Minimizing Multi-zone Orders in the Correlated Storage Assignment Problem

A Thesis
Presented to
The Academic Faculty

by

Maurice Garfinkel

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

School of Industrial and Systems Engineering
Georgia Institute of Technology
January 2005

Minimizing Multi-zone Orders in the Correlated Storage Assignment Problem

Approved by:

Dr. Gunter P. Sharp, Advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Joel S. Sokol, Co-advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Shamkant B. Navathe
College of Computing
Georgia Institute of Technology

Dr. Earl R. Barnes
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. John H. Vande Vate
School of Industrial and Systems Engineering
Georgia Institute of Technology

Date Appproved: January 2005

To my parents

for all their support

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my parents for all their support during my time in graduate school. Without everything they have done for me during my time at Georgia Tech and well before, I would not have gotten to this point. I also want to thank my brother Chaim for his support.

My primary advisor, Gunter Sharp, helped me define my research problem and get me on track when my previous research ideas did not work out. Joel Sokol agreed to be my co-advisor when Gunter went abroad for much of a summer. They were both integral to this dissertation's merger of optimization methodology with warehousing. My conversations with them shaped many of the ideas in this dissertation. I really appreciate their patience, insights, and support.

My committee members, Earl Barnes, Sham Navathe, John Bartholdi, and John Vande Vate, gave me valuable feedback on my research and write up. Earl served as my initial advisor at Georgia Tech and supported me during my first years. When I suggested a research problem outside his area of expertise, he directed me to Gunter. Sham helped suggest some applications for our problem outside of warehousing. As a member of the College of Computing, he also gave us the valuable perspective of someone from outside our domain. John Bartholdi served on my reading committee and gave me valuable feedback which helped define the direction of my research. When he told me that he was unable to attend my defense, John Vande Vate agreed to take his place just a few weeks before the defense date. I really appreciate his filling in on such short notice. His comments were valuable in clarifying the description of our problem in the write up.

I also appreciate the various faculty members of ISyE for their assistance on various aspects of my research. The faculty, staff, and students of this department contributed in many ways to this dissertation.

My time in Atlanta would not have been the same without the Jewish community in

Toco Hills. I feel immense gratitude to all the families that have invited me to their homes for meals. It is not possible to mention all the friends I have made in this community during my time at Georgia Tech. The advice and encouragement of many different people has kept me going at several difficult times during my studies. Some of my most enjoyable hours over the last few years was spent playing with the children of these families.

I have had many good friends during my time at Georgia Tech. My closest friend has been David Copeland who was my roommate for three years until he went on to a more permanent roommate, his wife Amanda. I have been blessed to know have them as friends the last couple of years. More recently, playing with their son Jac has been a favorite activity when I needed a break from my research.

When I needed a change of scenery, bringing my work to some non-standard places helped me gain insights into particularly challenging problems. Many ideas in my dissertation were worked out in the various kosher restaurants that have come (and gone) over the last few years. Other insights came to me surrounded by the nature at Stone Mountain. At times, taking walks around campus helped stimulate my thought process.

Tennis has been a valuable outlet for stress relief during my studies. I appreciate all the different tennis partners who I have played with over the last couple years.

Many more people who I have not been able to thank individually have helped me get to this point. I am grateful for everyone who had a direct or indirect influence on my dissertation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

A fundamental issue in warehouse operations is the storage location of the products it contains. Placing products intelligently within the system can allow for great reductions in order pick costs. This is essential because order picking is a major cost of warehouse operations. For example, a study by Drury [2] conducted in the UK found that 63% of warehouse operating costs are due to order picking. When orders contain a single item, the COI rule of Heskett [5] is an optimal storage policy. This is not true when orders contain multiple line items because no information is used about what products are ordered together. In this situation, products that are frequently ordered together should be stored together. This is the basis of the correlated storage assignment problem.

Several previous researchers have considered how to form such clusters of products with an ultimate objective of minimizing travel time. In this dissertation, we focus on the alternate objective of minimizing multi-zone orders. We present a mathematical model and discuss properties of the problem. A Lagrangian relaxation solution approach is discussed. In addition, we both develop and adapt several heuristics from the literature to give upper bounds for the model.

A cyclic exchange improvement method is also developed. This exponential size neighborhood can be efficiently searched in polynomial time. Even for poor initial solutions, this method finds solutions which outperform the best approaches from the literature.

Different product sizes, stock splitting, and rewarehousing are problem features that our model can handle. The cyclic exchange algorithm is also modified to allow these operating modes. In particular, stock splitting is a difficult issue which most previous research in correlated storage ignores. All of our algorithms are implemented and tested on data from a functioning warehouse. For all data sets, the cyclic exchange algorithm outperforms COI, the standard industry approach, by an average of 15%.

LIST OF SYMBOLS OR ABBREVIATIONS

a	length of space in first dimension	41
b	length of space in second dimension	41
C_z	capacity of zone $z \in Z$	15
C_z^f	capacity of zone $z \in Z$ remaining after the nonfractional products have been assigned	27
c	number of clusters in Rosenwein's formulation	20
$\tilde{c}$	constant for particle heuristic	40
c_m	cost of relocating a pallet from its current zone	18
c_{ij}	correlation between products i and j	38
c^{ij}	cost of the assignment from iteration i given the locations of iteration j	43
D	set of order/product pairs	15
d_{ij}	measure of how often products i, j occur on different orders in Rosenwein's formulation	20
EZ	expected number of zones visited for all orders	36
e_z	cost of entering zone $z \in Z$	15
f_r	ratio of frequency for order r measure used in Amirhosseini's method	38
G	general cyclic swap graph	47
G^0	graph to determine permutation of zones for cyclic swap	49
G^1	cyclic swap graph given a permutation of zones	50
G^N	cyclic swap graph given a permutation of zones extended to allow null moves	58
g	location of grid point g in space	41
h	location of head of force vector in space	41
(IP_{1pal})	integer program to determine optimal product layout when there is only one pallet of each product	18

(IP_{fix})	integer program to correct the fractional variables from solving the Lagrangian relaxation of (IP_{noSS})	27
(IP_{general})	integer program to determine optimal product layout in the general case	16
(IP_{noSS})	integer program to determine optimal product layout when stock splitting is not permitted	17
$(IP_{\text{set cover}})^r$	set cover problem to determine from which zones to pick order r	66
L	set of pallets	25
LB	best lower bound of (IP_{1pal})	28
$L(\mu)$	value of (L) for a given value of μ	25
(L)	Lagrangian relaxation of (IP_{1pal})	25
(LD)	dual of the Lagrangian relaxation (L)	32
(L_x)	version of (L) after simplification	26
(L_μ)	linear program to recover values for μ given x_{pz} and y_{rz}	32
l_{rp}	zone from which product p of order r is picked	68
m_d	slope between dimensions d and $d+1$	40
N_k	number of orders containing k products	36
n_r	number of occurrences of order $r \in R$	15
n_{rz}	number of products from order r contained in zone z	45
P	set of product sku's	15
$P_{y_{\text{feas}}}$	feasibility problem to determine if a feasible assignment exists for the given y variables	29
P^f	products containing fractional assignments	27
$\Pr(\|Z\|, k, l)$	probability of using exactly l out of $\|Z\|$ zones in a k-product order	36
p_l	product sku that pallet l represents	15
$p_{j\|i}$	conditional probability that product j appears in an order given product i appears	37

q_r	number of zones visited for order r	32
q_{ij}	1 if item i assigned to cluster j in Rosenwein's formulation	20
R	set of orders	15
R_p	set of orders containing product p	45
R^f	orders containing products with fractional assignments	27
s_r	number of different products in order r	15
s_{rz}	number of products from order r that are picked from zone z	68
s^*	maximum order size	32
t	time step	40
UB	best upper bound of (IP_{1pal})	30
UB_x	upper bound of (IP_{1pal}) computed from the x variables	29
UB_y	upper bound of (IP_{1pal}) computed from the y variables	29
u_z	vertex identified with one of the nodes not being occupied by a product	58
v_p	number of pallets of product $p \in P$ that need to be stored	17
w_{ij}	weight in graph of edge between nodes i and j	37
$w_{p_1p_2}$	edge weight corresponding to the change in objective due to the replacement of p_2 in z_2 with p_1	44
w_{prz}	1 if product $p \in P$ in order r is picked from zone z; 0 otherwise	15
x_{lz}	1 if pallet $l \in L$ is stored in zone z; 0 otherwise	15
x_{pz}	1 if product $p \in P$ is stored in zone z; 0 otherwise	17
x_p^i	assignment of product p to a cluster center in iteration i	43
y_z	1 if zone z is visited to fill the order from $(IP_{\text{set cover}})^r$; 0 otherwise	66
y_{rz}	1 if to fill order $r \in R$, one must visit zone z; 0 otherwise	15
y_z^i	location of cluster center z in iteration i	43

Z	set of zones	15
Z_r^f	for fractional order $r \in R^f$, zones not already containing a product from the previous assignment that had an integral value	27
z_l	zone to which pallet l has been assigned	68
α_{lz}	1 if pallet l is relocated to/from zone z; 0 otherwise	19
ζ_i	i^{th} zone in the permutation; $\lvert\zeta_i\rvert$ = number of products stored in zone ζ_i	50
η_j	1 if item j is chosen as a cluster median in Rosenwein's formulation	20
θ	dual variables corresponding to the assignment constraints of (L)	31
θ^k	step size for iteration k in the Lagrangian relaxation algorithm	30
κ_p	location of product p in space	42
$\kappa_{i,d}$	location of product i in dimension d	41
λ^k	parameter used in the Lagrangian relaxation algorithm	30
μ_{rpz}	multipliers in L; $(r,p) \in D, z \in Z$	25
π	dual variables corresponding to the capacity constraints of (L)	31
ρ_z	residual capacity of zone z	58
σ_i	popularity of product i	36
σ_{ij}	pairwise popularity of products i and j	36
τ	test statistic for Frazelle's method	37
ϕ_{pg}	magnitude of force exerted by all other products on product p projected onto grid point g	42
χ_{lz}	1 if pallet l is currently stored in zone z; 0 otherwise	18
ψ_d	proportion of magnitude of ϕ_{pg} applied to dimension d	42
$\omega_{z_1 z_2}$	edge weight between z_1 and z_2 in G^0	49

CHAPTER 1

INTRODUCTION

Order picking is a fundamental activity in a warehouse that involves identifying and retrieving products that belong to customer orders. The time to fill an order is spent traveling to storage locations, searching for the product, physically extracting the product, merging the products, recording the retrievals, and preparing the shipment. It may be possible to do some of these activities simultaneously. With product proliferation and increased customer service expectations, reducing order fill time is crucial. A study by Drury [8] in the UK found that 63% of warehouse operating costs are due to order picking. Therefore, reducing picking costs can significantly reduce overall operating costs in a warehouse.

Typically, a warehouse has a reserve storage area and a forward pick area. Products in reserve storage are generally not retrieved frequently, or have smaller product quantities stored in the forward pick area for more efficient picking. The majority of order picking thus occurs in the forward pick area.

Clearly, the location of the products has a major effect on the time to fill orders. If the products are located so that travel time is reduced, significant reductions in operating cost may be attained. Other activities in order processing, such as physically extracting the product from its storage location and recording the pick, are typically not affected by product location.

1.1 Storage Assignment

Products may be distributed in a warehouse by category, i.e. manufacturer, or class, i.e. all hats. If a warehouse does not employ this strategy or consider order history, a first available location rule will likely be used. That is, products will be placed in the first available location. This rule may reduce the time to stock a location and will guarantee that there are not too many open locations close to the shipping docks. Furthermore, it will reduce congestion since product location is independent of frequency or correlation between products. However, if the popular products are located in the back of the facility, then order picking times will be high.

As an alternative to this policy, early attempts to reduce travel time focused on the idea that products that are frequently ordered should be located close to the shipping dock of the warehouse. Also, products with small size should be placed near the shipping dock so that as many products as possible are in the most accessible places. That way, travel is minimized to those products whose locations are most frequently visited. Under this rule, one possibly suffers increased congestion. A second idea is the cube order index (COI) rule of Heskett [14]. This rule considers both the number of orders containing a given product as well as the size of the product. That is,

$$\text{COI} = \frac{\text{Number of orders containing a product}}{\text{Space needed to store product}}$$

Products with highest COI are closest to the shipping dock. This rule can also lead to increased congestion.

The previous two ideas to reduce travel time do not always work well if products are correlated in order requests. Consider an example where each order contains two products, a "base" product and a "feature" product. Each base product has a set of feature products that may be ordered with it, one of which must be selected. The set of all feature products is disjoint from the set of all base products. Specifically, the base product is chosen from the set $\{p_1, \ldots, p_n\}$ and the feature product for p_i from the set $\{p_i^1, \ldots, p_i^m\}$ where $m > 1$. Assume that each base product is ordered equally often and that the feature product ordered with each base product is ordered with the same frequency. In this case, the base products are ordered m times more often than the feature products. Therefore, COI will place the base products together and the feature products together. In this case, significant travel is required to fill the orders. The optimal solution is to store each base product together with its feature products so that minimal travel is necessary.

For a warehouse containing k products, there are $2^k - 1$ possible clusters. Since it is unreasonable to consider all clusters explicitly except for very small values of k, we must develop other methods to cluster products frequently ordered together.

1.2 Order Picking

Warehouses may employ different strategies to do order picking [25]. The simplest approach is for one order picker to process one order at a time. Two variations on this strategy are possible. First, one may combine several orders into a batch so that more than one order is retrieved simultaneously. Performing batch picking requires a sorting operation to separate the products from the batch into individual orders. There are two alternatives for sorting: sort-while-pick and downstream sorting (Figure 1). In sort-while-pick the order picker places the products into a multi-compartment vehicle or container designed to keep items separated by order. In downstream sorting, products are sent to a downstream sorting operation that separates the products into individual orders. Determining order batches which minimize picking time is NP-Complete [12] so heuristics have been discussed in the literature, for example [13] and [22].

The second variation is to have multiple order pickers process a single order. This is most appropriate when orders have many line items. It is common for this situation to occur in a warehouse that is divided into zones where order pickers operate exclusively in a single zone. Under a warehouse with zoning, order picking may be sequential or simultaneous (Figure 2). In sequential picking, an order picker retrieves the products from his zone and passes them to the order picker in the next zone who adds the products from his zone to the order. After the last zone, the order is ready to be prepared for shipping. With simultaneous picking, order pickers in each zone pick the products from the order in their zone and send the products to a downstream sorting operation. After sorting, orders can be prepared for shipping.

Combining the possibilities of zoning and batching, we have the following different order picking environments:

No batching, no zoning: One order picker picks all the products for one order.

No batching, simultaneous zoning: The order is split into sub-orders which are distributed to the order picker in each zone to pick. The completed sub-orders are sent

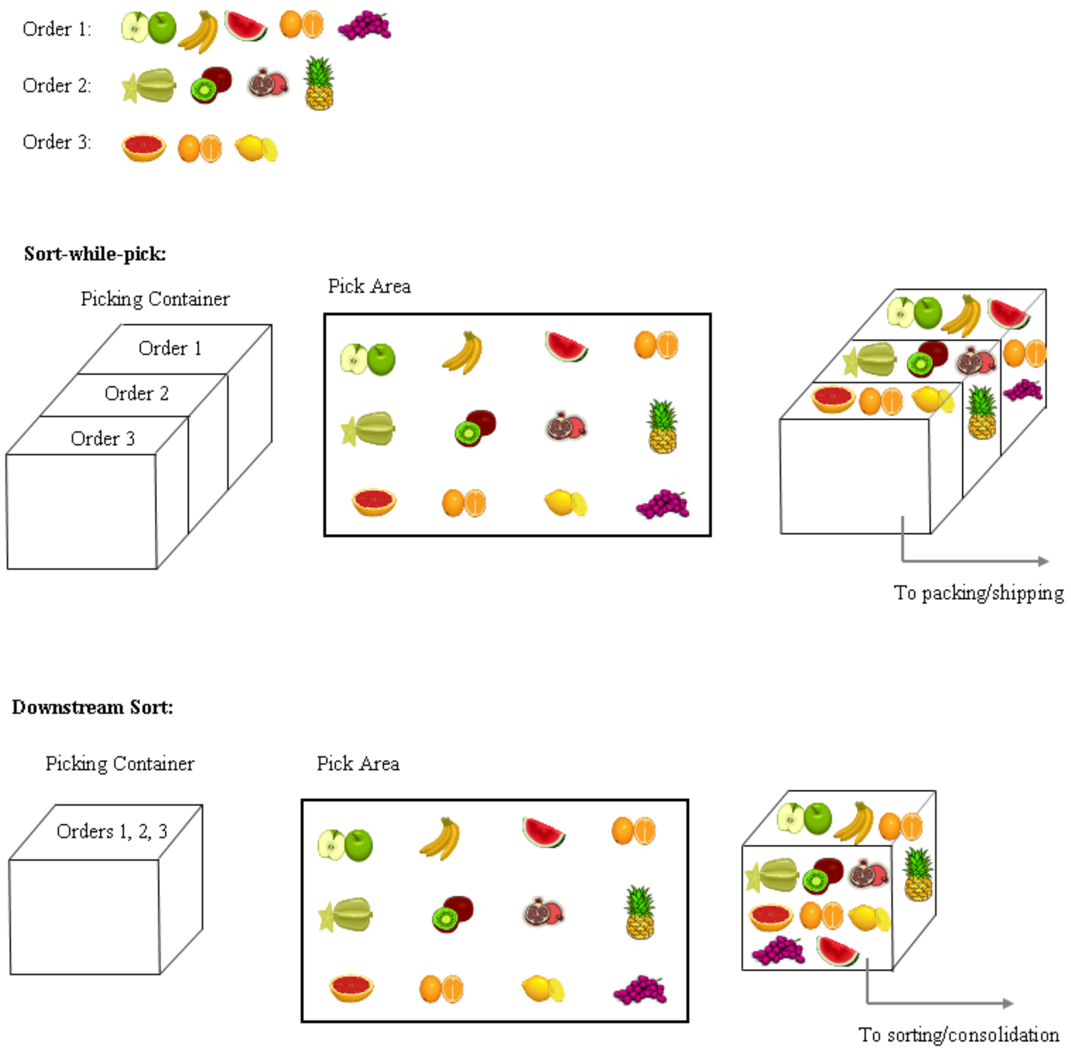

Figure 1: Sorting Alternatives for Batch Picking

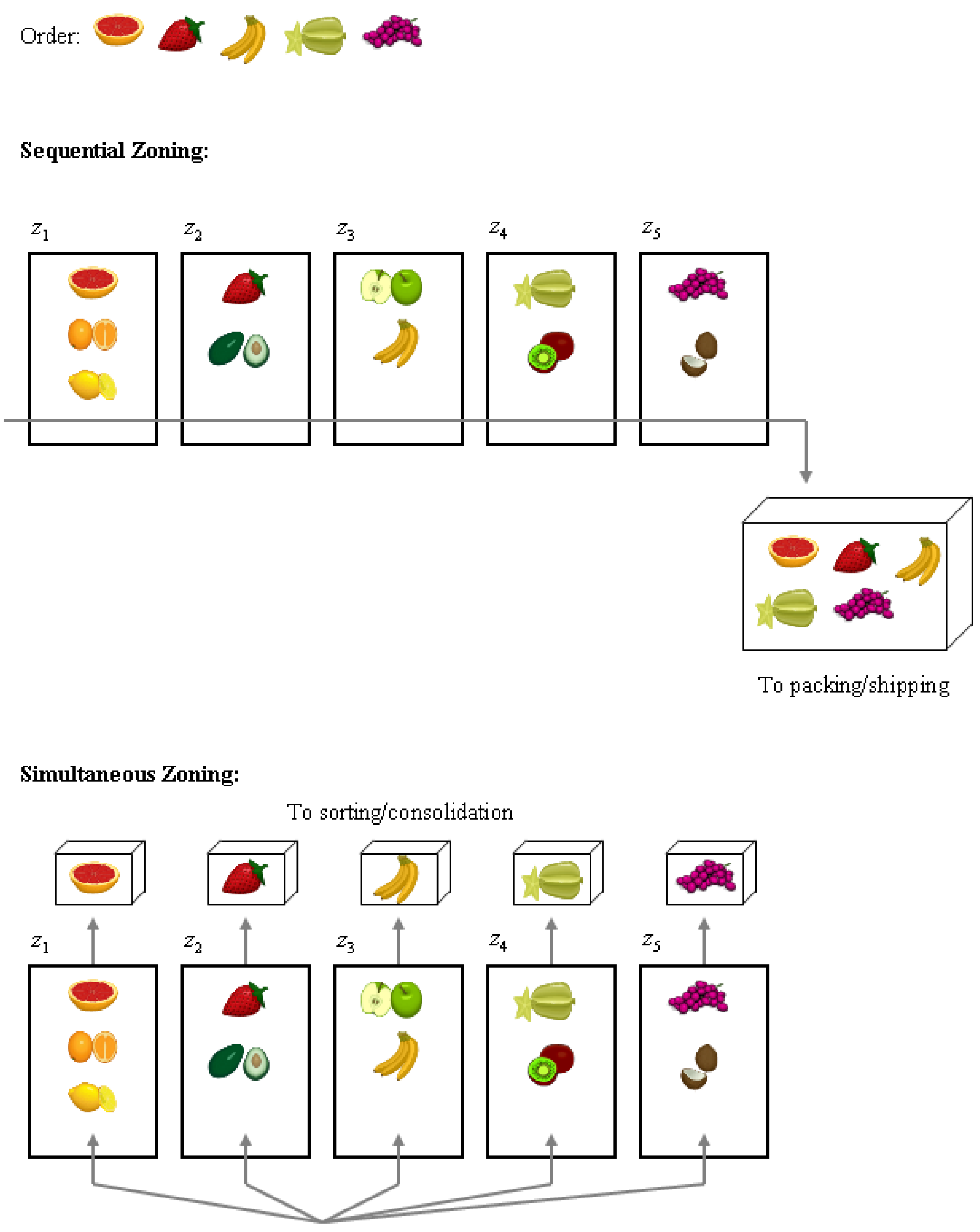

Figure 2: Alternatives for Zone Picking

to a downstream merging operation.

No batching, sequential zoning: A container for an order is passed from zone to zone. In each zone, the products from that zone are placed in the container.

Batching with downstream sorting, no zoning: A single order picker fills multiple orders and sends the products downstream, often on a conveyor, to a sorting system.

Batching with downstream sorting, simultaneous zoning: The batch is divided into sub-orders, one for each zone. Each order picker picks the products from his zone into a container and sends the container to a downstream sorting operation which sorts and merges the containers into orders.

Batching with downstream sorting, sequential zoning: A container for all the orders is passed from zone to zone. Each order picker picks the products from the batch in his zone. After the last zone, the container must go to a sorting system where the products are sorted into orders.

Batching with sort-while-pick, no zoning: One order picker fills several orders simultaneously using a vehicle or container with compartments for each order.

Batching with sort-while-pick, simultaneous zoning: The batch is divided into sub-orders, one for each zone. A vehicle or container with compartments for each order is used by the order picker in his zone. After each order picker picks the products from the batch in their zone, the containers are sent downstream so that the compartments from each zone corresponding to the same order can be merged. This order picking environment is rarely used.

Batching with sort-while-pick, sequential zoning: A vehicle or container with compartments for each order is passed from zone to zone. In each zone, products are placed into the compartment corresponding to the order(s) to which they belong. This order picking environment is rarely used.

1.3 Correlated Storage for Zone Picking

In our problem, we consider warehouses where many orders contain multiple products and batch picking is not possible or desirable. Batch picking may not be possible or desirable because orders need fast response time or because item sizes are large. Also, for our problem the pick area is divided into zones with order pickers typically operating in a single zone. The number of zones and their size is predefined. There are several reasons why order pickers may operate in a single zone. Doing so allows the picker to become familiar with the products in the zone. It also permits pickers to be trained and expert at special equipment used in a given zone. That is, the zones need not all have uniform storage equipment. One zone may have carousels, another a miniload, and a third aisles.

We require that the zones have uniform storage capabilities to the extent that any product may be stored in any zone. If products have storage incompatibilities, for example chemicals, refrigerated and non-refrigerated goods, etc. then we would consider each subset of products separately.

Another case when each order picker picks from a single zone is when the warehouse layout restricts order pickers from easily traveling between zones. For example, there may be conveyor systems which naturally divide the warehouse into zones. Also, it may be expensive to travel between zones so order pickers do not leave their zones. For example, in an environment where different products are stored in different buildings on a site, it is possible to travel between buildings, but it may be time intensive.

In environments where picking is simultaneous, each picker selects the products from the order in his zone and sends the products to a downstream sorting operation, which may be manual or mechanized. The process of merging products from different zones into the respective single orders requires human, machine, and data management resources and hence incurs overhead costs. In particular, if sorting is done manually, there may be limited space/capacity for sorting, products may not be conveyable, products may be fragile, or the sorting operation can be expensive. Additionally, all the products from each zone may be placed in a container with limited availability. Therefore, it is desirable to reduce the amount of sorting to be done by reducing the number of zones that are visited for each order. In this research we emphasize sorting workload rather than travel time. If we reduce the number of zones that must be entered, we reduce the number of product groups that must be sent to the downstream sort. Hence, it should cost less and take less time to transport all the products to the sorting station. In the case where conveyance is done mechanically, there may be little or no change in transportation cost as the number of zones visited changes.

When picking is sequential with sort-while-pick there is no downstream sorting. However, the following reasons for focusing on minimizing the number of zones visited still apply. In many systems, once a picker enters a zone, he is required to completely traverse it. This may be due to limitations on the way a picking machine operates. For example, in carousels it is common to traverse the entire carousel to retrieve products. In aisle systems, some picking vehicles are difficult to drive backwards because of the steering mechanism. The entire zone may also be traversed because multiple pickers are responsible for the zone and one-way travel is required to reduce congestion. It is also possible that the aisles are too narrow for two-way travel. It is also desirable to avoid visiting many different zones when there are expensive start-up costs associated with picking any item from a zone, for example, as in miniload and carousel systems.

In such cases, the location of each product within the zone is not as important as which products are in the zone. Travel time between picks in a zone is irrelevant when the entire zone is traversed. If one were concerned with the product placement in a zone, one could look at the products assigned to a zone and optimize the locations within the zone. This problem is not considered here.

Warehouses with multi-product orders, zone-exclusive pickers, and where batching is undesirable may occur in many different settings. For example, spare parts, automotive parts, and hardware supply warehouses often have these characteristics. In these environments, products are often too bulky for batch picking. Orders for smaller shops usually contain only a few line items. Many warehouses pick the large orders, such as those that occur for dealers, separately from the small orders. When picking the large orders, order pickers often need to visit every zone because of the order's size. For the small orders, solving our problem can provide significant cost savings. The warehouses we have described may store as many as several tens of thousands of different products and receive up to several hundreds of thousands or several million orders. As described, typical orders only have a few different line items. There are typically between twenty and forty zones.

In summary, we mainly consider the situation with zoning and no batch picking. We can also consider zoning and batch picking with downstream sorting if the batches are relatively small; in this case each batch can be treated as an order. For zoning and batch picking with sort-while-pick, it is also possible to consider our objective. The objective is to minimize the number of zones visited for all orders. This objective distinguishes our work from other related efforts which focus on reducing total travel time by placing correlated products together.

Warehouses with the following characteristics are likely candidates to benefit from optimizing according to this objective:

1. Typical orders are relatively small, i.e. 10 or fewer line items.
2. Batching is not desirable.
3. Sorting is expensive.
4. All items picked from a zone are placed in a container which is sent to the sorting station.

We can also view our problem as a clustering problem. The clustering problem mentioned previously is to group products together so that products in the same group have a high level of correlation and so that products not in the group have a low level of correlation with products in the group. In our case, we place the products in groups, one group for each zone, so that as few zones as possible need to be visited for each order. There is a very rich history of literature on the clustering problem, including the classic reference of Anderberg [5].

There are two main approaches to clustering, hierarchical and non-hierarchical. In hierarchical clustering, once a product joins a cluster it stays in that cluster. Non-hierarchical clustering allows a product to leave a cluster if it has a stronger bond with another cluster. Under hierarchical clustering, there are three main approaches: linkage, centroid, and minimum variance. Of these approaches, [23] remarks that the linkage approach is best suited for correlated storage in a warehouse. Among non-hierarchical clustering methods, the k-means method and its variants are most popular. For more discussion of cluster analysis as it relates to correlated storage in warehouses, see [23].

The classical measures used in clustering consider summary information about the relationships between products. They are fundamentally concerned with product information while we are concerned directly with order information. The product summary information used to form clusters loses crucial order information. Since we are specifically concerned with orders, the techniques we develop and analyze later work directly with this information to minimize our objective. The clustering methods address our objective only indirectly.

1.4 Literature Review

Correlated product assignment in a warehouse is a fairly new problem with a short literature history. Forming good product clusters is the focus of [18], [21], and [24]. These methods do not explicitly consider an objective related to operating costs. Minimizing travel time is the objective of [4], [10], and [15]. The objective of [24] is to minimize the number of pallets retrieved. None of these authors' clustering methods explicitly consider their respective objective. Only [23] explicitly considers cost factors when forming clusters. He includes rewarehousing and order picking costs. In this research, we minimize the alternative objective of multi-zone orders and present methods that do this explicitly.

The first treatment is by Shah [24] who discusses a problem in a miniload system where products of different sizes are stored in the same pallet. He considers the pallet assignment problem (PAP) which minimizes the expected number of pallets retrieved over a given time horizon. A particular time horizon contains several different orders as a super-order. That is, he combines correlated storage with batching. He shows that the PAP is NP-Hard if three or more products must be assigned to a pallet. Three heuristics are presented to solve this problem: one is a greedy heuristic, one is based on pairwise correlation, and the third on level of demand and correlation. The heuristics narrowly consider the products and do not consider how many pallets have to be retrieved for a given order. He briefly mentions a 2-exchange heuristic but only goes into minimal detail. His construction heuristics involve a prohibitively large amount of enumeration. No computational results are given.

Frazelle [10] formulates the stock location assignment problem (SLAP), which considers the travel time between two products in an order. Frazelle shows that the SLAP is NP-Hard and presents a heuristic solution method that computes the probabilities of the different ways that two products may occur in an order. An independence hypothesis test is then applied to filter out pairs with low correlation. He starts with the most popular product. Of all other products that are correlated, the one with the highest total correlation is added to the cluster. Products are no longer added to the cluster when a capacity constraint is violated. After clusters are generated, he places the clusters with highest total popularity closest to the shipping dock.

Sadiq [23] considers when it is worthwhile to reassign products to different locations under a dynamic system with correlated demand. His method is the only approach from the literature that explicitly considers operating costs in the cluster formation. The costs he includes are the cost associated with picking time and the rewarehousing costs. He proposes a heuristic, the dynamic stock location assignment algorithm (SLAA), that minimizes the sum of these costs. This algorithm uses demand and product forecasts, and only considers order history as a tiebreaker. The SLAA uses a hybrid clustering algorithm (HYCLUS) to determine the clusters. It is a hybrid method because first it merges products into clusters and then it considers if a product has a stronger relationship with another cluster and hence should be moved. He allows stock splitting.

Rosenwein [21] proposes a binary IP formulation to cluster products in a warehouse. He uses a branch and bound algorithm that solves a Lagrangian relaxation at each node. The proposed algorithm can be implemented efficiently. He discusses the impact of his approach on a problem containing 1,000 products, 75,000 orders, and 60 clusters.

Amirhosseini and Sharp [4] propose several correlation measures for clustering products. Among these measures is an order satisfying correlation measure (OSCM) that attempts to measure how likely the two products are to satisfy the demand of orders in which they appear. They also propose a clustering method that merges the attributes from the cluster with the new product added to it so at each stage the original cluster is nested inside the new one.

Liu [18] gives a correlation measure for products based on how often the products appear together versus the maximal order size where products appear together. Then he formulates an IP model to find the best clustering and gives a primal dual algorithm to solve it. He presents computational results for a problem containing only twenty products.

Ruben and Jacobs [22] discuss batching in settings with random storage, turnover-based storage, and family-based storage. In the family-based storage policy, "families of items are identified that are likely to appear on orders together." They do not give a method for accomplishing this.

Hua [15] considers clustering in a kitting area of a manufacturing facility. In this setting, each order can contain hundreds of different products. He uses a correlation measure based on the percentage of orders containing both products. Then, a genetic algorithm is used to determine the clusters. A cluster COI is computed and the clusters are assigned to locations along space-filling curves in increasing order of cluster COI's. He also considers adjusting the COI rule for correlation between clusters.

Our work presents an objective focused on minimizing multi-zone orders whereas previous authors' objective minimizes travel time. In addition, many previous authors' methods do not explicitly work with travel time; instead, they use one of several surrogate measures of cluster strength. We propose a Lagrangian relaxation that works explicitly with our objective. In addition, we adapt several approaches from previous authors for purposes of comparison. In addition to these construction approaches, we propose improvement algorithms that explicitly reduce the number of multi-zone orders.

Most previous authors allow products to have different sizes, but stock splitting is only permitted in [23]. In our research, we allow for both situations. Finally, we present results from a real data set with 10,644 products and almost 200,000 orders. Problem sizes reported in the literature contain no more than 1,000 products.

1.5 Outline of Thesis

In Chapter 2 we present a mathematical model for minimizing multi-zone orders in the correlated storage assignment problem and discuss properties of the problem. In Chapter 3 we present a Lagrangian relaxation of this model and discuss how to solve it. We develop heuristics in Chapter 4 to give upper bounds for the model. We also adapt heuristics from the literature which focus on different objectives for our problem. In addition, local improvement methods are developed. In Chapter 5, the model is generalized to deal with multiple pallets of storage for a single product. In this situation, stock splitting is considered and the local improvement methods are extended to allow stock splitting. Chapter 6 presents results for each algorithm discussed and discusses the relevant performance characteristics. Finally, Chapter 7 summarizes the contributions of this dissertation and discusses possible extensions and future work.

CHAPTER 2

MODEL

As discussed in Chapter 1, our objective is to minimize the total number of zones entered to fill each order. We assume that the products under consideration must be assigned to a zone. The picking area is defined to be the area containing all the zones. If some products in the warehouse are only in reserve storage and are not designated for the picking area, they will not be considered in the problem defined here.

Also, as mentioned in Section 1.3, we assume that all products under consideration can be stored in any of the zones. That is, the storage technology of each zone is compatible with each product. If a certain subset of the products could not be stored along with other products, then we would divide the problem into independent problems - one for each storage type.

We also assume that all product locations under consideration are empty so that there is no rewarehousing cost. If a storage decision required moving products to other zones, it would not be difficult to incorporate such a cost into the objective. We discuss this briefly in Section 2.4.

We use the warehouse's order history as a predictor of future orders. Only the orders representative of what the warehouse will be filling are considered. For example, in a clothing warehouse setting up for the fall season, orders from previous years' fall season would be relevant but orders from other seasons may not be included. In this dissertation, we take the order history as given. Our methodology is independent of how the order history has been determined.

2.1 General Problem

In addition to the modeling assumptions discussed in the introduction to Chapter 2, there are some generalizations that we make for our general formulation:

Generalization 1 A product may have several pallets stored in the picking area.

Generalization 2 Different pallets of the same product may be stored in different zones, i.e. stock splitting is permitted.

Following the general model, we will discuss some special cases that restrict these generalizations. The problem data is:

P	$=$ set of product skus
L	$=$ set of pallets
p_l	$=$ product sku that pallet l represents
R	$=$ set of orders
Z	$=$ set of zones
D	$=$ set of order/product pairs; that is, for each order, all the products in that order; for example $(r_1, p_1) \in D$ if product p_1 is in order r_1
n_r	$=$ number of occurrences of order $r \in R$
e_z	$=$ cost of entering zone $z \in Z$
s_r	$=$ number of different products in order r
C_z	$=$ capacity of zone $z \in Z$

Note that we measure product units in terms of pallets for convenience, but we could model product units by other measures such as cartons or individual items if desired. The variables are:

x_{lz}	$= 1$ if pallet $l \in L$ is stored in zone z; 0 otherwise
y_{rz}	$= 1$ if to fill order $r \in R$, one must visit zone z; 0 otherwise
w_{prz}	$= 1$ if product $p \in P$ in order r is picked from zone z; 0 otherwise

Given these generalizations, the general mathematical formulation is:

$$
\begin{array}{llll}
(IP_{\text{general}}) & \min & \displaystyle\sum_{r\in R}\sum_{z\in Z} n_r e_z y_{rz} & \\
& \text{s.t.} & \displaystyle\sum_{z\in Z} x_{lz} = 1 & \forall l \in L \quad (1) \\
& & \displaystyle\sum_{l\in L} x_{lz} \le C_z & \forall z \in Z \quad (2) \\
& & \displaystyle\sum_{l\in L:p_l=p} x_{lz} \ge w_{prz} & \forall (r,p) \in D, z \in Z \quad (3) \\
& & \displaystyle\sum_{z\in Z} w_{prz} \ge 1 & \forall (r,p) \in D \quad (4) \\
& & w_{prz} \le y_{rz} & \forall (r,p) \in D, z \in Z \quad (5) \\
& & x_{lz} \text{ binary} & \forall l \in L, z \in Z \\
& & w_{prz} \text{ binary} & \forall (r,p) \in D, z \in Z \\
& & y_{rz} \text{ binary} & \forall r \in R, z \in Z
\end{array}
$$

The objective minimizes the total cost of entering all zones to fill all orders. When $e_z = 1\ \forall z$, it minimizes the total number of zones entered. If some order appears multiple times, the objective reflects that. Also, we allow different zones to have different entry costs. Constraint (1) ensures that each pallet of a product is assigned to one zone. Constraint (2) ensures that more pallets are not assigned to a zone than the zone can store. Each pallet occupies the same amount of space, though each product may have a different number of pallets. We are considering an environment where each product is stored in a standard unit size. If unit sizes were non-standard, a parameter for the unit's size can easily be incorporated into this constraint. Constraint (3) allows us to pick a product for an order from a zone only if some pallet containing that product is stored in that zone. Constraint (4) forces each product in an order to be picked from some zone. Constraint (5) ties the assignment to the objective; if some product in an order is picked from a particular zone, then that zone must be visited when filling the order. Observe that this formulation has $|L| + |D| + |Z| * (1 + |D| + |R|)$ constraints and $|Z| * (|L| + |D| + |R|)$ variables. For our formulation, note that single-product orders will always be contained in exactly one zone regardless of the assignment. Hence, such orders need not be considered in the optimization. This helps reduce the problem size.

2.2 Special Case: No Stock Splitting

Here we present the formulation for the special case where Generalization 2 is restricted so that stock splitting is prevented. This situation occurs in many warehouses. In this case, a product is the basic unit of storage so we do not need to work with the set L. The simplified formulation is:

$$
\begin{array}{llll}
(IP_{\text{noSS}}) & \min & \sum_{r\in R}\sum_{z\in Z} n_r e_z y_{rz} & \\
 & \text{s.t.} & \sum_{z\in Z} x_{pz} = 1 \quad \forall p \in P & (1') \\
 & & \sum_{p\in P} v_p x_{pz} \le C_z \quad \forall z \in Z & (2') \\
 & & x_{pz} \le y_{rz} \quad \forall (r,p) \in D, z \in Z & (3') \\
 & & x_{pz} \text{ binary} \quad \forall p \in P, z \in Z & \\
 & & y_{rz} \text{ binary} \quad \forall r \in R, z \in Z &
\end{array}
$$

In this formulation:

x_{pz} = 1 if product $p \in P$ is stored in zone z; 0 otherwise

v_p = number of pallets of product $p \in P$ that need to be stored

The objective does not change in this case. Constraint (1') is identical to (1) except that the subscript $l \in L$ that has been replaced by $p \in P$. Constraint (2') has been modified from (2) to account for the difference in product storage sizes. Also, as with constraint (1'), the subscripts have been replaced. Constraint (3') replaces constraints (3)-(5) above. It relates the objective to the decision variables: for each order, if some product in that order is in a zone, then that zone must be visited when filling that order. Observe that this formulation has $|P| + |Z| + |Z| * |D|$ constraints and $|Z| * (|P| + |R|)$ variables.

2.3 Special Case: One Pallet For Each Product

This special case restricts Generalization 1 so that each product has exactly one pallet in the picking area. Generalization 2 is also restricted since stock splitting is no longer possible and the sets P and L are identical. The simplified formulation is:

$$
\begin{array}{llll}
(IP_{\text{1pal}}) & \min & \displaystyle\sum_{r\in R}\sum_{z\in Z} n_r e_z y_{rz} & \\
 & \text{s.t.} & \displaystyle\sum_{z\in Z} x_{pz} = 1 \qquad \forall p\in P & (1'') \\
 & & \displaystyle\sum_{p\in P} x_{pz} \le C_z \qquad \forall z\in Z & (2'') \\
 & & x_{pz} \le y_{rz} \qquad \forall (r,p)\in D, z\in Z & (3'') \\
 & & x_{pz} \text{ binary} \qquad \forall p\in P, z\in Z & \\
 & & y_{rz} \text{ binary} \qquad \forall r\in R, z\in Z &
\end{array}
$$

Constraints (1”) and (3”) are identical to the first special case. The v_p term from (2’) does not appear in constraint (2”) because $v_p = 1\ \forall p$ by assumption. That is, we can view this formulation as the special case of (IP_{noSS}) where $v_p = 1\ \forall p$. The formulation has $|P| + |Z| + |Z| * |D|$ constraints and $|Z| * (|P| + |R|)$ variables.

2.4 Rewarehousing Cost

We will now consider the case where pallets may already be stored in the warehouse so there is a cost associated with moving the pallets to a different location. Let us introduce the following data:

c_m = cost of relocating a pallet from its current zone

χ_{lz} = 1 if pallet l is currently stored in zone z; 0 otherwise

The parameter c_m includes the cost of labor and capital as well as the costs associated with human error such as incorrect transfers and breakage.

To incorporate the rewarehousing cost, the objective contains the following additional term:

$$\frac{1}{2}\sum_{l\in L}\sum_{z\in Z} c_m \left(x_{lz} - \chi_{lz}\right)^2$$

The term $(x_{lz} - \chi_{lz})^2$ will be equal to zero for all zones if the pallet does not change location. If the pallet changes location, this term will be equal to one for both the old and new zones, and zero for all other zones. This causes the term to be divided by two.

To avoid the quadratic term in the objective, we may introduce the following variable:

$\alpha_{lz} = 1$ if pallet l is relocated to/from zone z; 0 otherwise

The additional objective term becomes:

$$\frac{1}{2}\sum_{l\in L}\sum_{z\in Z} c_m \alpha_{lz}$$

In this case we need to add the following constraints:

$$\begin{aligned}
&\alpha_{lz} \geq x_{lz} - \chi_{lz} && \forall l \in L,\ \forall z \in Z && (6)\\
&\alpha_{lz} \geq \chi_{lz} - x_{lz} && \forall l \in L,\ \forall z \in Z && (7)\\
&\alpha_{lz} \text{ binary} && \forall l \in L,\ \forall z \in Z &&
\end{aligned}$$

Since we are minimizing, α_{lz} will be set to zero if possible. When $x_{lz} = \chi_{lz}$, this is possible. If $x_{lz} \neq \chi_{lz}$, the right hand side of either Constraint (6) or (7) will be equal to 1. The other constraint's right hand side will be equal to -1, so α_{lz} will be set to one. Hence these constraints ensure that α_{lz} is defined correctly. Observe that we must add $2|L| * |Z|$ constraints and $|L| * |Z|$ variables to the formulation to eliminate the use of a quadratic term.

In our discussion we have used pallets as the units of storage. In the special cases where we use products instead, the model extension is still valid. The only change is that the subscripts $l \in L$ become $p \in P$.

2.5 Review of Key Modeling Elements

We will now give a summary of important modeling details implicit to our formulations:

1. Any product can be stored in any location
2. Products come in standard container sizes (i.e. pallets, totes, etc.)
3. Replenishment costs are not considered
4. Cost to enter a zone is a fixed value independent of the number of stops in the zone (do not consider travel within zone)
5. Capacity of pick vehicle is sufficient to retrieve all products in the order from the zone in one visit

6. No restriction on which products can be put in the same zone
7. Stock splitting is into discrete, predefined units of storage

2.6 Cluster Strength Formulation

For comparison purposes, we now present Rosenwein's model [21], a typical formulation to optimize cluster strength:

$$\begin{array}{lll} \min & \sum_{i \in P, j \in P} d_{ij} q_{ij} & \\ \text{s.t.} & \sum_{j \in P} q_{ij} = 1 & \forall i \in P \\ & \sum_{j \in P} \eta_j = c & \\ & q_{ij} \le \eta_j & \forall i, j \in P \\ & q_{ij} \text{ binary} & \forall i, j \in P \\ & \eta_j \text{ binary} & \forall j \in P \end{array}$$

where:

q_{ij} = 1 if item i assigned to cluster j

η_j = 1 if item j is chosen as a cluster "median"

d_{ij} = measure of how often products i, j occur on different orders

c = number of clusters

Observe that this formulation has $|P|^2 + |P|$ variables and $|P|^2 + |P| + 1$ constraints. Also observe that it looks similar to (IP_{1pal}). The objective is to "select c items as medians such that the sum of distances from all items to their respective median is minimized" [21]. While d_{ij} takes into consideration how often pairs of items are ordered together, the objective does not look at how many clusters an order would have to visit to be filled. Although these are similar objectives, they are different.

2.7 Difficulty of Problem

As mentioned in Chapter 1, typical systems have between twenty and forty zones, tens of thousands of products, and several hundreds of thousands to millions of orders. Even after eliminating single product orders, the formulation ($IP_{1\text{pal}}$) has millions of constraints and millions of variables. Hence direct solution does not seem promising.

It is even worse than that. Previous authors have shown that similar problems are NP-Hard. Shah [24] shows that the PAP is NP-Hard if three or more products must be assigned to a pallet. Frazelle [10] also shows that the SLAP is NP-Hard. Now we will show that our problem is NP-Complete. Let us state the decision version for the special case of this problem where every order contains exactly two products and $e_z = 1\ \forall z$:

> MULTIZONE 2-ORDERS
>
> Given a set P of products, R of orders, Z of zones, zone capacity p, and an integer k, is there an assignment of products to zones with less than k total zone visits to fill all orders?

The 2 in the problem name indicates that each order has two products. Omitting the 2 will denote the general decision problem. Let us also state the following problem:

> GRAPH PARTITIONING
>
> Given a graph $G = (V, E)$, m subsets, maximum subset size j, and an integer l, is there a partition of vertices into m subsets of size at most j with less than l edges going between subsets?

Theorem 2.1 *MULTIZONE ORDERS is NP-Complete*

Proof We will show that the special case MULTIZONE 2-ORDERS is NP-Complete. It is straightforward to see that MULTIZONE 2-ORDERS is in NP. Given an assignment of products to zones do the following: for every order with both products in the same zone, add one to the objective function; for all other orders, add two to the objective function. Sum over all orders and verify if the total is less than k.

It remains to give a polynomial reduction. Hyafil and Rivest [16] show that GRAPH PARTITIONING is NP-Complete. Let each vertex be a product ($V \to P$), each edge $e = (u, v)$ represent an order containing products u and v ($E \to R$), each subset be a zone ($m \to Z$), and maximum subset size be the zone capacity ($j \to p$). It is clear that there is an

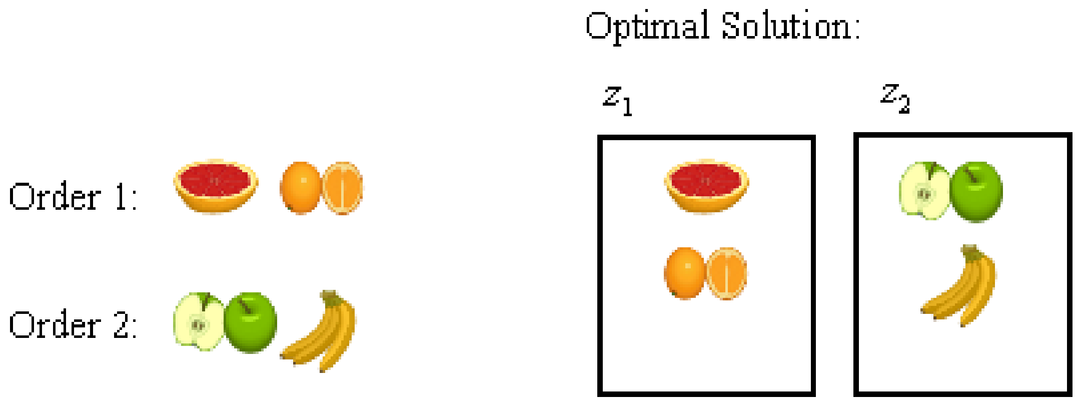

Figure 3: Small Example

assignment of products to zones with less than k total zone visits for all orders if and only if there is a partition of vertices into subsets with at most l edges between subsets. Therefore MULTIZONE 2-ORDERS is NP-Complete and consequently MULTIZONE ORDERS is as well. □

This proof focused on the special case of (IP_{1pal}) where $e_z = 1 \ \forall z$. By extension, this result demonstrates that (IP_{1pal}) and its two generalizations, (IP_{noSS}) and (IP_{general}), are NP-Complete as well.

2.8 Small Example

To illustrate our formulation, we present the following very small example (Figure 3). There are two orders each of which contain two distinct products and occur once. There are two zones each with capacity two and entry cost one. That is, $|R| = 2, |P| = 4, |Z| = 2, C_z = 2, e_z = 1 \ \forall z \in Z, n_r = 1 \ \forall r \in R$. The formulation is:

$$
\begin{aligned}
\min \quad & y_{11} + y_{12} + y_{21} + y_{22} \\
\text{s.t.} \quad & x_{11} + x_{12} = 1 \\
& x_{21} + x_{22} = 1 \\
& x_{31} + x_{32} = 1 \\
& x_{41} + x_{42} = 1 \\
& x_{11} + x_{21} + x_{31} + x_{41} \leq 1 \\
& x_{12} + x_{22} + x_{32} + x_{42} \leq 1 \\
& x_{11} \leq y_{11} \\
& x_{12} \leq y_{12} \\
& x_{21} \leq y_{11} \\
& x_{22} \leq y_{12} \\
& x_{31} \leq y_{21} \\
& x_{32} \leq y_{22} \\
& x_{41} \leq y_{21} \\
& x_{42} \leq y_{22} \\
& x_{pz} \text{ binary } \forall p \in P, z \in Z \\
& y_{rz} \text{ binary } \forall r \in R, z \in Z
\end{aligned}
$$

The first four constraints belong to the family (1"), the next two to the constraint family (2"), and the last eight to (3"). Note that the optimal solution is for $x_{11} = x_{21} = x_{32} = x_{42} = y_{11} = y_{22} = 1$ and for all other variables to be equal to zero.

2.9 Bounds on the Solution Quality

When each order occurs once, $n_r = 1 \ \forall r$, and each zone has unit entry cost, $e_z = 1 \ \forall z$, a trivial lower bound to (IP_{general}) is $|R|$. This corresponds to the assignment where for every order, all of its products are in a single zone. The trivial upper bound is $|D|$. This upper bound is based on an assignment where every product in an order is in a different zone. Such an assignment may not actually be possible but certainly one can do no worse than this. In fact this upper bound is asymptotically tight as shown in the following example:

Every order has exactly two products, one of which is product 1. The other product is different for each order. Let $v_p = 1\ \forall p$ and let $C_z = 2\ \forall z$. In this case $|D| = 2|R|$. The optimal assignment has product 1 and some other product in zone 1, and two products never ordered together in each other zone. Hence the optimal value is $2|R| - 1$.

CHAPTER 3

LAGRANGIAN RELAXATION APPROACH

3.1 Lagrangian Relaxation

The difficulty with (IP$_{\mathrm{noSS}}$) lies in the constraints that relate the objective to the assignment variables. Therefore, our idea is to consider the Lagrangian relaxation which drops these constraints explicitly. The following is the formulation of this Lagrangian relaxation problem:

$$
\begin{array}{llllll}
(L) & \max_{\mu\geq 0}\{\ L(\mu) = & \min_{x,y} & \displaystyle\sum_{r\in R}\sum_{z\in Z} n_r e_z y_{rz} + \sum_{(r,p)\in D}\sum_{z\in Z}\mu_{rpz}(x_{pz}-y_{rz}) & & \\
 & & \text{s.t.} & \displaystyle\sum_{z\in Z} x_{pz} = 1 & \forall p\in P & (1)\\
 & & & \displaystyle\sum_{p\in P} v_p x_{pz} \leq C_z & \forall z\in Z & (2)\\
 & & & x_{pz}\ \text{binary} & \forall p\in P, z\in Z & \\
 & & & y_{rz}\ \text{binary} & \forall r\in R, z\in Z\ \ \} &
\end{array}
$$

where μ_{rpz} $\forall (r,p)\in D, z\in Z$ are the Lagrange multipliers. Let us denote by $L(\mu)$ the value of the above problem for a given μ. Also, let $v^*(IP_{\mathrm{noSS}})$ be the optimal value of our original problem. For any vector $\mu\geq 0$, $L(\mu)$ is a lower bound for $v^*(IP_{\mathrm{noSS}})$. Hence if we find $\max_{\mu\geq 0} L(\mu)$ then we will obtain the Lagrangian relaxation lower bound.

3.2 Simplification

Note that in this formulation, the y variables do not appear in constraints (1) or (2). Since $\mu\geq 0$ and the problem is minimization, we can preprocess y_{rz} $\forall r\in R, z\in Z$ as follows:

$$
y_{rz} = \begin{cases} 1 & \text{if } \left(n_r e_z - \displaystyle\sum_{p:(r,p)\in D}\mu_{rpz}\right) < 0 \\ 0 & \text{otherwise} \end{cases}
$$

After the optimization, we can then add the contribution to the objective from the y terms. To determine the value of the x variables, we have two cases, described in Sections 3.2.1

and 3.2.2.

3.2.1 One Pallet for Each Product

In (IP_{1pal}), all products have the same size, i.e. $v_p = 1\ \forall p$, so the constraints (1) and (2) are network constraints. We can view the problem as a network flow over a bipartite graph where there is a node for each product with supply one, a node for each zone with demand C_z, and a node with supply $\sum_{z\in Z} C_z - |P|$ corresponding to the excess demand. There is a directed arc from every supply node to every demand node. Hence, the constraint matrix is totally unimodular and any solution of $L(\mu)$ with the x variables relaxed to be nonnegative will automatically be integer [20]. Furthermore, if we solve $\max_{\mu\geq 0} L(\mu)$ then we will get the solution to the LP relaxation of (L) where the x variables are relaxed. To obtain the x variables we solve:

$$
\begin{array}{lllll}
(L_x) & \min & \displaystyle\sum_{(r,p)\in D}\sum_{z\in Z}\mu_{rpz}x_{pz} & & \\
& \text{s.t.} & \displaystyle\sum_{z\in Z} x_{pz} = 1 & \forall p\in P & (1') \\
& & \displaystyle\sum_{p\in P} x_{pz} \leq C_z & \forall z\in Z & (2') \\
& & x_{pz}\geq 0 & \forall p\in P, z\in Z &
\end{array}
$$

The x variables should now be interpreted as the percentage of a product assigned to a given zone. This problem is an LP with $|P|+|Z|$ constraints and $|P|*|Z|$ variables that is amenable to a network simplex algorithm or the more specialized transportation simplex. Hence, it is not a difficult problem to solve. The only entity with large size is μ which has $|Z|*|D|$ components. This is a setup cost of the problem and does not affect the problem size.

3.2.2 No Stock Splitting

For (IP_{noSS}), the constraints are not a network polytope so integrality is not guaranteed when solving the relaxation (L_x) with constraint (2) instead of (2'). Still in this case one is guaranteed to not have too many fractional variables when one solves the relaxation. Since there are $|P|+|Z|$ constraints, a basis will be of size $|P|+|Z|$. For each of the $|P|$

products, at least one x_{pz} must be strictly positive. No x_{pz} can satisfy more than one of the constraints (1') so $|P|$ of the variables in the basis will be used to satisfy the constraints (1'). Therefore, there are at most $|Z|$ variables left that can be fractional. Since $|P|$ is much greater than $|Z|$, even in this case solving the relaxation will give a solution not too far from integral. If the solution obtained is not integral, solving the following problem will fix the fractional values to give an integral solution:

$$
\begin{array}{lllll}
(IP_{\text{fix}}) & \min & \displaystyle\sum_{r\in R^f}\sum_{z\in Z_r^f} n_r e_z y_{rz} & & \\
& \text{s.t.} & \displaystyle\sum_{z\in Z} x_{pz} = 1 & \forall p \in P^f & (1^f) \\
& & \displaystyle\sum_{p\in P^f} v_p x_{pz} \le C_z^f & \forall z \in Z & (2^f) \\
& & x_{pz} \le y_{rz} & \forall (r,p) \in D \text{ s.t. } r \in R^f, p \in P^f, z \in Z_r^f & (3^f) \\
& & x_{pz} \text{ binary} & \forall p \in P, z \in Z & \\
& & y_{rz} \text{ binary} & \forall r \in R, z \in Z &
\end{array}
$$

Where:

P^f = products containing fractional assignments

R^f = orders containing products with fractional assignments

Z_r^f = for fractional order $r \in R^f$, zones not already containing a product from the previous assignment that had an integral value

C_z^f = capacity of zone $z \in Z$ remaining after the nonfractional products have been assigned

Since $|P^f| \le |Z|$ and $|Z|$ is small, the above problem will be small so long as the number of affected orders, R^f, is small.

The solution obtained from appending this assignment to the assignment from solving the relaxation is a heuristic and not an optimal approach. The difference in cost between the merged assignment and the fractional assignment is expected to be small if R^f is small.

3.3 Quality of Lower Bound

For (IP_{noSS}) where each order occurs once, $n_r = 1\ \forall r$, and each zone has unit entry cost, $e_z = 1\ \forall z$, we now show that under the assumption that the problem is feasible, the solution to $\max_{\mu \geq 0} L(\mu) = |R|$:

Theorem 3.1 *If* $\sum_{p\in P} v_p \leq \sum_{z\in Z} C_z$, *then* $\max_{\mu\geq 0} L(\mu) = |R|$

Proof Consider the following solution:

$$x_{pz} = y_{rz} = \frac{C_z}{\sum_{u\in Z} C_u}\ \forall r \in R, p \in P, z \in Z$$

First we show that this solution achieves the lower bound:

$$\sum_{r\in R}\sum_{z\in Z} y_{rz} = \sum_{r\in R}\sum_{z\in Z} \frac{C_z}{\sum_{u\in Z} C_u} = \sum_{r\in R} 1 = |R|$$

Now we show that this solution is feasible to the LP relaxation of (IP_{noSS}):

1. $\sum_{z\in Z} x_{pz} = \sum_{z\in Z} \frac{C_z}{\sum_{u\in Z} C_u} = 1\ \forall p \in P$
2. $\sum_{p\in P} v_p x_{pz} = \frac{C_z}{\sum_{u\in Z} C_u} \sum_{p\in P} v_p \leq \frac{C_z}{\sum_{u\in Z} C_u} \sum_{u\in Z} C_u = C_z\ \forall z \in Z$
3. $x_{pz} = \frac{C_z}{\sum_{u\in Z} C_u} = y_{rz} \geq 0\ \forall (r,p) \in D, z \in Z$

□

Because (IP_{1pal}) is an instance of (IP_{noSS}) with $v_p = 1\ \forall p \in P$, Theorem 3.1 holds true for this problem as well. Theorem 3.1 tells us that since the optimal value for the Lagrangian relaxation equals the optimal value for the LP relaxation of (IP_{1pal}), the lower bound, LB, we obtain is no better than the trivial lower bound. Therefore, using a simple Lagrangian relaxation approach in this situation is not sufficient to obtain good lower bounds.

3.4 Upper Bounds

For these last two sections, we will only discuss the special case (IP_{1pal}) with $n_r = 1\ \forall r$ and $e_z = 1\ \forall z$. Each time we solve (L_x) for a given μ we obtain a feasible assignment. The y

variables from the solution are almost certainly not feasible since the $x_{pz} \leq y_{rz}$ constraints have been relaxed. Hence the objective value for (L_x) plus the contribution from the y terms will be incorrect. However, it is not difficult to compute the actual cost of the assignment from the x variables. We will call this upper bound, UB_x. For each order, if some product in that order is assigned to a zone then a cost of one is incurred by the objective. That is,

$$UB_x = \sum_{r \in R} \sum_{z \in Z} 1_{[\exists p:(r,p) \in D, x_{pz}=1]}$$

where 1_S is an indicator variable that indicates whether statement S is true. If it evaluates to true, the variable has value 1; otherwise it has value 0. This upper bound can be obtained for each μ.

A second upper bound may be available from the y variables, but this upper bound is not guaranteed for each μ. First, if $\sum_{r \in R, z \in Z} y_{rz} \geq UB_x$, then the upper bound from the y variables will almost surely be no better than that from the x variables so we do not look further. However, if $\sum_{r \in R, z \in Z} y_{rz} < UB_x$, the y variables may give a better lower bound but only if a feasible assignment can be found for these y variables. So, we must solve the following feasibility problem:

$$\begin{array}{llll}(P_{y_{\text{feas}}}) & \displaystyle\sum_{z \in Z} x_{pz} = 1 & \forall p \in P & (4) \\ & \displaystyle\sum_{p \in P} x_{pz} \leq C_z & \forall z \in Z & (5) \\ & x_{pz} = 0 & \forall p \in P \text{ s.t. } (r,p) \in D \text{ and } y_{rz} = 0, z \in Z & (6) \\ & x_{pz} \geq 0 & \forall p \in P, \forall z \in Z & \end{array}$$

The third constraint says that if to fill an order one does not visit a zone, then no product in that order can be assigned to that zone. If this problem is not feasible, then the y variables do not allow for a feasible assignment. If the problem is feasible, then an assignment has been found that gives the y variables. Since it is possible that some y_{rz} was unnecessarily positive, we compute the value of the assignment in the same way as we did to get UB_x. At this point we have found an upper bound from the y variables which we call UB_y .

Therefore, for each value for the multiplier μ we can get one, and potentially two upper bounds for (IP_{1pal}).

3.5 Algorithmic Details

To solve the problem, $\max_{\mu \geq 0} L(\mu)$, we use the subgradient optimization technique outlined in Ahuja et al. [1]. That is, we start with an initial μ^0. Then, $\forall (r,p) \in D, z \in Z$:

$$\mu_{rpz}^{k+1} = [\mu_{rpz}^{k} + \theta^k (x_{pz} - y_{rz})]^+$$

We use the following suggested heuristic for updating θ^k:

$$\theta^k = \frac{\lambda^k [UB - L(\mu^k)]}{\|x - y\|^2}$$

where UB is the best upper bound obtained so far. To start we use the trivial upper bound of $|D|$ and to update λ^k , use the following heuristic:

> Start with an initial λ^0 between 0 and 2. If the best Lagrangian objective found so far has not increased in a given number of iterations, then reduce λ^k by a given factor.

Even though the initial value of μ^0 does not affect convergence of the method, to get good upper bounds it helps to try multiple starting points. That is, we let the method proceed for some number of iterations from multiple starting points. After a predetermined number of iterations, we stop and go on to the next starting point. For one of the starting points, we let the method run until a terminating condition has been reached. The algorithm terminates when one of the following conditions occurs:

1. The iteration limit has been exceeded.

2. The upper bound and lower bound are the same - this is the ideal situation.

3. The value of the Lagrangian relaxation is the same in two consecutive iterations.

3.6 Computing Multipliers Given an Initial Solution

In Chapter 4 we will discuss several other ways to get feasible solutions. Now we will show how a set of multipliers for the Lagrangian relaxation problem can be obtained from any heuristic solution by solving an inverse optimization problem. Hence any feasible solution

Lagrangian Relaxation Upper Bounding Algorithm

```
For each starting point do
    While no terminating condition holds do
        Preprocess out the contribution to the objective, $L(\mu)$ from the $y$ terms.
        Solve $(L_x)$.
        Let $L(\mu) = (L_x) + y$ contribution.
        Update LB.
        Update the best Lagrangian objective found or the number of iterations
            in which it hasn't changed.
        Determine $UB_x$.
        If $\sum_{r\in R, z\in Z} y_{rz} < UB_x$ solve $(P_{y_{\text{feas}}})$. If it is feasible, compute $UB_y$.
        If $\min(UB_x, UB_y) < UB$, update UB and record the assignment.
        Update $\lambda,\ \theta,\ \mu$.
        If the current starting point should not go to completion and it has completed
            its allowed number of iterations, break and go to the next point.
    End while
End for
```

Figure 4: Overview of Lagrangian Relaxation Upper Bounding Algorithm

not only gives an upper bound to the number of zones visited but also gives a starting point for solving the Lagrangian relaxation problem. To show this, we will rewrite the Lagrangian relaxation problem together with its dual. In this section, we assume that all products have one pallet of storage, $v_p = 1\ \forall p$, each order occurs once, $n_r = 1\ \forall r$, and each zone has unit entry cost, $e_z = 1\ \forall z$. So we have:

$$
\begin{array}{llll}
(L) & \min & \displaystyle\sum_{r\in R}\sum_{z\in Z} y_{rz} + \sum_{(r,p)\in D}\sum_{z\in Z}\mu_{rpz}(x_{pz} - y_{rz}) & \\
& \text{s.t.} & \displaystyle\sum_{z\in Z} x_{pz} = 1 \quad \forall p \in P & (\theta) \\
& & \displaystyle\sum_{p\in P} x_{pz} \le C_z \quad \forall z \in Z & (\pi) \\
& & \mu_{rpz} \ge 0 \quad \forall (r,p)\in D, z\in Z & \\
& & x_{pz} \ge 0 \quad \forall p\in P,\ z\in Z & \\
& & y_{rz} \ge 0 \quad \forall r\in R,\ z\in Z &
\end{array}
$$

$$
\begin{array}{llll}
(LD) & \max & \sum_{p\in P}\theta_p + \sum_{z\in Z} C_z\pi_z & \\
& \text{s.t.} & \theta_p + \pi_z \le \mu_{rpz} & \forall p\in P, z\in Z \\
& & 0 \le 1 - \sum_{p\in P:(r,p)\in D} \mu_{rpz} & \forall r\in R, z\in Z \\
& & \pi_z \le 0 & \forall z\in Z
\end{array}
$$

Given a solution x_{pz}, y_{rz}, we can solve the following linear program to recover μ_{rpz}:

$$
\begin{array}{lllll}
(L_\mu) & \min & \sum_{\substack{(r,p)\in D, z\in Z: \\ x_{pz}=0, y_{rz}=1}} \mu_{rpz} & & \\
& \text{s.t.} & \theta_p + \pi_z \le \mu_{rpz} & \forall (r,p)\in D, z\in Z & \left.\right\} \text{(D) feasibility} \\
& & 0 \le 1 - \sum_{p\in P:(r,p)\in D} \mu_{rpz} & \forall r\in R, z\in Z & \\
& & \pi_z \le 0 & \forall z\in Z & \\
& & \mu_{rpz} \ge 0 & \forall (r,p)\in D, z\in Z & \left.\right\} \text{(P) feasibility} \\
& & \pi_z = 0 & \forall z\in Z : \sum_{p\in P} x_{pz} < C_z & \left.\right\} \text{Complementary Slackness} \\
& & \theta_p + \pi_z = \mu_{rpz} & \forall (r,p)\in D, z\in Z : x_{pz} > 0 & \\
& & 0 = 1 - \sum_{p\in P:(r,p)\in D} \mu_{rpz} & \forall r\in R, z\in Z : y_{rz} > 0 &
\end{array}
$$

The objective of (L_μ) looks for multiplier values that will maximize (L). However, the true goal is to find a feasible solution. In fact, we can give an analytical solution for such a feasible point. First we define the following:

q_r = number of zones visited for order r

s^* = maximum order size

Theorem 3.2 *The following point is feasible for (L_μ):*

$$
\mu_{rpz} = \begin{cases} \frac{1}{s^*} & \forall (r,p)\in D, z\in Z : x_{pz} = y_{rz} = 1 \\ \frac{1-\frac{n_{rz}}{s^*}}{s_r - n_{rz}} = \frac{1}{s^*}\frac{s^*-n_{rz}}{s_r-n_{rz}} & \forall (r,p)\in D, z\in Z : x_{pz} = 0, y_{rz} = 1 \\ \frac{1}{s_r} & \forall (r,p)\in D, z\in Z : x_{pz} = y_{rz} = 0 \end{cases}
$$

$$\pi_z = \quad 0 \quad \forall z\in Z$$

$$\theta_p = \quad \tfrac{1}{s^*} \quad \forall p\in P$$

and has an objective value for (L) of $\frac{1}{s^*}\sum_{r\in R} s_r$

Proof First observe that:

$$s^* \geq s_r \ \forall r \in R \qquad s_r \geq n_{rz} \ \forall r \in R, z \in Z \qquad s^* \geq n_{rz} \ \forall r \in R, z \in Z$$

Now let us restate the constraints as follows:

$$\begin{array}{lll}
\theta_p + \pi_z = \mu_{rpz} & \forall (r,p) \in D, z \in Z : x_{pz} = 1 & (1a) \\
\theta_p + \pi_z \leq \mu_{rpz} & \forall (r,p) \in D, z \in Z : x_{pz} = 0 & (1b) \\
\displaystyle\sum_{p \in P:(r,p) \in D} \mu_{rpz} = 1 & \forall r \in R, z \in Z : y_{rz} = 1 & (2a) \\
\displaystyle\sum_{p \in P:(r,p) \in D} \mu_{rpz} \leq 1 & \forall r \in R, z \in Z : y_{rz} = 0 & (2b) \\
\pi_z = 0 & \forall z \in Z : \displaystyle\sum_{p \in P} x_{pz} < C_z & (3a) \\
\pi_z \leq 0 & \forall z \in Z : \displaystyle\sum_{p \in P} x_{pz} = C_z & (3b) \\
\mu_{rpz} \geq 0 & \forall (r,p) \in D, z \in Z & (4)
\end{array}$$

We will verify that the constraints (1a) - (4) hold:

(1a) $x_{pz} = 1 \Rightarrow \mu_{rpz} = \dfrac{1}{s^*} = \theta_p$

(1b) $x_{pz} = y_{rz} = 0 \Rightarrow \mu_{rpz} = \dfrac{1}{s_r} \geq \dfrac{1}{s^*} = \theta_p$

$$x_{pz} = 0, y_{rz} = 1 \Rightarrow \mu_{rpz} = \frac{1}{s^*} \frac{s^* - n_{rz}}{s_r - n_{rz}} \geq \frac{1}{s^*} = \theta_p$$

(2a) $$y_{rz} = 1 \Rightarrow \sum_{p \in P:(r,p) \in D} \mu_{rpz} = \sum_{\substack{p \in P:(r,p) \in D \\ x_{pz}=1}} \mu_{rpz} + \sum_{\substack{p \in P:(r,p) \in D \\ x_{pz}=0}} \mu_{rpz}$$
$$= n_{rz} \frac{1}{s^*} + (s_r - n_{rz}) \frac{1 - \frac{n_{rz}}{s^*}}{s_r - n_{rz}} = 1$$

(2b) $$y_{rz} = 0 \Rightarrow \sum_{p \in P:(r,p) \in D} \mu_{rpz} = s_r \frac{1}{s_r} = 1$$

(3) $\pi_z = 0 \ \forall z \in Z$

(4) $x_{pz} = y_{rz} = 1 \Rightarrow \mu_{rpz} = \dfrac{1}{s^*} > 0$

$$x_{pz} = y_{rz} = 0 \Rightarrow \mu_{rpz} = \frac{1}{s_r} > 0$$

$$x_{pz} = 0, y_{rz} = 1 \Rightarrow \mu_{rpz} = \frac{1}{s^*} \frac{s^* - n_{rz}}{s_r - n_{rz}} \geq \frac{1}{s^*} > 0$$

Now we compute the objective value of (L) for this point:

$$\begin{aligned}
(L) &= \sum_{r\in R, z\in Z} y_{rz} + \sum_{(r,p)\in D, z\in Z} \mu_{rpz}(x_{pz} - y_{rz}) \\
&= \sum_{r\in R} q_r - \sum_{(r,p)\in D, z\in Z:\ x_{pz}=0, y_{rz}=1} \mu_{rpz} \\
&= \sum_{r\in R} q_r - \sum_{r\in R, z\in Z:\ x_{pz}=0, y_{rz}=1} (s_r - n_{rz}) \frac{1 - \frac{n_{rz}}{s^*}}{s_r - n_{rz}} \\
&= \sum_{r\in R} q_r - \sum_{r\in R, z\in Z:\ x_{pz}=0, y_{rz}=1} \left(1 - \frac{n_{rz}}{s^*}\right) \\
&= \sum_{r\in R} q_r - \sum_{r\in R} \left(q_r - \frac{s_r}{s^*}\right) \\
&= \frac{1}{s^*} \sum_{r\in R} s_r
\end{aligned}$$

□

Theorem 3.2 connects the approach from the current chapter to those in Chapter 4. The objective value (L) of the analytical point given by (x, y, μ) does not depend on the storage assignment (x, y). The value is the sum of the proportion of each order size to the maximum order size. This value will be greater than zero and cannot exceed $|R|$. Since (L) gives a lower bound of the optimal solution, we would like to maximize its value. We attain the maximal lower bound of $|R|$ when $s_r = s^*$, i.e. all orders are the same size.

CHAPTER 4

UPPER BOUND HEURISTICS

Earlier, we gave the trivial upper bound of $|D|$ on (IP_{general}). Typically, this upper bound is an overestimate so we would like to use intelligent heuristics to get better upper bounds. In this chapter, we will exclusively discuss the formulation (IP_{1pal}). For simplicity, we assume each order occurs once, $n_r = 1\ \forall r$, and each zone has unit entry cost, $e_z = 1\ \forall z$. In Chapter 5 we will discuss the two generalizations.

Below we discuss several ideas of our own as well as several ideas from the literature. Before presenting other researchers' methods, we must clarify how we will use their approaches on our problem since they consider a slightly different problem. They consider how to form product clusters and where to place them so that travel time is minimized. First, since they consider clusters and we consider zones, we use their notions of cluster formation to create the set of products to store in a zone. That is, clusters will logically be treated as zones. Second, we do not consider the placement of zones in the warehouse. Hence, we will ignore the problem of placing clusters in locations and only worry about the content of the clusters.

4.1 Construction Methods

In construction heuristics, a feasible solution is incrementally built up from an empty initial solution. In this section, we adapt several heuristics that were developed for travel time objectives and propose a new heuristic for our problem.

4.1.1 Random Assignment

A simple approach is to place products randomly in zones in the warehouse. This heuristic is useful as a performance baseline for future methods. To determine the expected performance, we let $\Pr(|Z|, k, l)$ be the probability of using exactly l out of $|Z|$ zones in a k-product order. Note that we make the logical assumptions that $l \leq |Z|$ and $l \leq k$. Then

$\Pr(|Z|, k, 0) = 0$ and for $l > 0$,

$$\Pr(|Z|, k, l) = \left(\frac{l}{|Z|}\right)^k \binom{|Z|}{l} - \sum_{i=1}^{l-1} \binom{|Z|-i}{l-i} \Pr(|Z|, k, i)$$

The first term is the number of different ways that the k products may be placed in l or fewer zones out of $|Z|$ total. The second term subtracts all the ways that the products could be in i zones for $i < l$. For each i, there are $|Z| - i$ zones that could be empty of products in the order and we need to choose $l - i$ zones to be empty.

Given the probabilities $\Pr(|Z|, k, l)$, the expected number of zones visited for all orders, EZ, is

$$\sum_k \left[N_k \sum_{l=1}^{k} l \Pr(|Z|, k, l) \right]$$

where N_k is the number of orders containing k products.

4.1.2 Popularity

One common method in industry is to use popularity. That is, we sort the products in order of their popularity. We place the most popular products together subject to the capacity on the zone and continue until all products have been assigned to a zone. When all products have the same size as we assume in this chapter, this is equivalent to the COI rule.

4.1.3 Sequential Clustering

Frazelle [10] proposes a sequential clustering approach where clusters are grown one at a time. He includes a congestion constraint in addition to the capacity constraint; we will modify his method to ignore this constraint as we do not consider congestion explicitly. First he determines whether for all pairs of products i and j, there is a statistically significant relation between them using the Chi-square, χ^2, test for independence. First define:

σ_{ij} = the number of orders in which products i and j appear together
(the pairwise popularity of products i and j)

σ_i = the number of orders in which product i appears
(the popularity of product i)

```
Sequential Clustering

While there is an unassigned product do
    Choose the most popular unassigned product, say $i$, as a cluster seed.
    Decrement the capacity of the zone by one.
    While the capacity of the zone is positive do
        Choose a neighbor of the cluster with highest weight, say $j$,
          and add it to the cluster.
        Merge nodes $i$ and $j$ by letting $w_{ik} := w_{ik} + w_{jk}\ \forall k \neq i, j$
          and removing vertex $j$ along with all its edges from the graph.
        Decrement the capacity of the zone by one.
        Let $\sigma_i := \sigma_i + \sigma_j$
    End while
End while
```

Figure 5: Sequential Clustering Algorithm

Then he defines the following test statistic:

$$\tau = \frac{\left(\sigma_{ij} - \frac{\sigma_i \sigma_j}{|R|^2}\right)^2}{\frac{\sigma_i \sigma_j}{|R|^2}} + \frac{\left(\sigma_i - \sigma_{ij} - \frac{\sigma_i(1-\sigma_j)}{|R|^2}\right)^2}{\frac{\sigma_i(1-\sigma_j)}{|R|^2}} + \frac{\left(\sigma_j - \sigma_{ij} - \frac{(1-\sigma_i)\sigma_j}{|R|^2}\right)^2}{\frac{(1-\sigma_i)\sigma_j}{|R|^2}}$$

If $\tau > \chi^2_{1-\alpha}(1)$ for a specified level α then the test fails to reject the hypothesis that products i and j are independent. Using this information, he creates a graph as follows. For each product, he creates a vertex i with label σ_i. For each pair where τ is statistically significant, he creates an edge with weight equal to the conditional probability that product j appears in an order given i appears. So the edge weight, w_{ij}, is

$$p_{j|i} = \frac{\sigma_{ij}}{\sigma_i}$$

Given this graph, the method uses the most popular unassigned product as a cluster seed. The unassigned product with strongest correlation to the cluster is iteratively added to the cluster subject to capacity restrictions. When the cluster has reached its size limit, the next cluster is formed in the same way. The algorithm proceeds until all products have been assigned to clusters. The method is summarized in Figure 5.

4.1.4 Simultaneous Clustering

Amirhosseini and Sharp [4] propose a clustering approach where multiple clusters are formed simultaneously. They define some correlation measures from the literature which we will also state in terms of the variables σ_i and σ_{ij} that we defined in Section 4.1.3:

$$\begin{aligned} c_{ij}^1 &= \frac{\text{\# of times products } i,j \text{ appear together in an order}}{\text{Total \# of orders}} \\ &= \frac{\sigma_{ij}}{|R|} \end{aligned}$$

$$\begin{aligned} c_{ij}^2 &= \frac{\text{\# of times products } i,j \text{ appear alone in an order}}{\text{Total \# of orders}} \\ &= \frac{\sigma_i + \sigma_j - 2\sigma_{ij}}{|R|} \end{aligned}$$

$$\begin{aligned} c_{ij}^3 &= \frac{\text{\# of times products } i,j \text{ appear together in an order}}{\text{\# of times products } i \text{ or } j \text{ or both occur in an order}} \\ &= \frac{\sigma_{ij}}{\sigma_i + \sigma_j - \sigma_{ij}} \end{aligned}$$

$$\begin{aligned} c_{ij}^4 &= \frac{\text{\# of times products } i,j \text{ appear together in an order}}{\text{\# of times only product } i \text{ occurs in an order}} \\ &= \frac{\sigma_{ij}}{\sigma_i - \sigma_{ij}} \end{aligned}$$

$$\begin{aligned} c_{ij}^5 &= \frac{\text{\# of times products } i,j \text{ appear together in an order}}{\text{\# of times only product } j \text{ occurs in an order}} \\ &= \frac{\sigma_{ij}}{\sigma_j - \sigma_{ij}} \end{aligned}$$

They then present their own measure, the Order-Satisfying Correlation Measure (OSCM):

$$\begin{aligned} c_{ij}^{\text{OSCM}} &= \frac{\text{sum of the fractions of orders filled by products } i \text{ or } j \text{ or both}}{\text{\# of times products } i \text{ or } j \text{ occur alone in an order}} \\ &= \frac{\sum_{r\in R} f_r}{\sigma_i + \sigma_j - 2\sigma_{ij}} \end{aligned}$$

where

$$f_r = \begin{cases} \dfrac{\text{number of products from } i \text{ and } j \text{ that appear in order } r}{s_r} & \text{if } f_L \leq f_r \leq f_U \\ 0 & \text{otherwise} \end{cases}$$

for parameters $0 \leq f_L \leq f_U \leq 1$.

Using a chosen correlation measure, he then does hierarchical clustering using a nearest neighbor method (NNC). The method starts with each product in its own cluster. The two clusters with the strongest correlation iteratively merge together. A cluster becomes ineligible for mergers when its size reaches the maximum. The main difference with the approach in Section 4.1.3 is that here multiple clusters are eligible to grow whereas before only one cluster grew at a time. Figure 6 outlines our implementation of this method.

```
Simultaneous Clustering

Compute the correlation for all pairs of products.
Initialize all products to be in their own cluster.
While there are more clusters than zones do
        Choose the cluster pair, say $i$ and $j$, with the maximum correlation measure
          and merge the clusters together.
        Update the correlation between the merged cluster and all other clusters $k$
          by taking $\max\{c_{ik}, c_{jk}\}$.
        If the cluster size is equal to the zone capacity, remove the cluster from
          further consideration.
End while
```

Figure 6: Simultaneous Clustering Algorithm

```
Particle Movement

Phase I
  Assign products to initial points in space
  While terminating condition has not yet been reached do
        Relocate products in space based on pairwise popularity
  End while

Phase II
  Assign cluster centers to initial points in space
  While the assignment is new do
        Assign products to cluster centers using transportation simplex
        Relocate cluster center to center of mass defined by products in the cluster
  End while
```

Figure 7: Overview of Particle Movement Algorithm

4.1.5 Particle Movement

In this approach, products in n-dimensional space iteratively move towards each other with magnitude depending on the correlation strength. When an equilibrium is reached so that products no longer move significantly, cluster centers are placed in space. Products are successively assigned to the cluster centers and the cluster centers relocated until an equilibrium is reached. The overview of the algorithm is given in Figure 7. This method is similar to those in Sections 4.1.3 and 4.1.4 in that it works directly with pairwise product popularity. Below we will discuss in detail how the movement and cluster center phases work as well as the implementation details.

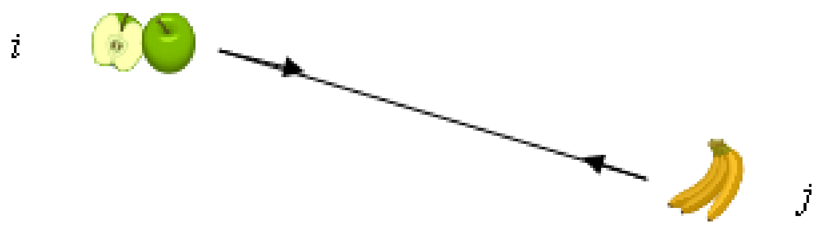

Figure 8: Product Movement in Phase I

The objective of Phase I is to place the products in space so that products with strong bonds are near each other. First, products are randomly placed at points in n-dimensional space. Given an assignment of products to points in space, we relocate the products so that products with high pairwise popularity move closer to each other. However, to ensure that the products do not all move to the same point, we include a repulsive force. Following is the total magnitude of movement for two products i, j:

$$t(\overbrace{\sigma_{ij}}^{\text{attractive force}} - \overbrace{\frac{\tilde{c}}{\text{distance}(i,j)}}^{\text{repulsive force}})$$

where t is a time step and $\tilde{c}$ is a constant. Observe that the repulsive force used here is inversely proportional to the distance between two points. We assign half of the magnitude of movement to each product. Let m_d be the slope between dimensions d and $d+1$ and dim be the number of spatial dimensions. Since the products move towards each other on the line connecting them (Figure 8), the following proportion of the movement is assigned to each dimension:

Dimension 1: $\dfrac{1}{\sqrt{\sum_{d=1}^{\text{dim}-1} m_d^2}}$

Dimension d $(d > 1)$**:** $\dfrac{m_{d-1}}{\sqrt{\sum_{d=1}^{\text{dim}-1} m_d^2}}$

where:

$$m_d = \frac{\kappa_{i,d+1} - \kappa_{j,d+1}}{\kappa_{i,d} - \kappa_{j,d}}$$

$\kappa_{i,d} =$ location of product i in dimension d

We sum the movement for each product over all its pairs and then simultaneously execute the moves. This process continues until we reach a stopping condition. Several different stopping criteria are proposed:

1. Iterate for a fixed number of time steps.
2. Continue while the total change in movement in an iteration exceeds some threshold.
3. Run while the percentage of change in movement exceeds a threshold.
4. Proceed until the total distance between all pairs exceeds a threshold.

The distance threshold in the last criterion is a percentage of the expected distance between the points if we were to randomly place the points in space. In the case when the dimension is two, Lazoff and Sherman [17] report the expected distance in closed form:

$$\frac{a^5 + b^5 - (a^4 - 3a^2b^2 + b^4)\sqrt{a^2+b^2}}{15a^2b^2} + \frac{a^2}{6b}\ln\left(\frac{b+\sqrt{a^2+b^2}}{a}\right) + \frac{b^2}{6a}\ln\left(\frac{a+\sqrt{a^2+b^2}}{b}\right)$$

where a, b are the lengths of the space in each of the two dimensions. When $a = b$ the equation reduces to

$$a\left(\frac{2+\sqrt{2}}{15} + \frac{\ln(1+\sqrt{2})}{3}\right) \approx 0.5214a$$

When a stopping criterion is reached, the products have reached an equilibrium in space and Phase I terminates. Note that the equilibrium is not guaranteed to be a globally optimal product distribution.

In Phase II clusters are formed using the distribution of products in space from Phase I. Throughout Phase II, the placement of products is fixed. First, cluster centers must be given initial positions in space. The simplest approach is to do this randomly. However, since we have placed the products in space in Phase I, we can use this information as follows. Create a grid in n-dimensional space and compute the force exerted on each grid point g by each product p. Let us define the point h to be the head of the force vector created by the cumulative force of every other product on p (Figure 9). Also, let

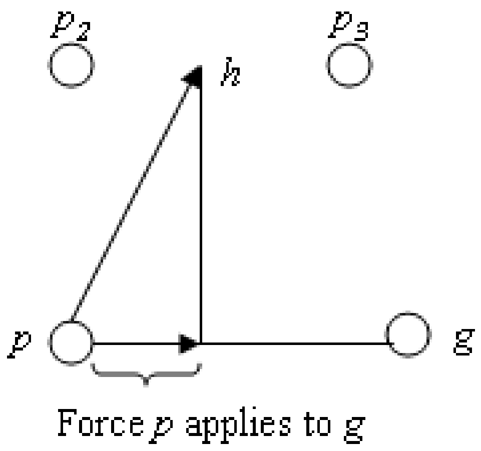

Figure 9: Force on Grid Point when dim = 2

κ_p = location of product p in space

Then,

$$\phi_{pg} = \frac{(\kappa_p + h) \cdot g}{\|g\|}$$

or the magnitude exerted by all other products on p projected onto the point g. We then break this magnitude down into the different dimensions of our space. Define

$$\begin{aligned} m_d &= \left| \frac{g_{d+1} - p_{d+1}}{g_d - p_d} \right| \\ \psi_d &= \begin{cases} 1 & d = 1 \\ m_d \psi_{d-1} & d > 1 \end{cases} \end{aligned}$$

for each dimension $d = 1, \ldots, \dim - 1$. Observe that ψ_d is defined recursively. The force each product p applies to grid point g in dimension d is $\phi_{pg}\psi_d$. So the total force on grid point g is $\sum_{p \in P} \sum_{d=1}^{d=\dim} \phi_{pg}\psi_d$. The $|Z|$ grid points with the highest total force become our initial cluster centers.

Once we have initial cluster centers, we optimally assign the products to the cluster centers using the transportation simplex algorithm. The objective is to minimize the total distance from products to their respective cluster centers. Note that this objective does not directly compute our objective of minimizing multizone orders, but we expect the objectives to be strongly correlated. It is simple to compute our objective for the assignment and compare its behavior over time to the objective used by the transportation simplex. If the assignment in an iteration is the same as an assignment from a prior iteration, the method terminates. Otherwise the cluster centers are relocated to the center of mass of the products which are assigned to it. In this case all products have identical mass.

Now we will show that Phase II converges. First define:

y_z^i = location of cluster center z in iteration i

x_p^i = assignment of product p to a cluster center in iteration i

c^{ij} = cost of the assignment from iteration i given the locations of iteration j

$= \sum_{p \in P} \text{distance}(\kappa_p, y^j_{x^i_p})$

Theorem 4.1 *Phase II of the particle movement heuristic converges monotonically to some limit c^*.*

Proof First we show that the objective is nonincreasing, that is, $c^{ii} \geq c^{i+1,i+1}\ \forall i$. Observe that $c^{ii} \geq c^{i+1,i}$ because the locations of the cluster centers have not changed so the new assignment is optimal and hence cannot have a cost higher than the previous one. Now observe that $c^{i+1,i} \geq c^{i+1,i+1}$. This is true because the cluster centers have moved to the center of mass so the total distance from all products in a cluster to their cluster center must decrease. Putting these two inequalities together shows that $c^{ii} \geq c^{i+1,i+1}\ \forall i$. Since the sequence is bounded below by zero, the Monotone Convergence Theorem [6] demonstrates that it converges to some limit c^*. This completes the proof. □

This theorem shows that Phase II terminates at an objective value. However, it is important to note that there is no guarantee that we get globally optimal clusters.

Now we will propose some implementation alternatives for the particle heuristic. In Phase I, products were initially placed randomly throughout the entire space. An alternative idea is to start with the products close together in the space and let them expand out into clusters in a "big bang" way. In Phase II, there are several alternative ideas on how to form clusters using the assignment of products to points in space.

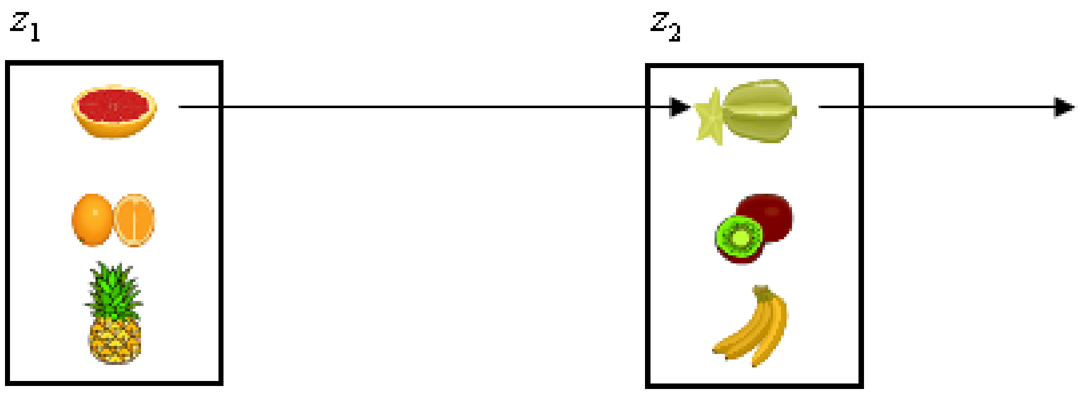

Figure 10: Exchange Move

1. Start with a single cluster and split the cluster into pieces until we have $|Z|$ clusters.

2. Run a vehicle routing algorithm on the location of the products in space to find $|Z|$ routes with minimal travel distance.

3. Use as many grid points as form distinct cluster centers whether it is more or less than $|Z|$. If some cluster has too many products in it or there are too many clusters, fix the content of the other clusters and run a smaller assignment problem on the excess products.

4.2 Improvement Methods

The upper bound heuristics discussed in Section 4.1 all construct solutions from scratch. Here we discuss methods that try to improve upon an an incumbent feasible solution. The basis of the improvement methods we discuss below is the exchange move where a product in one zone is replaced with a product from a different zone (Figure 10).

4.2.1 2-Exchange

We implement a 2-exchange algorithm where two products $p_1 \in z_1$ and $p_2 \in z_2$ swap zones during an iteration (Figure 11). First let:

$w_{p_1p_2}$ = edge weight corresponding to the change in objective due to the replacement

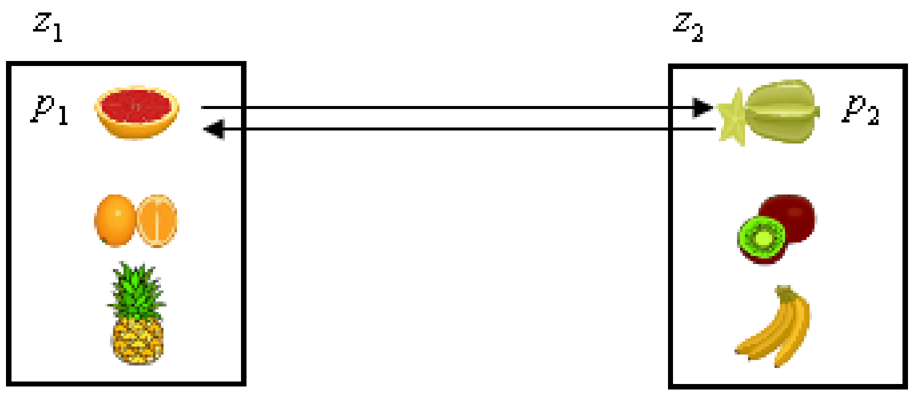

Figure 11: 2-Exchange Move

of p_2 in z_2 with p_1

R_p	= set of orders containing product p
n_{rz}	= number of products from order r contained in zone z

The edge weight is negative if this move reduces the objective and positive otherwise. Note that in general $w_{p_1p_2}$ does not equal $w_{p_2p_1}$. To compute $w_{p_1p_2}$, initialize $w_{p_1p_2} := 0$. Then for each order $r \in R_{p_1}$, do:

1. If $n_{rz_1} = 1$, let $w_{p_1p_2} := w_{p_1p_2} - 1$
2. If $n_{rz_2} = 0$, let $w_{p_1p_2} := w_{p_1p_2} + 1$
3. If $n_{rz_2} = 1$ and p_2 is the unique product from order r in zone z_2, let $w_{p_1p_2} := w_{p_1p_2} + 1$

In the first computation, if p_1 is the only product from the order contained in z_1, we no longer need to visit the zone to fill order r so the objective improves by one. If z_2 contained no products from order r, we now must visit this zone to retrieve the order so the second computation increases the objective by one. The third computation corrects the case when p_2 is the only product from r in z_2 but is replaced by p_1. In this situation, the weight for the edge from p_2 will mistakenly give a benefit for order r. Since any cycle has both an inbound and an outbound edge to p_2, the third computation offsets this case.

For each pair of zones, we exchange the pair of products that minimizes $w_{p_1p_2} + w_{p_2p_1}$ so long as the minimum is negative. The method terminates when there is no pair of

2-Exchange

```
While swapflag
    Set swapflag := false
    For each pair of zones $z_1$ and $z_2$
        For each pair of products such that $p_1 \in z_1$ and $p_2 \in z_2$
            Evaluate the improvement in objective from exchanging $p_1$ and $p_2$
        If the greatest improvement is positive
            Exchange this pair
            Set swapflag := true
        End if
    End for
End while
```

Figure 12: 2-Exchange Algorithm

zones with an improving 2-exchange. The algorithm (Figure 12) is guaranteed to finitely terminate because only improving exchanges are accepted.

4.2.1.1 Size of Search Space

Any two products in different zones may be exchanged. Hence the size of the search space in any iteration is $\Theta(|P|^2)$.

4.2.1.2 Running Time per Iteration

In the worst case, we must look at all pairs of products in different zones before finding an improving exchange or determining that none exists. There are $O(|P|^2)$ such pairs to examine per iteration. For each product pair, the edge weights composing the swap must be computed. As described in Section 4.2.1, we must consider $|R_p|$ orders for edge (p, p_1). On average, the value of $|R_p|$ is small compared to $|P|$. Therefore, in practice, an edge weight can be computed in constant time and the total running time is $O(|P|^2)$. So the search space and running time are identical.

4.2.2 Cyclic Exchange: Basic Case

The 2-exchange approach can be viewed as a search for cycles with two products from different zones. Next, we consider larger cycle sizes. We could do this using a k-exchange procedure for any $k < |Z|$. Performing a k-exchange procedure for each k involves $O(|P|^k)$

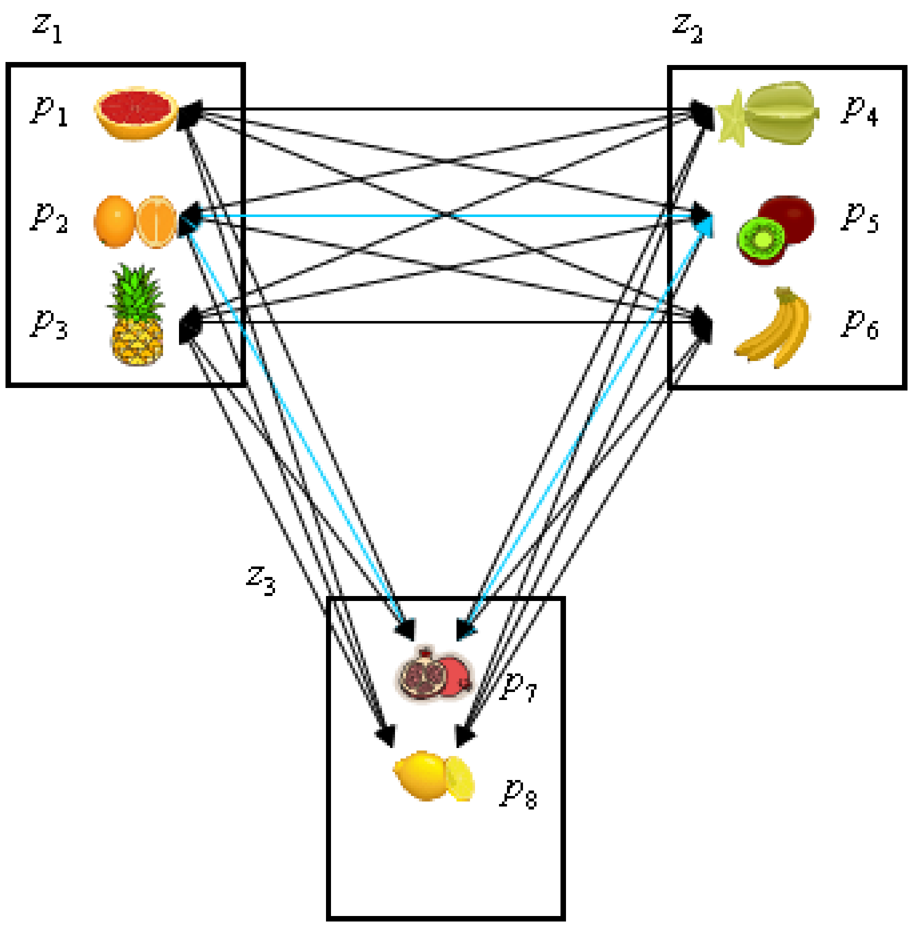

Figure 13: Structure of G

possible swaps. Below we discuss a cyclic exchange method to search an exponential number of swaps of sizes from 2 to $|Z|$ in polynomial time.

The theory of cyclic exchanges is developed in [26]. Cyclic exchanges are applied to the capacitated minimum spanning tree problem in an efficient manner yielding good solutions in [2] and [3]. In this work we are minimizing multi-zone orders in the storage assignment problem.

We define a graph G (Figure 13) as follows:

- There is a vertex for each product.
- There is a directed edge between each pair of products not contained in the same

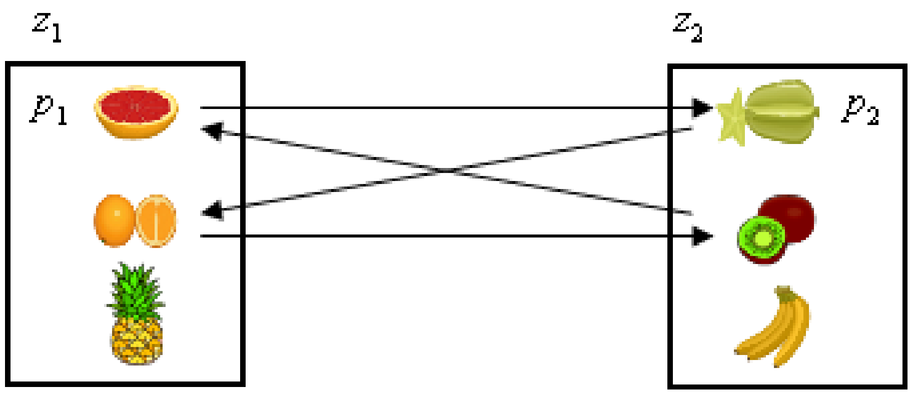

Figure 14: Illustration of a Double Cycle

zone.

- The edge weight $w_{p_1p_2}$ is as described for 2-exchanges.

In the 2-exchange case, an edge (p_1, p_2) is paired with the edge (p_2, p_1). The main issue with cyclic swaps is that we do not know the next edge in the cycle so the location of p_2 after the exchange is unknown. Our goal is to find the minimum weight negative cycle (MWNC) in this graph. However, there is a complication. The cycle detected could contain two products that are both located in the same zone (Figure 14). In this scenario, the edge weights computed earlier are incorrect. The computation for edge weight (p_1, p_2) assumed that p_2 was leaving z_2 but in this case some other product is leaving as well. Hence, if we get a cycle that repeats any zone, for example Figure 14, the reported objective improvement could be incorrect. Therefore we need to identify the minimum weight negative cycle that visits each zone at most once (MWNCZ1). Let us state the decision version of this problem:

MWNCZ1

Given the graph G defined above with a vertex for each product $p \in P$ and edge weights w_{p_i,p_j}, and an integer k, is there a directed negative weight cycle visiting each zone at most once with length $\leq k$?

Conjecture 4.1 *MWNCZ1 is NP-Complete.*

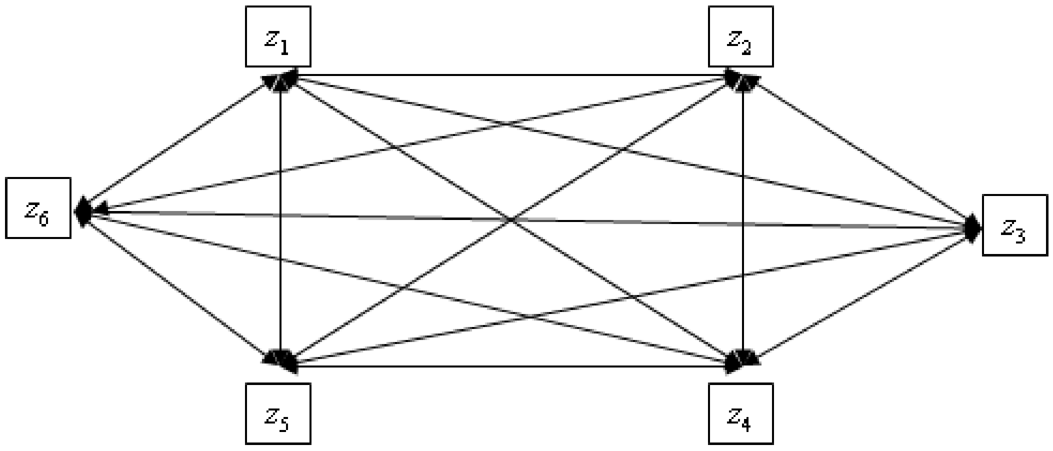

Figure 15: Structure of G^0

We believe that Conjecture 4.1 is true because the shortest circuit problem is NP-Complete when negative edge weights are allowed [11]. It is difficult to prove that the MWNCZ1 problem is NP-complete because we can only visit each zone at most once and because the edge weights have a special structure. Since we believe that MWNCZ1 is NP-Complete, we use heuristic methods to detect good improving cycles. First we determine a permutation of the zones. Next we create a directed acyclic graph (DAG) based on the permutation. Due to the method by which we create this graph, we find an improving cycle by solving a shortest path problem. Now we will describe each of these steps in further detail.

To determine which of the $|Z|!$ different permutation to choose, we construct the graph G^0 (Figure 15):

- There is a node for each zone.
- There is a directed edge between each pair of zones.
- The edge weight between zones z_1 and z_2,

$$\omega_{z_1 z_2} = \sum_{p_1 \in z_1, p_2 \in z_2} 1_{[w_{p_1 p_2} < 0]} w_{p_1 p_2}$$

  where $w_{p_1 p_2}$ is from G.

```
Permute

Choose the directed edge $e = (z_1, z_2)$ in $G^0$ with minimum weight
Let $z_1$ be the first zone and $z_2$ the second zone in the permutation
While there remains a zone not yet in the permutation
        Select the edge $e = (z_1, z_2)$ with minimum weight where $z_1$ is the zone
           most recently added to the permutation and $z_2$ is a zone not in the permutation
        Add $z_2$ to the end of the permutation
End while
```

Figure 16: Permutation Algorithm

To compute the edge weights $\omega_{z_1 z_2}$, we consider all pairs of products $p_1 \in z_1, p_2 \in z_2$. If the move of p_1 to the location of p_2 improves the objective, the objective change $w_{p_1 p_2}$ is added to $\omega_{z_1 z_2}$. Note that, by construction, all edge weights in G^0 are non-positive.

On the graph G^0 we do a greedy search for a permutation (Figure 16). First we select the edge with smallest weight and add the associated zones to the permutation. Then we iteratively choose the edge starting at the last zone in the permutation which terminates in a zone not yet selected which has minimum weight. The as yet unselected zone is added to the permutation. This terminates when every zone is included in the permutation.

Performing this algorithm on G^0 defines a sequence of zones $\zeta_1, \ldots, \zeta_{|Z|}$ where ζ_i is the i^{th} zone in the permutation. We let $|\zeta_i|$ denote the number of products stored in this zone. Now that we have found a permutation, the graph G^1 is constructed (Figure 17). For each product there is a node in G^1. There is a directed edge between each pair of products p_1, p_2 for which the zone of p_1 occurs earlier in the permutation than the zone of p_2. Note that unlike in G, in G^1 exactly one of the edges $(p_1, p_2), (p_2, p_1)$ occurs but not both. The edge weight is exactly the same edge weight as defined for G. In addition to the subgraph of G just described, a dummy sink node, s, is created. Let $p_0 \in \zeta_i$ denote the starting node for a cyclic exchange. There is an edge from every product $p \in \zeta_j$, $j > i$, to node s. The weight of these edges is w_{pp_0} so that these edges correspond to moving p to the location of p_0, the starting node.

Using the graph G^1, we can find a minimum weight cycle. We have constructed G^1 so that it is a DAG. A $\Theta(V + E)$ algorithm is given for solving the single source shortest path

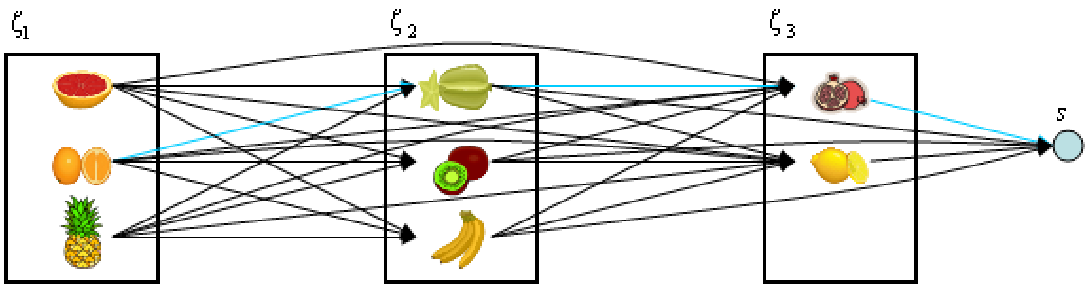

Figure 17: Structure of G^1

problem in a DAG in [7]. Running this algorithm to find a shortest path finds a cycle of products because s is identified with the first node on the path. By construction, the path must contain at least one product node besides the starting node p_0 and may contain up to $|Z|$ total product nodes. It cannot find self loops.

The DAG-shortest-paths algorithm works on a topological sort of the vertices. For each permutation, the edges from a particular vertex in the topological sort is the same. Only the locations of individual products in the topological sort changes. Therefore, to efficiently implement G^1, we construct the graph on the topological sort. The vertices and edges need only be defined once. However, the edge weights must be reevaluated each time the locations of products in the topological sort change.

On G^1, we search a restricted set of cyclic exchanges. For any starting node p_0 in zone ζ_i, we consider cyclic exchanges that include p_0 and products from all subsequent zones, but not products from zones preceding ζ_i. At most one product from each zone subsequent to ζ_i can be in the cycle, and the products included must be in ascending zone permutation order. In our algorithm we consider three cases for p_0. The first case lets the starting node p_0 be one particular product in ζ_1. The second case expands upon the first by letting p_0 be each possible product in ζ_1. The third case set expands upon the second by allowing p_0 to be a product from any of the first $|Z| - 1$ zones in the permutation. In the first case, we run the DAG shortest path algorithm once. In the second case, it runs $|\zeta_1|$ times. In the last case it runs $|P| - |\zeta_{|Z|}|$ times.

We have discussed the main issues involved in finding a good improving cycle. Now we

```
Cyclic Exchange

While found an improving cycle in last permutation or
        did not let the starting zone be any zone in last permutation
    Construct $G^0$ and run Permute
    Construct $G^1$
    If permutation is different from previous one
        Evaluate edge weights in $G^1$ for all edges not incident to origin
        While Find Cycle($\zeta_1$)
            Perform cyclic exchange
    Else
        For $i = 2$ to $|Z| - 1$
            Find Cycle($\zeta_i$)
        If cycle was found
            Perform cyclic exchange
    End if . . . else
End while


Find Cycle($z$)

For each product $p$ contained in $z$
    Set origin to $p$ and evaluate edge weights in $G^1$ for all edges incident to $p$
    Run DAG-shortest-paths
    If the path found has weight smaller than current cycle
        Set cycle to the one defined by this path
End for
If cycle has negative weight, return True
Else, return False
```

Figure 18: Cyclic Exchange Algorithm

present a summary of the algorithm in Figure 18 and discuss the implementation details. We use the second case for p_0 and perform the exchange indicated by the minimum weight cycle so long as the cycle weight is negative. Observe that when we vary the starting node p_0, all arcs in G^1 incident to p_0 must be updated. This includes both arcs of the form $w_{p_0 p}$ and w_{pp_0} for $p \neq p_0$. Also observe that if the cycle found does not include all zones, we can run the minimum weight cycle algorithm on the remaining zones to find an additional cycle. We do not implement this latter idea because it is unlikely to lead to significant additional improvement.

If the minimum cycle weight is nonnegative, we look for a new zone permutation. Cycles are then evaluated in the graph G^1 defined by this permutation. If no cycle can be found for a particular permutation, we extend the search space to the third one described. If a cycle is found in this search space, we return to the second search space. If no cycle is found in the largest search space, the algorithm terminates. As with 2-exchanges, the algorithm is guaranteed to finitely terminate because only improving exchanges occur.

Prior to terminating, there are several other neighborhoods one might try short of examining all permutations which would find any improving simple cycle. We might try to reverse the direction of all the arcs in the graph. Or we could pick a point in the permutation and move all zones prior to this point to the end of the permutation. The experimental results indicate that most improvements in the cyclic exchange algorithm come without expanding the search space to the largest neighborhood. Therefore, it seems unlikely that these approaches will significantly improve the objective while they will certainly increase the running time.

4.2.2.1 Size of Search Space

Now we will consider how large a search space our cyclic exchange algorithm examines. For a given permutation of zones, it considers all exchanges in which products from an earlier zone can be exchanged with products in a later zone. At most one product from any one zone may move, and one particular product from the first zone must be involved in the exchange.

Theorem 4.2 *Starting from a single starting product, the number of cycles in the search space is*

$$\prod_{i=2}^{|Z|}(|\zeta_i| + 1) - 1$$

Proof First, we comment that the $(|\zeta_i| + 1)$ term has the $+1$ to count exchanges of a product from ζ_i with a zone other than ζ_{i+1}. The -1 ensures that we do not count the self loop associated with moving the starting product to the dummy node.

The proof is by induction on $|Z|$. In the base case, $|Z| = 2$, it is clear that the starting product can exchange locations with any product in the second zone. Now we demonstrate this:

$$\prod_{i=2}^{|Z|}(|\zeta_i|+1)-1=(|\zeta_2|+1)-1=|\zeta_2|$$

Now, assume that the statement is true for $|Z|=k$. For $|Z|=k+1$ we have:

$$\left(\prod_{i=2}^{k}(|\zeta_i|+1)-1\right)+\left(\prod_{i=2}^{k}(|\zeta_i|+1)-1\right)|\zeta_{k+1}|+|\zeta_{k+1}|$$

where the first term is the number of possible cycles ignoring ζ_{k+1}, the second is the number of possible cycles using some other zones along with ζ_{k+1}, and the third is the number of possible cycles using only ζ_{k+1}. This can be rewritten as:

$$\begin{aligned}
& \left(\prod_{i=2}^{k}(|\zeta_i|+1)-1\right)(|\zeta_{k+1}|+1)+|\zeta_{k+1}| \\
= & \prod_{i=2}^{k+1}(|\zeta_i|+1)-(|\zeta_{k+1}|+1)+|\zeta_{k+1}| \\
= & \prod_{i=2}^{k+1}(|\zeta_i|+1)-1
\end{aligned}$$

which completes the proof. □

We then have the following corollaries:

Corollary 4.1 *There are $O\left((|P|/|Z|)^{|Z|}\right)$ cycles starting from a single starting product.*

Proof Since ζ_1 can be any zone so to maximize the product from Theorem 4.2 we would like to solve:

$$\begin{aligned}
\max \quad & \prod_{i=1}^{|Z|}(|z_i|+1) \\
\text{s.t.} \quad & \sum_{i=1}^{|Z|}|z_i|=|P| \\
& |z_i|\geq 0, \quad i=1,\ldots,|Z|
\end{aligned}$$

From [6] we know that when a_i are positive real numbers, $\prod_{i=1}^{m}a_i \leq (\sum_{i=1}^{m}a_i/m)^m$ with equality when $a_i=a_j\ \forall i,j$. Therefore for this problem the optimal solution sets $|z_i|=|z_j|\ \forall i,j$ which gives an objective value of $(|P|/|Z|+1)^{|Z|}$. Theorem 4.2 excludes one zone from the product so the maximum value is $(|P|/|Z|+1)^{|Z|-1}$. Also, in general, $|z_i|=|z_j|\ \forall i,j$. Therefore there are $O\left((|P|/|Z|)^{|Z|}\right)$ cycles. □

Corollary 4.2 *If the $|\zeta_i|$ are balanced, that is $|\zeta_i| \approx |P|/|Z|\ \forall i$, then there are $(|P|/|Z|+1)^{|Z|-1}-1$ or $\Theta\left((|P|/|Z|)^{|Z|-1}\right)$ cycles.*

Observe that Corollary 4.2 shows that the search space is exponential in the number of zones. We also note that in typical warehouses $|P|/|Z| > |Z|$ so the number of cycles is greater than $|Z|^{|Z|-1}$.

In the situation where no cycle can be found from the first zone in the permutation, we search a larger search space. Here we let the starting zone be any zone except the last one. From Theorem 4.2 it is easy to see that the search space in this situation is:

$$\sum_{j=2}^{|Z|}\left(\prod_{i=j}^{|Z|}(|\zeta_i|+1)-1\right) = \sum_{j=2}^{|Z|}\prod_{i=j}^{|Z|}(|\zeta_i|+1) - |Z| + 1$$

If we assume as before that the $|\zeta_i|$'s are balanced and that $|P|/|Z| > |Z|$, then the number of cycles is

$$\begin{aligned} & \sum_{i=1}^{|Z|-1}(|P|/|Z|+1)^i - |Z| + 1 \\ > & \sum_{i=1}^{|Z|-1}(|Z|+1)^i - |Z| + 1 \\ = & \sum_{i=2}^{|Z|-1}(|Z|+1)^i + 2 \end{aligned}$$

4.2.2.2 Running Time per Iteration

In Section 4.2.1.2 we argued that an edge weight can be computed in constant time. To populate graph G^1, $\Theta(|P|^2)$ edge weights must be computed as a setup cost. We also must run the Permute algorithm. Given the edge weights $w_{p_1p_2}$, we can compute the weights $\omega_{z_1z_2}$ for G^0 in $\Theta(|P|^2)$ time. The first step in Permute is to determine the most negative edge weight. This requires $|Z|(|Z|-1) = \Theta(|Z|^2)$ evaluations and adds two zones to the permutation. Successive steps in Permute add the most negative edge incident to the last zone in the permutation. When there are k zones still to be added to the permutation, this requires k evaluations. Since this must be done $|Z|-2$ times, there are $\sum_{k=1}^{|Z|-2} k = |Z|(|Z|+1)/2 - 2 = \Theta(|Z|^2)$ total evaluations. Hence the total run time for Permute is $\Theta\left(|P|^2 + 2|Z|^2\right) = \Theta(|P|^2)$ since $|Z| < |P|$.

In Section 4.2.2 we stated that we can search for the best exchange in a graph by solving the shortest path problem in $\Theta(V+E)$ time. For our graph $V = |P|+1$ and E is $\Theta(|P|^2)$ so the shortest path problem takes $\Theta(|P|^2)$ time to solve. In total, for a particular starting product we can find an improving cycle in $\Theta(|P|^2)$ time. In the case where no improving cycle is found from a product in zone ζ_1, we perform $|P| - \zeta_{|Z|}$ runs. Hence the total search time in this case is $\Theta\left(|P|^2(|P| - \zeta_{|Z|})\right) = \Theta\left(|P|^3\right)$. In either case we search an exponential space in polynomial time.

The difference in run time from the 2-exchange algorithm is that here it is $\Theta(|P|^2)$ starting from the first zone whereas for 2-exchanges it is $O(|P|^2)$. Another difference between the two algorithms is that the 2-exchange algorithm searches all possible 2 exchanges. The cyclic exchange algorithm searches all 2-exchanges as well as all cycles that follow the zone permutation as described in Section 4.2.2.

4.2.3 Cyclic Exchange: Null Moves

Here we extend the neighborhood defined above to include unused storage spaces in a zone. Previously, every cycle moved products from one zone to the next. That is, we were able to represent a cycle involving n zones as $C = (p_1, p_2, \ldots, p_n, p_1)$ where $p_i \in P\ \forall i$. Now we allow a cycle to include moves involving empty slots within a zone provided they exist (Figure 19). While the above representation of a cycle is still valid, we now include at most one node $p \notin P$ for each zone. We restrict our search to cycles where for each arc (p, q), at least one of p and q is in P. That is, we eliminate cycles that move an empty node to another empty node. We allow the following types of edges (p, q) (Figure 20):

1. $p, q \in P$ - the type of edge allowed previously
2. $p \in P, q \notin P$ - moves a product to an empty node
3. $p \notin P, q \in P$ - allows the cycle to continue with a product after including an empty node

Including null exchanges allows for path exchanges ($p_i \in P\ \forall i \neq 1$), standard cyclic exchanges ($p_i \in P\ \forall i$), and exchanges with multiple null moves.

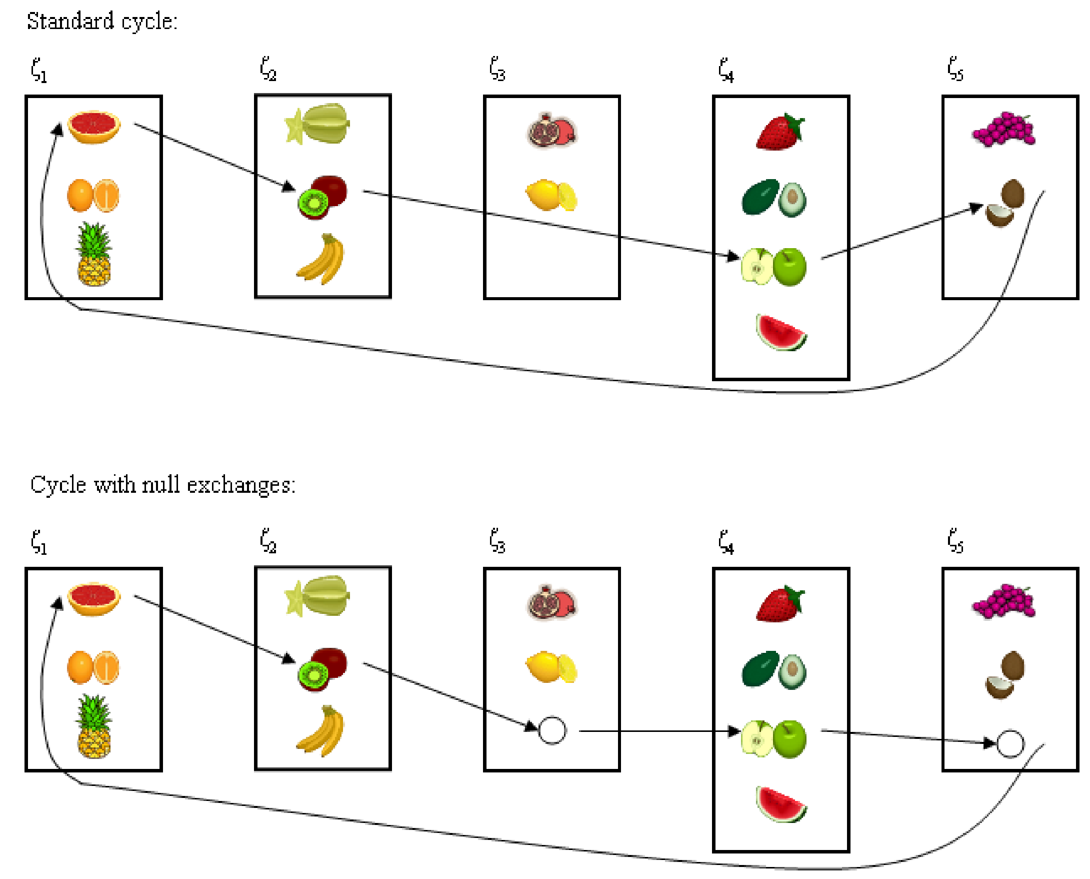

Figure 19: Illustration of Standard Cycle vs. Cycle with Null Exchanges

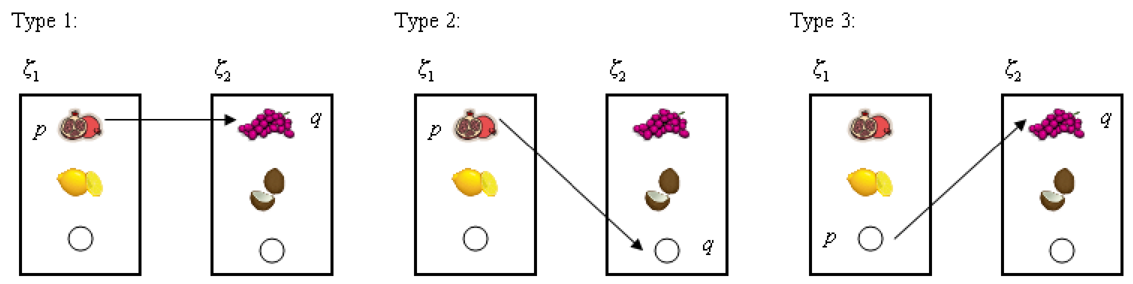

Figure 20: Possible Edges when Null Exchanges are Permissible

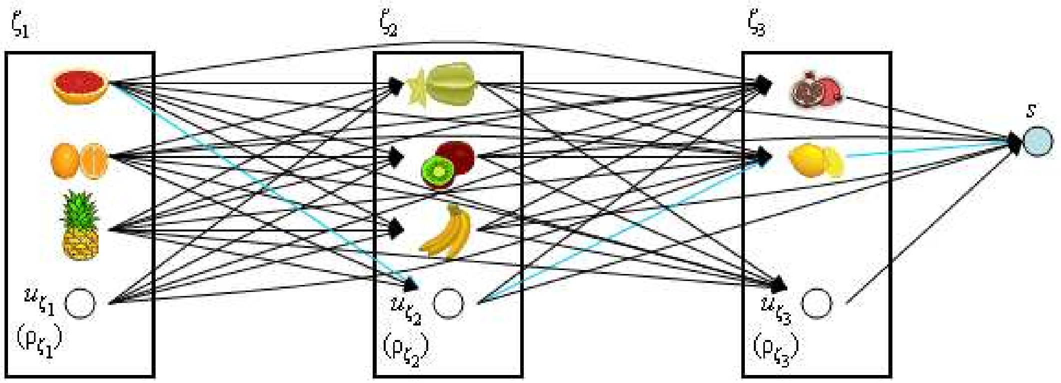

Figure 21: Structure of G^N

When we defined G^1 previously, we had a node for each product and a downstream arc between any pair of products in different zones. Structurally, we were able to create the node and arc structure once because a cycle only changed the product represented by a particular node. Now that we allow null moves, the number of products in a zone may change thereby changing the arc structure. To avoid potentially having to modify the graph each time an improving cycle has been detected, we define an extended skeletal graph. For each zone we have a node for each unit of capacity rather than for each product as in G^1. Aside from the extra vertices, the graph structure is identical. To determine an improving cycle, a subgraph of this skeletal graph is used which we refer to as G^N (Figure 21).

To form G^N, each product is assigned to a node in its respective zone. Remaining unassigned nodes correspond to empty slots within a zone. For each zone $z \in Z$ we define

u_z = vertex identified with one of the nodes not being occupied by a product

ρ_z = residual capacity of zone z

If $\rho_z = 0$, the zone is full and u_z is ignored as a vertex in G^N. Otherwise it is present and any null exchange involving zone z must go through u_z. This avoids the redundancy of considering a separate null exchange for each empty slot in z. To summarize, G^N contains a node for each $p \in P$, a node u_z $\forall z \in Z$, and the terminal node s. The edge structure is as follows. As in G^1, there is a downstream arc into s for every node not contained in zone ζ_1. There is a downstream arc between each pair of nodes in different zones with the following exceptions: if $\rho_z = 0$, there are no edges incident to u_z; there is no arc between u_z and $u_{z'}$, for $z \neq z'$.

The edge weight between two nodes containing products is the same as before. An edge going from an empty node to a node containing a product has zero weight. For an edge going from a product node to an empty node, the weight computation is as before except that the third computation is unnecessary.

The search for cyclic exchanges on the graph G^N is identical to the search on the graph G^1. By construction, the graph does not allow infeasible null moves. The graph update is unchanged except for one case. If an exchange involving u_{ζ_i} is contained in the best cycle, the node identified with u_{ζ_i} must be changed and ρ_{ζ_i} as well as $\rho_{\zeta_{i+1}}$ must be updated (Figure 22). Since the skeletal graph exists, it is not costly to change u_{ζ_i}.

4.2.3.1 Size of Search Space

As described, we have expanded the search space. Previously we had $|\zeta_i|$ nodes in the i^{th} zone. Now we have $|\zeta_i|+1$ nodes if $\rho_{\zeta_i} > 0$ and $|\zeta_i|$ nodes otherwise. We make the following claim about the number of cycles in the expanded search space:

Theorem 4.3 *The number of cycles in the search space is*

$$\Theta\left(\prod_{i=2}^{|Z|} |\zeta_i|\right)$$

Proof It is clear that there are at least as many cycles as when null moves are not allowed. Hence there are at least

$$\prod_{i=2}^{|Z|}(|\zeta_i|+1)-1$$

cycles in the search space.

If every zone has $|\zeta_i|+1$ nodes, there are fewer than

$$\prod_{i=2}^{|Z|}((|\zeta_i|+1)+1)-1 = \prod_{i=2}^{|Z|}(|\zeta_i|+2)-1$$

cycles because the formula treats the extra nodes as if they are product nodes. However, unlike product nodes, there are no arcs between u_z and $u_{z'}$, for $z \neq z'$. This completes the proof. □

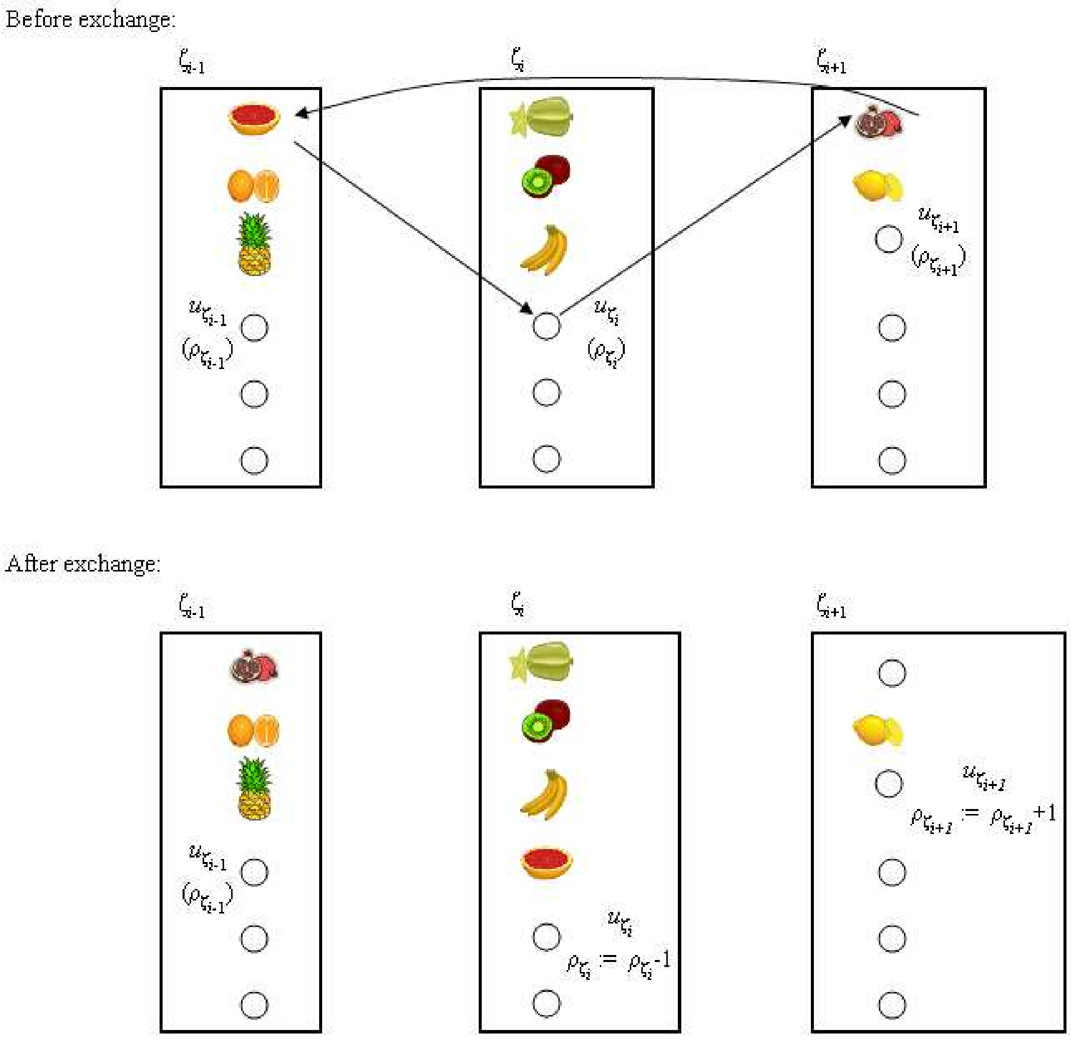

Figure 22: Update of G^N for a Null Move

We have corollaries similar to those in Section 4.2.2.1:

Corollary 4.3 *There are* $O\left((|P|/|Z|)^{|Z|}\right)$ *cycles starting from a single starting product.*

Corollary 4.4 *If the* $|\zeta_i|$ *are balanced, that is,* $|\zeta_i| \approx |P|/|Z|\ \forall i$*, then there are* $\Theta\left((|P|/|Z|)^{|Z|-1}\right)$ *cycles.*

4.2.3.2 Running Time per Iteration

When we have null swaps, the graph can have up to $|P|+|Z|$ vertices. The graph has $\Theta(|P|^2)$ edges between product nodes and $\Theta(|P|)$ edges incident to empty nodes. Therefore, there are still $\Theta(|P|^2)$ edges. Computing individual edge weights still takes constant time and is a setup cost of the problem. The run time for Permute is $\Theta(|P|^2)$ as before. The shortest path problem can still be solved in $\Theta(E) = \Theta(|P|^2)$ time. Therefore, for a particular starting product, it still takes $\Theta(|P|^2)$ time to find an improving cycle. In the case where no improving cycle is found from a product in zone ζ_1, we can perform up to $|P|+|Z|-\zeta_{|Z|}-1$ runs. Hence the total search time in this case is $\Theta\left(|P|^2(|P|+|Z|-\zeta_{|Z|}-1)\right) = \Theta\left(|P|^3\right)$. Here the search times are the same as in the basic case.

CHAPTER 5

GENERALIZATIONS

The algorithms in Chapter 4 focused on the special case where each product only had one pallet, (IP_{1pal}). In this chapter we will generalize these approaches to the models where products may have multiple pallets, (IP_{noSS}), and where stock splitting is allowed, (IP_{general}). For simplicity of discussion, we will assume each order occurs once, $n_r = 1 \; \forall r$, and each zone has unit entry cost, $e_z = 1 \; \forall z$.

5.1 Different Product Sizes

In this section we generalize the product storage requirements so that it may be necessary to store multiple pallets of the same product. We have restricted the situation so that all pallets of a product must be stored in the same zone, i.e. we do not allow stock splitting until Section 5.2. The main change from Chapter 4 where each product required exactly one unit of storage is the feasibility of product to zone assignment. Instead of ensuring that the number of products in a zone does not exceed the capacity as we did previously, we now must ensure that the volume of products assigned to a zone does not exceed the capacity. A smaller change is that where the algorithms used popularity, they now use COI to adjust for different product sizes. The construction approaches discussed previously are easily extended with these modifications. Below, we discuss in detail how the improvement methods can be generalized.

5.1.1 2-Exchange

Previously when doing an exchange, feasibility was not an issue. Any product could fit in the storage space of any other product, so any exchange was feasible. Now it is possible that a large product may be unable to fit in a zone replacing a small product. As we do when cyclic exchanges allow null swaps, for each zone we must keep track of the residual capacity, ρ_z in order to check feasibility. Recall from Section 2.2 that v_i is the number of

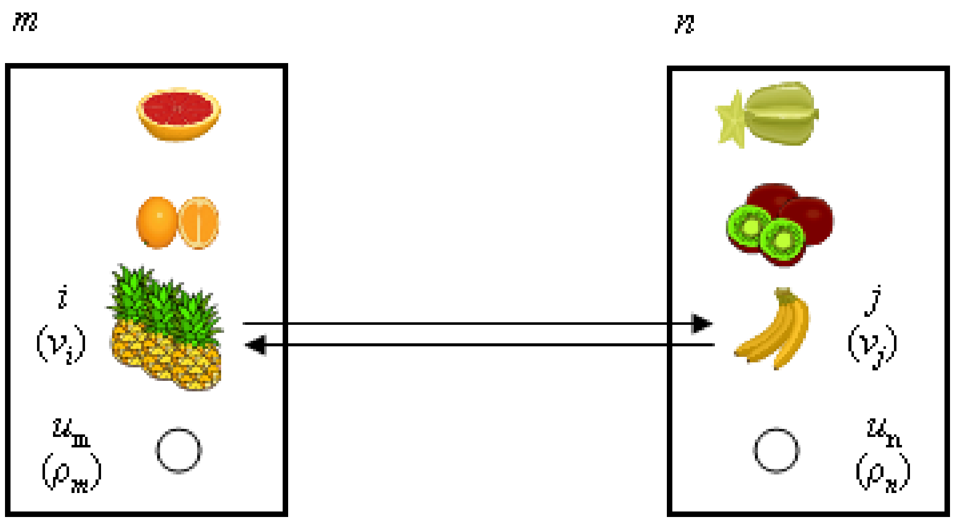

Figure 23: 2-Exchange Move when Products have Different Sizes

pallets of product i. An exchange involving product i from zone m and product j from zone n will be feasible if there is sufficient space in both zones, i.e. $v_i + \rho_m \geq v_j$ and $v_j + \rho_n \geq v_i$ (Figure 23). We can extend the 2-exchange approach to incorporate null exchanges, that is, moves of a single product to a different zone exchanging it with empty space. When one node is an empty space, the feasibility test is the same except that $v_i = 0$ for the empty node (Figure 24). Other than the modification of the feasibility test and updating ρ_z at each iteration, the algorithm proceeds as discussed in Section 4.2.1.

The search space includes any feasible exchange of two products from different zones. There are $O(|P|^2|)$ such moves. In addition, any product may move into a different zone with sufficient extra capacity. There are $\Theta(|P| * |Z|)$ such moves. Hence the search space in this case is $\Theta(|P|^2|)$ like before. Since the algorithm proceeds as in Section 4.2.1, and we must still consider at most $\Theta(|P|^2)$ possible moves to find an improving exchange, the running time is $O(|P|^2)$ as before.

5.1.2 Cyclic Exchange

We can treat cyclic swaps of different sized products as generalizations of cyclic swaps with null moves. We define G^N as in Section 4.2.3 (Figure 25). That is, we maintain the residual

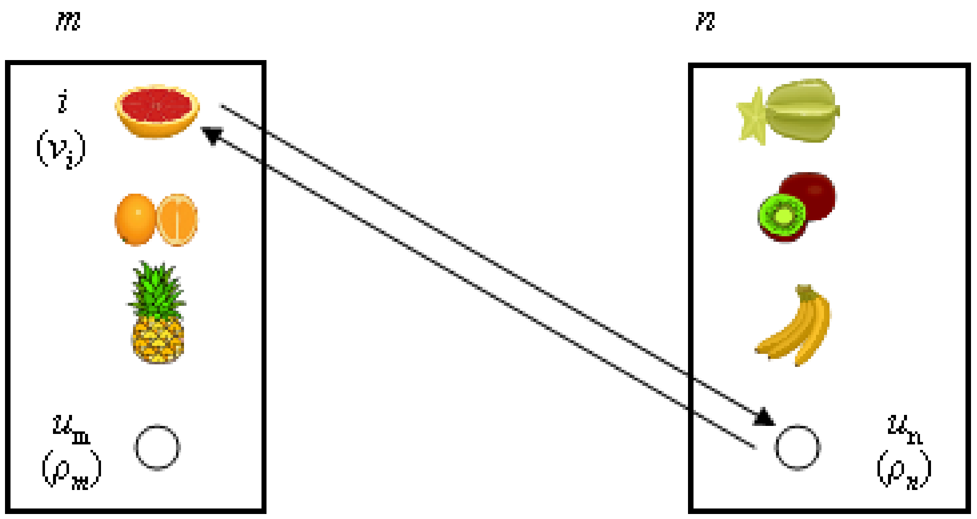

Figure 24: 2-Exchange Move with Empty Spaces

capacity of a zone, ρ_z, a node corresponding to extra capacity in a zone, u_z, and identify one underlying node with each product. The arc structure is also identical. The only difference, as with 2-exchanges is that certain edges will be infeasible due to the lack of space for different product sizes. For an arc replacing product j in zone n with product i in zone m, it is feasible if $v_j + \rho_n \geq v_i$. If the node from zone m corresponds to unused capacity u_m, the arc is feasible. If the node from zone n corresponds to unused capacity u_n, the arc is feasible if $\rho_n \geq v_i$. We do not allow arcs where both nodes represent empty slots. In this slightly modified graph with the infeasible arcs eliminated, we search for a cycle on G^N as before. To update the graph, we can follow the same approach as in the case with null swaps.

The search space and algorithm are the same as the cyclic exchange method with null moves from Section 4.2.3 except that some additional arcs are infeasible. The removal of these arcs does not effect the complexity of the search space, so it still contains $\Theta\left(\prod_{i=2}^{|Z|} |\zeta_i|\right)$ cycles. The complexity of the running time is also not effected by the removal of these arcs so it remains $\Theta\left(|P|^3\right)$.

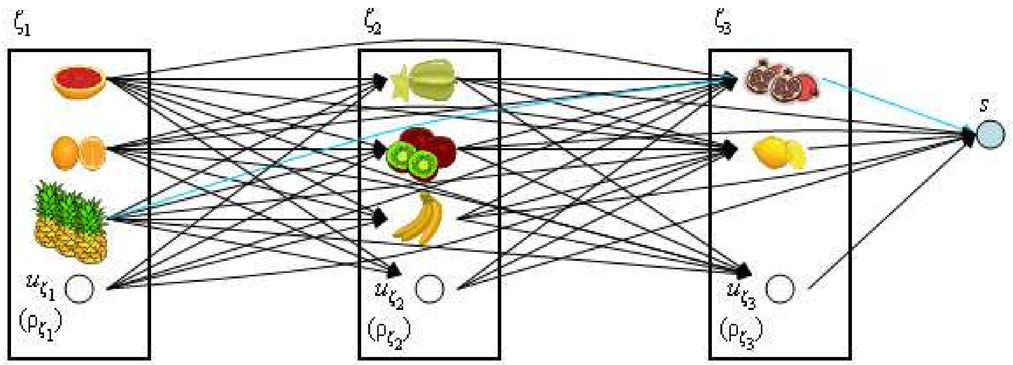

Figure 25: Cyclic Exchange Graph with Products of Different Sizes

5.2 Stock Splitting

The second modification we consider generalizes the case where one may have multiple pallets of the same product to allow stock splitting, i.e. pallets of the same product each stored in different zones. Stock splitting may be particularly beneficial for products that commonly occur with disjoint groups of products. In addition, stock splitting gives flexibility as to which zones are visited to fill an order.

Aside from [23], previous authors have not discussed stock splitting in the correlated storage assignment problem. It is more complicated to generalize the approaches of Sections 4.1.3 and 4.1.4 in this case. Allowing stock splitting extends the range of feasible assignments. However, we can still use a non-stock-splitting approach to construct a feasible solution and use our improvement methods to rearrange and split products in favorable ways.

Before discussing these methods in further detail, we must address one major complication that arises when we allow stock splitting. When all pallets of a product are stored together, it is not difficult to compute an order's contribution to the objective. One simply visits all zones that contain some product from the order. By allowing stock splitting we now potentially have a choice of which zones to visit. If some product has multiple pallets each stored in a different zone, we can now choose from which zone we wish to retrieve that product. Given a choice, we would like to visit as few zones as possible. That is, for each order $r \in R$ we now have the following set covering problem.

$$\begin{aligned}(IP_{\text{set cover}})^r = \quad \min \quad & \sum_{z \in Z} y_z \\ \text{s.t.} \quad & \sum_{l \in L : p_l = p} y_z \geq 1 \quad \forall p : (r,p) \in D \\ & y_z \text{ binary} \qquad \forall z \in Z\end{aligned}$$

where

$y_z = 1$ if zone z is visited to fill the order; 0 otherwise

The formulation minimizes the number of zones visited. For each product, at least one zone must be visited that contains a pallet of that product.

There are $|R|$ such problems and they are each NP-Complete. However, when the average order is small, these are small problems. We show in Chapter 6 that the computation time needed to solve these problems is reasonable for a large real world data set.

We now describe how we modify construction and improvement heuristics to obtain good stock-split solutions to the problem.

5.2.1 Random Assignment

When stock splitting is permitted, we use random assignment to place each pallet independently regardless of the product it contains. Each pallet is randomly assigned to a zone so long as the zone has remaining capacity. Thus, the initial solution allows for stock splitting. As before, if the zone is full, a different zone to store the product is randomly generated. This continues until all pallets have been assigned.

5.2.2 Popularity

The popularity heuristic proceeds much as before except that we exploit the ability to split products. Products are ranked from greatest COI to least COI and the zones are ordered $\zeta_1, \ldots, \zeta_n$. We proceed down the product list placing products in the active zone until it reaches maximum capacity. Then we move onto the next zone. In this case, if a product has more than one pallet of storage so that it is eligible for stock splitting we do not place all pallets in the active zone. The first pallet is placed in the active zone with each subsequent

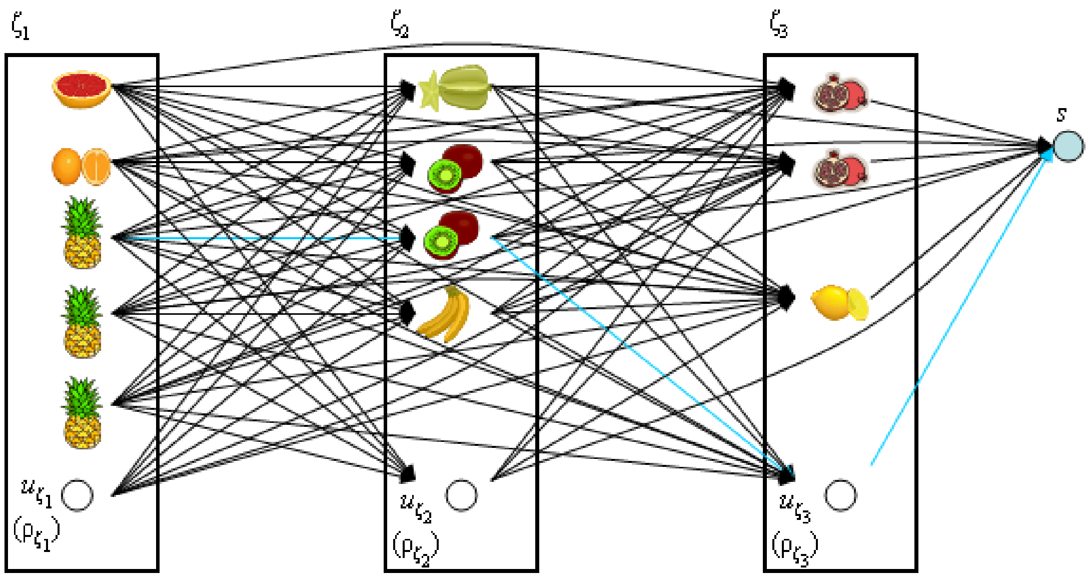

Figure 26: Cyclic Exchange Graph when Stock Splitting is Permitted

pallets placed in the successive zones in the zone ordering. If pallets remain after the final zone is reached, they are placed in the active zone subject to capacity. This approach ensures that at any point in the algorithm, for zones ζ_i and ζ_j with $i < j$, the utilized space of zone ζ_i is greater than or equal to that of zone ζ_j.

5.2.3 Cyclic Exchange

When stock splitting is permitted, we can use the cyclic exchange procedure to take either a split or non-split solution and generate a solution with stock splitting. The basic graph structure is the same as G^N where null moves are allowed but there is no stock splitting. The main difference between this case and the previous cases is that we now identify a node in the graph as containing a pallet rather than a product (Figure 26). This enables us to separate the pallets of a product since each pallet is now a logical unit.

Treating each pallet as a unit does not effect the presence of arcs in the skeletal graph underlying G^N but does impact the effective graph. Feasibility was complicated when we allowed products to have different storage sizes but moved all pallets of a product together. Here feasibility reverts back to the simpler case of null swaps with each product occupying one unit of space. The feasibility check is to verify that $\rho_z > 0$.

We have mentioned that stock splitting complicates the computation of the objective. This main difficulty for cyclic swaps is in the computation of the arc costs. When moving a pallet from zone i to zone j, we can no longer say with certainty that the product associated with that pallet was previously picked from zone i and is now picked from zone j. If we wanted to be able to make such a statement, we would have to solve $(IP_{\text{set cover}})^r$ $\forall r \in R$ before executing the move and again for the system resulting from the entire series of swaps in a particular cycle. This is too expensive to do for each potential cycle.

Our solution is to use arc costs so that executing improving cycles found with these costs improves the objective. Let us define the following data:

z_l	$=$ zone to which pallet l has been assigned
l_{rp}	$=$ pallet used to retrieve product p from order r
s_{rz}	$=$ number of products from order r that are picked from zone z
w_{ij}	$=$ cost of arc that moves pallet i to the location occupied by pallet j, assuming that l_{rp} does not change when this move is made

Of particular note is l_{rp}. By tracking this piece of data, we make the decisions specified by $(IP_{\text{set cover}})^r$.

We can solve $(IP_{\text{set cover}})^r$ $\forall r \in R$ before executing any cyclic swaps and reevaluate it after every iteration. However, this is computationally intensive, so in practice we only want to reevaluate it every k iterations, for some fixed parameter k.

Theorem 5.1 *Suppose we have used our cyclic exchange procedure to identify a sequence of k cyclic swaps since the last solution of $(IP_{\text{set cover}})^r$ $\forall r \in R$. If we have used w_{ij} as arc costs in the cyclic exchange graph, then executing the sequence of k cyclic exchanges will improve the solution.*

Proof Let x be the initial assignment of pallets to zones.

Case 1: $k = 1$, i.e., we have just solved the set covering problems. Therefore, we have calculated l_{rp}, the optimal pallets to pick each product for each order, with true cost z.

The conservative edge weights, w, represent the edge weights if l_{rp} does not change, i.e. one picks the same pallets that were selected in the set covering problems. If an improving cycle C was detected using these edge weights, then $v(C) < 0$ where $v(C)$ is the weight of the cycle. Therefore, if one picks the same pallets for the orders, i.e. , uses the same l_{rp}, the new solution x' has cost $z' = z + v(C) < z$. However, we might do better if we re-solve the set covering problems to obtain l'_{rp}, the optimal pallets to pick each product in each order, with true cost z''. Since l_{rp} with value z' is feasible to the set covering problems, $z'' \leq z'$. Therefore $z'' \leq z' < z$ and C is an improving cycle under real edge weights as well as for conservative edge weights.

Case 2: Since last solving the set covering problems to obtain l_{rp}, we have found $k > 1$ improving cycles, $C_1, C_2, \ldots, C_k$, using conservative edge weights. The solution resulting from executing all the cycles, x', has cost $z' = z + v(C_1) + v(C_2) + \ldots + v(C_k)$. Since these cycles were improving under the conservative edge weights, $v(C_i) < 0$ $i = 1, \ldots, k$, we have

$$\begin{aligned} z &> z + v(C_1) \\ &> z + v(C_1) + v(C_2) \\ &\vdots \\ &> z + \sum_{i=i}^{k} v(C_i) \\ &= z' \end{aligned}$$

When we re-solve the set covering problems to obtain l'_{rp}, the optimal pallets to pick each product in each order, it has value z''. Since l_{rp} is feasible to the set covering problems, $z'' \leq z'$. Therefore $z'' \leq z' < z$ and we have improved the solution since last solving the set covering problems under both conservative and real edge weights. □

Theorem 5.1 shows the benefit of solving the set covering problems more often. Case 1 shows that for $k = 1$, re-solving the set covering problem after every improving cycle, we are guaranteed to improve the objective. Case 2 demonstrates that multiple cycles collectively improve the objective with no guarantee that each one individually improves the objective. Therefore, for smaller values of k, it is more likely that each cycle improves the objective.

Now we will describe how to compute w_{ij} and update s_{rz}. For arc $(i, j), \forall r \in R : (r, p) \in D, p_i = p, l_{rp} = i$ let $s_{rz_i} := s_{rz_i} - 1$ and $s_{rz_j} := s_{rz_j} + 1$. Start with $w_{ij} = 0$ and do the following:

1. If $s_{rz_i} = 1$, let $w_{ij} := w_{ij} - 1$
2. If $s_{rz_j} = 0$, let $w_{ij} := w_{ij} + 1$
3. If $s_{rz_j} = 1$, $(r, p_j) \in D$, and $l_{rp_j} = j$, let $w_{ij} := w_{ij} + 1$

The update is similar to the update from Section 4.2.1 where there was no stock splitting. We consider orders r that contain p_i and use pallet i to retrieve p_i for the order. In the first computation, if p_i is the only product from r contained in z_i, we no longer need to visit the zone to fill the order so the objective improves by one. In the second computation, if z_2 contained no pallet of a product used to fill order r, we now must visit this zone to retrieve p_i so the second computation increases the objective by one. The third computation adjusts for the case where p_j is the only product from r contained in z_j and pallet j is used to fill product p_j for the order. In this case, the weight for the edge from p_2 will mistakenly give a benefit for order r. Since any cycle has both an inbound and an outbound edge to p_2, the third computation offsets this case.

This update may not give the best choice of l_{rp} because moving pallet i from zone z_i to z_j may result in the selection of p_i from an entirely new zone for the order. It is also possible that all other products from the order currently picked from z_i should also be picked from zone z_j. That is, we should set $l_{rp'} = z_j$ for every p' such that $(r, p') \in D$ and $l_{rp'} = z_i$ and not just for p.

Once we have determined the costs for each each arc in the graph, the cyclic swap heuristic proceeds in the same way as it did when we did not have stock splitting. Note that to reduce the search space, we consider only the first pallet of a product in a particular zone for edge cost computation and for the DAG algorithm; every other pallet would have exactly the same set of improving edges.

5.2.3.1 Size of Search Space

The search space is similar to the case of cyclic swaps with null moves. The difference in this case is that we redefine $|\zeta_i|$ to be the number of pallets stored in the i^{th} zone instead of the number of products. After this definition modification, the search space still contains $\Theta\left(\prod_{i=2}^{|Z|} |\zeta_i|\right)$ cycles.

5.2.3.2 Running Time per Iteration

When the values of $l_{rp} \forall (r,p) \in D$ are given, the running time for the cyclic swap algorithm is the same as for cyclic swap with null moves. Only the arc cost computation has changed, and it is of the same difficulty as it was before. Therefore, for given l_{rp} the total search time is $\Theta\left(|P|^3\right)$. As stated in Section 5.2, solving $(IP_{\text{set cover}})^r \ \forall r \in R$ requires solving $|R|$ small NP-Complete problems. So iterations which require solving $(IP_{\text{set cover}})^r \ \forall r \in R$ take $O(|R| 2^{s^*})$ time while the other iterations take $\Theta\left(|P|^3\right)$ time.

5.3 Rewarehousing

In Section 2.4 we showed how to extend the formulations to incorporate a rewarehousing cost. Here we briefly discuss how to include the rewarehousing cost in the improvement techniques. To search for an improving cycle, we evaluated edge weights w_{ij} which indicate the change in objective from moving pallet i to the location of pallet j. From this value we will subtract the rewarehousing cost, c_m, if the pallet is currently stored in the incumbent location. When a pallet has moved and we wish to consider returning it to its incumbent location, we add c_m. With this minor modification of the edge weights, the algorithms proceed exactly as described earlier.

CHAPTER 6

COMPUTATIONAL RESULTS

In this chapter, we present computational results for the algorithms we discussed in Chapters 3, 4, and 5.

6.1 Data

The data set used to test the model was obtained from a potential client of a global third-party logistics provider who cannot be revealed due to confidentiality. The data for this warehouse has 12,845 products, 293,404 orders, and 508,006 distinct order-product pairs. Of this total, 10,644 products are used in 74,202 multi-product orders for a total of 288,870 order-product pairs. Hence there are an average of just fewer than four products per multi-product order. We treat duplicate orders that come at different times as separate for the purposes of order picking.

The data was in a MS Access database which we cleaned up using queries. Since there was no information about the number of zones, we assumed that there were 40 zones for the complete data set as this is a common number for warehouses of this size. We also assumed that all zones had equal capacity, and that the overall system had 10% excess capacity spread equally across all the zones. Excess storage capacity is common in warehouses. In our case it is required for the products which do not appear in multi-product orders. We also considered subsets of the complete data set. The problem sizes are summarized in Table 1.

For purposes of comparison, Table 2 summarizes the problem sizes of some of the work discussed in Section 1.4. Observe that the problem sizes we consider are significantly larger than those reported in the literature. Only [21] gives results for problem sizes close to ours.

Table 1: Data Used in Computational Tests

Products	Multi-product orders	Line items	Zones
100	711	1,922	10
1,000	14,308	44,540	20
2,000	21,827	74,089	20
5,000	52,131	199,035	30
10,644	74,202	288,870	40

Table 2: Problem Sizes of Previous Authors

Author	Products	Orders	Zones
Frazelle [10]	n/a	5,000	300
Sadiq [23]	300	n/a	n/a
Rosenwein [21]	1,000	75,000	60
Amirhosseini et al. [4]	500	1,000	n/a
Liu [18]	20	n/a	n/a

6.2 Larger Data Sets

Recall from Chapter 4 that to compute the edge weights (p, q) in the improvement graphs, the popularity of a product, and the pairwise popularity of p, q we had to consider all orders containing a product. Also, to update the multipliers in the Lagrangian relaxation approach we had to consider all orders (Chapter 3). It is possible for the order history to contain many millions of orders. In such cases, one may consider a subset of the order history containing ”good” orders so as to speed up the algorithms. Further good orders could then be included during the construction or improvement algorithm being used. Leaving out orders gives an objective below the actual value but does not affect the feasibility of an assignment.

There are several ways to select a good subset of orders. One approach is to consider which orders will net the largest savings by being added to the algorithm. To do this, we can use the probabilities from Section 4.1.1 to determine the expected number of zones necessary to fill an order if the algorithm does not consider it. This value is compared to a target number of zones that likely can be obtained if the order is considered in the algorithms. To implement this approach, one must estimate reasonable targets.

Another idea to consider only those orders containing products that are frequently ordered together. To determine such products we can use clustering approaches. However, this alone may not be sufficient. If two items are frequently ordered together but almost always occur with additional items, then putting these items in the same zone is not enough because one will have to leave the zone to retrieve the other items. Hence, one needs to consider how often the products, if placed together, will come close to filling an order [4]. That is, we would divide the number of times the products occur together by the average order sizes in which the products occur. There are many other methods to determine candidate products. Once the candidate products have been determined, orders containing those products are added to the algorithm.

These methods were not required for the data in this study.

6.3 Results

We implemented all our algorithms in the C programming language with use of the CPLEX callable library. In addition, as discussed in Chapter 4, we adapted the algorithms of [10] and [4]. We refer to Amirhosseini and Sharp's algorithm by just Amirhosseini in the tables. The performance of their algorithm is presented for each of the six correlation measures they propose. In the tables, the best performance in each column is in bold. The experiments were run on a Dual Xeon 2.4 GHz machine with 2GB of RAM.

6.3.1 One Pallet for Each Product

In our first computational experiment, we tested the basic case where each product requires a single pallet of storage. Tables 3, 4, 5, and 6 present the number of zones visited for each data set using each algorithm. To compute the expected number of zones using random assignment for the 100 product data set, we use the information in Tables 8 and 9. Similar information is used for the other data sets.

Limited results are presented for the full data set due to limits on machine memory. The clustering approaches have to store information about 113,284,092 product pairs which requires more memory than is available on our machines. The Lagrangian relaxation (L) has 10,684 constraints, 425,760 x variables, 2,968,080 y variables, and 11,554,800 μ values.

Table 3: Comparison of Heuristics: 100 Products

Algorithm		Initial	2-Exchange	Cyclic	Cyclic: Null Moves
Random		1731	948	794	784
Popularity		888	821	806	806
Lagrangian		846	791	780	793
Frazelle		907	837	795	818
Particle		1202	959	794	799
Amirhosseini	1	785	780	**769**	**769**
	2	1069	831	812	812
	3	**775**	**775**	775	775
	4	787	780	776	776
	5	790	787	782	783
	OSCM	786	777	775	771
Upper Bound		1922			
Lower Bound		711			
Optimal		764			

The optimal solution for the 100 product problem requires 3.5 weeks to terminate. Therefore it is unlikely that the larger problems can be solved optimally.

6.3.1.1 Construction Heuristics

We observe that the first and third correlation measure perform best for Amirhosseini and Sharp's algorithm, and outperform each of the other algorithms as well. Note that these measures are not the new one they define, OSCM. It is also interesting that following this algorithm, the order of the best performers is: Popularity, Lagrangian relaxation, Frazelle, Particle, Random. Random assignment is worst as expected. One would expect popularity to perform poorly because it does not consider any notion of product correlation but it does reasonably well. The particle heuristic performs unexpectedly poorly. Due to this and its sensitivity to its parameters, we do not give results for larger problem sizes. Frazelle's heuristic is motivated by a travel distance objective rather than minimizing the number of zones visited. It is possible that this hurts its performance when using our objective. The Lagrangian relaxation method's performance is sensitive in the parameters used. Therefore, we might be able to improve its performance with more experimentation on its parameters.

Table 4: Comparison of Heuristics: 1000 Products

Algorithm		Initial	2-Exchange	Cyclic
Random		41071	26608	22830
Popularity		25808	24839	22615
Lagrangian		29756	26606	22294
Frazelle		31635	28875	22366
Amirhosseini	1	**22462**	**22359**	22226
	2	26240	25229	22502
	3	22948	22391	**22184**
	4	31215	28791	22230
	5	31400	28351	22241
	OSCM	25517	24144	22241
Upper Bound		44540		
Lower Bound		14308		
Optimal		n/a		

Table 5: Comparison of Heuristics: 2000 Products

Algorithm		Initial	2-Exchange	Cyclic
Random		66836	37072	35471
Popularity		40166	39156	35973
Lagrangian		47668	38843	35139
Frazelle		56069	51185	34530
Amirhosseini	1	**35260**	**34889**	34409
	2	40736	39438	35900
	3	37694	36508	34668
	4	49504	44852	**34328**
	5	51738	47225	34724
	OSCM	46145	43684	34855
Upper Bound		74089		
Lower Bound		21827		
Optimal		n/a		

Table 6: Comparison of Heuristics: 5000 Products

Algorithm		Initial	2-Exchange	Cyclic
Random		180801	105217	92958
Popularity		114459	111542	97446
Lagrangian		135380	104941	92394
Frazelle		148518	139157	95542
Amirhosseini	1	**99211**	**98202**	94902
	2	114371	111118	96305
	3	112268	108367	**90002**
	4	134612	122697	90113
	5	136979	125591	91513
	OSCM	120856	115139	90762
Upper Bound		199035		
Lower Bound		52131		
Optimal		n/a		

Table 7: Comparison of Heuristics: 10644 Products

Algorithm		Initial
Random		266665
Popularity		**174624**
Lagrangian		215881
Frazelle		n/a
Amirhosseini	1	n/a
	2	n/a
	3	n/a
	4	n/a
	5	n/a
	OSCM	n/a
Upper Bound		288870
Lower Bound		74202
Optimal		n/a

Table 8: Probabilities ($z = 10$)

k	l						
	1	2	3	4	5	6	7
2	0.1	0.9					
3	0.01	0.27	0.72				
4	0.001	0.063	0.432	0.504			
5	0.0001	0.0135	0.18	0.504	0.3024		
6	0.00001	0.00279	0.0648	0.3276	0.4536	0.1512	
7	0.000001	0.000567	0.021672	0.1764	0.42336	0.31752	0.06048

Table 9: Distribution of Order Sizes: 100 Products

k	number of orders, N_k
2	242
3	58
4	16
5	4
6	4
7	1

For the full data set, the Lagrangian relaxation algorithm has a running time of approximately 10 hours while the popularity based heuristic takes under 2 minutes to run.

6.3.1.2 Improvement Heuristics

Both the 2-exchange and cyclic exchange algorithms improve upon the initial solutions found by the construction algorithms. For the same initial solution, the cyclic exchange algorithm always outperforms the 2-exchange algorithm. In addition, it makes arbitrarily bad initial solutions, such as those from random assignment, competitive. Including null moves in the cyclic exchange algorithm has little effect on the algorithm's performance. For this reason, we do not report the performance of this variation for larger problem sizes. We also note that using the inverse optimization approach from Section 3.6 to improve a solution yields no benefit.

The best performance over all instances is from using Amirhosseini and Sharp's construction algorithm with the first correlation measure together with the cyclic exchange improvement algorithm. In particular, comparing the performance of the best solution to popularity, we see an improvement of 13.4% for 100 products, 14% for 1000 products, 14.5% for 2000 products, and 21.4% for 5000 products. Since popularity is a standard measure used in industry, using our approach can lead to significant savings. In addition, the trend indicates that the potential savings increase as the warehouse size increases.

Graphs 27-36 illustrate the change in objective during the run of the cyclic exchange algorithm for each construction heuristic. Each of these graphs starts with an initial sharp decrease of objective. In general, the rate of change decreases after some number of iterations. However, it is difficult to conclude that the cyclic exchange method should terminate once the rate of decrease slows down. Graphs 30, 33, 34, 35, and 36 all have one or more kinks after the graph initially flattens out. Recall that the cyclic exchange algorithm allows the starting product to be any product in any zone if no improving cycle can be found from a product in the first zone in the permutation. The kinks in the graph almost always occur when this happens. Since significantly more work is necessary for these steps, it is undesirable to perform the expanded search at every iteration. We also observe that Graphs 27, 28, 29, and 31 do not have this behavior. With further experimentation, we may find that certain construction heuristics lead to smoother graphs. For smooth graphs, it is easier to identify a break point after which significant additional improvement is unlikely.

The running time of the improvement methods is negligible for the 100 product data set. The cyclic exchange algorithm takes 7-19 minutes for 1000 products. For the 5000 product problem, the 2-exchange algorithm takes 4-39 minutes while the cyclic exchange algorithm takes 23 to 65 hours. The upper bounds for the run time are for the random heuristic. The other construction methods have running times near the lower bound. These run times are not unreasonable since they are for warehouses that are starting with an empty system. Warehouses completely reconfigure on a seasonal rather than daily basis. Hence a running time of several days is acceptable. For the 5000 product problem, our implementation of the 2-exchange algorithm and the cyclic exchange algorithm require on the order of 1GB of memory. This limits the problem sizes that can be solved using the current implementation of these methods.

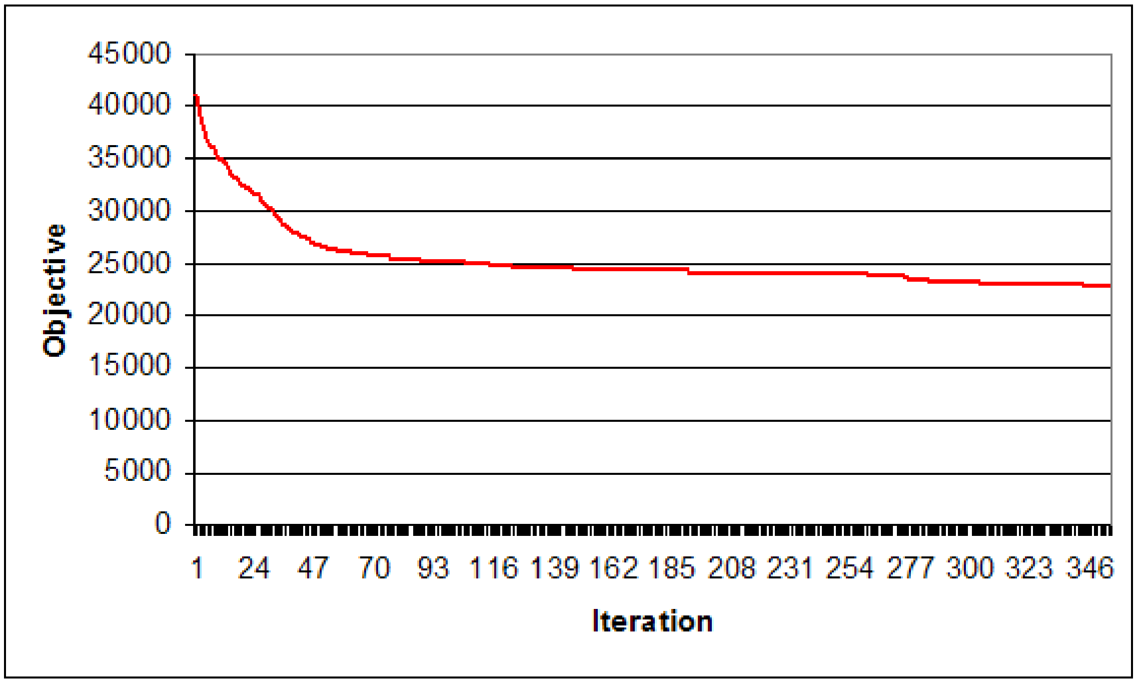

Figure 27: Objective Change: Random, Cyclic Exchanges (1000 Products)

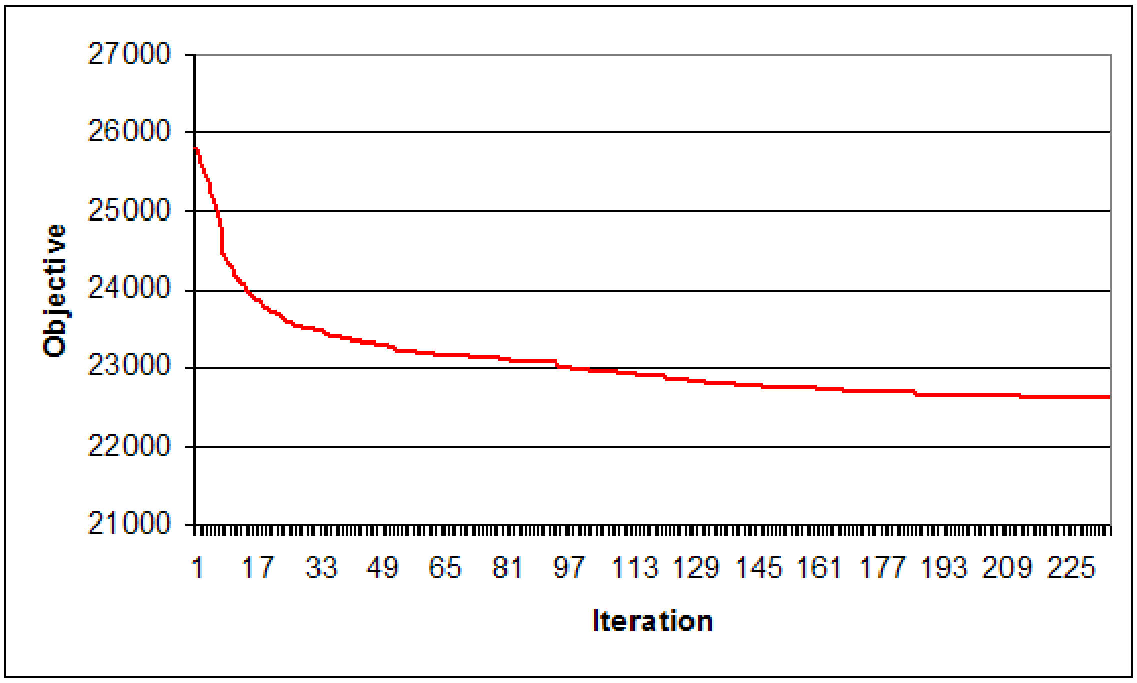

Figure 28: Objective Change: Popularity, Cyclic Exchanges (1000 Products)

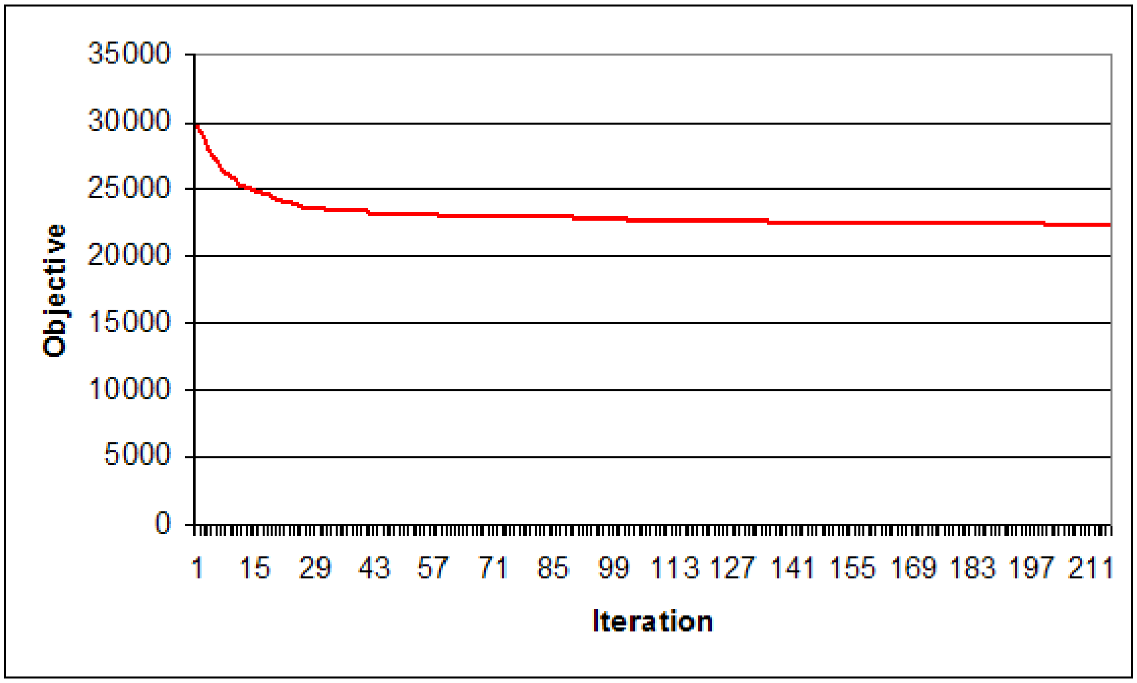

Figure 29: Objective Change: Lagrangian Relaxation, Cyclic Exchanges (1000 Products)

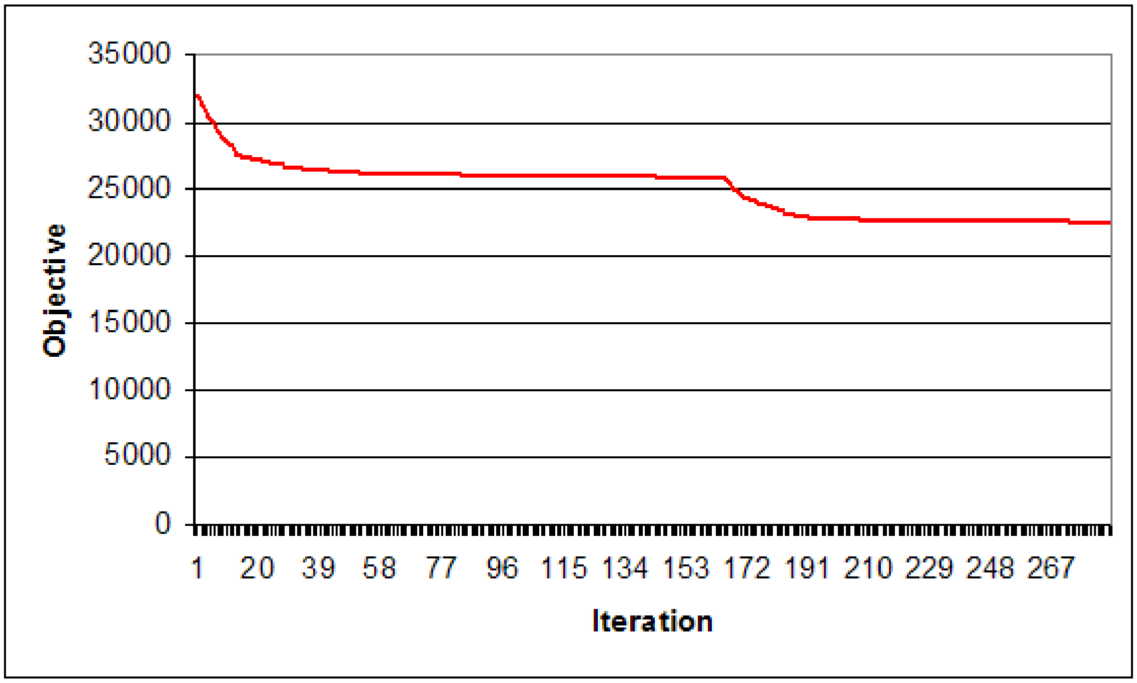

Figure 30: Objective Change: Frazelle, Cyclic Exchanges (1000 Products)

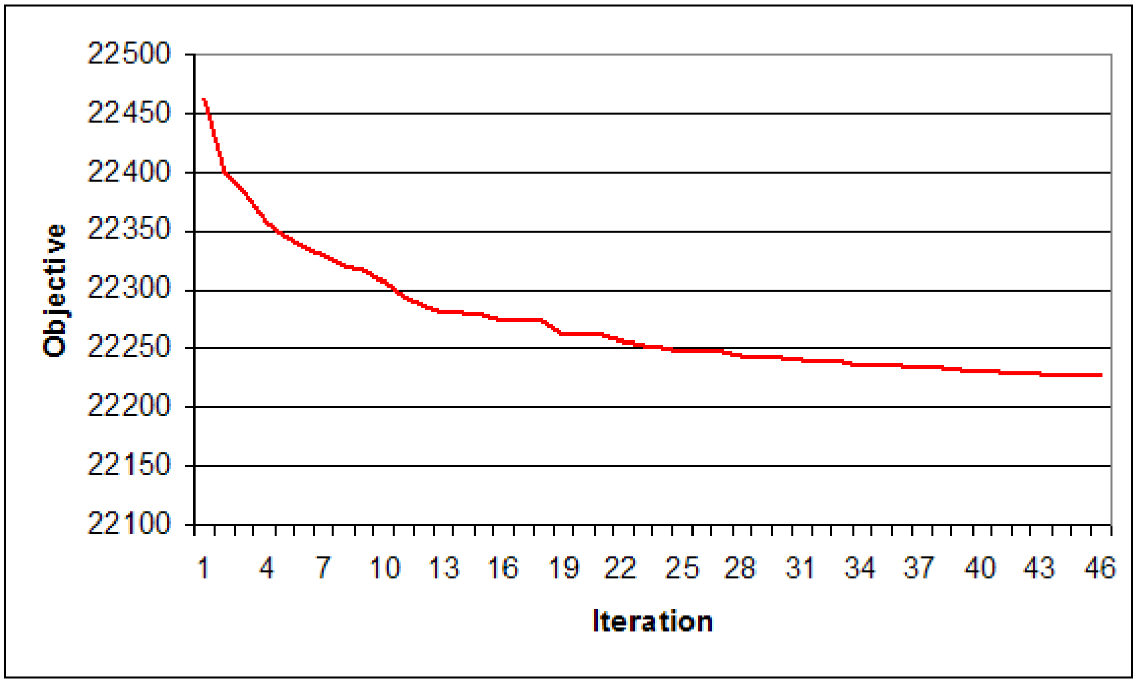

Figure 31: Objective Change: Amirhosseini 1, Cyclic Exchanges (1000 Products)

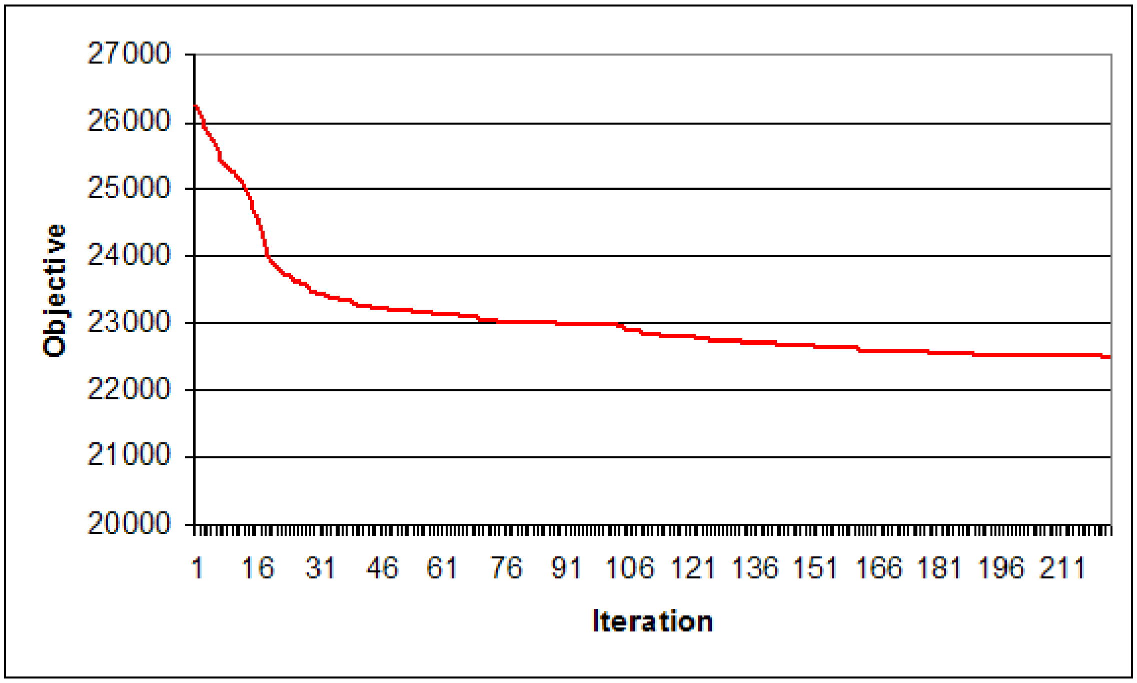

Figure 32: Objective Change: Amirhosseini 2, Cyclic Exchanges (1000 Products)

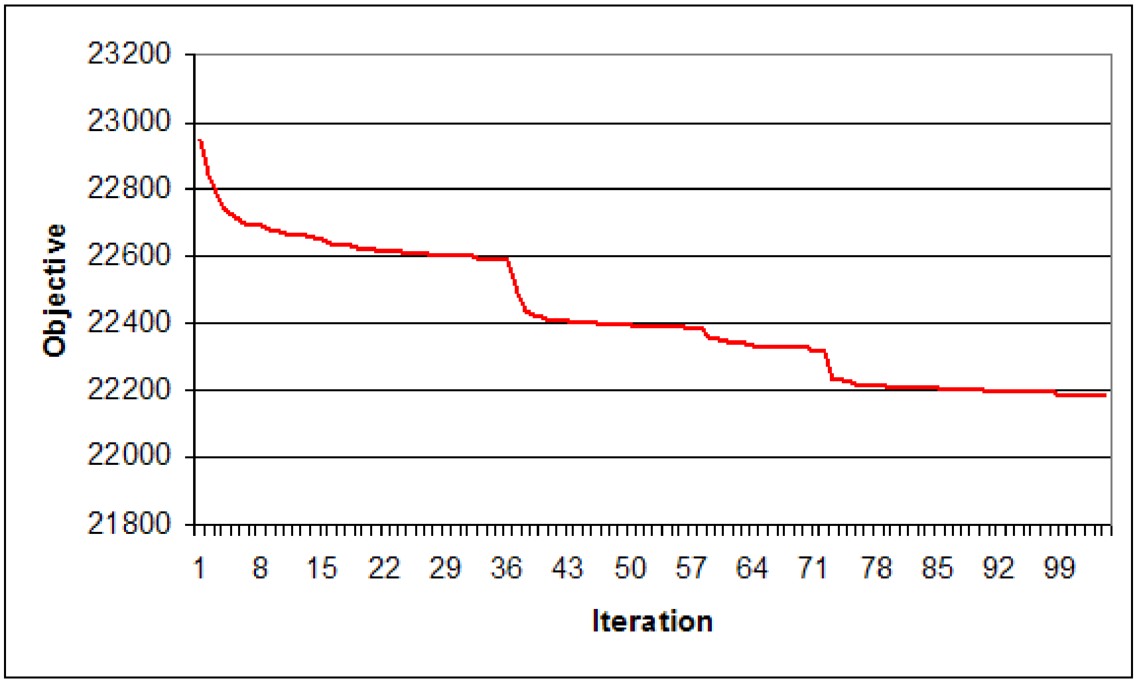

Figure 33: Objective Change: Amirhosseini 3, Cyclic Exchanges (1000 Products)

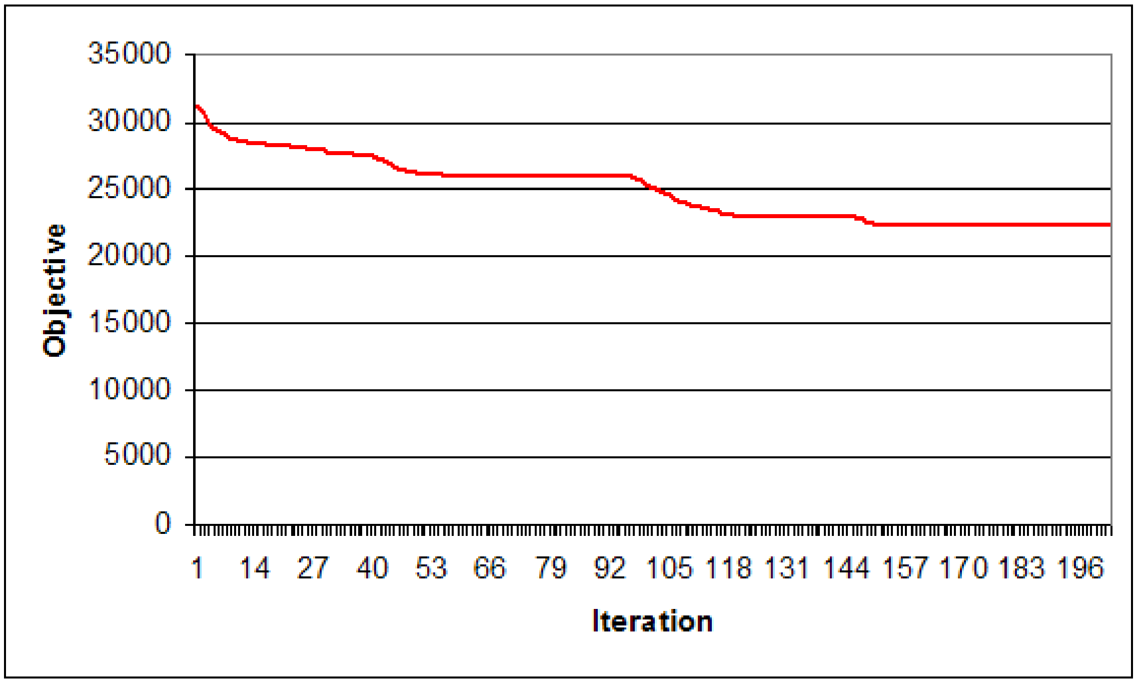

Figure 34: Objective Change: Amirhosseini 4, Cyclic Exchanges (1000 Products)

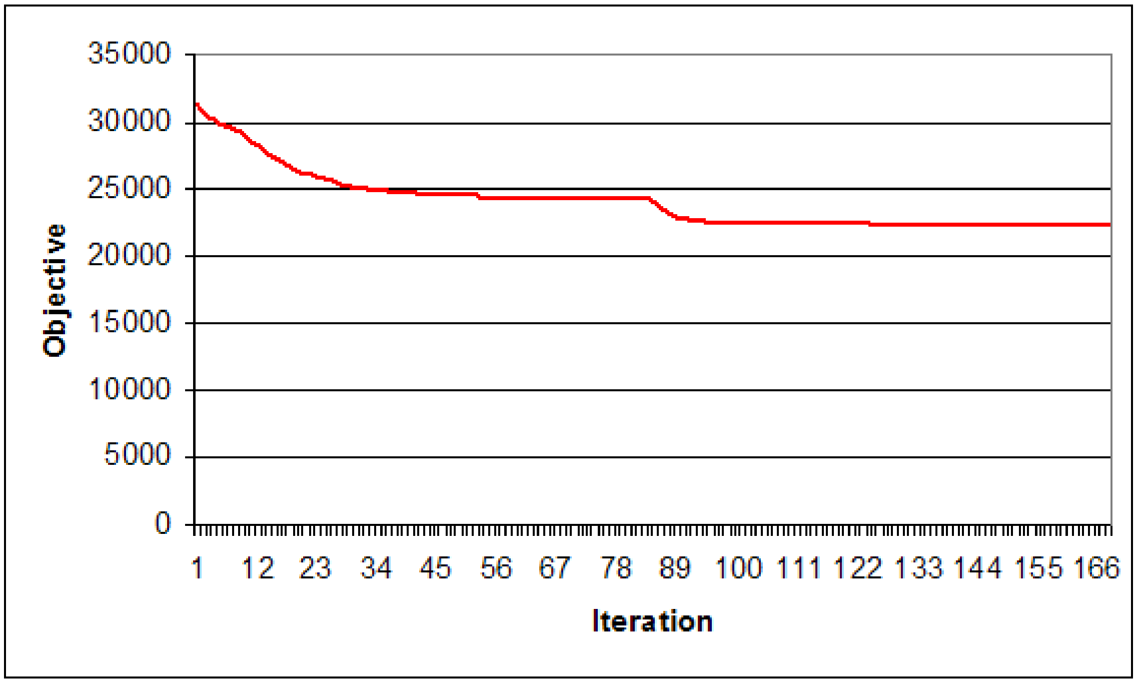

Figure 35: Objective Change: Amirhosseini 5, Cyclic Exchanges (1000 Products)

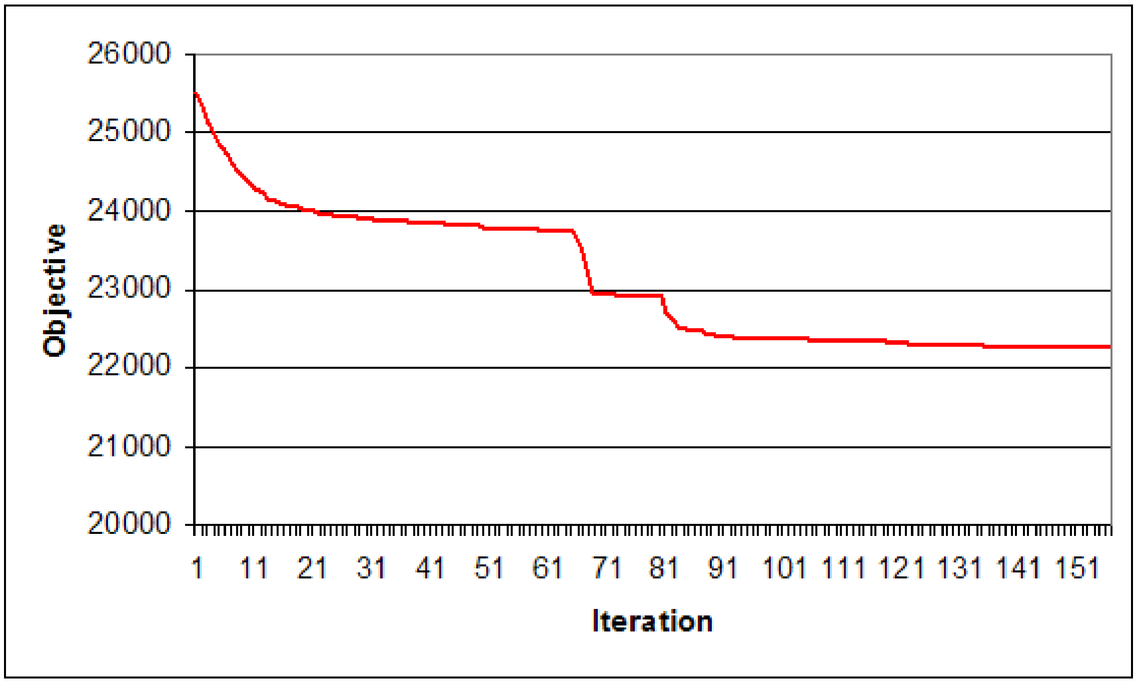

Figure 36: Objective Change: Amirhosseini OSCM, Cyclic Exchanges (1000 Products)

Table 10: Distribution of Product Sizes

Number of Pallets	Percent of Products
1	75%
2	10%
3	10%
4	5%

Table 11: Comparison of Heuristics: 100 Products with Different Sizes

Algorithm		Initial	2-Exchange	Cyclic	Cyclic: SS
Random		1608	1077	794	767
Popularity		847	804	788	782
Frazelle		869	814	789	764
Amirhosseini	1	780	776	**772**	**753**
	2	949	841	793	761
	3	**777**	775	773	764
	4	786	785	780	764
	5	783	780	779	770
	OSCM	781	**774**	775	**753**
Upper Bound		1922			
Lower Bound		711			
Optimal		n/a			

6.3.2 Multiple Pallets per Product

When we allow multiple pallets for each product, we consider the effect of allowing stock splitting. For our data set there was no information about the number of pallets per product in the picking area so we assumed the distribution in Table 10.

In Tables 11 and 12 we present results that show the effect of permitting stock splitting. We let Cyclic: SS denote the version of the cyclic exchange algorithm that allows stock splitting. The Lagrangian relaxation algorithm is not implemented for this case because (L) no longer has the network structure that enabled us to solve it quickly (Section 3.2.2). Also, because the particle heuristic performed poorly and is sensitive in its parameters, we do not implement it here. Note that for every construction heuristic, the initial solution does not contain any products with split stock.

The relative performance of the heuristics is the same as when all products had the

Table 12: Comparison of Heuristics: 1000 Products with Different Sizes

Algorithm		Initial	2-Exchange	Cyclic	Cyclic: SS
Random		41255	27603	22868	21588
Popularity		25837	25024	22436	22221
Frazelle		33304	30523	22329	20762
Amirhosseini	1	**22523**	**22422**	**22287**	**19869**
	2	26522	25632	23015	22727
	3	23001	22536	22337	20654
	4	31252	28624	22479	20677
	5	31193	28120	22364	20190
	OSCM	25956	24456	22556	20672
Upper Bound		44540			
Lower Bound		14308			
Optimal		n/a			

same size. As before, both the 2-exchange and cyclic exchange algorithms improve upon the initial solutions. The cyclic exchange algorithm outperforms the 2-exchange algorithm and makes arbitrarily bad initial solutions competitive. The best performance again comes from pairing Amirhosseini and Sharp's construction algorithm using the first correlation measure with the cyclic exchange improvement algorithm.

In this situation, we observe that when stock splitting is allowed, cyclic exchanges always produce a better solution than in the non-stock splitting case. Since the initial solutions do not contain stock splitting, we rely on the improvement heuristic to do stock splitting. For the results presented here, we re-solve the set covering problem at every iteration, that is, k=1.

The running time of the improvement methods is negligible for the 100 product data set. For 1000 products, the 2-exchange algorithm takes a few seconds. Without stock splitting, the running times for the cyclic exchange are similar to those when all products had the same size, i.e., under 20 minutes. When stock splitting is allowed, the running times are 3-6 hours, an order of magnitude larger.

CHAPTER 7

CONCLUSIONS AND EXTENSIONS

In this chapter we summarize the major contributions of this research and recommend directions for future work.

7.1 Contributions

Following is a summary of the contributions of this research which we will discuss in more detail below.

1. Modeled problem using direct measure of zone visits.
2. Developed Lagrangian relaxation approach.
3. Introduced powerful local search heuristics.
4. Incorporated the ability to solve problems that include different product sizes, stock splitting, and rewarehousing.
5. Solved significantly larger problems than previous authors.
6. Provide superior solution quality compared to previous approaches.
7. Performed comparative study of existing methods.

The objective we have presented for the correlated storage assignment problem is based on the number of zones that must be visited to fill an order. Our objective directly models order picking costs, unlike previous authors who use cluster strength as a proxy to measure travel time. We have also discussed properties of our problem and showed that it is NP-Complete.

To solve the storage assignment problem with this objective, we have proposed Lagrangian relaxation and particle heuristic approaches. In addition we have adapted several approaches from the literature. A 2-exchange and cyclic exchange algorithm were proposed to improve the assignments from the construction approaches. The strength of the cyclic exchange algorithm is that it searches an exponential neighborhood in polynomial time. Previous authors have not discussed local search approaches for correlated storage.

In addition, we discuss generalizations that allow a single product to have multiple pallets of storage as well as stock splitting. The issue of rewarehousing is another important situation that we have discussed. The stock splitting and rewarehousing scenarios are important cases for which many previous authors do not account. We presented mathematical models for each problem variation.

Also, we report results for the special case as well as the generalizations for problems containing up to several thousand products, several hundred thousand orders, and forty zones. These are problem sizes which are realistic for operating warehouses and exceed the problem sizes discussed by previous authors. In addition, we perform a comparative study of our approaches as well as the approaches of several other authors. No other studies exist which consider the approaches of different authors on the same problem.

Our solution methods give superior results. The best solution using our approach outperforms COI, the standard used by many companies, by an average of 15%. Even when we start with a poor quality initial solution like random assignment, our solution is better than that from using COI.

7.2 Future Work

Now we will outline some possible directions for future research on this problem based on our findings.

7.2.1 Related Applications

We would like to explore how our problem could be adapted to settings outside of warehousing. The core idea of our problem is of work that can be partitioned into multiple zones with substantial costs caused by accessing multiple zones. There are several problems which appear to fit into this paradigm. One application is organizing manufacturing processes into virtual manufacturing cells. Here we would like to assign processes to cells so that jobs can be completed using a minimal number of cells. Another application is assigning tasks to parallel processors. In this case, the problem is to divide the work among processors so that minimal information must be sent between processors to complete the work.

There are also applications for our problem in computer science. A standard problem is storing information on multiple disks and/or on multiple partitions on a disk for use in one application or report. In this situation there are costs associated with accessing multiple disks and moving between partitions on a single disk. Another problem arises in vertical partitioning of databases. Vertical partitioning refers to splitting the attributes of records into fragments [19]. Partitioning can be beneficial to performance since fragments are smaller than the entire data so fewer memory pages are necessary to fill a transaction. To maximize performance, we want to minimize the number of fragments required to fill a database transaction.

7.2.2 Improvement Heuristics

Solving problems with more than 5000 products using the cyclic exchange method is difficult due to machine memory limits and running time complexity. There are several methods we can incorporate into the cyclic exchange algorithm to alleviate these problems. For example, we can eliminate arcs with high costs from the graph. We also might consider partial enumeration of 2-exchanges and/or 3-exchanges before, or within, general cyclic exchanges. Another possibility is to include kick moves where some products are randomly moved to other zones. We could also try implementing some of the ideas from Section 6.2 to determine a good subset of orders.

In the cyclic exchange algorithm with stock splitting, we might perform further experiments on algorithm performance for different values of k. We also can experiment solving the set covering problem with preprocessing and enumeration instead of giving it to CPLEX directly.

7.2.3 Lower Bound Heuristics

The various algorithms we have discussed focus on finding good upper bounds to our problem. It is also important to determine good lower bounds. Since we would like to close the gap between the upper and lower bounds, it is worth exploring lower bound heuristics.

7.2.4 Additional Data Sets

Further consideration should be given to data sets with various characteristics. The comparative performance of the different construction heuristics may depend on the nature of the set. Additionally, the improvement heuristics may have varying degrees of impact.

REFERENCES

[1] Ahuja, R.K., Magnanti, T.L., Orlin, J.B., 1993. Network Flows: Theory, Algorithms, and Applications. Prentice Hall.

[2] Ahuja, R.K., Orlin, J.B., Sharma, D., 2001. Multi-exchange neighborhood search algorithms for the capacitated minimum spanning tree problem. Mathematical Programming 91, 71-97.

[3] Ahuja, R.K., Orlin, J.B., Sharma, D., 2003. A composite very large-scale neighborhood structure for the capacitated minimum spanning tree problem. Operations Research Letters 31, 185-194.

[4] Amirhosseini, M.M., Sharp, G.P., 1996. Simultaneous analysis of products and orders in storage assignment. Manufacturing Science and Engineering, MED 4, 803-811.

[5] Anderberg, M.R., 1973. Clustering Analysis for Applications. Academic Press.

[6] Bartle, R.G., 1976. The Elements of Real Analysis. John Wiley & Sons.

[7] Cormen, T.H., Leiserson, C.E., Rivest, R.L., 1999. Introduction to Algorithms. McGraw-Hill.

[8] Drury, J., 1988. Towards more efficient order picking. IMM Monograph No. 1, The Institute of Materials Management, Cranfield, United Kingdom.

[9] Frazelle, E.H., Sharp, G.P., 1989. Correlated Assignments: A Stock Location Assignment Strategy to Dramatically Reduce Order Picking Times. Industrial Engineering (April), 33-37.

[10] Frazelle, E.H., 1990. Stock location assignment and order picking productivity. Ph.D. Dissertation, Georgia Institue of Technology.

[11] Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co.

[12] Gademann, A.J.R.M., Van Den Berg, J.P., Van Der Hoff, H.H., 2001. An orrder batching algorithm for wave picking in a parallel-aisle warehouse. IIE Transactions 33, 385-398.

[13] Gibson, D.R., Sharp, G.P., 1992. Order batching procedures. European Journal of Operational Research 58, 57-67.

[14] Heskett, J.L., 1963. Cube-per-order index - a key to warehouse stock location. Transportation and Distribution Management 3, 27-31.

[15] Hua, W., 2001. Cluster based storage policies in kitting area. Ph.D. Dissertation, Georgia Institue of Technology.

[16] Hyafil, L., Rivest, R.L., 1973, Graph partitioning and constructing optimal decision trees are polynomial complete problems. Report No. 33, IRIA-Laboria, Rocquencourt, France.

[17] Lazoff, D.M., Sherman, A.T., 1994. An exact formula for the expected wire length between two randomly chosen terminals. Technical Report TR CS-94-08, Computer Science Department, University of Maryland Baltimore County. 14 pages.

[18] Liu, C.M., 1999. Clustering techniques for stock location and order-picking in a distribution center. Computers & Operations Research 26, 989-1002.

[19] Navathe, C.B., Ceri, S., Weiderhold, G., Dou, J., 1984. Vertical partitioning algorithms for database design. ACM Transactions on Database Systems 9, 680-710.

[20] Nemhauser, G.L., Wolsey, L.A., 1988. Integer and Combinatorial Optimization. John Wiley & Sons.

[21] Rosenwein, M.B., 1994. An application of cluster-analysis to the problem of locating items within a warehouse. IIE Transactions 26, 101-103.

[22] Ruben, R.A., Jacobs, F.R., 1999. Batch construction heuristics and storage assignment strategies for walk/ride and pick systems. Management Science 45, 575 - 596.

[23] Sadiq, M., 1993. A hybrid clustering algorithm for reconfiguration of dynamic order picking systems. Ph.D. Dissertation, University of Arkansas.

[24] Shah, P., 1988. Decision problems in mini-load automatic warehousing systems. Ph.D. Dissertation, Purdue University.

[25] Sharp, G.P., 2001. Warehouse Management, in Salvendy, G. (Ed.), Handbook of Industrial Engineering, 3rd Ed., John Wiley & Sons, 2083-2109.

[26] Thompson, P.M., Orlin, J.B., 1989. The theory of cyclic transfers. Working Paper OR200-89, Operations Research Center, Massachusetts Institute of Technology.

VITA

Maurice Garfinkel was born in San Francisco, California, USA on April 26, 1976. After living in Berkeley, California for the first eight years of his life, his family moved to Los Angeles, California. Maurice graduated from Cornell University in 1998 with a B.A. in Mathematics and a concentration in Operations Research. After spending the summer in Fort Worth, Texas working for Lockheed Martin Tactical Aircraft Systems, he began a doctoral program at Georgia Institute of Technology. He completed a M.S. in Operations Research in 2001 and a Ph.D. in 2005. In his spare time, Maurice is an avid tennis player.