A Computational Software System to Design Order Picking Warehouses

S. G. Ozden[a,*], A. E. Smith[b], K. R. Gue[c]

[a]*Information Sciences & Technology, Penn State Abington, Abington, PA 19001, USA*
[b]*Department of Industrial & Systems Engineering, Auburn University, Auburn, AL 36849, USA*
[c]*Research and Development, Fortna, Louisville, KY 40292, USA*

Abstract

Even though order picking is the most costly operation in a warehouse, current design practices have used the same principles (straight rows with parallel pick aisles and perpendicular cross aisles) to reduce travel distances between pick locations for more than sixty years. We present an open-source computational software system for facilitating the design of warehouse layouts to near-optimality considering average walking distance of the picker as the objective function. This software is particularly novel because a wide variety of traditional and innovative designs are automatically generated and evaluated. For the warehouse design parameters we consider the rectangular aspect ratio of the floorplan, the number and location of cross aisles, the number and location of pick aisles, and the location of a single input/output location. The main components of the design system are importing pick list profile data, creating the warehouse layout design as a network, product allocation (slotting) of SKUs through the warehouse, routing of pickers on a sample of orders using an exact routing algorithm, and design optimization using a meta-heuristic. We provide both mathematical and computational descriptions of the algorithms used by the software system, describe the types of problems that can be solved, and summarize our computational experience. This software is open source available on a GitHub website under an MIT license.

Keywords: Warehouse Design, Open Source, Computational Tool, Optimization

1. Introduction

Order picking is the most costly operation in a warehouse (Tompkins, 2010). It is strategic to every supply chain because slow performance results in unsatisfactory customer experience and high costs (Henn, 2012) especially vital in today's logistics environment. However, warehouses are still designed with the same design principles that have been in place for many decades, namely:

*Corresponding author
Email addresses: gokhan@psu.edu (S. G. Ozden), smithae@auburn.edu (A. E. Smith), kevingue@fortna.com (K. R. Gue)

straight rows with parallel pick aisles and perpendicular cross aisles (Vaughan and Petersen, 1999; Petersen, 1999). Gue and Meller (2009) challenged these assumptions by proposing the fishbone layout, which achieved reductions in travel distance of up to 20% in unit-load warehouses. The fishbone layout has been effectively applied to newly built warehouses (Meller and Gue, 2009). Even given these innovations, the research and practice in finding superior designs for order picking warehouses are lacking, especially by using a software design system.

Both research and practice are sparse largely because the problem is very complex and open-source software is not available. In this paper, we present an open source warehouse design and optimization system that can be used to find superior designs for different types of order picking operations. For ease of reference, we have named this system "GABAK." GABAK gives the user the ability to import sets of pick list data and optimize the layout to minimize the average travel distance. Along with identification of the warehouse block layout and appropriate product allocation/slotting, users can compare alternative designs explicitly or develop a design of experiments in a spreadsheet to import to GABAK to ascertain performance indicators (e.g., average travel distance and warehouse area). This is the first known open source warehouse design system and thus represents a major step forward. After a literature review, we give the overall approach in descriptive and mathematical terms, the details of each component of the software system, some test cases, and conclusions and recommendations for future research. Appendix A contains additional details regarding some algorithms and software components.

2. Literature Review

The common practice for warehouse layout design is to use computer aided design (CAD), with only a small percentage of the warehouse designers using a specialized warehouse design software (Baker and Canessa, 2009). Many companies design warehouses based on experience and rules of thumb. They then use a CAD software package to draft layouts. Baker and Canessa (2009) concluded that there are some techniques available to facilitate warehouse design, but they focus on assisting experienced warehouse designers rather than being comprehensive and fairly autonomous.

There are very few open access tools available for research and educational use related to warehouse design. Roodbergen et al. (2008) presented a model that minimizes travel distances in the picking area by identifying an appropriate layout structure consisting of one or more blocks of parallel aisles. This tool is accessible on Roodbergen's website (Roodbergen, 2019b) and it determines the optimal number of aisles and blocks but is limited to traditional layouts and assumes the depot location is fixed at the lower left corner. Other important assumptions are that products are slotted/allocated with a random storage policy and order pickers follow an S-shape routing for

picking. Another tool developed by Roodbergen (2019a) analyzed different routing methods in a traditional warehouse. This web based tool lets users to set the layout based on the number of blocks, the number of aisles, the number of storage locations, and the depot location, then test different routing methods for a given pick list (i.e., a set of pick locations).

The aforementioned papers assume a greenfield, or new, warehouse design. However, warehouses that are already constructed may realize a high return on investment with a warehouse re-design if there is increased efficiency in order picking operations. Berglund and Batta (2012) stated that cross aisle configurations can be changed without incurring prohibitive costs. Some order picking operations are performed by picking from pallet storage, which can be readily re-configured to re-orient the cross aisle and pick aisle positions. For this reason, GABAK can be employed for re-designing warehouses as well as for greenfield design.

3. Methodology

3.1. General Framework

The goal of GABAK is to identify superior layout designs for order picking warehouses. By superior, it is meant to have minimal travel distance/time while adhering to standard practices in warehouse designs and operations. Superior is also characterized by designs that move beyond the standard orthogonal warehouse designs with no or a few cross aisles. Using the characterization of the products (i.e., SKUs) and their popularity (i.e., the demand of the items and typical order lists), the system considers the design variables of floorplan aspect ratio, locations of cross aisles, locations of pick aisles, and location of the input/output (or depot) for a given warehouse capacity. The capacity is stated in interchangeable (that is, same size and shape) storage locations. The system returns values for each of the design parameters along with the calculated average travel distance of the pickers.

Our approach has the steps as given in Figure 1. First we import the pick list profile data of the warehouse (this could be historical data from the enterprise or simulated data generated from a pick list profile). Then we define a few search parameters (e.g., cross aisle width) which are fixed throughout the optimization and a few search boundaries (e.g., warehouse maximum aspect ratio) which create lower and upper bounds for some variables. Then the evolutionary strategies (ES) meta-heuristic, described in full later, automatically generates a wide variety of designs and returns the best one to minimize average expected distance traveled by order pickers. Distance is correlated with cost and is the typical metric used to assess warehouse designs in the literature.

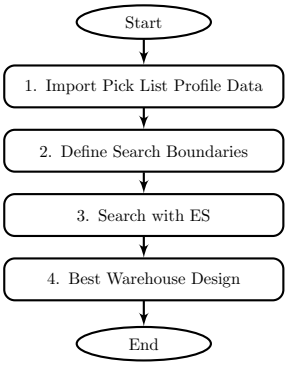

Figure 1: The solution approach

3.2. Assumptions

The order picking process is subject to a number of practical and common assumptions. First, we use a newly developed modified turnover-based storage (slotting or allocation) policy described in detail in Section 3.6.4. For warehouses that keep product popularity information updated on a timely basis, turnover-based storage is superior if there is little congestion. We assume that the turnover frequency of each product is known and constant through time. We also assume that each Stock Keeping Unit (SKU) can only be assigned to a single storage location and the capacity of this storage location is sufficient to store this SKU. Each storage location is of same size similar to assumptions given in (Roodbergen and De Koster, 2001a; Öztürkoğlu et al., 2014; Petersen and Aase, 2004). In practice, demand rates are varying. Therefore, warehouses using a turnover-based storage policy may need to reassign products to storage locations periodically.

Second, we consider only the straight line distance within an aisle, and not the lateral movements within a picking aisle. This is a common assumption in the warehouse literature (Goetschalckx and Ratliff, 1988).

Third, the picking route is assumed to start and end at a single depot or input/output location, located anywhere along the periphery of the warehouse.

Fourth, the product allocation/slotting and routing computations do not account for similar or correlated products stored at different locations. In other words, an SKU cannot be placed in multiple storage locations.

Finally, the capacity of the order picker is assumed to be sufficient for all items to be selected during a single tour. For a given pick list, only a single route is necessary for the picker to travel.

3.3. Importing Pick List Data

In the first phase, we use either simulated pick lists generated from a distribution using Bender's model (Bender, 1981) or use historical pick list data. The import phase only needs these two

parameters: pick list ID and SKU number. In Appendix B, we give an example of pick list data.

After the pick list data is imported, each unique SKU number and pick list ID is extracted and the average number of SKUs per pick list is calculated, which is later used by the product allocation/slotting algorithm.

3.4. Warehouse Design Classes

A warehouse design is classified according to three components: exterior nodes, interior nodes, and cross aisle segments (Öztürkoğlu et al., 2014). A node is defined as a point of intersection of a cross aisle segment and the exterior boundary of the design space, or the intersection of two or more cross aisle segments within the interior. We do not allow interior nodes with degree less than two. Cross aisle and picking aisle segments must be straight lines. We consider 19 design classes (shown in Figure 2) during a single search to find a best layout. Collectively, these classes comprise all possible designs that can be created with up to four exterior nodes (termination point of a cross aisle) and up to one interior node (intersection of multiple cross aisles). Note that a single exterior node is not possible. The set of four exterior and one interior nodes enables a huge number of possible layouts for warehouses including any that would actually be built. Add more possible nodes would be mathematically possible of course but not practical in warehousing.

3.5. Model Formulation

We now present the formal mathematical model used in the design optimization. Recall that we working with a rectangular area with up to four exterior nodes (ends of cross aisles) and up to one interior node (also the end of one or more cross aisles). There is a single depot which can be located anywhere along the perimeter of the rectangle. Nodes are placed in continuous, that is real valued, space. Pick aisles within a region defined by cross aisles are always parallel and of a set distance apart. Pickers are routed from the depot using a minimal path to gather all items and return to the depot. The decision variables are the location of the depot, the aspect ratio of the bounding rectangle, the locations of any cross aisle segments, and the angles of the pick aisles within each region. First, we define the notation used. Note that p is an exterior/interior node designator.

Table 1: Some important notation

Name	Description
C	set of connected exterior or interior nodes that form cross aisles
E	set of exterior nodes

Continued on next page

Table 1 – continued from previous page

Name	Description
D	distance matrix for storage locations
i	index of item (SKU)
j	index of item (secondary index)
k	index of storage location ($k = 1$ is for the depot location)
t	index of pick list

Table 2: Warehouse design parameters (set by the user)

Name	Description
SLW	storage location width
SLD	storage location depth
CAW	cross aisle width
PAW	pick aisle width

Table 3: Other parameters used (calculated by the system)

Name	Description
n	number of SKUs
m	number of pick lists
L_t	set of SKUs contained in pick list t
p	index of interior or exterior nodes where $p = 5$ is an interior node
s	index of interior or exterior nodes where $s = 5$ is an interior node (second index)

Table 4: Design decision variables

Name	Description
X	interior node X axis (standardized between 0 and 1, represented by $p = 5$)
Y	interior node Y axis (standardized between 0 and 1, represented by $p = 5$)
E_p	exterior node (one dimensional variable standardized between 0 and 1), $E_p\ 2\ E; p = 1; 2; 3; 4$
WA	warehouse area
AR	aspect ratio

Continued on next page

Table 4 – continued from previous page

Name	Description
DL	depot location (one dimensional variable standardized between 0 and 1)
A_r	angle of pick aisles in region r, A_r 2 $A; r = 1; :::; 8$. A is the set of region angle values
H_r	horizontal adjuster for region r, H_r 2 $H; r = 1; :::; 8$. H is the set of horizontal adjuster values for each region
V_r	vertical adjuster for region r , V_r 2 $V; r = 1; :::; 8$. V is the set of vertical adjuster values for each region
C_{ps}	binary variable equal to 1 iff node p is connected to node s, C_{ps} 2 $C; p$ 6= s
$x_{i;k}$	binary variable equal to 1 iff item i is is at location k
$l_{i;j;t}$	binary variable equal to 1 iff list t requires to go from item i to item j when performing the picking operation
$u_{i;t}$	the picking order of item i in list t

Table 5: Calculated values for a given warehouse design

Name	Description
NL	number of storage locations
$D_{k;l}$	distance between locations k and l

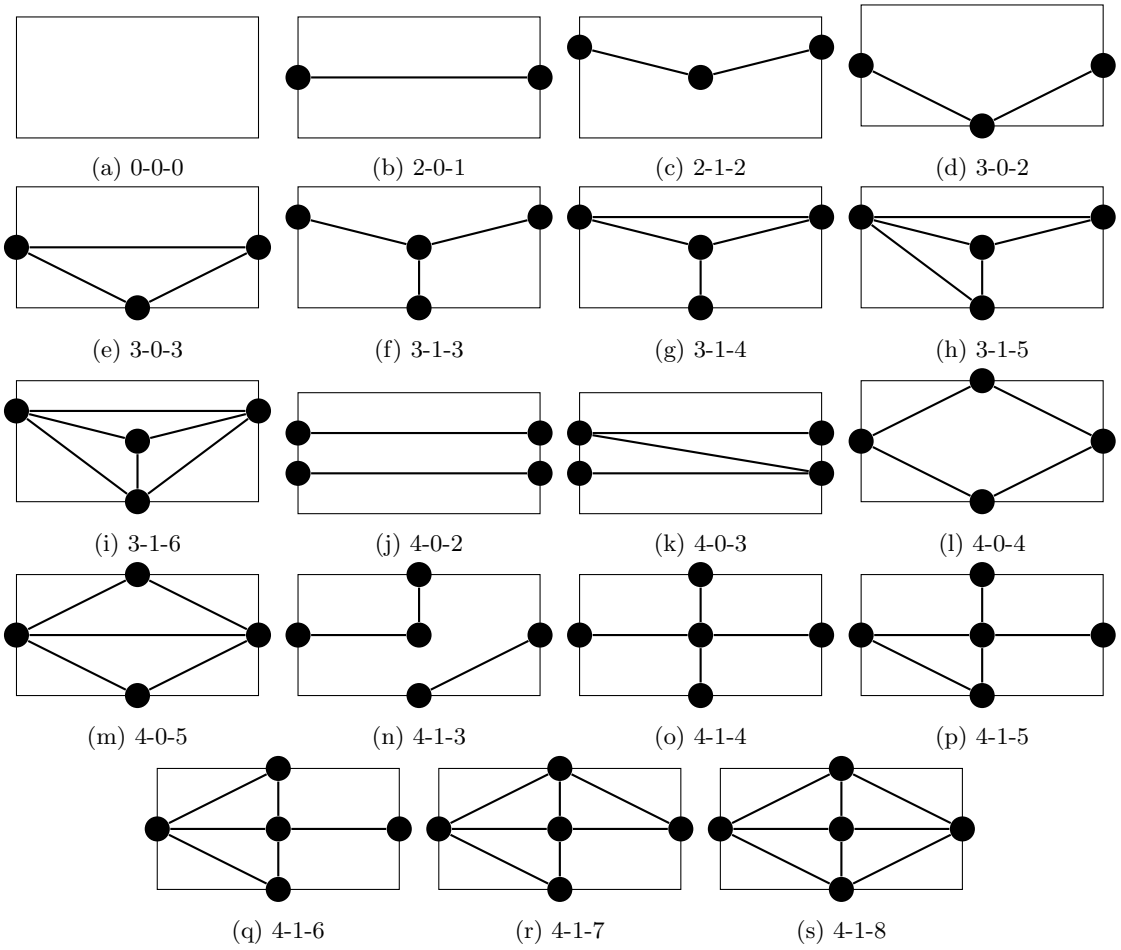

Figure 2: The 19 design classes searched using GABAK. For example 3-0-2 represents a warehouse design class with 3 exterior nodes, no interior nodes, and 2 cross-aisle segments. This design has 3 regions and each region has its own angled pick aisles (pick aisles are not depicted in this figure for simplicity). See Figure 3 for a detailed 3-0-2 design class example.

Model:

$$\text{Min} \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n}\sum_{t=1}^{m} x_{i;k}x_{j;l}l_{i;j;t}D_{k;l} \tag{1}$$

$$\text{s.t.} \sum_{j=1}^{n} x_{i;j} = 1; \quad 1 \quad i \quad n \tag{2}$$

$$\sum_{i=1}^{n} x_{i;j} = 1; \quad 1 \quad j \quad n \tag{3}$$

$$x_{1;1} = 1 \tag{4}$$

$$\sum_{j \in L_t} l_{i;j;t} = 1; \quad i \; 2 \; L_t; 1 \quad t \quad m \tag{5}$$

$$\sum_{i \in L_t} l_{i;j;t} = 1; \quad j \; 2 \; L_t; 1 \quad t \quad m \tag{6}$$

$$l_{i;i;t} = 1; \quad 1 \quad i \quad n; 1 \quad t \quad m \tag{7}$$

$$u_{i;t} = 1; \quad 1 \quad t \quad m \tag{8}$$

$$u_{i;t} \quad u_{j;t} + nl_{i;j;t} \quad n \quad 1; \quad 1 \quad i \quad n; 2 \quad j \quad n; 1 \quad t \quad m; i \neq j \tag{9}$$

$$NL + 1 = n \tag{10}$$

$$0 \quad X; Y; DL; AR; E_p; H_r; V_r \quad 1; \quad 1 \quad p \quad 4; 1 \quad r \quad 8 \tag{11}$$

$$WA \quad 0 \tag{12}$$

$$C_{ps}; x_{i;j}; l_{i;j;t} \; 2 \; f0; 1g; \quad 1 \quad p; s \quad 5; 1 \quad i; j \quad n; 1 \quad t \quad m \tag{13}$$

The aim here is to minimize the distance traveled by pickers when performing the order picking operation. The model is a modified version of the mathematical model proposed by Beroule et al. (2017). The objective function (eq. 1) represents the sum of the distances between each SKU to be picked when they are ordered as a TSP minimum distance tour. The picker starts and ends at the depot, which is denoted by $x_{1;1}$. The distance can either be the traditional aisle center to aisle center method or the more realistic visibility graph method according to the user's choice (Ozden et al., 2020). GABAK offers both distance metrics. The mathematical descriptions of the distance calculations are found in Appendix A.

Equations (2) and (3) force each SKU to be placed in a unique location and a location to be occupied by a unique SKU, respectively. For simplicity, location number 1 is the depot as shown in equation (4). Equations (5) and (6) enforce that for each pick list each SKU is preceded and followed by a unique SKU, respectively. Pick lists are considered cyclic, that is, each pick list starts and ends with the SKU number 1 (the depot). Equation (7) is used to prevent a pick list from containing the same SKU more than once. Equations (8) and (9) are used for sub tour elimination

for pick lists. Equation 10 ensures that the number of storage locations is equal to number of SKUs (plus one is for the depot since depot is represented by $x_{1;1}$). NL is calculated in Appendix A.

3.6. Searching the Design Space

A non-linear objective function, an extremely large search space, and continuous decision variables motivate a meta-heuristic that can effectively identify a near-optimal solution and is appropriate for continously valued variables. Evolutionary Strategies (ES) has been shown to be effective for warehouse design (Reehuis and Bäck, 2010). It is a very efficient algorithm with few tuneable parameters for optimization in continuous (real valued) space. Moreover, it self-adapts the search strategy ranging from diverse, global exploration to focused, local search based on the search progress.

Öztürkoğlu et al. (2014) introduced the use of a warehouse encoding of continuous variables. Our encoding is an extended version of Öztürkoğlu et al. (2014) that can search over multiple design classes. A region is an area bounded by cross aisles. Each design class has a different number of regions separated by cross aisle segments. For example in Figure 2, the 0-0-0 design class has only one region and the 2-0-1 design class has two regions separated by a single cross aisle segment. Our encoding uses a string of continuous variables that defines locations of the cross aisle endpoints, the angles of picking aisles in each region, the aspect ratio of the warehouse, and the adjusting amounts for the storage locations in each region, and the location of the depot. An exterior node is defined in one dimensional standardized space (between 0 and 1) where the upper-left corner is arbitrarily defined as the origin 0. The upper-right, lower-right, and lower-left corners are defined as 0.25, 0.5, 0.75, respectively. An interior node is defined in two dimensional standardized space (between 0 and 1) where the upper-left corner is defined as the origin (0, 0) and the lower-right corner is defined as (1, 1). Note that the coordinate definitions for exterior nodes and for interior nodes are (necessarily) different.

An example of the encoding and the represented layout are given in Table 6 and Figure 3, respectively. The "type" column in Table 6 shows the independent variables as type 1, the parameters as type 2, and the dependent variables as type 3. E1, E2, E3, and E4 reflect the position of exterior nodes along a clockwise path of length 1 beginning and ending at the upper left corner. IX and IY are the normalized coordinate locations of the interior node (if there is one). DE is the location of the depot using the same encoding system as used for exterior nodes. A1 through A8 are the angles of the picking aisles for pick aisle regions. HA1 to HA8 and VA1 to VA8 are horizontal and vertical adjuster variables, respectively. Each adjuster variable very slightly shifts the parallel pick aisles inside a region in horizontal or vertical directions without changing their

angle. These adjuster variables determine the precise positions for the pick aisles. These are fully defined mathematically in the Appendix A.

Recall that with up to four exterior nodes and one interior node, there exists the possibility of many cross aisles in all sorts of configurations. But most warehouse designs will be simpler, that is not use all possible cross aisles. Whether a cross aisle is created or not is controlled with PC parameters (PC stands for "probability of cross-aisle."). For example, the likelihood of having a cross aisle segment between the E1 and E2 nodes is determined by the PC12 parameter. A value of 1 means that a cross aisle segment between those two points will always be present whereas a 0 means it will never be present. Probabilities between 0 and 1 determine the likelihood of the cross aisle being present or not. The optimization algorithm decreases or increases the probability of these variables to find a superior design. Of course, to define the actual design, we need the realizations of the corresponding connections which are denoted by C12, C13, C14, C15, C23, C24, C25, C34, C35, C45 in this case.

In our encoding not every variable is used for every design class. For example, the 0-0-0 design class has only one region so it only uses A1, HA1, and VA1 and it does not use E1, E2, E3, E4, IX, nor IY. However, GABAK still stores these variables since the optimizer may change to a different design class that requires them.

The 19 design classes (see Figure 2) are considered because comprise all possible designs that can be built with up to four exterior nodes and up to one interior node. However, the GABAK source code has no limitation on the number of interior or exterior nodes.

Table 6: Encoding example

Name	Type	Range	Value	Description
SLW	2	$(0, \mathbf{1})$	4	Storage Location Width
SLD	2	$(0, \mathbf{1})$	4	Storage Location Depth
CAW	2	$(0, \mathbf{1})$	12	Cross Aisle Width
PAW	2	$(0, \mathbf{1})$	12	Pick Aisle Width
WW	3	$(0, \mathbf{1})$	400	Warehouse Width
WD	3	$(0, \mathbf{1})$	200	Warehouse Depth
WA	2&3	$(0, \mathbf{1})$	80000	Warehouse Area
AR	1	$(0, \mathbf{1})$	0.5	Aspect Ratio
E1	1	[0,1)	0.0416	Exterior Node 1
E2	1	[0,1)	0.2083	Exterior Node 2

Continued on next page

Table 6 – continued from previous page

Name	Type	Range	Value	Description
E3	1	[0,1)	0.6250	Exterior Node 3
E4	1	[0,1)	0.6250	Exterior Node 4
IX	1	(0,1)	0.5000	Interior Node X Axis
IY	1	(0,1)	0.5000	Interior Node Y Axis
DE	1	(0,1)	0.625	Depot
A1	1	[0,1)	0.1777	Region 1 Angle (32°)
A2	1	[0,1)	0.8166	Region 2 Angle (147°)
A3	1	[0,1)	0.5000	Region 3 Angle (90°)
A4	1	[0,1)	0.5000	Region 4 Angle (90°)
A5	1	[0,1)	0.5000	Region 5 Angle (90°)
A6	1	[0,1)	0.5000	Region 6 Angle (90°)
A7	1	[0,1)	0.5000	Region 7 Angle (90°)
A8	1	[0,1)	0.5000	Region 8 Angle (90°)
HA1	1	[0,1)	0.5000	Horizontal Adjuster 1
HA2	1	[0,1)	0.5000	Horizontal Adjuster 2
HA3	1	[0,1)	0.5000	Horizontal Adjuster 3
HA4	1	[0,1)	0.5000	Horizontal Adjuster 4
HA5	1	[0,1)	0.5000	Horizontal Adjuster 5
HA6	1	[0,1)	0.5000	Horizontal Adjuster 6
HA7	1	[0,1)	0.5000	Horizontal Adjuster 7
HA8	1	[0,1)	0.5000	Horizontal Adjuster 8
VA1	1	[0,1)	0.5000	Vertical Adjuster 1
VA2	1	[0,1)	0.5000	Vertical Adjuster 2
VA3	1	[0,1)	0.5000	Vertical Adjuster 3
VA4	1	[0,1)	0.5000	Vertical Adjuster 4
VA5	1	[0,1)	0.5000	Vertical Adjuster 5
VA6	1	[0,1)	0.5000	Vertical Adjuster 6
VA7	1	[0,1)	0.5000	Vertical Adjuster 7
VA8	1	[0,1)	0.5000	Vertical Adjuster 8
PC12	1	[0,1]	0	Probability of E1-E2 connection
PC13	1	[0,1]	1	Probability of E1-E3 connection

Continued on next page

Depot

Figure 3: Corresponding representation of the encoding

Table 6 { continued from previous page

Name	Type	Range	Value	Description
PC14	1	[0,1]	0.0001	Probability of E1-E4 connection
PC15	1	[0,1]	0	Probability of E1-I connection
PC23	1	[0,1]	1	Probability of E2-E3 connection
PC24	1	[0,1]	0	Probability of E2-E4 connection
PC25	1	[0,1]	0	Probability of E2-I connection
PC34	1	[0,1]	0	Probability of E3-E4 connection
PC35	1	[0,1]	0	Probability of E3-I connection
PC45	1	[0,1]	0	Probability of E4-I connection
C12	3	0 or 1	0	Realization of E1-E2 connection
C13	3	0 or 1	1	Realization of E1-E3 connection
C14	3	0 or 1	0	Realization of E1-E4 connection
C15	3	0 or 1	0	Realization of E1-I connection
C23	3	0 or 1	1	Realization of E2-E3 connection
C24	3	0 or 1	0	Realization of E2-E4 connection
C25	3	0 or 1	0	Realization of E2-I connection
C34	3	0 or 1	0	Realization of E3-E4 connection
C35	3	0 or 1	0	Realization of E3-I connection
C45	3	0 or 1	0	Realization of E4-I connection

3.6.1. Re-sizing the Warehouse and Penalizing Excess Locations

In the encoding scheme there is a possibility that a warehouse may have a few more or less storage locations than the number of SKUs. If the warehouse does not have enough locations, it is an infeasible design (with positive in nite cost). If the warehouse has excess storage locations, the system shows the number of excess locations (where no SKUs are assigned). The system has two methods to minimize the number of excess locations and the user can choose the alternative preferred. The rst is \resize to t" and performs a linear search for the warehouse area variable to eliminate extra locations (see Algorithm 1). Since this is a heuristic optimality is not guaranteed but it eliminates excess locations.

The other method is based on the \Near Feasibility Threshold, (NFT)" (Coit et al., 1996) an adaptive penalty function corresponding to a promising region close to feasibility. Excess locations are penalized but still considered feasible. We use a modi ed version of the NFT to penalize designs with excess locations dynamically (early iterations have less penalty and later iterations have more penalty for the same number of excess locations). Our slotting algorithm places any excess locations at distant locations from the depot since they are not used for picking as they are vacant. The NFT works as follows:

$$v_t = \mathrm{totalnumberoflocations}_t \quad \mathrm{totalnumberofSKUs} \tag{14}$$

where v_t is the number of excess locations at iteration t. NFT_t is calculated as follows:

$$NFT_t = \frac{NFT_0}{1 + \quad t}; \tag{15}$$

where NFT_t is the value of the near feasibility threshold at iteration t, NFT_0 is an upper bound for the NFT_t, is a user de ned positive parameter, and t is the iteration counter. The penalty p is calculated as follows:

$$p = \quad \frac{v}{NFT}^{2} \tag{16}$$

This adaptive penalty function permits the design space to be searched e ectively by allowing designs with few excess locations at early iterations and enforcing designs with no excess locations at later iterations. Compared to \resize to t" implementation, NFT is used during optimization, not afterwards. Figure 4 shows the increase of the penalty function with increasing number of iterations for $NFT_0 = 10$ and $= 0:05$

Algorithm 1 Pseudo-code of the Linear Search for Resize to Fit

```
function RESIZETOFIT (n,WA,AR)
    increased = 0
    decreased= 0
    warehousefit = false
    finalize = false
    while warehousefit == false do
        WW = sqrt(WA/AR)
        WD = sqrt(WA AR)
        Create a warehouse with width WW and depth WD
        if increased > 0 and decreased >0 then
            if number of storage locations< n then      . This check is necessary because at last
iteration it could have decreased and become infeasible
                WA = WA / 0.99                                                    . Increase area
            else if number of storage locations> n or finalize = true then
                warehousefit = true
                break the loop
            end if
        end if
        if number of storage locations< n then
            WA = WA / 0.99                                                        . Increase area
            increased = increased + 1
        else if number of storage locations> n then
            WA = WA * 0.99                                                       . Decrease area
            decreased = decreased + 1
        else if number of storage locations == n then
            warehousefit = true
        end if
    end while
end function
```

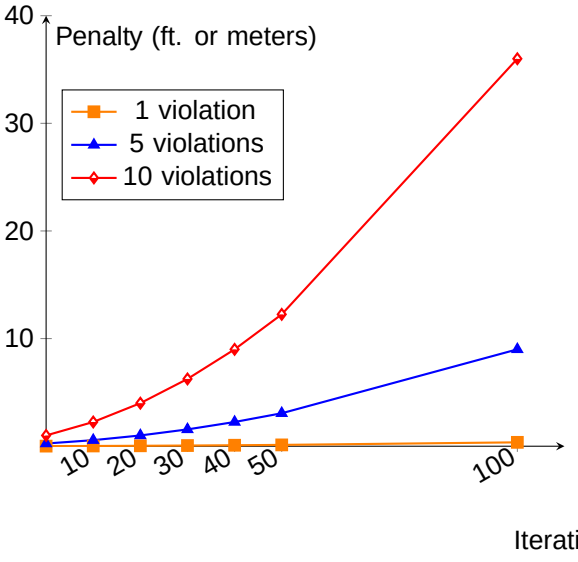

Figure 4: Comparison among different number of violations for penalty function

3.6.2. Evolutionary Strategies Meta-heuristic

Evolutionary Strategies (ES) was introduced by Rechenberg (1965) and Schwefel (1965) to imitate the principles of natural evolution as a method to solve continuous parameter optimization problems. ES is a population-based meta-heuristic optimization algorithm that uses biology-inspired mechanisms including mutation and survival of the ttest to re ne a set of solution candidates iteratively. ES is known for being e cient especially in continuous space and it operates with very few tuneable parameters. The search self adapts to diversify or concentrate the search based on the search progress.

GABAK uses an implementation of the standard ES algorithm, (+)-ES is the population strategy. The (+)-ES uses mutation from parent individuals to create o spring. From the joint set of parents and o spring, only the ttest ones survive (Schwefel, 1975, 1977). The default values in GABAK are a parent size of 20 and the number of children of 120. Therefore each parent creates six children at each iteration for these default values. O spring are generated through perturbations using a Gaussian/Normal probability distribution with mean 0 about the variables in the parent solution. The value (standard deviation of the Gaussian / Normal distribution) determines how similar or disimilar an o spring is from the parent. We use a single value for the population. The value changes according to a modi ed $\frac{1}{5}$ rule. Rechenberg $\frac{1}{5}$ success rule states that the ratio of successful mutations (where children are better than the parent) to all mutations should be $\frac{1}{5}$. If the ratio is greater than $\frac{1}{5}$, the which controls the step size is increased to nd better regions (exploration), and if it is less than $\frac{1}{5}$, the is decreased to focus the search more around the parents (exploitation). We reduce the ratio because each iteration's success rate is much lower than $\frac{1}{5}$ (usually around 5-10% at early iterations and 1% or lower at later iterations). Therefore we use $\frac{1}{20}$ as the success ratio for this rule. That is, at every 10 iterations (successratecounter = 10), if the ratio of successful mutations is larger than 5%, the sigma is increased by $\frac{1}{0.85}$. Otherwise, it is decreased by 0.85.

The e-vector contains all of the Type 1 decision variables described in Table 6, such as AR, E1, E2, E3, E4, and so on. The y-vector holds the modi ed/evolved version of the e-vector values with the added normal random values from the z-vector. Any value (y_k) of the y-vector outside the bounds is repaired to bring it back inside the boundary value by 10% of the variable range. We use a termination rule of maximum iterations and earlier if the optimization does not improve the best solution by more than 0.5% for 100 iterations. Figure 5 depicts the main steps of the ES algorithm. Algorithm 2 shows the pseudo-code of the ES meta-heuristic optimizer. The search automatically identi es new warehouse designs as it progresses by choosing values of the decision variables.

Algorithm 2 Pseudo-code of the ES algorithm

```
for all parents i in population of size do
    Initialize e-vector randomly between its bounds
    Calculate e-vector- tness value f (e)
end for
while maximum iterations not reached do
    for all o spring j in children population do
        Select a parente randomly
        Draw a z-vector from the normal distribution N(0, 2)
        y-vector = e-vector + z-vector
        for all values y_k in y-vector do //Repair y-vector if some values are outside the bounds
            if y_k < L_k then
                y_k = 0.1 * ( U_k - L_k)
            end if
            if y_k > U_k then
                y_k = 0.9 * ( U_k - L_k)
            end if
        end for
        if f (y) < f (e) then
            increase success rate
        else
            decrease success rate
        end if
    end for
    Join parent and children populations and select the ttest for the next generation
    successratecounter = successratecounter + 1
    if successratecounter = 10 then
        successratecounter = 0 //Reset counter
        if successrate > 0:05 then
                = = 0:85 //Increase sigma
        else
                =  0:85 //Decrease sigma
        end if
    end if
    if Less than 0.5 percent improvement over last 100 iterations then
        break;
    end if
end while
```

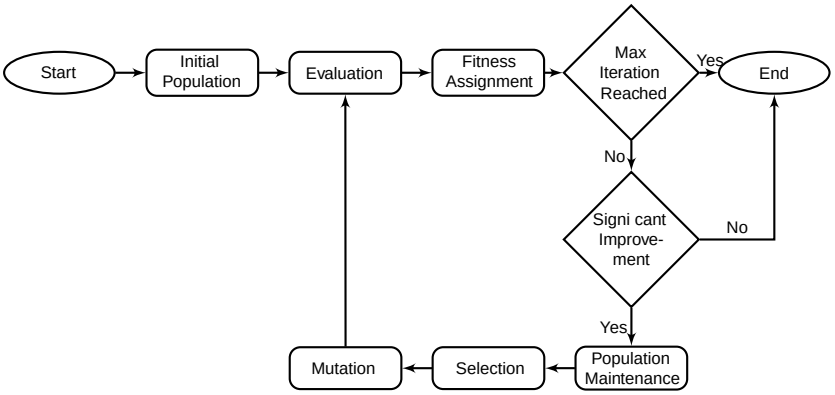

Figure 5: ES algorithm

3.6.3. Evaluation

A graph representing the warehouse as a network (see Figure 6) is created for each new design. This gure only depicts one example of each edge type and node type since showing the whole network would make the gure cluttered. There are two di erent network representations depending on the distance metric selected: aisle centers or visibility graph. With the aisle centers method, order pickers follow the centers of the aisles to perform picking. This modeling method is very common in the warehouse literature but it can overestimate tour distances especially for non-traditional (angled) layouts. With the visibility graph method, order pickers follow shortest paths by avoiding obstacles (such as racks and pallets) in a warehouse. The visibility graph considers the size of the picker as a parameter since it needs to create a space between the picker and the obstacles. Without such a bu er distance, the picker would literally walk along side obstacles such as racking with no space at all. This would not be realistic behavior, hence the need for a de ned size of picker or equivalently a bu er distance. (These distance methods are described in detail in Ozden (2017) and Ozden et al. (2020).)

The nodes of the network include the depot location (n1), exterior nodes (n2) and interior nodes (n3) for cross aisles, the beginning or ending points of pick aisles (n4), pick locations (n5), and corner nodes that connect the exterior boundary edges (n6) (to determine the aspect ratio). The edges of the network include exterior boundary edges (ed1), region edges (ed2), pick aisles (ed3), pick aisle end points connection edges (ed4), depot connection edges (ed5), and pick location connection edges (ed6). A region is an area bounded by region edges that is used to place angled pick aisles parallel to each other. A pick location is placed on the center line of the appropriate pick aisle and it provides access to the center of the corresponding storage locations. The distance between a pick location and its connected storage locations is assumed to be zero because these storage locations are actually served from the same coordinate.

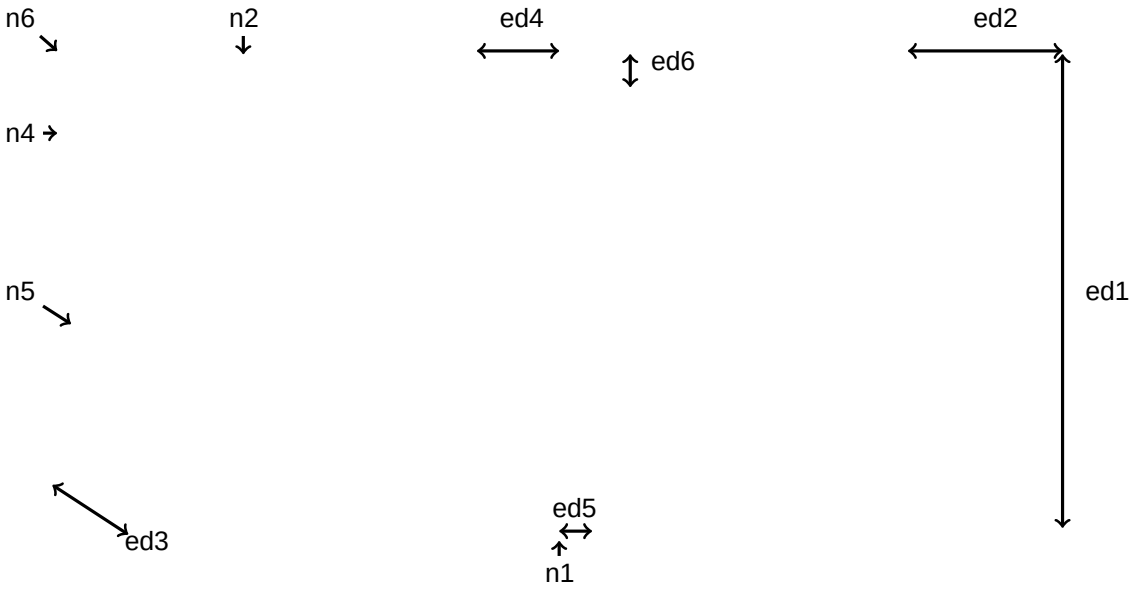

Figure 6: Warehouse graph based network representation

A connection between two nodes creates an edge, and the types of nodes connected de ne the types of edges created. Table 7 lists the connections between the types of nodes and the types of edges created by these connections. These show the node set and the edge set. There are two path nding methods (aisle centers and visibility graph) and they both use Dijkstra's shortest path algorithm to nd the shortest distances between any two pick locations or any pick location and the depot location.

3.6.4. Storage (Product Allocation / Slotting)

This section explains how the SKUs are assigned to storage locations within the new warehouse design. This must be done prior to evaluating the design using the average travel distance since the distance depends on where items are located. The many-to-many problem, also called the many-to-many shortest path, is to calculate the pick location to pick location shortest path distances to nd locations that are closest to any other location, on average. In other words, by calculating many-to-many distances, pick locations that are close to the centroid of the design space are highly desired. Dijkstra's algorithm can calculate many-to-many distances and, if n is the total number of pick locations, the worst-case running time is $O(mn \log n)$. After calculating the total travel distances to every location for each storage location, the locations are sorted based on total travel distance from smallest to largest to use in the product allocation algorithm described below.

A turnover-based storage policy is used because it is the most e cient policy among random and class-based storage policies (Petersen and Aase, 2004). In a turnover-based storage, the most

Table 7: List of node connections and the names of the resulting edges

Node 1	Node 2	Edge	Description
n6	n6	ed1	Connection of two corner nodes is an exterior boundary edge.
n6	n6	ed2	If there is no exterior node on the exterior boundary edge, then an exterior boundary edge becomes a region edge also.
n6	n2	ed2	Connection of an exterior node and a corner node is a region edge.
n2	n2	ed2	Connection of two exterior nodes is a region edge.
n2	n3	ed2	Connection of an exterior node and an interior node is a region edge.
n4	n4	ed3	Connection of two beginning and ending points of pick aisles is a pick aisle if the beginning or ending points of pick aisles are located on di erent region edges.
n4	n4	ed4	Connection of two beginning and ending points of pick aisles is a pick aisle connection edge if the beginning or ending points of pick aisles are located on the same region edges.
n1	n2	ed5	Connection of a depot location and an exterior node is a depot connection edge.
n1	n6	ed5	Connection of a depot location and a corner node is a depot connection edge.
n1	n4	ed5	Connection of a depot location and a beginning or ending point of a pick aisle is a depot connection edge.
n4	n5	ed6	Connection of a pick location and beginning (or ending) points of pick aisles is a pick location connection edge

frequently picked item is stored in the most convenient (most easily reached) location. In single-command travel (pick lists with a single item), the convenience of locations is based on its distance to depot. However, when there is more than a single item to pick, the order picker also travels between pick locations. Travel between pick location increases as the pick list size lengthens and also the location of the depot becomes less important (Petersen, 1997). The most convenient location for multiple-command travel for the turnover-based storage policy is not known in the literature (Pohl et al., 2011). Therefore, we developed an algorithm to calculate the convenience of locations based on not only considering distance to depot but also considering distance to the centroid of the warehouse (i.e., the location that has the least average travel distance to other locations as described above). Once the conveniences of the locations are calculated, the most frequently picked item is assigned to the most convenient location, and so on, down to the least frequently ordered SKU. There are other possible slotting algorithms in the literature (Accorsi et al., 2012; Walter et al., 2013) however, this one is common and computationally expedient.

In Figure 6, some pick locations (n5) have storage locations on both sides and some pick locations have storage locations only on one side. Either a single SKU or two SKUs can be assigned to one pick location depending if a pick location has a storage location on one side or has storage locations on both sides. An SKU cannot be stored in more than one pick location.

When importing pick list pro le data, the average pick list size is calculated. For any pick list that consists of $n > 1$ picks, an order picker needs to travel between $n - 1$ pick locations. Let do_k and dm_k denote single command and many-to-many travel distances for each pick location k. These distances are normalized so they can be ranked. The normalized single command sdo_k and many-to-many travel distances sdm_k for each pick location k are found by using Equations 17 and 18, respectively.

$$sdo_k = \frac{do_k - do_{min}}{do_{max} - do_{min}} \tag{17}$$

where do_{min} and do_{max} are the minimum and maximum values for single command travel distances, respectively.

$$sdm_k = \frac{dm_k - dm_{min}}{dm_{max} - dm_{min}} \tag{18}$$

where dm_{min} and dm_{max} are the minimum and maximum values for many-to-many travel distances, respectively.

These normalized values are between 0 and 1 where 0 means the most favorable location and 1 means the least favorable location. A linear combination of the normalized values of one-to-many

and many-to-many travel distance values is used. Let denote the fraction of the travel between distances (many-to-many problem) of expected travel distances and apls denote the average pick list size. is calculated by using:

$$= \frac{\text{apls} \quad 1}{\text{apls} + 1}: \tag{19}$$

This in turn is used to calculate the convenience c_k of each pick location k as a linear combination of the single command and the many-to-many problem in Equation 20:

$$c_k = (1 \quad) \quad \text{sdo}_k + \quad \text{sdm}_k \tag{20}$$

According to this equation, the most convenient locations have values close to 0 and the least convenient locations have values close to 1. Then SKUs are allocated by popularity (demand) from most to least convenient pick locations.

3.6.5. Selecting a Routing Method

Routing is the most computationally time consuming part of GABAK. For a given number of pick lists n, n TSP problems must be solved to calculate the objective function of average distance traveled. There are two options in GABAK for routing: Lin-Kernighan-Helsgaun (LKH) (Helsgaun, 2000) for achieving near-optimal solutions and Concorde (Applegate et al., 2019) for optimal solutions. Pansart et al. (2018) extended a dynamic programming algorithm initially proposed by Ratli and Rosenthal (1983) for one block traditional layouts. Their exact algorithm is faster than Concorde but it cannot be used in GABAK since it only works for traditional layouts using rectilinear or Manhattan distance. GABAK e ciently solves the TSP routings of a set of pick lists using parallel and distributed computing paradigms as described in (Ozden et al., 2017).

3.6.6. Optional Dynamic Sampling of Pick Lists

GABAK can perform dynamic pick list sampling during optimization as an option. Early iterations use a small sample from a set of pick lists to calculate the average travel distance per pick list, and later iterations increase the sample size dynamically. This saves computational time during early iterations where the optimization is exploring the entire warehouse design space rather than focusing on promising areas. Users should de nitely use this feature when they are optimizing large warehouses with a large set of pick lists (i.e., more than 1000 SKUs and 1000 pick lists).

3.6.7. User Interface

The graphical user interface enables a user to conduct analyses, perform optimization, or simply explore designs visually (see Figure 9). The system including a graphical user interface written in the C# programming language. The user enters various parameters (e.g., warehouse capacity, pick list data generation parameters) and selects options (e.g., distance method, routing algorithm). The user has the option to print designs to a printer or save designs as image files, including the vector based svg format, using the top menu. The computer processing status is reported at the bottom of the control panel.

3.6.8. Validation of the Optimization Algorithm

To validate the ES meta-heuristic, we followed a similar approach performed by Öztürkoğlu et al. (2014) and considered a warehouse situation where the Chevron design is the known optimal (Öztürkoğlu et al., 2012). This is a single, centrally located P&D point and one cross aisle with 1000 SKUs and the demand for those SKUs is uniform. The ES algorithm terminated after 362 iterations since there were no significant improvement over the last 100 iterations. GABAK returned a design very similar to a Chevron (see Figure 8), however the pick aisles were marginally different from the optimal angles in Chevron. The angle of the pick aisles of the right and left regions were 45.2° and 133.3°, instead of 45° and 135° as in the Chevron. The P&D point slightly moved to the left with a coordinate of 0.1235 compared to the exact middle position of 0.125 and the cross aisle has end points at 0.1232 and 0.6179 compared to the exactly centered middle aisle end points of 0.125 and 0.625. We performed the same validation with four other random number seeds with similar results. Recall that the ES is searching over a multitude of designs (formed by all 19 design classes) so identification of the simple, best design is a good indicator of its effectiveness. This test gives good evidence that the ES meta-heuristic is a powerful optimizer for warehouse design and can be relied on to give superior designs. The color coding of storage locations (see Figure 7) is from most (red) to least (purple) convenient location based on Equation 20. Color coding uses the wavelength of the visible spectrum below (Hardy and Steeb, 2008).

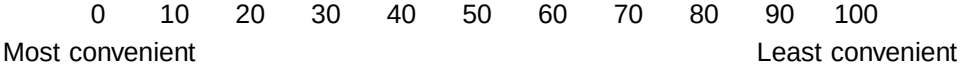

Figure 7: Color coding from wavelength

Figure 8: Validation of the ES returns a design almost identical to the known optimum

4. Example Instances and Usage

In this section, we describe several ways to use GABAK for warehouse design analysis and optimization. In the rst example, we create two xed warehouse layouts (traditional one block and two block) and compare the average travel cost of these two designs for a set of pick lists given by a company. We use this example to follow the GABAK work- ow. In the second example, we use a generated set of pick lists and nd the layout design that minimizes the average travel cost. In our nal example, we create a design of experiments to compare one block and two block traditional layouts with di erent pick list sizes. All experiments were performed on a Windows 10 computer with 8GB of RAM and Ryzen 7 1700X 8 core CPU.

4.1. Comparing One Block and Two Block Traditional Layouts

In this example, we compare one block and two block traditional layouts for a given set of pick lists from real data. The process begins by clicking the \Import RD" button and selecting the pick list data (an example pick list data le is provided in Appendix B). Then, input the warehouse parameters such as location width, location depth, and aspect ratio. To create a one block traditional layout, the Angle1 parameter is set to 90 degrees. There is the option to use the visibility graph method or the aisle centers method for estimating the distances between two locations. We selected the visibility graph method for this example and set the picker size to 5. There is also the option to choose LKH or Concorde for the routing algorithm. We use LKH for this example. The \Resize to Fit" option eliminates empty storage locations as much as possible using the line search described earlier and we select this option. The Adjuster and Pick Adjuster parameters use default values but can be changed them if desired. The remaining parameters do not need to be changed for a one block traditional layout so we keep the default values (the default aspect ratio is 0.5 and the default depot location is 0.625 which means it is located at the bottom center). After setting the parameters, click the \Create" button. Once the system nishes the calculations it shows the design and the key performance indicator \Average Distance" value which is 734.0 ft in this case. The warehouse area is 47,511 sq. ft and this warehouse can

accommodate 887 SKUs. This is the smallest size that can be achieved by using the \Resize to Fit" option, but is not provably optimal. The evaluation of this single xed warehouse design took 1 minute and 38 seconds. Figure 9 shows the GUI of GABAK as well as the traditional one block layout settings and the results of the experiment.

Figure 9: GABAK graphical user interface with traditional one block layout settings and results

Next, we discuss steps to create a two block layout. Change E1 to 0.875, E2 to 0.375, Angle2 to 90, and click the \C12" check box to add a cross aisle between E1 and E2. Then, click the \Create" button again and wait for the calculations to nish. GABAK shows the design in Figure 10. The \Average Distance" value is 652.6 ft. with a warehouse area of 51,935 sq. ft. Even though the area increased by about 9.3%, the two block design has 12% shorter picker routes on average. This is the common trade o between area of the warehouse and picker travel because as cross aisles are

added, the former increases while the latter decreases.

Figure 10: Traditional two block layout representation using the given parameters

4.2. Optimizing a Warehouse Design for Generated Pick Lists

In this example, GABAK is used to generate a set of pick lists for use in the design optimization process. The model by Bender (1981) calculates the probability of demand p_i for each SKU i for N total number of SKUs. The notation \100 x=100 F (x) indicates that 100x percent of the items represents 100 $F(x)$ percent of the total demand where $F(x)$ is the cumulative percent demand and x is a fraction of the total number of items stored. The model uses \(px) percent of the total number of items stored" and \(ptd) percent of the total demand" as parameters to calculate S as a shape factor which determines the skewness of the demand frequency curve. For a demand skewness pattern of 20/40 (i.e., twenty percent of the most frequently picked SKUs constitute forty percent of the picks), the value of S is 0.60. S is 0.20 and 0.07 for the demand skewness patterns of 20/60 and 20/80, respectively. The random storage policy can also be represented by this model as a 20/20 demand skewness pattern with a large value of S (Pohl et al., 2011). The shape factor S is determined from:

$$S = \frac{px^{\frac{ptd\ \ px}{100}}}{ptd\ \ px} \tag{21}$$

Bender (1981) represents the demand frequency curve with the model:

$$F(x) = \frac{(1 + S)x}{S + x} \tag{22}$$

The probability p_i of demand for SKU i is determined from

$$p_i = F\left(\frac{i}{N}\right) \quad F\left(\frac{i\ \ 1}{N}\right) \tag{23}$$

First, select \Generate" from the drop-down list. In this way, the system will generate pick list data instead of importing it from a spreadsheet. To achieve this, the parameter values in Table 8 are used:

Table 8: Parameter values for pick list data generation in this example

Parameter	Value
Percent of the total number of items stored (px)	20
Percent of the total demand (ptd)	80
# of SKUs (N)	1000
# of Pick Lists	266
Pick List Size	30
Order Generation Random Seed	0

Pick list size determines how many SKUs are in a single pick list. The seed number for random number generation creates di erent sets of pick lists with the same skewness ( ), and pick list size.

Once these parameters are set, click the \Import SKUs" button to generate SKUs and then click the \Import Order" button to generate the 266 pick lists. The \Max Iteration" parameter for the optimization algorithm (ES) is set to 500 which is a good balance between global and local search with the default values. \Mu" and/or \Lambda" values can be increased to perform a more exhaustive search but this will increase the computational time and memory usage. To start the optimization process, click the \Solve ES" button. GABAK will then automatically generate and evaluate many warehouse designs over the 19 design classes, iterating to nd the nal design. This optimization took 18 hours 22 minutes 57 seconds on a 8-core (16 logical core) computer with 8GB RAM. Computers with more CPU cores will achieve a faster execution because GABAK can use those additional cores during optimization (Ozden et al., 2017). During the optimization process, GABAK creates a folder (see Figure 11) on the computer so the user can see the graphics of the best designs found at each iteration (it does not create a new drawing if the best design does not change at the next iteration). GABAK also creates a spreadsheet in the same folder that logs the optimization process. Figure 12 shows the best design at the end of the optimization process.

Figure 11: Experiment folder for optimization shows the di erent best designs identi ed over the course of the optimization and the log

GABAK can also create animated videos to present how warehouse layouts evolve throughout

Figure 12: Best design at the end of the optimization

the optimization process. To do that, select from top Menu \Chart" and then select \New...". This will open a new window for the experiment folder and select the \experiment.csv" le. Once the le is selected, the system creates a video in the same experiment folder (see Figure 13).

Figure 13: A screen shot from the video output. The video shows the best design found so far (below) and the best objective function value at each iteration (above)

4.3. Comparing One and Two Block Traditional Warehouses with Varying Pick List Sizes

GABAK can import a spreadsheet to perform design of experiments batch optimization. The user enters optimization parameters in a spreadsheet for each experiment (e.g., lower and upper bounds for each variable such as aspect ratio or depot location, ES parameters such as maximum iterations or initial sigma, and/or pick list data generation parameters such as number of SKUs or number of pick lists). The system performs batch optimization and nds the best design for each

experiment. The user can then assess the robustness of designs to changes in variable values. We show an example below.

We will examine how varying sizes of pick list a ects the best layout from one block and two block traditional layouts. In this example, we develop a design of experiments using a spreadsheet and import this to GABAK to calculate the performance indicators for each experiment. Instead of setting parameters for each experiment via the user interface, a spreadsheet saves time and reduces possible data entry errors. In this design of experiments, we assumed a uniform demand among 1000 SKUs and tested pick list sizes of 1, 2, 3, 5, 10, and 30. Roodbergen and De Koster (2001b) performed a similar experiment with varying pick list sizes from 1 to 50. They found that the two block traditional layout performs better for all pick list sizes (except size 1).

Table 9: Design of experiments results

	Average Travel Distance (ft.)		
Pick List Size	One Block	Two Block	% Di
1	279	284	1.53
2	464	441	-5.04
3	590	542	-8.11
5	780	684	-12.2
10	1110	919	-17.17
30	1881	1524	-18.97

The traditional one block layout performed better than traditional two block layout (see Table 9) when a single item is picked from the warehouse, consistent with the ndings of Roodbergen and De Koster (2001b). In every other case, the traditional two block layout had shorter average travel distances per pick list. Researchers can use this experimental design feature to investigate layout sensitivity and robustness to changes such as altering the depot location or modifying the demand skewness.

4.4. Summary of Other Features and Potential Uses

In this section, we summarize other features of GABAK and their potential uses:

1. Along with the slotting algorithm already de ned, GABAK can allocate products using a dedicated storage policy. The rst SKU goes to rst location (based on location ID), the second SKU goes to the second location, and so on. The rst SKU is chosen on its order of appearance if the data is imported. Each storage location gets an ID based on the order created by the system. In this way, an organization which already has product allocation de ned can use this to explore designs and their performance.
2. Each optimization process produces a spreadsheet to keep a log of the best design found at each iteration. These logs include all of the variables (such as aspect ratio, exterior

nodes, pick list size) needed to re-create the design. GABAK can also import designs from a spreadsheet. A user can use this feature to ne tune the designs produced by the optimizer without re-entering them manually.

5. Conclusions and Future Research

Warehouse design optimization for order picking operations is a complex task that involves many variables and many algorithms. Open source tools for warehouse layout optimization did not exist until the recently published GABAK under an MIT License. The main challenges of developing GABAK were to design a comprehensive but compact representation of a warehouse design, to specify the overall system architecture, and to develop e cient and e ective algorithms for each task, as well as to integrate the algorithms e ciently. Note that developing this system was a very signi cant undertaking and GABAK contains more than 10,000 lines of code.

Although GABAK was designed for warehouse layout optimization, other applications can bene t from it. Researchers could develop a module to export designs to a CAD design software or export designs to an augmented reality software to test them in a simulated real world environment. The GABAK visualization capabilities could be used to understand the e ects of di erent shapes of warehouses and the video animation feature could assist researchers in warehouse layout optimization analysis.

In future work, we plan to extend GABAK to di erent routing algorithms and storage policies. Another enhancement is to add the option for more than one depot. Also we plan to use graphical processing units to decrease the computational time of calculations. Finally, we are working on a warehouse augmented reality software so that designs can be virtually tested for operational practicality.

Acknowledgment

This research was supported in part by the National Science Foundation under Grant CMMI-1200567. Any opinions, ndings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily re ect the views of the National Science Foundation. The authors would like to thank undergraduate and graduate research assistants Michael Robbins, Ataman Billor, and Akhil Varma Jampana for data collection and testing the code. Source code and binary les can be downloaded fromhttps://github.com/gokhanozden/gabak .

References

Accorsi, R., Manzini, R., Bortolini, M., 2012. A hierarchical procedure for storage allocation and assignment within an order-picking system. a case study. International Journal of Logistics Research and Applications 15 (6), 351{364.

Applegate, D., Bixby, R., Chvatal, V., Cook, W., 07 2019. Concorde tsp solver.http://www.math.uwaterloo.ca/tsp/concorde.html , online Accessed 07.18.2019.

Baker, P., Canessa, M., 2009. Warehouse design: A structured approach. European Journal of Operational Research 193 (2), 425{436.

Bender, P. S., 1981. Mathematical modeling of the 20/80 rule: theory and practice. Journal of Business Logistics 2 (2), 139{157.

Berglund, P., Batta, R., 2012. Optimal placement of warehouse cross-aisles in a picker-to-part warehouse with class-based storage. IIE Transactions 44 (2), 107{120.

Beroule, B., Grunder, O., Barakat, O., Aujoulat, O., 2017. Order picking problem in a warehouse hospital pharmacy. IFAC-PapersOnLine 50 (1), 5017{5022.

Coit, D. W., Smith, A. E., Tate, D. M., 1996. Adaptive penalty methods for genetic optimization of constrained combinatorial problems. INFORMS Journal on Computing 8 (2), 173{182.

Goetschalckx, M., Ratli , H. D., 1988. Order picking in an aisle. IIE Transactions 20 (1), 53{62.

Gue, K. R., Meller, R. D., 2009. Aisle con gurations for unit-load warehouses. IIE Transactions 41 (3), 171{182.

Hardy, A., Steeb, W.-H., 2008. Mathematical tools in computer graphics with C# implementations. World Scienti c Publishing Company.

Helsgaun, K., 2000. An e ective implementation of the Lin-Kernighan traveling salesman heuristic. European Journal of Operational Research 126, 106{130.

Henn, S., 2012. Algorithms for on-line order batching in an order picking warehouse. Computers & Operations Research 39 (11), 2549{2563.

Meller, R. D., Gue, K. R., 2009. The application of new aisle designs for unit-load warehouses. In: NSF Engineering Research and Innovation Conference. pp. 1{8.

Ozden, S. G., 2017. A computational system to solve the warehouse aisle design problem. Ph.D. thesis, Auburn University.
URL http://etd.auburn.edu/handle/10415/5761

Ozden, S. G., Smith, A. E., Gue, K. R., 2017. Solving large batches of traveling salesman problems with parallel and distributed computing. Computers & Operations Research 85, 87–96.

Ozden, S. G., Smith, A. E., Gue, K. R., 2020. A novel approach for modeling order picking paths. Naval Research Logistics.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.21966

Öztürkoğlu, Ö., Gue, K. R., Meller, R. D., 2012. Optimal unit-load warehouse designs for single-command operations. IIE Transactions 44 (6), 459–475.

Öztürkoğlu, Ö., Gue, K. R., Meller, R. D., 2014. A constructive aisle design model for unit-load warehouses with multiple pickup and deposit points. European Journal of Operational Research 236, 382–394.

Pansart, L., Catusse, N., Cambazard, H., 2018. Exact algorithms for the order picking problem. Computers & Operations Research 100, 117–127.

Petersen, C. G., 1997. An evaluation of order picking routing policies. International Journal of Operations & Production Management 17, 1098–1111.

Petersen, C. G., 1999. The impact of routing and storage policies on warehouse efficiency. International Journal of Operations & Production Management 19, 1053–1064.

Petersen, C. G., Aase, G., 2004. A comparison of picking, storage, and routing policies in manual order picking. International Journal of Production Economics 92 (1), 11–19.

Pohl, L. M., Meller, R. D., Gue, K. R., 2011. Turnover-based storage in non-traditional unit-load warehouse designs. IIE Transactions 43 (10), 703–720.

Ratliff, H. D., Rosenthal, A. S., 1983. Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. Operations Research 31, 507–521.

Rechenberg, I., August 1965. Cybernetic Solution Path of an Experimental Problem. Royal Aircraft Establishment Library Translation No. 1122, Farnborough.

Reehuis, E., Bäck, T., 2010. Mixed-integer evolution strategy using multiobjective selection applied to warehouse design optimization. In: Proceedings of the 12th annual conference on Genetic and evolutionary computation. pp. 1187–1194.

Roodbergen, K. J., Mar. 2019a. Interactive warehouse. Online.
URL http://www.roodbergen.com/warehouse/frames.htm

Roodbergen, K. J., Feb. 2019b. Warehouse optimizer by Kees Jan Roodbergen. Online.
URL http://www.roodbergen.com/whopt/

Roodbergen, K. J., De Koster, R., 2001a. Routing methods for warehouses with multiple cross aisles. International Journal of Production Research 39, 1865–1883.

Roodbergen, K. J., De Koster, R., 2001b. Routing order pickers in a warehouse with a middle aisle. European Journal of Operational Research 133(1), 32–43.

Roodbergen, K. J., Sharp, G. P., Vis, I. F., 2008. Designing the layout structure of manual order picking areas in warehouses. IIE Transactions 40 (11), 1032–1045.

Schwefel, H., 1965. Kybernetische evolution als strategie der exprimentellen forschung in der strmungstechnik. Master's thesis, Technical University of Berlin.

Schwefel, H., 1975. Evolutionsstrategie und numerische optimierung. Ph.D. thesis, Technical University of Berlin.

Schwefel, H., 1977. Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie: mit einer vergleichenden Einfhrung in die Hill-Climbing-und Zufallsstrategie. Vol. 26. Birkhäuser Basel.

Tompkins, J. A., 2010. Facilities Planning. John Wiley & Sons.
URL http://books.google.com/books?id=-xBIq6Qm2SQC

Vaughan, T. S., Petersen, C. G., 1999. The effect of warehouse cross aisles on order picking efficiency. International Journal of Production Research 37, 881–897.

Walter, R., Boysen, N., Scholl, A., 2013. The discrete forward–reserve problem–allocating space, selecting products, and area sizing in forward order picking. European Journal of Operational Research 229 (3), 585–594.

Appendix A. Number of Locations and Distance Matrix Calculations

Table A.1: Notation for Appendix

Name	Description
R	set of regions
P_r	set of pick aisles in region r
K_r^p	set of pick locations in pth pick aisle in region r
SL_{rpk}	kth pick location's left storage location node in region r's pth pick aisle
SR_{rpk}	kth pick location's right storage location node in region r's pth pick aisle
$TEMPNODES$	a set used for storing pick aisle intersection nodes with region edges

Algorithm A.1 Detailed Pseudo-code of the Algorithm that Returns the Total Number of Locations

function NumberofLocations(X,Y,E,PD,A,H,V,C,WA,AR,SLW,SLD,CAW,PAW)
  $WW = \sqrt{\frac{WA}{AR}}$
  $WD = \sqrt{WA \quad AR}$
  CreateCornerEdges(CAW,WW,WD)
  AddExteriorNodes(E)
  AddInteriorNode($X;Y$)
  AddPickandDepositNode(PD) . This function is similar to AddExteriorNodes function
  Connect all exterior and interior nodes based on C and add them as region edges
  $NL = 0$; . Counter for total number of storage locations
  if Any region edges intersect **then**
    return -1 . Infeasible design, return a negative value
  else
    for $r = 1; r \quad jRj; r = r + 1$ **do** . Create pick aisles, pick locations, and storage locations
      Find all region edges for region r and add them to region edge list RE_r
      FillRegion($r; A_r; H_r; V_r$)
    end for
    for $r = 1; r \quad jRj; r = r + 1$ **do**
      for $p = 1; p \quad jP_rj; p = p + 1$ **do** . P_r is the set of pick aisles in region r
        for $k = 1; k \quad jK_r^pj; k = k + 1$ **do** . K_r^p is the set of pick locations in pth pick aisle in region r
          if SL_{rpk} exists **then** . If storage location on the left of pick location exists
            $NL = NL + 1$. Increase number of storage locations counter
          end if
          if SR_{rpk} exists **then** . If storage location on the right of pick location exists
            $NL = NL + 1$. Increase number of storage locations counter
          end if
        end for
      end for
    end for
  end if
  return NL
end function

Algorithm A.2 Detailed Pseudo-code of the Algorithm that Creates Corner Nodes and Edges

function CreateCornerEdges(CAW,WW,WD)

$n6_1^x = \frac{CAW}{2}$. Top left corner x-axis

$n6_1^y = \frac{CAW}{2}$. Top left corner y-axis

$n6_2^x = WW \quad \frac{CAW}{2}$. Top right corner x-axis

$n6_2^y = \frac{CAW}{2}$. Top right corner y-axis

$n6_3^x = WW \quad \frac{CAW}{2}$. Bottom right corner x-axis

$n6_3^y = WD \quad \frac{CAW}{2}$. Bottom right corner y-axis

$n6_4^x = \frac{CAW}{2}$. Bottom left corner x-axis

$n6_4^y = WD \quad \frac{CAW}{2}$. Bottom left corner y-axis

Connect $n6_1$ and $n6_2$. Top outer cross-aisle

Connect $n6_2$ and $n6_3$. Right outer cross-aisle

Connect $n6_3$ and $n6_4$. Bottom outer cross-aisle

Connect $n6_4$ and $n6_1$. Left outer cross-aisle

return Corner nodes and corner edges

end function

Algorithm A.3 Detailed Pseudo-code of Adding Exterior Nodes

```
function AddExteriorNodes(E)
    for i = 1; i < 5; i = i + 1 do
        templeft =  1
        tempright =  1
        if E_i == 0 then
            n2_i^x = n6_1^x                          . Assign it to top left corner x-axis
            n2_i^y = n6_1^y                          . Assign it to top left corner y-axis
            templocation = 0                         . Assign it to a smallest possible value
            for j = 1; j < i; j = j + 1 do           . If exists, find the closest exterior node on the left
                if E_j  0:75 and E_j < 1 then
                    if templocation < E_j then
                        templocation = E_j
                        templeft = j
                    end if
                end if
            end for
            templocation = 1                         . Assign it to a largest possible value
            for j = 1; j < i; j = j + 1 do           . If exists, find the closest exterior node on the right
                if E_j > 0 and E_j  0:25 then
                    if templocation > E_j then
                        templocation = E_j
                        tempright = j
                    end if
                end if
            end for
            if templeft >  1 and tempright >  1 then
                Disconnect n2_templeft and n6_1
                Disconnect n2_tempright and n6_1
                Connect n2_templeft and n2_i
                Connect n2_tempright and n2_i
            end if
            if templeft >  1 and tempright ==  1 then
                Disconnect n2_templeft and n6_1
                Disconnect n6_1 and n6_2
                Connect n2_templeft and n2_i
                Connect n6_2 and n2_i
            end if
            if templeft ==  1 and tempright >  1 then
                Disconnect n2_tempright and n6_1
                Disconnect n6_1 and n6_4
                Connect n2_tempright and n2_i
                Connect n6_4 and n2_i
            end if
            if templeft ==  1 and tempright ==  1 then
                Disconnect n6_1 and n6_2
                Disconnect n6_1 and n6_4
                Connect n6_4 and n2_i
                Connect n6_2 and n2_i
            end if
        end if
```

Algorithm A.3 Detailed Pseudo-code of Adding Exterior Nodes (Part 2)

```
if E_i > 0 and E_i < 0:25 then
    n2_i^x = (n6_2^x  n6_1^x)  (E_i / 0:25) + n6_1^x        . Assign x-axis somewhere on top outer aisle
    n2_i^y = n6_1^y                                          . Assign it to top left corner y-axis
    templocation = 0                                         . Assign it to a smallest possible value
    for j = 1; j < i; j = j + 1 do      . If exists, find the closest exterior node on the left
        if E_j  0 and E_j  0:25 then
            if templocation < E_j then
                templocation = E_j
                templeft = j
            end if
        end if
    end for
    templocation = 1                                         . Assign it to a largest possible value
    for j = 1; j < i; j = j + 1 do      . If exists, find the closest exterior node on the right
        if E_j  0 and E_j  0:25 then
            if templocation > E_j then
                templocation = E_j
                tempright = j
            end if
        end if
    end for
    if templeft >  1 and tempright >  1 then
        Disconnect n2_templeft and n2_tempright
        Connect n2_templeft and n2_i
        Connect n2_tempright and n2_i
    end if
    if templeft >  1 and tempright =  1 then
        Disconnect n2_templeft and n6_2
        Connect n2_templeft and n2_i
        Connect n6_2 and n2_i
    end if
    if templeft =  1 and tempright >  1 then
        Disconnect n2_tempright and n6_1
        Connect n2_tempright and n2_i
        Connect n6_1 and n2_i
    end if
    if templeft =  1 and tempright =  1 then
        Disconnect n6_1 and n6_2
        Connect n6_1 and n2_i
        Connect n6_2 and n2_i
    end if
end if
```

Algorithm A.3 Detailed Pseudo-code of Adding Exterior Nodes (Part 3)

```
if E_i == 0.25 then
    n2_i^x = n6_2^x                                  . Assign it to top right corner x-axis
    n2_i^y = n6_2^y                                  . Assign it to top right corner y-axis
    templocation = 0                                 . Assign it to a smallest possible value
    for j = 1; j < i; j = j + 1 do                   . If exists, find the closest exterior node on the left
        if E_j  0 and E_j < 0.25 then
            if templocation < E_j then
                templocation = E_j
                templeft = j
            end if
        end if
    end for
    templocation = 1                                 . Assign it to a largest possible value
    for j = 1; j < i; j = j + 1 do                   . If exists, find the closest exterior node on the right
        if E_j > 0.25 and E_j  0.5 then
            if templocation > E_j then
                templocation = E_j
                tempright = j
            end if
        end if
    end for
    if templeft >  1 and tempright >  1 then
        Disconnect n2_templeft and n6_2
        Disconnect n2_tempright and n6_2
        Connect n2_templeft and n2_i
        Connect n2_tempright and n2_i
    end if
    if templeft >  1 and tempright ==  1 then
        Disconnect n2_templeft and n6_2
        Disconnect n6_2 and n6_3
        Connect n2_templeft and n2_i
        Connect n6_3 and n2_i
    end if
    if templeft ==  1 and tempright >  1 then
        Disconnect n2_tempright and n6_2
        Disconnect n6_2 and n6_3
        Connect n2_tempright and n2_i
        Connect n6_1 and n2_i
    end if
    if templeft ==  1 and tempright ==  1 then
        Disconnect n6_1 and n6_2
        Disconnect n6_3 and n6_2
        Connect n6_1 and n2_i
        Connect n6_3 and n2_i
    end if
end if
```

Algorithm A.3 Detailed Pseudo-code of Adding Exterior Nodes (Part 4)

```
if E_i > 0.25 and E_i < 0.5 then
    n2_i^x = n6_2^x                                   . Assign it to top right corner x-axis
    n2_i^y = (n6_3^y  n6_2^y)  ((E_i − 0.25)/0.25) + n6_2^y   . Assign y-axis somewhere on right outer aisle
    templocation = 0                                  . Assign it to a smallest possible value
    for j = 1; j < i; j = j + 1 do      . If exists, find the closest exterior node on the left
        if E_j  0.25 and E_j  0.5 then
            if templocation < E_j then
                templocation = E_j
                templeft = j
            end if
        end if
    end for
    templocation = 1                                  . Assign it to a largest possible value
    for j = 1; j < i; j = j + 1 do     . If exists, find the closest exterior node on the right
        if E_j  0.25 and E_j  0.5 then
            if templocation > E_j then
                templocation = E_j
                tempright = j
            end if
        end if
    end for
    if templeft >  1 and tempright >  1 then
        Disconnect n2_templeft and n2_tempright
        Connect n2_templeft and n2_i
        Connect n2_tempright and n2_i
    end if
    if templeft >  1 and tempright =  1 then
        Disconnect n2_templeft and n6_3
        Connect n2_templeft and n2_i
        Connect n6_3 and n2_i
    end if
    if templeft =  1 and tempright >  1 then
        Disconnect n2_tempright and n6_2
        Connect n2_tempright and n2_i
        Connect n6_2 and n2_i
    end if
    if templeft =  1 and tempright =  1 then
        Disconnect n6_2 and n6_3
        Connect n6_2 and n2_i
        Connect n6_3 and n2_i
    end if
end if
```

Algorithm A.3 Detailed Pseudo-code of Adding Exterior Nodes (Part 5)

```
if E_i == 0.5 then
    n2_i^x = n6_3^x                                  . Assign it to bottom right corner x-axis
    n2_i^y = n6_3^y                                  . Assign it to bottom right corner y-axis
    templocation = 0                                 . Assign it to a smallest possible value
    for j = 1; j < i; j = j + 1 do      . If exists, find the closest exterior node on the left
        if E_j  0.25 and E_j < 0.5 then
            if templocation < E_j then
                templocation = E_j
                templeft = j
            end if
        end if
    end for
    templocation = 1                                 . Assign it to a largest possible value
    for j = 1; j < i; j = j + 1 do     . If exists, find the closest exterior node on the right
        if E_j > 0.5 and E_j  0.75 then
            if templocation > E_j then
                templocation = E_j
                tempright = j
            end if
        end if
    end for
    if templeft >  1 and tempright >  1 then
        Disconnect n2_templeft and n6_3
        Disconnect n2_tempright and n6_3
        Connect n2_templeft and n2_i
        Connect n2_tempright and n2_i
    end if
    if templeft >  1 and tempright ==  1 then
        Disconnect n2_templeft and n6_3
        Disconnect n6_4 and n6_3
        Connect n2_templeft and n2_i
        Connect n6_4 and n2_i
    end if
    if templeft ==  1 and tempright >  1 then
        Disconnect n2_tempright and n6_3
        Disconnect n6_2 and n6_3
        Connect n2_tempright and n2_i
        Connect n6_2 and n2_i
    end if
    if templeft ==  1 and tempright ==  1 then
        Disconnect n6_2 and n6_3
        Disconnect n6_3 and n6_4
        Connect n6_2 and n2_i
        Connect n6_4 and n2_i
    end if
end if
```

Algorithm A.3 Detailed Pseudo-code of Adding Exterior Nodes (Part 6)

```
if E_i > 0.5 and E_i < 0.75 then
    n2_i^x = n6_3^x   (n6_3^x   n6_4^x)   ((E_i − 0.5)/0.25)   . Assign x-axis somewhere on bottom outer aisle
    n2_i^y = n6_3^y                                            . Assign it to bottom right corner y-axis
    templocation = 0                                           . Assign it to a smallest possible value
    for j = 1; j < i; j = j + 1 do      . If exists, find the closest exterior node on the left
        if E_j   0.5 and E_j   0.75 then
            if templocation < E_j then
                templocation = E_j
                templeft = j
            end if
        end if
    end for
    templocation = 1                                           . Assign it to a largest possible value
    for j = 1; j < i; j = j + 1 do      . If exists, find the closest exterior node on the right
        if E_j   0.5 and E_j   0.75 then
            if templocation > E_j then
                templocation = E_j
                tempright = j
            end if
        end if
    end for
    if templeft >   1 and tempright >   1 then
        Disconnect n2_templeft and n2_tempright
        Connect n2_templeft and n2_i
        Connect n2_tempright and n2_i
    end if
    if templeft >   1 and tempright =   1 then
        Disconnect n2_templeft and n6_4
        Connect n2_templeft and n2_i
        Connect n6_4 and n2_i
    end if
    if templeft =   1 and tempright >   1 then
        Disconnect n2_tempright and n6_3
        Connect n2_tempright and n2_i
        Connect n6_3 and n2_i
    end if
    if templeft =   1 and tempright =   1 then
        Disconnect n6_3 and n6_4
        Connect n6_3 and n2_i
        Connect n6_4 and n2_i
    end if
end if
```

Algorithm A.3 Detailed Pseudo-code of Adding Exterior Nodes (Part 7)

```
if E_i == 0.75 then
    n2_i^x = n6_4^x                                   . Assign it to bottom left corner x-axis
    n2_i^y = n6_4^y                                   . Assign it to bottom left corner y-axis
    templocation = 0                                  . Assign it to a smallest possible value
    for j = 1; j < i; j = j + 1 do      . If exists, find the closest exterior node on the left
        if E_j  0.5 and E_j < 0.75 then
            if templocation < E_j then
                templocation = E_j
                templeft = j
            end if
        end if
    end for
    templocation = 1                                  . Assign it to a largest possible value
    for j = 1; j < i; j = j + 1 do     . If exists, find the closest exterior node on the right
        if E_j > 0.75 and E_j < 1 then
            if templocation > E_j then
                templocation = E_j
                tempright = j
            end if
        end if
    end for
    if tempright ==  1 then      . Check if closest on the right is on the top left corner
        for j = 1; j < i; j = j + 1 do
            if E_j == 0 then
                tempright = j
            end if
        end for
    end if
    if templeft >  1 and tempright >  1 then
        Disconnect n2_templeft and n6_4
        Disconnect n2_tempright and n6_4
        Connect n2_templeft and n2_i
        Connect n2_tempright and n2_i
    end if
    if templeft >  1 and tempright ==  1 then
        Disconnect n2_templeft and n6_4
        Disconnect n6_4 and n6_1
        Connect n2_templeft and n2_i
        Connect n6_1 and n2_i
    end if
    if templeft ==  1 and tempright >  1 then
        Disconnect n2_tempright and n6_4
        Disconnect n6_3 and n6_4
        Connect n2_tempright and n2_i
        Connect n6_3 and n2_i
    end if
    if templeft ==  1 and tempright ==  1 then
        Disconnect n6_1 and n6_4
        Disconnect n6_3 and n6_4
        Connect n6_4 and n2_i
        Connect n6_1 and n2_i
    end if
end if
```

Algorithm A.3 Detailed Pseudo-code of Adding Exterior Nodes (Part 8)

```
        if E_i > 0.75 and E_i < 0.1 then
            n2_i^x = n6_4^x                                   . Assign it to bottom left corner x-axis
            n2_i^y = n6_4^y   (n6_4^y   n6_1^y)   ((E_i − 0.75)/0.25)   . Assign y-axis somewhere on left outer aisle
            templocation = 0                                  . Assign it to a smallest possible value
            for j = 1; j < i; j = j + 1 do      . If exists, find the closest exterior node on the left
                if E_j   0.75 and E_j < 1 then
                    if templocation < E_j then
                        templocation = E_j
                        templeft = j
                    end if
                end if
            end for
            templocation = 1                                  . Assign it to a largest possible value
            for j = 1; j < i; j = j + 1 do    . If exists, find the closest exterior node on the right
                if E_j   0.75 and E_j < 1 then
                    if templocation > E_j then
                        templocation = E_j
                        tempright = j
                    end if
                end if
            end for
            if tempright ==   1 then     . Check if closest on the right is on the top left corner
                for j = 1; j < i; j = j + 1 do
                    if E_j == 0 then
                        tempright = j
                    end if
                end for
            end if
            if templeft >   1 and tempright >   1 then
                Disconnect n2_templeft and n2_tempright
                Connect n2_templeft and n2_i
                Connect n2_tempright and n2_i
            end if
            if templeft >   1 and tempright =   1 then
                Disconnect n2_templeft and n6_1
                Connect n2_templeft and n2_i
                Connect n6_1 and n2_i
            end if
            if templeft =   1 and tempright >   1 then
                Disconnect n2_tempright and n6_4
                Connect n2_tempright and n2_i
                Connect n6_4 and n2_i
            end if
            if templeft =   1 and tempright =   1 then
                Disconnect n6_4 and n6_1
                Connect n6_4 and n2_i
                Connect n6_1 and n2_i
            end if
        end if
    end for
end function
```

Algorithm A.4 Detailed Pseudo-code of Adding an Interior Node

function AddInteriodNode($X; Y$) . y-axis increases by going downwards

$n3^x = (n6_2^x \quad n6_1^x) \quad X + n6_1^x$. Assign x-axis for interior node

$n3^y = (n6_3^y \quad n6_2^y) \quad Y + n6_2^y$. Assign y-axis for interior node

end function

Algorithm A.5 Detailed Pseudo-code of Populating a Region with Pick Aisles, Pick Locations, and Storage Locations

function FillSingleRegion($pR; pA; pH; pV$) . pR is region index,pA is the angle of the region, pH is the horizontal adjuster parameter, pV is the vertical adjuster parameter
  $bx = 0$. base coordinate x-axis (used for rotation)
  $by = 0$. base coordinate y-axis (used for rotation)
  for $i = 1; i$ $jed2_{pR}j; i = i + 1$ **do** . $ed2_{pR}$ is set of region edges in region pR
    Rotate $ed2^i_{pR}$ by angle pA . Rotate them to create angled pick aisles
  end for
  $minx = M$. M is a big number
  $miny = M$
  $maxx = M$
  $maxy = M$
  . Find the bounding box (circumscribed rectangle) of the region
  for $i = 1; i$ $jed2_{pR}j; i = i + 1$ **do**
    if $ed2sx^i_{pR} > maxx$ **then** . $ed2sx^i_{pR}$ is the starting node x-axis for region edge i
      $maxx = ed2x^i_{pR}$. Update bounding edges' largest x-axis
    end if
    if $ed2sy^i_{pR} > maxy$ **then** . $ed2sy^i_{pR}$ is the starting node y-axis for region edge i
      $maxy = ed2sy^i_{pR}$. Update bounding edges' largest y-axis
    end if
    if $ed2ex^i_{pR} > maxx$ **then** . $ed2ex^i_{pR}$ is the ending node x-axis for region edge i
      $maxx = ed2x^i_{pR}$. Update bounding edges' largest x-axis
    end if
    if $ed2ey^i_{pR} > maxy$ **then** . $ed2ey^i_{pR}$ is the ending node y-axis for region edge i
      $maxy = ed2y^i_{pR}$. Update bounding edges' largest y-axis
    end if
    if $ed2sx^i_{pR} < minx$ **then** . $ed2sx^i_{pR}$ is the starting node x-axis for region edge i
      $minx = ed2x^i_{pR}$. Update bounding edges' smallest x-axis
    end if
    if $ed2sy^i_{pR} < miny$ **then** . $ed2sy^i_{pR}$ is the starting node y-axis for region edge i
      $miny = ed2sy^i_{pR}$. Update bounding edges' smallest y-axis
    end if
    if $ed2ex^i_{pR} < minx$ **then** . $ed2ex^i_{pR}$ is the ending node x-axis for region edge i
      $minx = ed2x^i_{pR}$. Update bounding edges' smallest x-axis
    end if
    if $ed2ey^i_{pR} < miny$ **then** . $ed2ey^i_{pR}$ is the ending node y-axis for region edge i
      $miny = ed2y^i_{pR}$. Update bounding edges' smallest y-axis
    end if
  end for
  $gap = PAW + 2$ SLD
  $i = 0$
  . Add angled pick aisles considering horizontal adjuster
  while $miny + i$ $gap + pH$ gap $maxy$ **do**
    $startx = minx$. Starting x coordinate for temporary edge
    $starty = miny + i$ $gap + pH$ gap . Starting y coordinate for temporary edge
    $endx = maxx$. Ending x coordinate for temporary edge
    $endy = miny + i$ $gap + pH$ gap . Ending y coordinate for temporary edge
    Create a temporary edge $tempedge$ for above coordinates
    Rotate back temporary edge by angle pA
    Create a temporary set of intersection nodes $TEMPNODES$

Algorithm A.5 Detailed Pseudo-code of Populating a Region with Pick Aisles, Pick Locations, and Storage Locations (Part 2)

for $j = 1; j$ *jed*$2_{pR}j; j = j + 1$ **do** . Check how many times *tempedge* intersects with the region edges for region pR
if *tempedge* intersects with *ed*2^{j}_{pR} **then**
Add intersection point to set $TEMPNODES$
end if
end for
if $jTEMPNODESj == 2$ **then** . Two intersections means only one pick aisle
Create temporary pick aisle with $TEMPNODES_1$ & $TEMPNODES_2$. $TEMPNODES_1$ is the first element of the set
if AddPickAisleEdge($TEMPNODES_1; TEMPNODES_2; pR; pA; pV$) == true **then**
Add this temporary pick aisle to P_{pR} set . $PICKAISLE_{pR}$ set includes all the pick aisles in region pR
end if
end if
if $jTEMPNODESj == 4$ **then** . Four intersections means two pick aisles
Create temporary pick aisle with $TEMPNODES_1$ & $TEMPNODES_2$
if AddPickAisleEdge($TEMPNODES_1; TEMPNODES_2; pR; pA; pV$) == true **then**
Add this temporary pick aisle to P_{pR} set
end if
Create temporary pick aisle with $TEMPNODES_3$ & $TEMPNODES_4$
if AddPickAisleEdge($TEMPNODES_3; TEMPNODES_4; pR; pA; pV$) == true **then**
Add this temporary pick aisle to P_{pR} set
end if
end if
. We don't need to check for 6 intersection points because we only have one interior node, it can only create two picking aisle edges from a single *tempedge*
$i = i + 1$
end while
end function

Algorithm A.6 Detailed Pseudo-code of Adding a Picking Aisle Inside a Region

function AddPickAisleEdge($tempnode1; tempnode2; pR; pA; pV; p$) . y-axis increases by going downwards

$XS = tempnode1^x$. Assign x-axis for interior node

$XE = tempnode2^x$

$YS = tempnode1^y$

$YE = tempnode2^y$. Assign y-axis for interior node

$lengthofaisle = \sqrt{(XS - XE)^2 + (YS - YE)^2}$

$numberofpicklocations = d lengthofaisle/SLWe$

$k = 1$

$incx = (XE - XS)/(lengthofaisle/SLW)$

$incy = (YE - YS)/(lengthofaisle/SLW)$

$PLX = XS + incx * pV$. Shift pick location vertically

$PLY = YS + incy * pV$. Shift pick location vertically

for $i = 1; i < numberofpicklocations; i = i + 1$ **do**

. Calculate four corners of left storage location below

$XI1 = PLX - incx/2 * (PAW/2 + SLD) * \cos pA$

$YI1 = PLY - incy/2 * (PAW/2 + SLD) * \sin pA$

$XI2 = PLX - incx/2 * (PAW/2) * \cos pA$

$YI2 = PLY - incy/2 * (PAW/2) * \sin pA$

$XI3 = PLX + incx/2 * (PAW/2 + SLD) * \cos pA$

$YI3 = PLY + incy/2 * (PAW/2 + SLD) * \sin pA$

$XI4 = PLX + incx/2 * (PAW/2) * \cos pA$

$YI4 = PLY + incy/2 * (PAW/2) * \sin pA$

. Check if the left storage location is completely inside the region without touching any cross aisles (region edges) surrounding the region pR

if All four corners are inside the region pR **then**

Add the storage location's pick location to set K^p_{pR} . Set K^p_{pR} only contains the pick locations

Add the storage location and its four corner coordinates to object SL_{pRpk} . Object SL_{pRpk} contains all left side storage location information for region pR's pth pick aisle

$k = k + 1$

end if

. Calculate the four corners of right storage location below

$XR1 = PLX - incx/2 * (PAW/2) * \cos pA$

$YR1 = PLY - incy/2 * (PAW/2) * \sin pA$

$XR2 = PLX - incx/2 * (PAW/2 + SLD) * \cos pA$

$YR2 = PLY - incy/2 * (PAW/2 + SLD) * \sin pA$

$XR3 = PLX + incx/2 * (PAW/2) * \cos pA$

$YR3 = PLY + incy/2 * (PAW/2) * \sin pA$

$XR4 = PLX + incx/2 * (PAW/2 + SLD) * \cos pA$

$YR4 = PLY + incy/2 * (PAW/2 + SLD) * \sin pA$

. Check if the right storage location is completely inside the region without touching any cross aisles (region edges) surrounding the region pR

if All four corners are inside the region pR **then**

Add the storage location's pick location to set K^p_{pR} . Set K^p_{pR} only contains the pick locations

Add the storage location and its four corner coordinates to object SR_{pRpk} . Object SR_{pRpk} contains all right side storage location information for region pR's pth pick aisle

$k = k + 1$

end if

PLX = PLX + incx

PLY = PLY + incy

end for

Algorithm A.6 Detailed Pseudo-code of Adding a Picking Aisle Inside a Region (Part 2)

```
    if k > 1 then                . At least one storage location has been created in this pick aisle
        return true
    else
        return false
    end if
end function
```

Algorithm A.7 Detailed Pseudo-code of the Algorithm that Calculates the Distances Between Locations

function LOCATIONDISTANCES(X,Y,E,PD,A,H,V,C,WA,AR,SLW,SLD,CAW,PAW)
  $WW = \sqrt{\frac{WA}{AR}}$
  $WD = \sqrt{WA \quad AR}$
  CreateCornerEdges(CAW,WW,WD)
  AddExteriorNodes(E)
  AddInteriorNode($X;Y$)
  AddPickandDepositNode(PD) . This function is similar to AddExteriorNodes function
  Connect all exterior and interior nodes based on C and add them as region edges
  $count = 0$;
  if Any region edges intersect **then**
    return NULL . Infeasible design, return a null value
  else
    for $r = 1; r \quad jRj; r = r + 1$ **do** . Create pick aisles, pick locations, and storage locations
      Find all region edges for region r and add them to region edge list RE_r
      FillRegion($r; A_r; H_r; V_r$)
    end for
    Add pick and deposit node to set G . Set G contains all visibility graph nodes, first element in G is pick and deposit node
    Add all pick location nodes to set G
    Add all storage location nodes to set G
    for $m = 1; m \quad jGj; m = m + 1$ **do**
      for $n = 1; n \quad jGj; n = n + 1$ **do**
        if mth node is visible to nth node **then** . Check if any storage location is blocking the visibility between two nodes
          Connect these two nodes
        end if
      end for
    end for
    Calculate all-pairs shortest distances in set G using Djikstra's Algorithm and store the distances in VD_{mn} . VD_{mn} stores the shortest path distances between two nodes
    $i = 0$
    $j = 0$
    for $m = 1; m \quad jGj; m = m + 1$ **do**
      if mth node is a pick and deposit node or a pick location node **then**
        $i = i + 1$
      end if
      for $n = 1; n \quad jGj; n = n + 1$ **do**
        if mth node is a pick and deposit node or a pick location node **then**
          $j = j + 1$
          $D_{ij} = VD_{mn}$. Assign distance between locations i and j
        end if
      end for
    end for
  end if
  return Distance Matrix D
end function

Appendix B. Example Pick List Data

Table B.2: Example of pick list data

Pick List ID	SKU Number
554468267	5161503
554468267	5161484
554468267	5161233
554468267	5138240
554468267	5161479
554468267	5161508
554468267	5161509
554468267	5161500
554468267	5162844
554468267	5161514
554468267	5151474
554468267	5161880
554468267	5162903
554468267	5161274
554468267	5161506
554468267	5162856
554468267	5130474
554469595	5140361
554469595	5058449
554469595	5127923
554469595	5062709
554469595	5138570
554469595	8551077
554469595	5124380
554469621	5136796
554469621	5136575
554469621	8661263
554469636	5157448
554469636	8681215
554469636	5140634
554469636	8681340
554469636	5142120
554469636	5117450
554469636	5014508
554469636	5011400
554469636	5011292
554469636	5017113
554469636	5125939
554469636	8681056
554469636	5158308
554469636	5012778
554469636	5127696
554469636	5156275
554469636	5126655
554469636	5117448
554469636	8681002
554469636	5158113
554469636	8681264

Appendix C. Explanation of the Region Finding Algorithm

GABAK has an algorithm that finds the closed regions in a plane (and labels them as Region 1, Region 2, Region 3, etc.). However, the algorithm does not work in a clockwise or a counter-clockwise manner. Therefore, we cannot label regions in a straightforward way. We explain this with an example below.

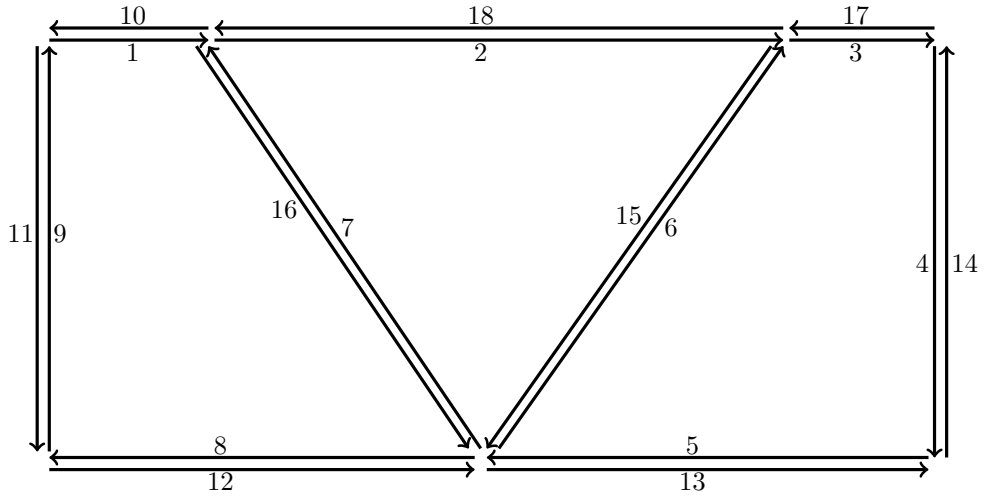

Figure C.1: A depiction of directed edges

Figure C.1 is an example of a 3-0-2 design. Three exterior nodes are located at $E_1 = 0.0416$ (top left end of the cross aisle), $E_2 = 0.2083$ (top right end of the cross aisle), and $E_3 = 0.625$. E_1 and E_3 are connected forming a cross aisle (the C_{13} variable is set to 1). E_2 and E_3 are also connected forming another cross aisle (the C_{23} variable is set to 1). The region finding algorithm finds three regions in the following way:

All region edges are identified from the definitions of the exterior nodes and interior nodes and their connections with cross aisles. The region finding algorithm creates two directed edges from these undirected region edges. The direction of each edge is arbitrary and does not change the number of regions but may change the order of the labeling. We add all positive and negative directed edges to a list of unvisited edges. Then we pick one unvisited directed region edge (the starting edge changes the order of the labeling as well, and we always pick the first one in the unvisited list) but it is not yet marked as visited (i.e., it is not removed from the list). We start to visit the rightmost unvisited directed edge and mark it as visited by removing it from the unvisited list, and keep visiting to next rightmost turn unvisited directed edge until we come back to the original starting directed edge and mark the starting directed edge as visited (i.e., remove it from the unvisited list). All of the edges we visited will be removed from the list so they are not visited

again. This is a region. The algorithm continues to find all regions similarly. At the end, all edges are visited and an additional region is removed; this is the whole warehouse itself. Below shows how the algorithm calculates the regions for the above example (assuming the first directed edge in the unvisited edges list is the edge number 1 in Figure C.1).

- Starting edge is #1: 16 *!* 8 *!* 9 *!* 1 (Region #1)
- Starting edge is #2: 15 *!* 7 *!* 2 (Region #2)
- Starting edge is #3: 4 *!* 5 *!* 6 *!* 3 (Region #3)
- Starting edge is #10: 11 *!* 12 *!* 13 *!* 14 *!* 17 *!* 18 *!* 10 (Outer Region, largest area among others - the entire warehouse, is removed from the set of regions)