Pick System Python Package documentation

Mohammadnaser Ansari

Contents

1 User's Guide ... 3
2 Introduction ... 3
2.1 Data Model ... 3
2.2 Python Modules ... 3
2.3 Data model and Python Module Relation ... 4
3 Basics of Pick System ... 4
3.1 Preparing the warehouse ... 4
3.1.1 Designing the warehouse ... 4
3.1.2 Assigning SKUs to Locations ... 5
3.2 Warehouse Servicing ... 5
3.2.1 Orders ... 5
3.2.2 Batching ... 5
3.2.3 Picking ... 5
4 Tutorial ... 5
4.1 Warehouse Design ... 6
4.1.1 twobyone ... 7
4.1.2 draw_whouse ... 7
4.2 generator ... 8
4.2.1 sku ... 8
4.2.2 order_datebound ... 8
4.2.3 order_normal_datebound ... 8
4.2.4 line_item_fixn ... 9
4.3 graph ... 9
4.3.1 nx_create ... 9
4.3.2 nx_create-db ... 9
4.3.3 nx_dijkstra_dm ... 10
4.3.4 nx_draw_graph ... 10

4.4 locap 10
4.4.1 random 10
4.4.2 coi 11
4.5 pickseq 11
4.5.1 order_in_one_all 11
4.5.2 fixed_batch_size_all 12
4.5.3 sku_to_node_pick 12
4.6 tsp 13
4.6.1 tsp_solver_master 13
4.6.2 google_sku 13
4.6.3 google_node 14
4.6.4 google_distance_matrix 14
4.6.5 gurobi_sku 15
4.6.6 gurobi_node 15
4.6.7 gurobi_distance_matrix 16
5 Data model 16
5.1 SKU 17
5.2 Orders 17
5.3 Line_item 18
5.4 Node 18
5.5 Arc 19
5.6 Slot 19
6 Dependent Packages 19
6.1 NetworkX 19
6.2 PyX 19
6.3 Google ortools 20
6.4 Gurobi and gurobipy 20
6.5 MySQL and mysql.connector 20
6.6 matplotlib.pyplot 20
7 Pick System Package Index 20

1 User's Guide

"Pick System" is a library for facilitating the process of programming a warehouse in python. The package is an integration of MySQL database with Python module. The idea in developing the package is for the researchers and academia working in supply chain area to have access to a pre-defined structure that they can work on.

The package consists of multiple modules, separated based on their role in the pick system. The package does not need installation and importing the Python files into the project will give the user the access to all modules and their functions. Although most of the functions in the package are completely new and were created by the authors, some utilizes outsider packages. Packages such as NetworkX, Matplotlib, Google Ortools, and Gurobi can be named as base function for some of the functions. This means that to get the full functionality of the package, some packages needs to be installed.

2 Introduction

As it was mentioned, the package consists of two main sectors, (1) a data model developed in MySQL and (2) Python modules. The data model works as the foundation while the module are the pillars of the package. In this section we will explain the structure of each section and at the end the relation between them.

2.1 Data Model

The data model consists of a set of related information stores and is implemented using a relational database. The data model is the back bone of package holding all modules together and at the same time gives the ability of being modular to the package by eliminating the need of programming language as a storage for data. The data model consists of several tables each playing a role in pick system package. A more detailed explanation of the data model is provided in section 5.

2.2 Python Modules

In the first step of developing the pick system data model and its related software package, Python has been chosen because of its popularity, simplicity, and its currently extensive library of packages that can be utilized in developing the pick system package. As it was mentioned in 4.2, the package is designed in a way that user comments and more importantly, future sample models can be easily integrated.

The pick system package is designed in 6 main modules:

1. graph: the graph module of package is responsible for creating a graph from the corresponding database
2. whousedesigns: Warehouse Designs (simply referred in the package as whousedesigns) is responsible for warehouse designs based on the specified parameters.
3. locap: Location Assigning Problem (simply referred in the package as locap) is responsible for assigning SKUs to warehouse Slots
4. sample: the sample module creates sample order, SKU, and line-item.
5. batch: Pick Sequence (in the package will be referred as pickseq) module groups the pick lines into a single pick list.
6. tsp: Traveling Salesman Problem or TSP module of package is responsible for generating pick paths using the pick list and graph.

In the following sections we tried to provide a document for each module of the package

2.3 Data model and Python Module Relation

The relation between the data model and the python modules can be described in a car manufacturing example. The chassis represents the data model in which parts of the car are mounted on, and the modules are the parts. The developed model will give different options for each part but using this structure give the freedom to the users to develop their own part, mount it on the data model and create a complete program without the need to create or code the whole thing. Back to the example, this will give a user an ability to test a customized or newly developed engine on a car, without the need to produce all of it. Moreover, the data model will give the user the freedom to use other programming languages (Java, C, C++, and etc.) alongside the Python package. Figure 1 "Data model in relation to Python Modules" illustrates the structure of database in the python package.

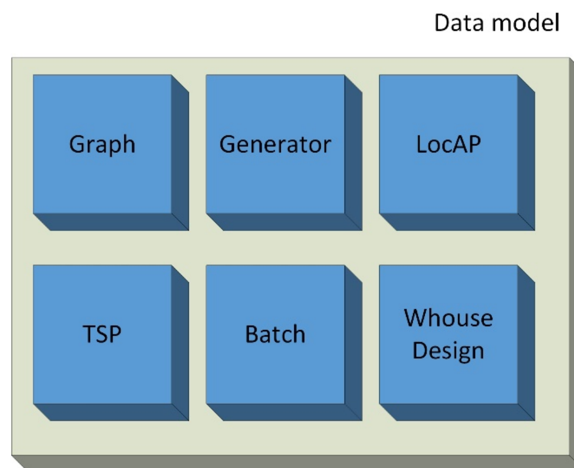

Figure 1 "Data model in relation to Python Modules"

3 Basics of Pick System

The first step in using the package is to understand the processes on a warehouse. The warehouse function can be described in two main categories, preparing and servicing. Preparing is referred to processes such as creating the warehouse and putting the products in their location, while the servicing is the step in which the warehouse is functional (receives orders, picks them, packs them and ships them).

3.1 Preparing the warehouse

Preparing the warehouse for service can be broken further broken down into designing the warehouse and allocating the locations in the warehouse to stock keeping units (SKUs)

3.1.1 Designing the warehouse

What is the warehouse size? How many aisles does it have? How are the aisles shape? What is the width of each aisle? And other questions like these are answered in this section. Developing a warehouse design is the first step in creating a pick system. In this step a warehouse should be created based on a width, length, and height. Then the aisles should be added to the warehouse with a specific width, length, and shape (traditional, chevron, flying V and …). In the last step, the racks, each containing multiple slots (as a location to keep the SKUs in them) are added to the warehouse.

3.1.2 Assigning SKUs to Locations

The next step in preparing the warehouse before it is ready for service is assigning SKUs to locations in the warehouse (stocking them). There are different methods in allocating the locations such as category based (like Walmart or most grocery stores) or a more optimal methods such as Customer Order Index (COI).We will talk about these methods in the section 3.2.

3.2 Warehouse Servicing

After the preparation stages are done, the warehouse can be open for service. The service process consists of receiving orders, each for a number of SKUs from the warehouse, batching the SKUs that should be picked together, and assigning the pick to a warehouse worker (called picker). The picker later picks the SKUs, brings them all to the packing station in which he SKUs for each order are packed together and getting ready to be shipped. In this section we will explain each step in a more detailed manner.

3.2.1 Orders

Each order consists of different information. The name of the customer, the date that the order was placed, the date that the customer desires to receive it, and the order number. The order also contains the SKUs that the customer wants and the quantity for each one.

3.2.2 Batching

The batching is the process of assigning the SKUs that needs to be picked into a batch that will be picked in one trip of the picker in the warehouse. There are several methods in batching based on different criteria. Batching based on size or the weight of the SKUs for the limitations of picker and cart, batching based on the location of the SKUs in the warehouse to decrease the trip length, batching based on the order to ease the process of packing and shipping and many more methods.

3.2.3 Picking

After the SKUs that should be picked together are put in one batch, the picker should start a trip in the warehouse to pick all of them. The picking process facilitates different methods in order to reduce the travel time. Methods such as S-Shaped, Traversal, and Sweep walking are the primary solutions, however, because of the similarities of this problem with Traveling Salesman Problem (TSP), more sophisticated and optimal solutions can be implemented.

4 Tutorial

The module consists of tools to give the ability of creating a complete pick system without the need for extensive coding. As it was discussed, the package was developed for academia and researcher to be able to create, test, or evaluate a specific method in the pick system without the need to code the whole pick system.

In this section we will review a step by step process that a user needs to take in order to create a pick system solely based on the modules and the functions of the package. The end user can replace the functions in any of the processes with other functions that play similar role from the package or add a new function with just some minor modification.

The following flow chart visualizes the algorithm in creating a pick system.

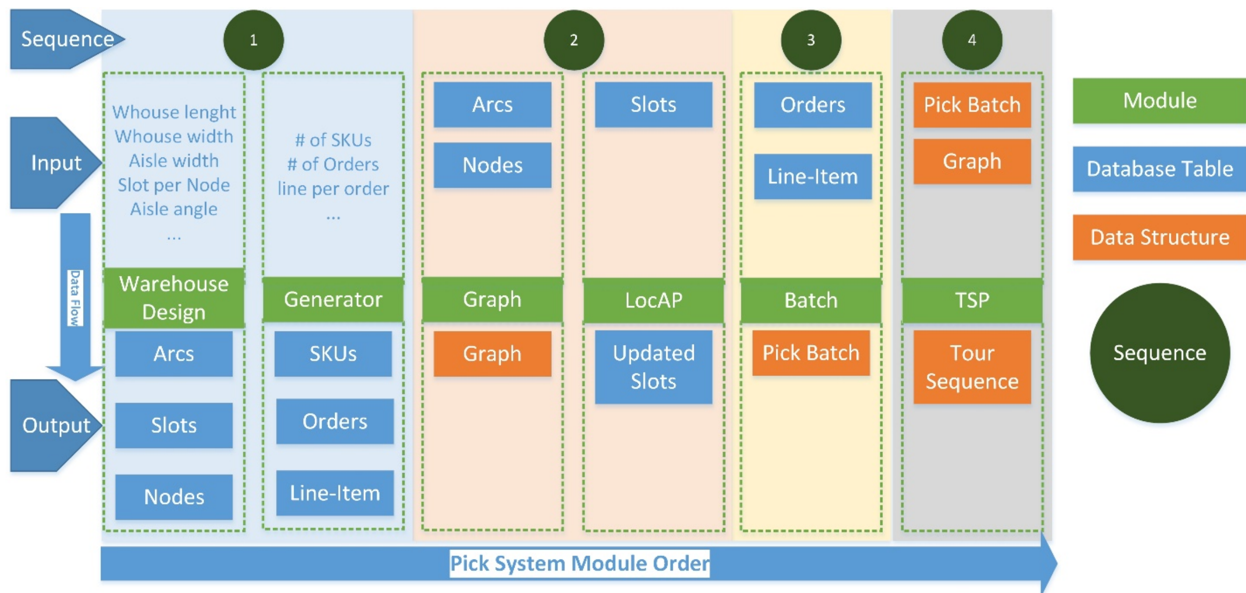

Figure 2 Pick System Flow Chart

Figure 1 Pick System Flow Chart identifies the steps that the user should take in order to create a complete pick system. As you can see in the figure, there are 4 sequence of jobs (color coded). The tasks in each sequence can be done simultaneously or interchangeably. The input/output of each module can be broken down into 4 main categories:

1. **Parameters** are identified by blue text. These values are user defined and are used in creating the samples and/or designs.
2. **Database tables** which represents a table from the SQL database. A more detailed breakdown of each table will be discussed in Data model section.
3. **Data Structures** which are kept in memory rather than database. The data structures are different type of objects (e.g. Graph is a graph object while pick batch and tour sequence are list of lists).

It should be noted that the figures highly simplified the process. Each function in the module is responsible or a part of output and the figure illustrates the cumulative responsibilities of all functions in each module.

Further in this section we will discuss each module and its function in more detail based on its sequence superiority. A table of all modules and their functions is provided in Pick System Package Index

4.1 Warehouse Design

The warehouse design (named whousedesign in python package) module of the package is responsible for designing and creating a warehouse based on the specified type. Table arc, node, and slot in the pick system database can be filled with the output of this module. The following functions can be utilized to create different type of design based on the user preferences.

4.1.1 twobyone

The twobyone function creates a two-by-one warehouse based on the parameters set by the user. The user can also set a non-zero aisle angle which will transform the warehouse from a traditional to chevron design

Inputs

- width: [float] the width of the warehouse
- length: [float] the length of the warehouse
- node_distance: [float] the desirable distances between each consecutive node
- center_aisle_width: [float] the width of the center aisle of the warehouse
- bottom_aisle_width: [float] the width of the bottom aisle of the warehouse
- aisle_width: [float] the width of each aisle (or pick aisle) of the warehouse
- aisle_angle: [float] the degree of deviation of the ailses from the traditional designs (parallel). The angle should be in degree, meaning 0 will return traditional horizontal aisles and 90 will return vertical aisles
- slots_per_node: [integer] the number of slots that the user will decide to assign to each node. This number can depend on the size of SKUs and the node distance. The higher the number, the smaller the slots

Output

- arcs: [list of lists] containing all the information of the arcs in the warehouse. Each list contains id, travel_factor, head_node_id, and tail_node_id respectively.
- nodes: [list of lists] containing all the information of the nodes in the warehouse. Each list contains id, name, x, y, z, and type of each node respectively.
- Slots: [list of lists] containing the id and node_id of each slot in the mentioned order

4.1.2 draw_whouse

This function visualizes the warehouse based on the exact location of the nodes, arcs and slots. Keep in mind that this function solely draws the nodes, arcs, and slots and not the graph. The output can be used to verify the warehouse design.

Inputs

- arcs: [list of lists] an exact replica of the arc table containing the id, travel_factor, head_node_id, and tail_node_id respectively
- nodes: [list of lists] containing all the information of the nodes in the warehouse. Each list contains id, name, x, y, z, and type of each node respectively.
- slots: [list of lists] containing the id and node_id of each slot in the mentioned order

Output

- none

4.2 generator

The generator module is responsible for creating sample order, SKU, and line-item for testing purposes. Real orders and line-item might have some characteristics such as correlation which are not generated using the current sample generators.

4.2.1 sku

The sku generator function, creates a list of lists of SKUs in which each list contains a unique id, a name which is generated by adding the "sku" before the id of each sku, a blank description and a class of A for all SKUs.

Input

- Sku_count: [integer] this number represents the number of the SKUs that the user wants to create using the sku function

Output

- sku_list: [list of lists] a list containing all the information needed to fill the sku table of the pick system database. Each list contains id, name, description and class of each SKU in the mentioned order

4.2.2 order_datebound

The order function creates random order list based on the specified start date, end date and the average number of orders per day.

Inputs

- order_per_day: [integer] average order per day that user wants the generator to create.
- start_date: [date] the date that the user wants the first order to be in.
- end_date: [date] the date the user wants the last order to be in.
- pick_date_deviation: [integer] the deviation between the date the order was received and the date that the order was processed for delivery.

Output

- order_list: [list of lists] containing the id, order_date, pick-date, costumer and order_number for each order respectively.

4.2.3 order_normal_datebound

The order function creates random order list based on the specified start date, end date, the average number of orders per day, and standard deviation of the normal distribution. Number of orders in each day follow a normal distribution with the average number of orders as the distribution mean and the standard deviation (stdev) as the standard deviation of that normal distribution.

Inputs

- avg_order_per_day: [integer] average order per day that user wants the generator to create.
- stdev: [float] The standard deviation for distribution of the average order per day
- start_date: [date] the date that the user wants the first order to be in.
- end_date: [date] the date the user wants the last order to be in.

- pick_date_deviation: [integer] the deviation between the date the order was received and the date that the order was processed for delivery.

Output

- order_list: [list of lists] containing the id, order_date, pick-date, costumer and order_number for each order respectively.

4.2.4 line_item_fixn

The line_item generator, generates random lines per each order. The number of lines per order and the quantity of the SKU in each line is asked from the user. The generator selects random SKUs from the sku_list for each line.

Inputs

- line_per_order: [integer] the number of lines per order that the user desires
- quantity: [integer] The average quantity of a SKU in each line of the order
- sku_id_list: [list] containing the id of all the SKUs.
- order_id_list: [list] containing the id of all orders.

Output

- line_list: [list of lists] a list of lists containing all the needed information by the pick system database; id, order_id, and sku_id respectively.

4.3 graph

The graph module of the package utilizes the available packages in python designed to develop graph as a tool to create a graph representation of the warehouse. The following is a list of available function in this module:

4.3.1 nx_create

Creates a graph using the package NetworkX.

Inputs

- arcs: [list of lists] containing id, travel_factor, head_node_id, and tail_node_id in the same order as mentioned.
- nodes: [list of lists] containing id, name, x, y, z, and type of each node respectively.
- graph_type: [string] type of graph can be “bidirectional” or “unidirectional”

Output

- NetworkX graph object

4.3.2 nx_create-db

Creates a graph using the package NetworkX but it only gets the database connection as an input and fetched the data itself

Inputs

- cnctn [mysql.connector database connection object]: a connection to the database holding the pick system data
- graph_type: [string] type of graph can be “bidirectional” or “unidirectional”

Output

- NetworkX graph object

4.3.3 *nx_dijkstra_dm*

The distance matrix (DM) in this function is calculated via Dijkstra algorithm (a well-known algorithm in calculating node distances in a graph). The output of this function can be used for TSP calculations.

Inputs

- graph: [NetworkX graph object]
- pickseq: [list] each list contains the pick list of a tour

Output

- Distance Matrix: [list of lists] containing the distance of each 2 sets of the pick list (e.g. distance_matrix[0][1] is the distance of 1st and 2nd node in the pick sequence)

4.3.4 *nx_draw_graph*

This function visualizes the graph as a plot as well as saving the plot in .png format. This function can be used to as a visual aid in creating the graph and the warehouse

Inputs

- graph: [graph object of NetworkX] The graph object created by the NetworkX package

Output

- none

4.4 *locap*

The location assigning problem (named as locap in the python package) in the warehouse is the process of assigning the SKUs to Slots. The smarter the assignment, the shorter the pick process which itself can save money. The locap module of the package contains functions that does it entirely or at least eases the process.

4.4.1 *random*

The random function of the locap is creating a random assignment of SKUs to locations.

Inputs

- slot_list: [list of lists] each list should contain the id and node_id of each slot
- sku_list: [list] a list containing the id of the SKUs

Or

- cnctn [mysql.connector database connection object]: a connection to the database holding the pick system data

Output

- slots: [list of lists] the updated slots list containing the sku_id for each corresponding slot and a default value for quantity

4.4.2 coi

This functions utilizes one of the most famous location assigning methods called Customer Order Index (COI) in assigning the SKUs to Slots in the warehouse. The COI method assigns the most popular SKUs (The one that has been ordered the most) to the best locations in the warehouse (The ones closest to the depot point).

Inputs

- graph: [graph object of NetworkX] The graph object created by the NetworkX package
- line_item_sku: [list] a list containing all the sku id from the line_item list.
- slots: [list of lists] each list should contain the id and node_id of each slot
- sku_id_list: [list] a list containing the id of the SKUs
- depot_node_id: [integer] the id of the depot node (usualy node id 1 represents the depot node)

Or

- graph: [graph object of NetworkX] The graph object created by the NetworkX package
- cnctn [mysql.connector database connection object]: a connection to the database holding the pick system data
- depot_node_id: [integer] the id of the depot node (usualy node id 1 represents the depot node)

Output

- slots_list: [list of lists] the updated slots list containing the sku_id for each corresponding slot.

4.5 pickseq

The pickseq module of the package is responsible for assigning the line-items that should be picked in one trip into a group. The pick sequencing can be based on the picker capacity, the items relevance (e.g. the chemicals should not be picked in the same tour as groceries in some cases), pick tour limitations, and many more. This module of the package can be used to implement different pick sequencing. Following is the available module in the current version of the package

4.5.1 order_in_one_all

order_in_one_all creates pick lists based on the order in which the lines belong to. Each outputted pick sequence contains all the lines in a single order. The output of this function creates a list of list assigning all items to pick batches.

Inputs

- item_list: [list of lists] this variable should contain a list of lists, being an exact replica of the line-item table of the pick system database. Each list in this variable contains quantity, order_id, and sku_id in the mentioned order.

- Order_list: [list of lists] this variable should contain a list of lists, being an exact replica of the order table of the pick system database. Each list in this variable contains id, order_date, pick_date, customer, and order_number respectively.

Or

- cnctn [mysql.connector database connection object]: a connection to the database holding the pick system data

Output

- Sku_pick_list: [list of lists] each list in this list of lists contains a pick sequence

4.5.2 fixed_batch_size_all

fixed_batch_size_all creates pick lists based on a fixed batch size amount. All batches will be exactly the same size (Except the last batch which assigns all the remaining items in one batch even if the count is smaller than the batch size). The output of this function creates a list of list assigning all items to pick batches.

Inputs

- item_list: [list of lists] this variable should contain a list of lists, being an exact replica of the line-item table of the pick system database. Each list in this variable contains quantity, order_id, and sku_id in the mentioned order.
- Order_list: [list of lists] this variable should contain a list of lists, being an exact replica of the order table of the pick system database. Each list in this variable contains id, order_date, pick_date, customer, and order_number respectively.
- Batch_size: [integer] the size of each batch which is a fixed number for each batch.

Or

- cnctn [mysql.connector database connection object]: a connection to the database holding the pick system data
- batch_size: [integer] the size of each batch which is a fixed number for each batch.

Output

- Sku_pick_list: [list of lists] each list in this list of lists contains a pick sequence

4.5.3 sku_to_node_pick

The sku_to_node_pick transforms the SKU pick list (which contains the SKUs that should be picked in each tour) into their corresponding nodes. Each node possesses a physical address in the warehouse and so in the graph representation of the warehouse. The outputs of this function can be used in order to calculate the shortest path that the picker can take to pick all SKUs as well as guiding the picker to the exact location (slot) of the SKU. The function automatically ignores the SKUs that should be picked but there is no stock left for that SKU.

Inputs

- Sku_pick_seq: [list] containing the SKUs that should be picked in a single picking trip

- Slots: [list of lists] each list contains a row the slot table from pick system database. Each row should contain slot_id, node_id, sku_id, and quantity respectively

Or

- sku_pick_seq: [list] containing the SKUs that should be picked in a single picking trip
- cnctn [mysql.connector database connection object]: a connection to the database holding the pick system data

Output

- node_pick: [list] containing the nodes that correspond to the SKU that the picker is assigned to pick.
- slot_pick: [list] containing the slots that should be visited to pick each SKU.

4.6 tsp

The Traveling Salesman Problem (TSP) calculates the best order of the nodes that should be visited in a graph to reduce the total traveling time or distance. In this concept, the corresponding node to each slot (which itself was selected based on the SKU that the picker wanted to pick) will play the node in the graph role and the picker will be the traveling salesman. Utilizing such algorithms can benefit us in achieving the best (or in some cases close-to-best) path the picker should take. The following are the functions that can be used to achieve this goal in the current version of the package.

4.6.1 tsp_solver_master

This TSP solver solves the routing problem and creates an order of nodes to be visited in each trip. This solver can be used for a fast solution rather than an accurate and close-to-optimal one.

Input

- sku_pick_seq: [list] containing the list of SKUs that should be picked in a single trip
- slots: [list of lists] each list contains the id, node id, SKU id, and quantity respectively. This variable is used in calling sku_to_node_pick function for transforming the SKUs to their corresponding node.
- graph: [graph object] the graph object of NetworkX which is used in this function to call nx_dijkstra_dm function for calculating the distance matrix.
- depot_node_id: [integer] the id of the depot node in the warehouse. This id is used in identifying the start and end point of each trip.

Output

- route: [list of dictionaries] each dictionary contains SKU, Slot, and node (e.g. {‘SKU’:sample_sku_id,’Slot’:sample_slot_id, ‘Node’:sample_node_id}). The sequence of the dictionaries corresponds to the order that they should be visited.

4.6.2 google_sku

The google function of TSP module utilizes a TSP algorithm that google incorporates in google maps website and app to find the shortest path between two locations. It should be mentioned that the method uses a heuristic algorithm which does not guarantee the optimal solution, but considering the

really low difference between the google output and the optimal solution as well as the speed of the function, we decided to include that in the current version of the package.

Input

- sku_pick_seq: [list] containing the list of SKUs that should be picked in a single trip
- slots: [list of lists] each list contains the id, node id, SKU id, and quantity respectively. This variable is used in calling sku_to_node_pick function for transforming the SKUs to their corresponding node.
- graph: [graph object] the graph object of NetworkX which is used in this function to call nx_dijkstra_dm function for calculating the distance matrix.
- depot_node_id: [integer] the id of the depot node in the warehouse. This id is used in identifying the start and end point of each trip.

Output

- route: [list of dictionaries] each dictionary contains SKU, Slot, and node (e.g. {‘SKU’:sample_sku_id,’Slot’:sample_slot_id, ‘Node’:sample_node_id}). The sequence of the dictionaries corresponds to the order that they should be visited.

4.6.3 google_node

Has the same functionality of the google_sku function. The only difference is that in this case, the function receives the nodes, and slots that it should visit as input rather than the SKUs that it should pick and list of all slots and the SKUs they are holding.

Input

- node_pick_seq: [list] containing the list of nodes that the target SKUs are connected to.
- slot_pick_seq: [list] containing the list of slots that the target SKUs are stored in.
- sku_pick_seq: [list] containing the list of SKUs that should be picked in a single trip
- graph: [graph object] the graph object of NetworkX which is used in this function to call nx_dijkstra_dm function for calculating the distance matrix.
- depot_node_id: [integer] the id of the depot node in the warehouse. This id is used in identifying the start and end point of each trip.

Output

- route: [list of dictionaries] each dictionary contains SKU, Slot, and node (e.g. {‘SKU’:sample_sku_id,’Slot’:sample_slot_id, ‘Node’:sample_node_id}). The sequence of the dictionaries corresponds to the order that they should be visited.

4.6.4 google_distance_matrix

Has the same functionality of the google_sku and google_node function. The only difference is that in this function, rather than using the nx_dijkstra_dm function to calculate the distance matrix, it gets the already calculated distance matrix. This function can be useful when the user wants to use another method rather than the default for calculation the distance matrix.

Input

- distance_matrix: [list of list] a 2 dimensional matrix containing the distance from each node in the node_pick_seq to the others.
- node_pick_seq: [list] containing the list of nodes that the target SKUs are connected to.
- slot_pick_seq: [list] containing the list of slots that the target SKUs are stored in.
- sku_pick_seq: [list] containing the list of SKUs that should be picked in a single trip
- depot_node_id: [integer] the id of the depot node in the warehouse. This id is used in identifying the start and end point of each trip.

Output

- route: [list of dictionaries] each dictionary contains SKU, Slot, and node (e.g. {'SKU':sample_sku_id,'Slot':sample_slot_id, 'Node':sample_node_id}). The sequence of the dictionaries corresponds to the order that they should be visited.

4.6.5 gurobi_sku

The gurobi function calculates the optimal route that the picker should take. The function uses the Gurobi python package. The parameters and constraints in this function can be manually changed in order to relax or add some constraints.

Input

- sku_pick_seq: [list] containing the list of SKUs that should be picked in a single trip
- slots: [list of lists] each list contains the id, node id, SKU id, and quantity respectively. This variable is used in calling sku_to_node_pick function for transforming the SKUs to their corresponding node.
- graph: [graph object] the graph object of NetworkX which is used in this function to call nx_dijkstra_dm function for calculating the distance matrix.
- depot_node_id: [integer] the id of the depot node in the warehouse. This id is used in identifying the start and end point of each trip.

Output

- route: [list of dictionaries] each dictionary contains SKU, Slot, and node (e.g. {'SKU':sample_sku_id,'Slot':sample_slot_id, 'Node':sample_node_id}). The sequence of the dictionaries corresponds to the order that they should be visited.
- obj_value: [float] the total distance that the picker must travel enable to pick all SKUs. This value represent the optimal distance based on the Gurobi output.

4.6.6 gurobi_node

Has the same functionality of the gurobi_sku function. The only difference is that in this case, the function receives the nodes, and slots that it should visit as input rather than the SKUs that it should pick and list of all slots and the SKUs they are holding.

Input

- node_pick_seq: [list] containing the list of nodes that the target SKUs are connected to.
- slot_pick_seq: [list] containing the list of slots that the target SKUs are stored in.
- sku_pick_seq: [list] containing the list of SKUs that should be picked in a single trip

- graph: [graph object] the graph object of NetworkX which is used in this function to call nx_dijkstra_dm function for calculating the distance matrix.
- depot_node_id: [integer] the id of the depot node in the warehouse. This id is used in identifying the start and end point of each trip.

Output

- route: [list of dictionaries] each dictionary contains SKU, Slot, and node (e.g. {'SKU':sample_sku_id,'Slot':sample_slot_id, 'Node':sample_node_id}). The sequence of the dictionaries corresponds to the order that they should be visited.
- obj_value: [float] the total distance that the picker must travel enable to pick all SKUs. This value represent the optimal distance based on the Gurobi output.

4.6.7 gurobi_distance_matrix

Has the same functionality of the gurobi_sku and gurobi_node function. The only difference is that in this function, rather than using the nx_dijkstra_dm function to calculate the distance matrix, it gets the already calculated distance matrix. This function can be useful when the user wants to use another method rather than the default for calculation the distance matrix.

Input

- distance_matrix: [list of list] a 2 dimensional matrix containing the distance from each node in the node_pick_seq to the others.
- node_pick_seq: [list] containing the list of nodes that the target SKUs are connected to.
- slot_pick_seq: [list] containing the list of slots that the target SKUs are stored in.
- sku_pick_seq: [list] containing the list of SKUs that should be picked in a single trip
- depot_node_id: [integer] the id of the depot node in the warehouse. This id is used in identifying the start and end point of each trip.

Output

- route: [list of dictionaries] each dictionary contains SKU, Slot, and node (e.g. {'SKU':sample_sku_id,'Slot':sample_slot_id, 'Node':sample_node_id}). The sequence of the dictionaries corresponds to the order that they should be visited.
- obj_value: [float] the total distance that the picker must travel enable to pick all SKUs. This value represent the optimal distance based on the Gurobi output.

5 Data model

In this section we will provide a more detailed view on structure of the SQL database. The data base consists of 6 relational tables. Figure 3 illustrates the EER diagram of the SQL database.

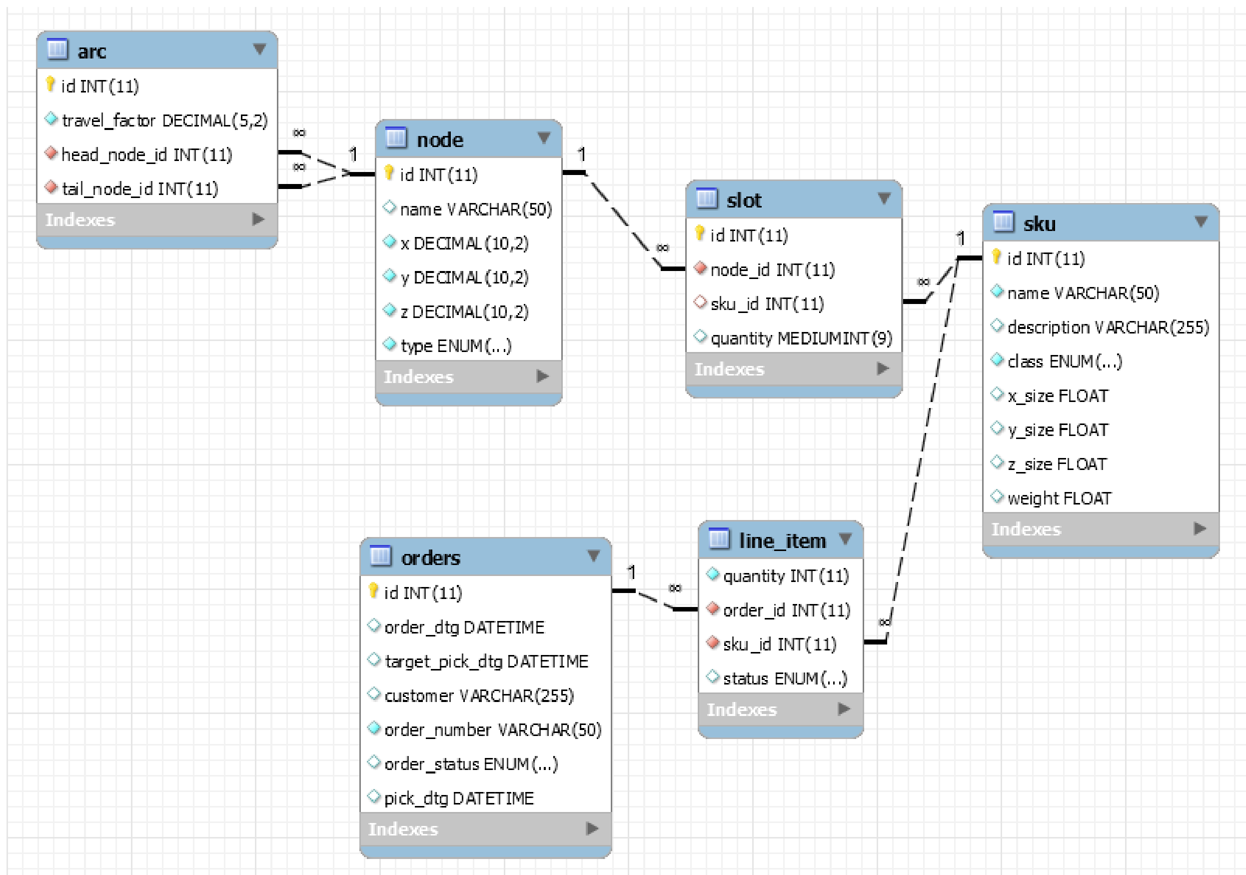

Figure 3 "Database EER Diagram"

5.1 SKU

The SKU table keeps record of the SKUs in our model. For each SKU the following information are stored in the table:

1. id [INT]: a unique number to identify each SKU
2. name [VARCHAR 50]: name of the SKU
3. description [VARCHAR 255]: a description on the SKU
4. Class [ENUM]: the class of the products that the SKU belongs to
5. x_size [Float]: the size of the SKU in X axis
6. y_ size [Float]: the size of the SKU in Y axis
7. z_size [Float]: the size of the SKU in Z axis
8. weight [Float]: The weight of each SKU (the unit is not important as long as it is consistent)

❖ The SKU table does not have any foreign keys.

5.2 Orders

The orders table store the information related to each order. The orders table consists of 7 columns:

1. id [INT]: a unique number to identify each order

2. order_dtg [DATETIME]: The time and date that order was placed
3. target_pick_dtg [DATETIME]: The time and date planned or requested for the order to be picked
4. customer [VARCHAR 255]: name of the costumer
5. order_number [VARCHAR 50]: The actual order number. This number is usually assigned by the sales department and does not affect the warehouse process
6. order_status [ENUM]: The current status of the order. The order can have one of these four statuses at any time:
   a. Received: The order was received, but is not processes yet
   b. Picking: The order is being picked
   c. Packing: The order is being packed
   d. Shipped: All the processes are done on the order and it has been shipped
7. pick_dtg [DATETIME]: The actual time and date that order was picked

❖ The orders table does not have any foreign keys

5.3 Line_item

The line_item table stores the lines in all the orders. The line_item table consists of 5 columns:

1. id [INT]: a unique number to identify each item
2. quantity [INT]: The quantity of the SKUs asked by the customer
3. order_id [INT]: The ID of the order that the line belongs to
4. sku_id [INT]: The ID of the SKU that the customer is asking for
5. status [ENUM]: order_status [ENUM]: The current status of the item. The item can have one of these four statuses at any time:
   a. Received: The request for the item was received, but is not processes yet
   b. Picking: The item is being picked
   c. Packing: The item is being packed
   d. Shipped: All the processes are done on the order and it has been shipped

❖ Order_id, and sku_id are foreign keys with 1→N relation

5.4 Node

Node table stores the node section of the information regarding the warehouse graph. Nodes can be defined as the location the picker can stop on in comparison to arcs that can only be used as a travel path. The node table consists of 6 columns:

1. id [INT]: a unique number to identify each node
2. name [VARCHAR 50]: name of the node
3. x [Decimal]: the location of node in the X axis

4. y [Decimal]: the location of node in the Y axis
5. z [Decimal]: the location of node in the Z axis
6. type [ENUM]: Type of node. The node type can be either "Picker", "Input", "Output" or "Other"

- ❖ The SKU table does not have any foreign keys.

5.5 Arc

The arcs in a graph illustrate the paths. In Arc table, the information about the arcs in the warehouse graph is stored. The arc table consists of 4 columns:

1. id [INT]: a unique number to identify each arc
2. travel_factor [Decimal]: The speed factor in that arc
3. head_node_id [INT]: Identifies the node that the arc is started from
4. tail_node_id [INT]: Identifies the node that the arc is ended in

- ❖ head_node_id and tail_node_id are foreign keys with 1→N relation

5.6 Slot

Slots in warehouse can be defined as space that the SKU is kept in while in the warehouse. Slot table keeps the information regarding all available slots, what SKU and in what quantity it is holding. The slot table consists of 4 columns:

1. id [INT]: a unique number to identify each slot
2. node_id [INT]: Identifies the node that the worker will be standing on while picking from this slot
3. sku_id [INT]: Identifies the SKU that is currently occupying this slot
4. quantity [MEDIUMINT]: The quantity of the SKU the slot is holding

- ❖ node_id is the foreign key with 1→N relation

6 Dependent Packages

As it was mentioned before, some of the functionalities of this package relies on other packages. Here is a list of packages that needs to be installed for the package to be fully functional as well as their role is provided.

6.1 NetworkX

The NetworkX package is an important part of the graph creating, graph drawing, and distance matrix calculation functions. The user can always use other graph packages or create his own graph functions and the tool is flexible in accepting any graph object whether it is NetworkX or any other package

6.2 PyX

The package behind the draw_whouse function. This function is not a necessity when it comes to creating a pick system, but help the process of creating warehouse designs.

6.3 Google ortools

This package is responsible for google TSP calculations. This uses the same algorithm behind Google Map guidance system.

6.4 Gurobi and gurobipy

Gurobi is a well-known software when it comes to calculating the optimal solution to TSP. In order to use the gurobi TSP functions in the package, the software and the python package should be installed.

6.5 MySQL and mysql.connector

The database software is a crucial part of package. MySQL and its python package (mysql.connector) are used in this case, but it can be replaced with any other SQL database software.

6.6 matplotlib.pyplot

This python package (imported as plt) is used in nx_draw_graph to draw the graph and save a .png copy of it. Again such as draw_whouse, this is not a crucial part of the pick system, but facilitates the process of designing the warehouse.

7 Pick System Package Index

In this section we will provide a table view of all the function in the package, their role, input, and output.

<table>
<tr><th colspan="2">Function Name</th><th>Inputs</th><th>Outputs</th></tr>
<tr><td rowspan="8">whousedesign.
twobyone</td><td rowspan="8">creates a twobyone warehouse based on the parameters set by the user</td><td>width: [float] the width of the warehouse</td><td rowspan="3">arcs: [list of lists] containing all the information of the arcs in the warehouse. Each list contains id, travel_factor, head_node_id, and tail_node_id respectively.</td></tr>
<tr><td>length: [float] the length of the warehouse</td></tr>
<tr><td>node_distance: [float] the desirable distances between each consecutive node</td></tr>
<tr><td>center_aisle_width: [float] the width of the center aisle of the warehouse</td><td rowspan="3">nodes: [list of lists] containing all the information of the nodes in the warehouse. Each list contains id, name, x, y, z, and type of each node respectively.</td></tr>
<tr><td>bottom_aisle_width: [float] the width of the bottom aisle of the warehouse</td></tr>
<tr><td>aisle_width: [float] the width of each aisle (or pick aisle) of the warehouse</td></tr>
<tr><td>aisle_angle: [float] the degree of deviation of the ailses from the traditional designs (parallel). The angle should be in degree, meaning 0 will return traditional horizontal aisles and 90 will return vertical aisles</td><td rowspan="2">Slots: [list of lists] containing the id and node_id of each slot in the mentioned order</td></tr>
<tr><td>slots_per_node: [integer] the number of slots that the user will decide to assign to each node. This number can depend on the size of SKUs and the node distance. The higher the number, the smaller the slots</td></tr>
<tr><td>whousedesign.
draw_whouse</td><td>Visualizes the warehouse based on the exact</td><td>arcs: [list of lists] an exact replica of the arc table containing the id, travel_factor, head_node_id, and tail_node_id respectively</td><td>None</td></tr>
</table>

	location of the nodes	nodes: [list of lists] containing all the information of the nodes in the warehouse. Each list contains id, name, x, y, z, and type of each node respectively.	
		slots: [list of lists] containing the id and node_id of each slot in the mentioned order	
generator. sku	Generates a random list of SKU based on the number provided as input	Sku_count: [integer] this number represents the number of the SKUs that the user wants to create using the sku function	sku_list: [list of lists] a list containing all the information needed to fill the sku table of the pick system database. Each list contains id, name, description and class of each SKU in the mentioned order
generator. order_datebound	The order function creates random order list based on the specified start date, end date and the average number of orders per day	order_per_day: [integer] average order per day that user wants the generator to create.	order_list: [list of lists] containing the id, order_date, pick-date, costumer and order_number for each order respectively.
		start_date: [date] the date that the user wants the first order to be in.	
		end_date: [date] the date the user wants the last order to be in.	
		pick_date_deviation: [integer] the deviation between the date the order was received and the date that the order was processed for delivery	
generator. order_normal_datebound	The order function creates random order list based on the specified start date, end date, the average	avg_order_per_day: [integer] average order per day that user wants the generator to create.	order_list: [list of lists] containing the id, order_date, pick-date, costumer and order_number for each order respectively.
		stdev: [float] The standard deviation for distribution of the average order per day	
		start_date: [date] the date that the user wants the first order to be in.	

	number of orders per day, and standard deviation of the normal distribution	end_date: [date] the date the user wants the last order to be in. pick_date_deviation: [integer] the deviation between the date the order was received and the date that the order was processed for delivery.	
generator. line_item_fixn	Generates random lines per each order	line_per_order: [integer] the number of lines per order that the user desires quantity: [integer] The average quantity of a SKU in each line of the order sku_id_list: [list] containing the id of all the SKUs. order_id_list: [list] containing the id of all orders.	line_list: [list of lists] a list of lists containing all the needed information by the pick system database; id, order_id, and sku_id respectively.
graph. nx_create	Creates a graph using the package NetworkX. (Package should be installed)	arcs: [list of lists] containing id, travel_factor, head_node_id, and tail_node_id in the same order as mentioned. nodes: [list of lists] containing id, name, x, y, z, and type of each node respectively. graph_type: [string] type of graph can be "bidirectional" or "unidirectional"	NetworkX graph object
graph. nx_dijkstra_dm	The distance matrix (DM) in this function is calculated via Dijkstra algorithm	graph: [NetworkX graph object] pickseq: [list] each list contains the pick list of a tour	Distance Matrix: [list of lists] containing the distance of each 2 sets of the pick list (e.g. distance_matrix[0][1] is the distance of 1st and 2nd node in the pick sequence)
graph. nx_draw_graph	This function visualizes the graph as a plot as	graph: [graph object of NetworkX] The graph object created by the NetworkX package	None

	well as saving the plot in .png format		
locap. random	The random function of the locap is creating a random assignment of SKUs to locations.	slot_list: [list of lists] each list should contain the id and node_id of each slot sku_list: [list] a list containing the id of the SKUs	slots: [list of lists] the updated slots list containing the sku_id for each corresponding slot and a default value for quantity
locap. coi	This functions utilizes one of the most famous location assigning methods called Customer Order Index (COI) in assigning the SKUs to Slots in the warehouse.	graph: [graph object of NetworkX] The graph object created by the NetworkX package line_item_sku: [list] a list containing all the sku id from the line_item list. slots: [list of lists] each list should contain the id and node_id of each slot sku_id_list: [list] a list containing the id of the SKUs depot_node_id: [integer] the id of the depot node (usualy node id 1 represents the depot node)	slots_list: [list of lists] the updated slots list containing the sku_id for each corresponding slot.
pickseq. order_in_one_all	creates pick lists based on the order in which the lines belong to. The output of this function creates a list of list assigning all items to pick batches.	item_list: [list of lists] this variable should contain a list of lists, being an exact replica of the line-item table of the pick system database. Each list in this variable contains quantity, order_id, and sku_id in the mentioned order. Order_list: [list of lists] this variable should contain a list of lists, being an exact replica of the order table of the pick system database. Each list in this variable contains id, order_date,	Sku_pick_list: [list of lists] each list in this list of lists contains a pick sequence

		pick_date, customer, and order_number respectively.	
pickseq. fixed_batch_size_all	creates pick lists based on a fixed batch size amount. All batches will be exactly the same size (Except the last batch which assigns all the remaining items in one batch even if the count is smaller than the batch size). The output of this function creates a list of list assigning all items to pick batches.	item_list: [list of lists] this variable should contain a list of lists, being an exact replica of the line-item table of the pick system database. Each list in this variable contains quantity, order_id, and sku_id in the mentioned order.	Sku_pick_list: [list of lists] each list in this list of lists contains a pick sequence
		Order_list: [list of lists] this variable should contain a list of lists, being an exact replica of the order table of the pick system database. Each list in this variable contains id, order_date, pick_date, customer, and order_number respectively.	
		Batch_size: [integer] the size of each batch which is a fixed number for each batch.	
pickseq. sku_to_node_pick	transforms the SKU pick list (which contains the SKUs that should be picked in each tour) into their correspondi ng nodes	Sku_pick_seq: [list] containing the SKUs that should be picked in a single picking trip	node_pick: [list] containing the nodes that correspond to the SKU that the picker is assigned to pick.
		Slots: [list of lists] each list contains a row the slot table from pick system database. Each row should contain slot_id, node_id, sku_id, and quantity respectively	slot_pick: [list] containing the slots that should be visited to pick each SKU.
tsp. tsp_solver_master	This TSP solver solves the routing problem and	sku_pick_seq: [list] containing the list of SKUs that should be picked in a single trip	route: [list of dictionaries] each dictionary contains SKU, Slot, and node (e.g. {‘SKU’:sample_sku_id,’Slot’:s

<table>
<tr><td rowspan="3"></td><td rowspan="3">creates an order of nodes to be visited in each trip.

This function uses the tsp package in python package index</td><td>slots: [list of lists] each list contains the id, node id, SKU id, and quantity respectively. This variable is used in calling sku_to_node_pick function for transforming the SKUs to their corresponding node.</td><td rowspan="3">ample_slot_id, ‘Node’:sample_node_id}). The sequence of the dictionaries corresponds to the order that they should be visited.</td></tr>
<tr><td>graph: [graph object] the graph object of NetworkX which is used in this function to call nx_dijkstra_dm function for calculating the distance matrix.</td></tr>
<tr><td>depot_node_id: [integer] the id of the depot node in the warehouse. This id is used in identifying the start and end point of each trip</td></tr>
<tr><td rowspan="4">tsp.
google</td><td rowspan="4">The google function of TSP module utilizes a TSP algorithm that google incorporates in google maps website and app to find the shortest path between two locations.

This packages uses google tsp package</td><td>sku_pick_seq: [list] containing the list of SKUs that should be picked in a single trip</td><td rowspan="4">route: [list of dictionaries] each dictionary contains SKU, Slot, and node (e.g. {‘SKU’:sample_sku_id,’Slot’:sample_slot_id, ‘Node’:sample_node_id}). The sequence of the dictionaries corresponds to the order that they should be visited.</td></tr>
<tr><td>slots: [list of lists] each list contains the id, node id, SKU id, and quantity respectively. This variable is used in calling sku_to_node_pick function for transforming the SKUs to their corresponding node</td></tr>
<tr><td>graph: [graph object] the graph object of NetworkX which is used in this function to call nx_dijkstra_dm function for calculating the distance matrix</td></tr>
<tr><td>Depot_node_id: [integer] the id of the depot node in the warehouse. This id is</td></tr>
</table>

		used in identifying the start and end point of each trip	
tsp. gurobi	The gurobi function calculates the optimal route that the picker should take. The function uses the Gurobi python package.	sku_pick_seq: [list] containing the list of SKUs that should be picked in a single trip	route: [list of dictionaries] each dictionary contains SKU, Slot, and node (e.g. {'SKU':sample_sku_id,'Slot':sample_slot_id, 'Node':sample_node_id}). The sequence of the dictionaries correspond to the order that they should be visited.
		slots: [list of lists] each list contains the id, node id, SKU id, and quantity respectively. This variable is used in calling sku_to_node_pick function for transforming the SKUs to their corresponding node.	
		graph: [graph object] the graph object of NetworkX which is used in this function to call nx_dijkstra_dm function for calculating the distance matrix.	obj_value: [float] the total distance that the picker must travel enable to pick all SKUs. This value represent the optimal distance based on the Gurobi output.
		Depot_node_id: [integer] the id of the depot node in the warehouse. This id is used in identifying the start and end point of each trip.	