like to note that the program has a modular structure, so it is possible to add further functionality without understanding and working with already implemented code.

Appendix A

Application source code

To check application source code visit project repository. To visit project repository use QR code above(it is clickable).

Bibliography

[1] WAREHOUSE AND DISTRIBUTION SCIENCE, Release 0.98.1 [online]. © 1998–2017 John J. BARTHOLDI, III and Steven T. HACKMAN. https://www.warehouse-science.com/book/editions/wh-sci-0.98.1.pdf

[2] Visualize warehouse travel [online]. © 2019 John J. BARTHOLDI III, Georgia Institute of Technology. https://www.warehouse-science.com/apps/pickpath/index.html

[3] Warehouse Optimization - Algorithms For Picking Path Optimization [online]. © 2019 Ruthie Bowles, Logiwa. https://www.logiwa.com/blog/picking-path-optimization-algorithm

[4] The traveling salesman problem [online]. © Britannica, The Editors of Encyclopaedia. Encyclopedia Britannica, https://www.britannica.com/science/traveling-salesman-problem

[5] The shortest path problem [online]. © 2021 HackerEarth https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/

[6] Dijkstra's algorithm © 1959 Dijkstra, E. W. A note on two problems in connexion with graphs. Numerische Mathematik, 1(1), 269–271.

[7] Floyd-Warshall algorithm © 2004 Sandeep Kumar Shukla, "Floyd-Warshall algorithm", in Dictionary of Algorithms and Data Structures. https://xlinux.nist.gov/dads/HTML/floydWarshall.html

[8] Johnson's algorithm [online]. © 2004 Paul E. Black, , "Johnson's algorithm", in Dictionary of Algorithms and Data Structures. https://xlinux.nist.gov/dads/HTML/johnsonsAlgorithm.html

[9] Dijkstra's Shortest Path algorithm description [online]. © Brilliant.org https://brilliant.org/wiki/dijkstras-short-path-finder/

[10] Floyd-Warshall algorithm description [online]. https://www.baeldung.com/cs/floyd-warshall-shortest-path

[11] Johnson's algorithm description © Brilliant.org https://brilliant.org/wiki/johnsons-algorithm/

[12] JavaFX [online]. © 2021 Oracle https://www.oracle.com/java/technologies/javase/javafx-overview.html

[13] Java Swing [online]. © 1993, 2020, Oracle and/or its affiliates. https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html

[14] JFXPanel [online]. © 2008, 2015 Oracle and/or its affiliates. https://docs.oracle.com/javase/8/javafx/api/javafx/embed/swing/JFXPanel.html

[15] SQLite [online]. https://www.sqlite.org/index.html

[16] Johnson Algorithm psjava [online]. © 2014 psjava team. https://psjava.org/algo/Johnson_Algorithm

[17] Floyd Warshall Algorithm psjava [online]. © 2014 psjava team. https://psjava.org/algo/Floyd_Warshall_Algorithm

[18] PICK PATH OPTIMIZATION [online]. © 2019 Katalyst Technologies Inc. https://katalysttech.com/white-paper/pick-path-optimization/

[19] Example Warehouse Layout [online]. © 2021 Fit Small Business https://fitsmallbusiness.com/wp-content/uploads/2019/10/Screenshot_Example_Warehouse_Layout_of_Packed_Tables_Placed_in_the_aisles_Between_Pallet_Racks.jpg