COMPETITIVE LEARNING WITH ADDITIONAL INFORMATION FOR SOLVING MULTI-FACILITY WEBER PROBLEM

by

Kerem Can Özkısacık

B.S., Industrial Engineering, Galatasaray University, 1998

M.S., Industrial Engineering, Boğaziçi University, 2001

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Industrial Engineering

Boğaziçi University

2007

ACKNOWLEDGEMENTS

This thesis is the result of almost six years of work whereby I have been accompanied and supported by many people. It is a pleasant aspect that I have now the opportunity to express my gratitude for all of them.

If there is one person to be mentioned concerning this study, it is Professor İ. Kuban Altınel. It is him who pushed me back when I was about to desert. Without him this project would have never ended. I would like to express my deepest thanks to him for his guidance and his continuous support throughout this study. I owe him a lot.

I would like to thank Assoc. Prof Necati Aras who kept an eye on the progress of my work and always was available when I needed his advises. Zealously he became my unofficial co-advisor during these years and his presence can be felt in every idea and sentence of the study.

Another driving force of this dissertation is Prof. Ethem Alpaydın. His contribution is hard to ignore, and I am especially grateful to him for his efforts to broaden my horizons.

I would also like to thank the other members of my PhD committee Prof.G ulay Barbarosoğlu and Assoc. Prof İlker Birbil who monitored my work and devoted time in reading and providing me with valuable comments. For their efforts I can only offer my humble appreciations.

I would to express my gratitude to my comrades, Albert and Fatih, for hours of discussion and wonderful insights. There was someone who brought me peace and serenity during endless hours of study and also she provided a model for my post-study plans: eating, sleeping, and to lead a careless life. Thank you Mırmır, my cat, my ever sleepy companion.

Mom and Dad: this work is dedicated to you, since you have dedicated all your life. This is the least I can do.

And last but not least Esra: not necessarily for coming along at the right time and pushing me to do my job, but for the very special person she is.

This research has been partially supported by Boğaziçi University Research Fund. Grant No: 06HA304D.

ABSTRACT

COMPETITIVE LEARNING WITH ADDITIONAL INFORMATION FOR SOLVING MULTI-FACILITY WEBER PROBLEM

It is possible to remark that the mathematical formulation of clustering analysis problem is similar to a certain class of location problems. Known as the multi–facility Weber problem, it is a non–convex continuous optimization problem where a predetermined number of facilities are located on the plane such that demand weighted summation of distances between each customer and its closest facility is minimized.

With further investigation, we pointed out that the clustering analysis formulation is a special case of this facility location problem mentioned above. Motivated by this relation, we revised two algorithms of competitive learning, namely vector quantization and Kohonen type networks, to use additional information and applied them to solve the Weber problem. The results obtained with the new techniques are superior than those reported previously in the literature.

In the second part of the study, we investigated the probabilistic version of the Weber problem. In this version, customer locations are not fixed and they are assumed to be distributed randomly.

Methods proposed previously are revised to handle probabilistic case as well. To the best of our knowledge, first experimental results are reported for this type of the problem and traditional techniques are compared with newly proposed methods. Furthermore, we proposed an approximation scheme for dealing with more general probability distributions.

ÖZET

EK BİLGİ KULLANAN REKABETÇİ ÖĞRENME İLE ÇOK TESİSLİ WEBER PROBLEMİNİN ÇÖZÜMÜ

Topak çözümlemesinde karşımıza çıkan problem gösteriminin belirli bir sınıf yer seçimi probleminde de benzer şekilde ifade edildiği gözlemlenmektedir. Weber problemi olarak da bilinen bu tip konveks olmayan sürekli yer seçimi problemlerinde belirli sayıdaki tesisin düzlemdeki konumları aranarak her müşteri ve ona hizmet veren en yakın tesis arasındaki mesafenin müşterinin talebi ile ağırlıklandırılması sonucu oluşan toplam amaç fonksiyonun enküçüklenmesi gerçekleştirilmektedir.

İncelendiğinde topak cözümlemesindeki problemin yukarıda bahsi geçen problemin özel bir hali olduğu görülür. Bu ilişkiden yola çıkarak yapay zeka yazınında ve topak cözümlemesinde uygulanan rekabetçi ögrenme yöntemlerinden vektör nicemleme ve Kohonen tipi yapay sinir ağı yöntemleri uzaklık bilgisi gibi ek bilgileri de kullanacak şekilde geliştirilmiş ve Weber probleminin cözümune uyarlanmıştır. Elde edilen yeni yöntemler yazında yer alan önceki çalışmalara göre daha iyi sonuçlar üretmektedir.

Çalışmanın ikinci kısmında Weber probleminin olasılıksal sürümü ele alınmıştır. Bu problem çeşidinde müşteri yerlerinin sabit olmadığı, konumlarına dair sadece birer olasılïk dağılımının bilindiği varsayılmaktadır.

Önceki bölümde önerilen yöntemler bu problem tipini de çözebilecek şekilde geliştirilmiştir. Bilindiği kadarıyla ilk defa hesaplamaya dayalı sonuçların üretildiği bu problem için yeni yöntemlerle elde edilen sonuçların geleneksel yöntemlere göre iyileştirmeleri raporlanmıştır. Ayrıca problemin olasılıksal modelini basitleştirmek icin dağılımdan bağımsız yaklasıklama yöntemleri önerilmistir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

$\boldsymbol{a}_j$	Coordinates of customer j
$\mathcal{A}$	Set of all codebook vectors
b	Number of random solutions generated in the k–neighborhood of the current solution
$B_{j^*(t)}$	Activation bubble
c	index for the closest codebook to the input signal
$c(\xi)$	Best matching unit or the "*winner*"
D	Set of dissimilar input pairs
$d\left(x, y\right)$	Distance between vectors x and y
$E[\cdot]$	Expectation operator
$H\left(\cdot, \cdot, \cdot\right)$	Confluent hypergeometric distribution
h_j	Demand of the customer j
i	Index for facility i
j	Index for customer j
k_{max}	Parameter for neighborhood structure for VNS
m	Number of facilities to be located
n	Number of customers
q	Dimension of input vectors
R_j	Voronoi set of the unit j
s	Codebook vector for competitive networks
S	Set of similar input pairs
S_j	Set of sample vectors for customer j
v	Input vector
$\boldsymbol{w}_s$	weight of the codebook vector
Y_j	Neuron j's codebook vector
Λ_{ci}	Neighborhood function on the network between units c and i
∇_{Y_j}	Gradient with respect to the codebook vector Y_j of neuron j
Φ_i	Contribution of i to the current objective value

Ψ_i	Demand weighted sum of distances between customers without a facility nd their second closest facility
ξ	Input signal
$\|\| \cdot \|\|$	L_r norm
AC	Measure of accuracy to compare the performances of clustering algorithms
AE	Approximation error between the exact and the approximated expectations
ALA	Alternate allocation–location
ASC	Add by second closest
DA	Drop–add moves
DSC	Drop by second closest
EX	Exchange moves, also known as swap or interchange moves
LA	Location allocation
LAP	Location–allocation problem
MFWP	Multi–facility Weber problem
PALA	Probabilistic ALA
PMFWP	Probabilistic multi–facility Weber problem
RSOM	Revised self-organizing maps
RVQ	Revised vector quantization
SOM	Self-organizing maps
SSOM	Standard self-organizing maps
VNS	Variable neighborhood search
VQ	Vector quantization

1. INTRODUCTION

Location-Allocation problems arise in many practical settings such as emergency service systems, distribution, telecommunication networks or public services. It is stated as follows: given the location of a set of customers or demand centers and their associated demands, find the location of supply centers and the corresponding allocation of the demand to them so that certain cost function is minimized. Besides its practical implications, the problem is also attractive academically because of its nature. Even in the simplest case, where we assume infinite capacities for facilities, allocating each customer to the closest facility so that total distance between facilities and customers is minimized is NP-hard [72]. Exact algorithms for this problem do exist (e.g. [23, 57]), but the NP-hard nature of the problem makes heuristics the natural choice for solving larger instances.

Heuristics are essential tools for the solution of many difficult optimization problems. Their main aim is to provide, in reasonable time, near-optimal solutions to constrained or unconstrained optimization problems. Moreover, they may also be very useful to provide bounds, initial solutions, etc. to be used in exact algorithms. General frameworks for building heuristics, known as metaheuristics, have been extensively studied in the last two decades. For detailed surveys and extensive bibliography it is possible to cite works by Reeves [83] and Osman and Laporte [78] as well.

Basically, heuristics seek a feasible solution. While traditional heuristics, such as, simple descent methods, are blocked in the first local optimum found, this is not the case for heuristics built within the metaheuristics paradigms. All of them provide methods, deterministic or stochastic, for getting out of poor local optima.

In contrast to this success in practice, the theory of metaheuristics is advancing more slowly. While good heuristics are often obtained, with some ingenuity and a lot of parameter setting, reasons why they work as well as they do are mostly unknown [44]. Hence some reflection on desirable properties of metaheuristics, which would guarantee

both their practical and theoretical interest may be preferred.

We should remark that assuming uncapacitated facilities, uniform demand distribution and setting the optimization criterion to minimizing the total sum of the distance between the location of customers and the supply centers, the problem can also be stated as a clustering problem facilities representing cluster centers. We highlight the mathematical relation between location-allocation problems and the problem of finding better clustering given a deterministic or probabilistic data set and we try to unify these two era of scientific research that are advancing independently of each other. This established relation enables us to apply efficient techniques of competitive learning to improve existing solution methods.

1.1. Competitive learning for facility location

Whenever a researcher starts to work on a data set, it may be to his advantage to group similar objects together, and work on a smaller set. This reduction usually implies to a grouping or formally to a clustering of the data. There are multiple algorithms proposed in clustering literature and vector quantization is one of the most studied ones. As expressed in [59], there is a slight difference between vector quantization and clustering. A vector quantization method tries to minimize a properly defined quantization error while clustering methods have an objective of finding "good" groupings of the samples. Nevertheless once a clustering algorithm finishes its execution and results with clusters, it is usually possible to compute the related quantization error. Many algorithms have been developed and proposed in vector quantization and clustering literature and several excellent surveys give details of theses algorithms (e.g. [49],[22],[73]). The performance of these algorithms is measured using final quantization error, which is subjectively evaluated according to several criteria such as topology preservation, density estimation, etc.

Actually, the relation between methods that solve the Weber problem and competitive learning techniques is quite strong. Location-Allocation problems occur whenever a given number of facilities need to be located to serve a set of demand centers and

it is not known or fixed prior to their allocation to supply centers. When the transportation costs are considered to be proportional to the distance travelled and the minimization function is selected to be the sum of distances, the problem is recognized as identical to certain clustering analysis and vector quantization problem.

There are two sets of locations in facility location problems: the set of customers and the set of facilities. According to these sets problems are classified to be probabilistic or deterministic. If the problem is deterministic, customer locations set is fixed and is given as input. Otherwise, they are distributed according to a probability density. The problem consists of finding the locations of the facilities of the second set, and mapping customers to facilities with the objective of minimizing the total cost of serving the customers. Usually, supply centers such as plants and warehouses constitute the facilities and retailers and dealers are considered as customers. When the facilities are restricted to be in a set of pre-determined candidate locations, the corresponding location-allocation (LA) problem becomes a discrete optimization problem. Otherwise, we have a continuous LA problem where facilities can be located anywhere in the Euclidean plane. The latter problem is referred to as Fermat-Weber problem or simply Weber problem. The objective function of the Weber problem is found to be non–convex, hence solving the Multi-facility Weber Problem (MFWP) becomes a difficult task [26]. The difficulty of the MFWP increases with the problem size, i.e., with the number of existing customers, n, and the number of facilities to be located, m. This is due to the fact that the number of local minimum solutions grows exponentially with increasing n and m.

In fact it is possible to remark the relation between partitioning points into k clusters or assigning customers to some facilities, if we consider the same objective function. Later in Chapter 4 and 7 we will give more rigorous formulations about this relation in both deterministic and probabilistic scenarios. Moreover in recent years, there has been a positive growth in the academic researches concerning competitive learning and new algorithms and techniques are developed in parallel. Thus it is possible to exploit this relation and combine techniques from both areas.

1.2. Organization of this document

This dissertation starts with a brief summary of some competitive learning algorithms. This summary is focused mostly on the unsupervised learning techniques that are relevant in facility location applications.

Then in the next chapter we give several approaches dealing with how we may incorporate our information about problem or data space into the competitive learning applications. In most of the location models distance information of the input space is fixed and is given. For some competitive learning techniques this constraint or knowledge may lead to some modifications in the algorithms. Finally we summarize approaches where the statistical information about data is considered.

In Chapter 4 we will explain the nature of the location-allocation problem and discuss its mathematical formulation in the deterministic case where the customer locations are given. We propose here our competitive learning approach for solving this type of problem and summarize experimental results.

In the next chapter, we review the case of location-allocation problems where the customer locations are not known precisely. We explore the probabilistic formulations of the problem in that chapter and discuss its specific cases. Different solution approaches including approximations for investigating the probabilistic nature of the problem are also investigated in this chapter.

In the subsequent chapters, we propose solution techniques for dealing with the probabilistic Multi-Facility problem. We start by proposing a variable neighborhood search technique, which gives accurate results for the deterministic case and continue with a vector quantization approach for solving the problem. Implementation details and some experimentation results are given in both chapters.

Finally, the last chapter summarizes our overall contributions and findings. We also suggests possible research avenues for further research.

2. COMPETITIVE LEARNING TECHNIQUES

The area of competitive learning includes a large number of models. They have similar objectives but different learning algorithms. The main goal of those models is to distribute a certain number of vectors such that the distribution of these vectors reflect the probability distribution of the input signals. These vectors are called usually as codebook vectors [54], and in general the probability distribution of the input signals is not given explicitly but estimated only through a sample set so that the codebook vectors try to reflect that estimated distribution.

We will review several methods related to competitive learning in the following sections. Note that there is a huge number of techniques in the area of competitive learning, therefore we focus on a relatively small group. These are the ones whose learning algorithms can be used in solving location problems. For a detailed surveys showing different competitive learning techniques and applications we refer to [37], [49] and [59].

We start by describing the goals of competitive learning and we continue on hard competitive learning problem and algorithms. The soft competitive learning algorithms are reviewed next. A special type of soft competitive learning algorithm, The Self-Organizing Map algorithm is presented in detail in the following section. Next, we explain the models where the dimensionality and number of codebook vectors is not fixed a priori.

2.1. Objective functions in Competitive Learning

Usually two basic goals are pursued to achieve in competitive learning: error minimization and entropy maximization. These goals are expressed as the objective of the related optimization problems. To be consistent with the competitive learning we refer to any models as *network* even if the model does not correspond to what is usually understood as a neural network.

Let us note that each *network* consists of a set of N units:

$$\mathcal{A} = \{s_1, s_2, \ldots, s_N\}, \tag{2.1}$$

and each unit s has an associated *codebook vector* $\mathbf{w}_s \in \mathbb{R}^q$ indicating its position in the input space. The input signals are assumed to be q–dimensional and generated either according to a continuous probability density function $p(\xi), \xi \in \mathbb{R}^q$ or from a finite training data set $\mathcal{D} = \{\xi_1, \ldots, \xi_M\}, \xi_i \in \mathbb{R}^q$.

For a given input signal ξ the *winner* $c(\xi)$ among the units in $\mathcal{A}$ is defined as the unit with the nearest codebook vector as

$$c(\xi) = \arg\min_{s \in \mathcal{A}} \|\xi - \mathbf{w}_s\|, \tag{2.2}$$

where $\|\cdot\|$ denotes the vector norm. In a case where there are several units, one of them is chosen randomly to be the winner. It is possible to denote the current winner only by c (omitting the dependency on ξ). If not only the nearest but also the second–nearest or even more distant units are of interest, we denote with c_i the i–nearest unit (c_1 is the winner, c_2 is the second nearest unit, etc.).

At this point, a fundamental concept from computational geometry needs to be noted. It is called as the *Voronoi Tessellation*: given a set of vectors $\mathbf{w}_1, \ldots, \mathbf{w}_N \in \mathbb{R}^q$, the Voronoi Region V_i of a particular vector $\mathbf{w}_i$ is defined as the set of all points in $\mathbb{R}^q$ for which $\mathbf{w}_i$ is the nearest vector. We can note this as follows:

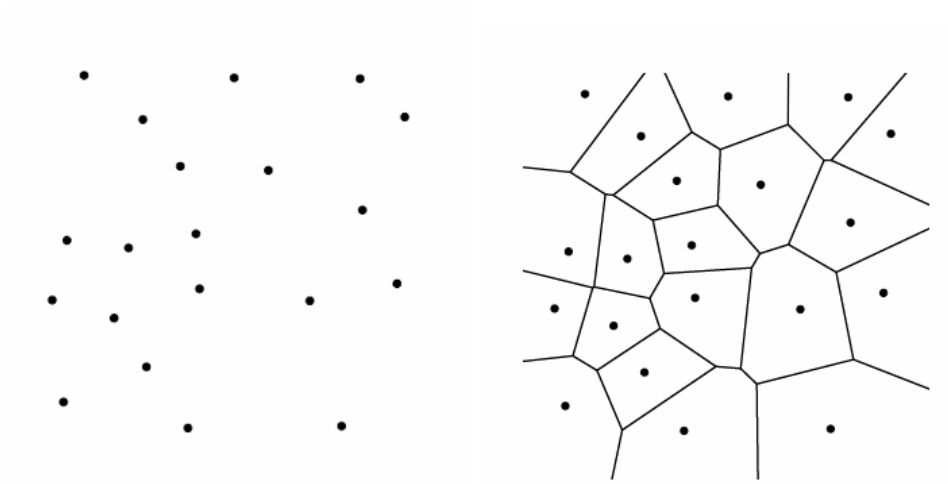
$$V_i = \{\xi \in R^q \text{ such that } i = \arg\min_{j \in \{1,\ldots,N\}} ||\xi - w_j||\} \tag{2.3}$$

In order for each data point to be associated to exactly one Voronoi region we define that in case of equality the corresponding point is mapped at random to one of the nearest codebook vectors. It is known that each Voronoi region V_i is a convex area,

namely,

$$\xi_1 \in V_i \text{ and } \xi_2 \in V_i \Rightarrow \xi_1 + \alpha\,(\xi_2 - \xi_1) \in V_i \tag{2.4}$$

for $0 \leq \alpha \leq 1$. The partition of $\mathbb{R}^q$ formed by all Voronoi polygons is called *Voronoi Tessellation* or *Dirichlet Tessellation* and efficient algorithms to compute them are only known for two–dimensional data sets [81]. An example of a Voronoi tessellation is illustrated in Figure 2.1 where $\|\cdot\|$ is chosen as the Euclidean norm.



(a) Points in $\mathbb{R}^2$ (b) Voronoi Tessellation

Figure 2.1. Voronoi tessellation corresponding to sample points in $\mathbb{R}^2$

2.1.1. Error minimization

A frequent goal is the minimization of the expected quantization (or distortion) error. Let $\mathcal{A}$ be the set of all codebook units. In the case of a continuous input signal distribution $p(\xi)$ this amounts to finding values for the reference vectors $\mathbf{w}_j$, $j \in \mathcal{A}$ such that the error

$$E(p(\xi), A) = \sum_{j \in A} \int_{V_j} \|\xi - w_j\| p(\xi) d\xi \tag{2.5}$$

with V_j is the Voronoi region of unit j is minimized with respect to the vector norm $\|\cdot\|$. Correspondingly, in the case of a finite data set $\mathcal{D}$ the error

$$E(\mathcal{D}, A) = 1/|\mathcal{D}| \sum_{j \in A} \sum_{\xi \in R_j} \|\xi - \mathbf{w}_j\| \tag{2.6}$$

has to be minimized with R_j being the Voronoi set of the unit j.

A typical application where error minimization is important arises in signal processing and compression. In this problem, data is transmitted over limited bandwidth communication channels by transmitting for each data vector only the index of the nearest reference vector. The set of codebook vectors (which is called reference in this context) is assumed to be known both to sender and receiver. Therefore, the receiver can use the transmitted indexes to retrieve the corresponding vector. Obviously, there is an information loss in this case which is equal to the distance of current data vector and nearest reference vector. The error minimization goal is actually represented by the expectation value of this error as described by equations (2.5) and (2.6). This problem is called as vector quantization [61],[40] and in particular if the data distribution is clustered (contains subregions of high probability density), dramatic compression rates can be achieved with vector quantization with relatively little distortion.

2.1.2. Entropy maximization

Sometimes the codebook vectors should be distributed such that each vector has the same chance to be winner for a randomly generated input signal ξ. In other words,

$$P\left(c\left(\xi\right) = s\right) = \frac{1}{|\mathcal{A}|} \quad (\forall s \in \mathcal{A}), \tag{2.7}$$

where $c(\xi)$ denote the best-matching unit or the "winner". If we interpret the generation of an input signal and the subsequent mapping onto the nearest unit in $\mathcal{A}$ as random experiment which assigns a value $x \in \mathcal{A}$ to the random variable X then (2.7)

becomes equivalent to maximizing the entropy

$$H(X) = -\sum_{x \in A} P(x) \log(P(X)) = E\left[\log(\frac{1}{P(x)})\right], \tag{2.8}$$

with $E[\cdot]$ being the expectation operator. If the data is generated from a continuous probability distribution $p(\xi)$, then (2.8) is equivalent to

$$\int_{V_c} P(\xi) d\xi = \frac{1}{|\mathcal{A}|} \quad \forall c \in \mathcal{A}. \tag{2.9}$$

An advantage of choosing reference vectors such as to maximize entropy is the inherent robustness of the resulting system. The removal (or "failure") of any reference vector affects only a limited fraction of the data.

Entropy maximization and error minimization can in general not be achieved simultaneously. In particular if the data distribution is highly non-uniform both goals differ considerably. Consider, e.g., a signal distribution $p(\xi)$ where 50 percent of the input signals come from a very small (point-like) region of the input space, whereas the other fifty percent are uniformly distributed within a huge hypercube. To maximize entropy half of the reference vectors have to be positioned in each region. To minimize quantization error however, only one single vector should be positioned in the point-like region (reducing the quantization error for the signals to zero) and all others should be uniformly distributed within the hypercube.

2.1.3. Topology preservation

With some architectures it is possible to map high-dimensional input signals onto a lower-dimensional structure in such a way, that some similarity relations present in the original data are still present after the mapping. This has been denoted feature mapping and can be useful for data visualization. A prerequisite for this is that the network used has fixed dimensionality.

A related question is, how topology-preserving is the mapping from the input data space onto the discrete network structure, i.e. how well are similarities preserved? Several quantitative measures have been proposed to evaluate this like the topographic product (Bauer and Pawelzik [9]) or the topographic function (Villmann et al. [93]).

It is also possible to consider other goals for competitive learning tasks, but we are not interested through out this study. In fact, we focus on the error minimization criteria since it seems to arise in the context of location-allocation problems.

Before starting to examine several techniques we also need to point out that the optimization of one of the above goals is considered as "learning". There are two general learning paradigms, supervised and unsupervised. In supervised learning the network compares its output with known correct answers. In other words, it receives feedback from outside about any errors. We can also say that there is a guide. In unsupervised learning no guide exists to tell the network whether the outputs are correct or not. The only available information is the input data. In such a case, the network has to discover the features of the input data from their correlations. In the following sections, we summarize three well–known algorithms of unsupervised learning techniques.

2.2. K-means

The "k-means" algorithm [69] is one of the most known clustering analysis technique. Basically the procedure consists of starting with k groups, each of which consisting of a single random point, and thereafter adding the points one after another to the group whose mean is the nearest. After a point is added to a group, the mean of that group is adjusted so as to take account of the new point. Thus at each stage there are in fact k means, one for each group. After the sample is processed in this way, the points are classified on the basis of nearness to the final means. The portions which result tend to be efficient in the sense of having low within class variance. In other terms, each codebook vector $\mathbf{w}_s$ is assigned always to the exact arithmetic mean of the input signals $\xi_1^c, \ldots, \xi_t^c$, it has been the winner so far. The sequence of successive values of $\mathbf{w}_c$ is as follows:

$$
\begin{aligned}
\mathbf{w}_c(0) &= \text{(random signal according to } p(\xi)) \\
\mathbf{w}_c(1) &= \mathbf{w}_c(0) + \epsilon(1)(\xi_1^c - \mathbf{w}_c(0)) \\
&= \xi_1^c \\
\mathbf{w}_c(2) &= \mathbf{w}_c(1) + \epsilon(2)(\xi_2^c - \mathbf{w}_c(1)) \\
&= \frac{\xi_1^c + \xi_2^c}{2} \\
&\vdots \\
\mathbf{w}_c(t) &= \mathbf{w}_c(t-1) + \epsilon(t)(\xi_t^c - \mathbf{w}_c(t-1)) \\
&= \frac{\xi_1^c + \cdots + \xi_t^c}{t}.
\end{aligned}
\tag{2.10}
$$

Here, a separate learning rate $\epsilon(t)$ is considered for each unit c and it is set according to the harmonic series: $\epsilon(t) = 1/t$.

One should note that the set of input signals $\xi_1^c, \xi_2^c, \ldots, \xi_t^c$ for which a particular unit c has been the winner may contain elements which lie outside the current Voronoi region of c since each adaptation of $\mathbf{w}_c$ changes the borders of the Voronoi region V_c. Therefore although $\mathbf{w}_c(t)$ represents the arithmetic mean of the signals it has been the winner, at time t some of these signal may well lie in Voronoi regions belonging to other units.

Both batch and online versions of k-means are used in many applications. Convergence analysis of the algorithm is available in [12]. Applications of k–means are suggested initially for the problems of non-linear prediction, efficient communication, non-parametric tests of independence, similarity grouping, and automatic file construction [69]. However due to its simplicity it is used as a default benchmark clustering algorithm in most of the real life problems.

2.3. Vector quantization

Another application of unsupervised learning is vector quantization (VQ) for clustering. The idea of VQ is to categorize a given set or distribution of input vectors into a number of classes and then represent any input vector by the *prototype* or *codebook* vector of the class. Determining the class of a given input vector requires finding the closest codebook vector using an appropriate distance measure. This divides up the input space into a Voronoi tessellation (see Figure 2.1). The dots in this figure represent the codebook vectors. From another point of view, we can say that VQ approximates the discrete or continuous probability distribution of the input vectors using a (finite) number of codebook vectors. Consequently, VQ reduces to finding the locations of the codebook vectors, which is solved by employing a network that is trained by competitive learning.

It is clear that as soon as we determine the codebook vectors, clustering of the input vectors becomes a trivial task. Therefore, dividing the input vectors into non–overlapping clusters is an output of VQ. Figure 2.2 shows the structure of a competitive neural network which performs vector quantization by dividing a given set of 5-dimensional input vectors into three clusters.

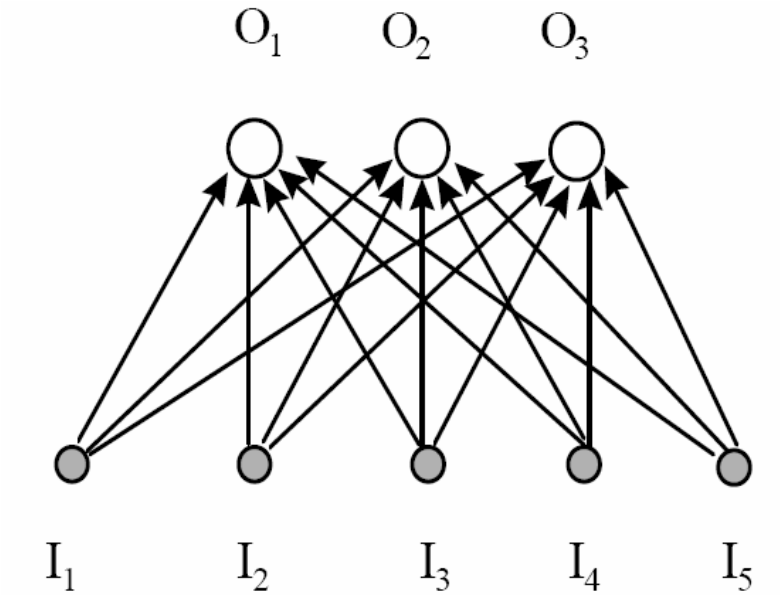


Figure 2.2. A competitive network for vector quantization.

Such a network has a number of output units each representing a cluster denoted as O_k in the figure. These units which are also called winner-take-all units compete for being fired upon the presentation of an input vector to the network. The requirement is that similar input data vectors should be classified as being in the same cluster, and therefore should fire the same output unit. In other words, the output units tend to fire for different domains of the input vectors. It should become clear from the presentation that output units are associated with clusters. That is output unit k represents cluster k. Similarly, the weight vector $\mathbf{w}_k$ of an output unit O_k defines the codebook vector of the cluster represented by O_k. Upon presentation of a q-dimensional input vector $\mathbf{v} \in \mathbb{R}^q$ to the network, each output unit O_k computes the distance between its weight vector $\mathbf{w}_k \in \mathbb{R}^q$ and $\mathbf{v}$. The one with the closest distance (with respect to some distance measure), referred to as the "winner" unit, is fired while other output units remain inactive. Namely,

$$O_c = \begin{cases} 1 & \text{if } d(\mathbf{w}_c, \mathbf{v}) = \min\limits_k \{d(\mathbf{w}_k, \mathbf{v})\} \\ 0 & \text{otherwise} \end{cases} . \tag{2.11}$$

where c is the index of the winner unit. The distance measure $d(\cdot)$ is usually taken as the Euclidean distance, i.e., $d(\mathbf{w}_k, \mathbf{v}) = \|\mathbf{w}_k - \mathbf{v}\|$, but other distance measures such as the rectilinear distance are possible choices.

Learning (or *training*) is completed when the input vectors are presented to the network a large number of times. A common measure, called cycle or epoch, is used to quantify the degree of neural network learning. An epoch is the presentation of all the input vectors once to the network. The required number of epochs for learning is most of the time application dependent. After learning is completed, we approximate the distribution of input vectors $\mathbf{v}$ by the weight vectors of output units (or by the codebook vectors representing the clusters). The error of such a reconstruction can be computed by summing up over all inputs the differences between each input vector $\mathbf{v}$ and the weight vector $\mathbf{w}_c$ of the closest output unit, i.e., the winner. In terms of a general distance function $d(\cdot)$, the quantization (also called distortion or reconstruction) error

as mentioned previously in (2.6) becomes

$$E(\mathbf{w}) = \sum_{\mathbf{v}} d(\mathbf{w}_c, \mathbf{v}) \tag{2.12}$$

where the index c of the weight vector of the winner is a function of the input vector $\mathbf{v}$. That is $d(\mathbf{w}_c, \mathbf{v}) = \min_k d(\mathbf{w}_k, \mathbf{v})$.

Vector quantization described in this framework is often called as "Hard Vector Quantization". The term 'hard' is used to express the *winner-take-all* strategy. That type of learning comprises methods where each input signal only determines the adaptation of one unit, the winner. Different specific methods can be obtained by performing either batch or on-line update. In batch methods all possible input signals (which must come from a finite set in this case) are evaluated first before any adaptations are done. This is iterated a number of times. On-line methods, on the other hand, perform an update directly after each input signal is presented to the network. Empirical study shows that both updating strategies have different benefits on their own, but on-line updating seems to stuck less in local optimals rather than batch updating. Among the on-line methods variants with constant adaptation rate can be distinguished from variants with decreasing adaptation rates of different kinds. Adaptation rates are also known as 'learning step' or 'line search step', and are reviewed in [10], [68].

A general problem occurring with hard competitive learning is the possible existence of "dead units". These are units which - perhaps due to inappropriate initialization - are never winner for an input signal and, therefore, keep their position indefinitely. Those units do not contribute to whatever the network purpose is (e.g. error minimization) and must be considered harmful since they are unused network resources. A common way to avoid dead units is to use distinct sample vectors according to $p(\xi)$ to initialize the reference vectors. The following problem, however, remains: if the reference vectors are initialized randomly according to $p(\xi)$, then their expected initial local density is proportional to $p(\xi)$. This may be rather suboptimal for certain goals. For example, if the goal is error minimization and $p(\xi)$ is highly non-uniform, then it is better to undersample the regions with high probability density (i.e., use

less reference vectors there than dictated by p(ξ) and oversample other regions. One possibility to adapt the distribution of the reference vectors to a specific goal is the use of local statistical measures for directing insertions and possibly also deletion of units.

Another problem of hard competitive learning is that different random initializations may lead to very different results. The purely local adaptations may not be able to get the system out of the poor local minimum where it was started. One way to cope with this problem is to change the "winner-take-all" approach of hard competitive learning to the "winner-take-most" approach of soft competitive learning. In this case not only the winner but also some other units are adapted . In general this decreases the dependency on initialization.

2.4. Self-organizing maps

A Self-organizing map (SOM) consists of neurons located on a regular, usually one or two-dimensional grid. Each neuron i of SOM is represented by an n-dimensional weight or reference vector (codebook vector) $\mathbf{w}_i = \{\mu_{i1}, \mu_{i2}, \ldots, \mu_{iq}\}$ where q is equal to the dimension of the input vectors. Although it is possible to use higher dimensional grids they are not preferred since their visualization becomes impossible. Usually the map topology is a rectangle; but different topologies have also been used successfully.

Let $\xi \in \mathbb{R}^q$ be a stochastic data vector and $p(\xi)$ be its probability density function. In the process of self-organization, vector ξ is compared with every $\mathbf{w}_i$, by using a given metric, and the closest codebook vector is selected as the "*best matching codebook vector*", signified by the subscript c. In fact the subscript c identifies the winning neuron of this competition.

So given that metric,

$$c = \arg\min_i \left\{ \|\xi - \mathbf{w}_i\|_2^2 \right\} \tag{2.13}$$

or

$$\|\xi - \mathbf{w}_c\|_2^2 = \min_i \left\{ \|\xi - \mathbf{w}_i\|_2^2 \right\}. \tag{2.14}$$

One kind of optimal selection of the w_i, minimizes the expected squared quantization error, defined as

$$E = \int \|\xi - \mathbf{w}_c\|^2 p(\xi)\, d\xi. \tag{2.15}$$

Here the metric used to measure the distance between vector ξ and the winner codebook vector $\mathbf{w}_c$ is the Euclidean norm. Applying gradient-descent, this error function is minimized by the following update formula, as shown by Kohonen [54],[56].

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \alpha(t).\Lambda_{ci}[\xi(t) - \mathbf{w}_i(t)] \tag{2.16}$$

Here t = 0,1,2, ... represents discrete time coordinates and Λ_{ci} is the neighborhood function. This function describes the interaction between the winning neuron and neuron i. At the beginning it is possible to initialize codebook vectors $\mathbf{w}_i$ randomly or set them arbitrarily to the mean value of the data. During the iterations the best matching codebook vector and its topological neighbors are moved closer to the input vector in the input space.

The codebook vectors $\mathbf{w}_k(t)$ used in VQ actually develop into a set of *feature-sensitive detectors* when the update mechanism given in Equation (2.16) is adopted. Furthermore, codebook vectors act independently. Therefore the order in which they are assigned to the different domains of input vectors $\mathbf{v}$ is more or less haphazard, strongly depending on the initial values of $\mathbf{w}_k(0)$. In SOM, the output units are organized as a linear array in one dimension or as a grid in two dimensions. The typical architecture of a two-dimensional SOM network is illustrated in Figure 2.3 where the output units have a rectangular grid structure.

The SOM algorithm follows a competitive learning technique with lateral excita-

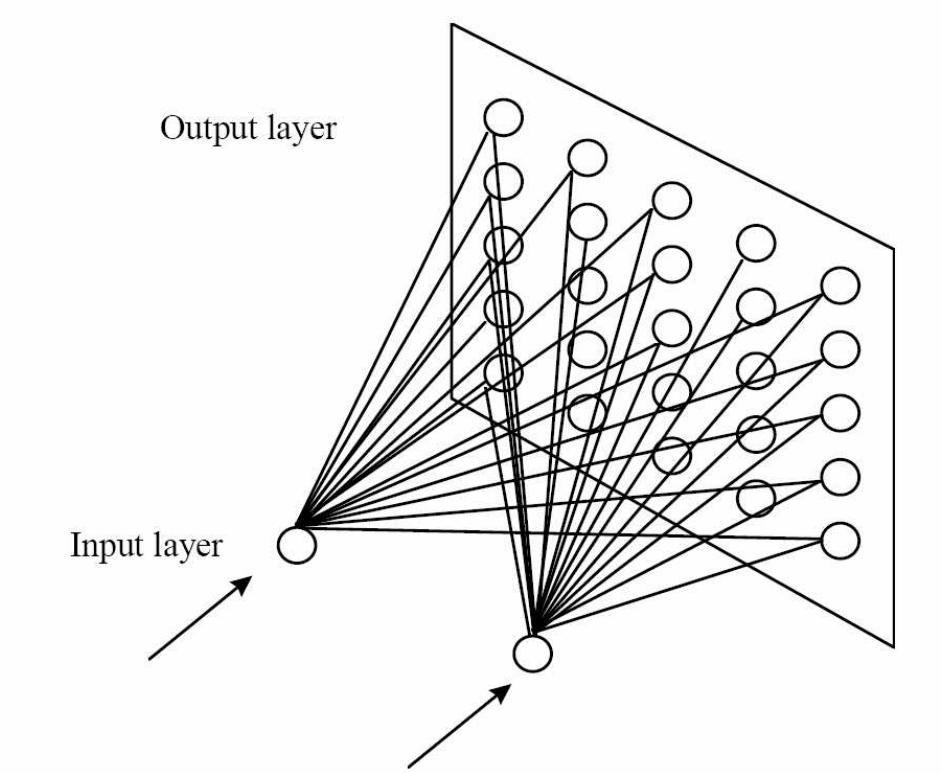


Figure 2.3. Typical architecture of a two-dimensional SOM.

tion. As in VQ, when an input vector $\mathbf{v}$ is presented to the input units of the network, the closest output unit (with respect to some distance measure) becomes the winner. Unlike the VQ, however, not only the weight vector of the winner but also weight vectors of a group of output units in the neighborhood of the winner are updated. Thus, in addition to their positions in the input space defined by their weight vectors, the output units also have a position among themselves in one or two dimensional space depending on the structure of the output units. For example, the output units of the network given in Figure 2.3 are organized as a rectangular grid on the output layer and the neighborhood relationships are defined in this two dimensional space.

As a consequence, the output units adapt in such a way that the topological relationships in the input vector space are preserved. Apparently, physically close output units will respond to naturally similar input vectors by projecting the topological (neighborhood) information inherent in the high-dimensional input data to a map of one or two dimensions. This results in a local smoothing effect on the weight vectors of the neurons in this neighborhood.

SOM has been proposed and developed by Kohonen [55],[54],[56]. It was first developed for the visualization of nonlinear relations of multidimensional data. SOM resembles other Vector Quantization (VQ) algorithms, such as k-means [49]. Missing details about SOM can be found in [55].

SOM is viewed as an artificial neural network model of ordered "maps" in the cortex. There exists a lot of neurophysiological evidence to support the idea that SOM captures some of the fundamental processing principles of the brain. In the computer science literature, SOM is considered as a model of unsupervised learning, and as an adaptive knowledge representation scheme [45]. Besides it is nowadays often used as a statistical tool for multivariate analysis [45]. SOM is both a projection method, which maps high-dimensional data space into low-dimensional space, and a clustering method so that similar data samples tend to be mapped to nearby neurons. SOM has a huge number of applications. An excellent catalog of them is given in [89].

2.5. Other competitive learning algorithms

One common property of the models of the previous sections, is that a topology is imposed on the network. In neural gas network there is no topology at all. The neural gas algorithm [71] sorts for each input signal ξ the units of the network according to the distance of their reference vectors to ξ. Based on this "rank order" a certain number of units is updated. Both the number of updated units and the adaptation strength are decreased according to a fixed schedule.

The winner of the competitive learning task has to be selected within a set of possible codebook vectors. It is possible to extend the idea by modifying this possibility set. In the literature, researchers investigated some models where instead of a fixed set of codebook vectors, adaptive structures are used. In these examples, during the learning process it is possible to add or delete some of the codebooks according to their efficiency. Growing Neural Gas is an example of such a method. In this method [35], [36] the number of units is changed (mostly increased) during the self-organization process. Starting with very few units new units are inserted successively. To determine

where to insert new units, local error measures are gathered during the adaptation process. Each new unit is inserted near the unit which has accumulated most error.

There is also another model denoted as Growing Cell Structures [34] and it is similar to growing neural gas model. The main difference is that the network topology is constrained to consist of k-dimensional simplices whereby k is some positive integer chosen in advance. The basic building block and also the initial configuration of each network is a k-dimensional simplex. This is, e.g., a line for $k = 1$, a triangle for $k = 2$, and a tetrahedron for $k = 3$.

3. USING ADDITIONAL INFORMATION

In usual applications of competitive learning the similarity, or the dissimilarity, between two given vectors of $\mathbb{R}^n$ is measured using Euclidean distance. However we remark in the literature that there are many works where it is proposed to use squared Euclidean distance as the dissimilarity measure for arithmetical purposes or to satisfy certain mathematical properties.

Despite its oftenness there exist some problems imposing their dissimilarity measures as distance functions instead of the squared Euclidean. Especially in location theory, problems are mostly classified according to the distance function they are based on (i.e. "Manhattan" distance problem, Euclidean problem, etc.). Hence, it is obligatory, rather than a necessity, to use distance function imposed by the problem.

Motivated by this fact, we try to enrich the learning rules of the SOM and the VQ algorithms by embedding the prior knowledge we have about the data space into its metric properties. In other words, if it is known a priori that the problem space has one specific dissimilarity measure, then improved update rules will enable its use, and subsequently give better results.

In this chapter, we introduce several ways to incorporate additional information we have about the data space into the techniques we presented in the previous chapter. Information we have considered may be given as a prior knowledge, such as the distance knowledge, or it may be incorporated according to the data dissimilarity as a side information or as a statistical information. In the sequel, we give three approaches dealing with additional information and explain how it is possible to revise VQ and SOM algorithms to benefit this knowledge.

3.1. Distance information in VQ and SOM

3.1.1. Explicit distance function

What essentially VQ or Kohonen's SOM do is to locate M codebook vectors so that (2.5) is minimized. Now let us assume that we are given a distance function $d(\cdot)$, instead of the squared Euclidean distance of (2.5) for a given data vector X. Then, the vector quantization objective becomes the minimization of

$$E(X) = \int d(X, Y_{j^*})^2 p(X) dX. \tag{3.1}$$

Here $d(X, Y_j)$ is the distance between the input data vector X and the neuron j's codebook vector Y_j, $p(X)$ is the continuous probability density function of the data vector X, dX is the volume differential in the input space, and j^* is the index of the winner neuron;

$$j^* = \arg\min_j \{d(X, Y_j)\}. \tag{3.2}$$

In other words, the SOM produces an approximation to a continuous probability density function $p(X)$ of the vector variable X using a finite number of codebook vectors Y_j. It has been shown that (3.1) determines locations for the codebook vectors in the space of input vectors X such that their density function is an approximation to $(p(X))^{q/(q+r)}$, when $d(X, Y_{j^*}) = \|X - Y_{j^*}\|_r$ with q denoting the dimension of X and Y_j , and $\|\cdot\|_r$ denoting the L_r–norm [101].

In addition, for $r = 2$, (3.1) has convenient mathematical properties: the update formula

$$Y_j(t+1) = Y_j(t) + \Lambda_{(j,j^*)}(X_i - Y_j(t)) \tag{3.3}$$

represent a steepest decent move for minimizing the quantization error (3.1). As a result, this convenience with the fact that $q \gg r$ holds in most applications and the

optimal locations of the codebook vectors approximate $p(X)$. This has been the only rationale behind the use of the square of the Euclidean metric to express expected quantization error.

The performance of SOM depends critically on the selection of a good metric over the input space. For instance the use of the squared Euclidean metric in the clustering of a given data set will probably result in a poor clustering when the weighted ℓ_r–norm

$$d(x,y) = k\,\|x-y\|_r = k\left(\sum_{i=1}^{d}|x_i-y_i|^r\right)^{1/r} \tag{3.4}$$

is more suitable to model distances between vectors x and y. Suppose we are given $d(x,y)$ with known $k \geq 1$ and $r \geq 1$ as an ideal model for the distances in the space of the input data vectors X. Such $d(x,y)$ clearly satisfies metric properties. Then we should revise SOM so that the winner is selected as

$$j^* = \arg\min_j \{d(X, Y_j)\} \tag{3.5}$$

and the neurons are updated as

$$Y_j(t+1) = Y_j(t) - \Lambda(j,j^*)\nabla_{Y_j} d(X, Y_j(t)). \tag{3.6}$$

In fact it is possible to show that updates (3.6) represent moves in the steepest descent direction of the error function (3.1) not only for the weighted L_r-norm, but also for any differentiable distance function $d(x,y)$. Here the operator ∇_{Y_j} denotes the gradient with respect to the codebook vector Y_j of neuron j.

3.1.2. Revising VQ and SOM to use additional distance information

We derived the update equations for the case that the dissimilarity is measured by a general distance function. In order to do this, we start by formulating the algorithm using the quantization error (2.5) for any arbitrary distance function. Then

we follow Kohonen's proof for the squared Euclidean case, and apply first-order necessary conditions. These equations are derived for vector quantization, but one can obtain Kohonen's learning rule simply by embedding the neighborhood function $\Lambda_{cj}(\cdot)$ between the unit j and the winner unit c. Details of that derivation can be seen as follows:

Let x be an n-dimensional stochastic input vector, *p(x)* its probability density function and $w_j \in \Re^n \;\; j = 1, \ldots, n$ the codebook vectors. Then the quantization error is defined as :

$$d(x, w_c) = \min_j \{d(x, w_j)\} \tag{3.7}$$

where c is the index of the closest reference vector to x in the space of input signals with respect to the distance function $d(x, y)$.

The error is then described as

$$e = \sum_{j \in L} \Lambda_{cj} d\left(x, w_j\right) \tag{3.8}$$

where L denotes the index set of codebook vectors, Λ_{cj} denotes the neighborhood function that determines the interaction between the winner codebook m_c and the rest during adaptation. As a consequence, the expected reconstruction error can be written as

$$E = \int ep\left(x\right) dx = \int \sum_{j \in L} \Lambda_{cj} d\left(x, w_j\right) p\left(x\right) dx \tag{3.9}$$

Now, we present a lemma which is used in the derivations to obtain the updates for the revised SOM (RSOM).

Lemma 3.1. *Let $\{a_j\}_{j=1}^n$ be a set of positive real scalar numbers. Then*

$$\lim_{r\to-\infty}\left[\sum_j a_j^r\right]^{\frac{1}{r}} = \min_j\{a_j\}$$

Proof. Let $r = -u$ then

$$\lim_{r\to-\infty}\left[\sum_{j=1}^n a_j^r\right]^{\frac{1}{r}} = \lim_{u\to\infty}\left[\sum_{j=1}^n \left(a_j^{-u}\right)\right]^{-1/u} = \lim_{u\to\infty}\left(\frac{1}{\left[\sum\limits_{j=1}^n \left(1/a_j\right)^u\right]}\right)^{1/u}$$

which is equal to

$$\frac{\lim\limits_{u\to\infty} 1^{1/u}}{\lim\limits_{u\to\infty}\left[\sum\limits_{j=1}^n \left(1/a_j\right)^u\right]^{1/u}} = \frac{1}{\max\limits_{j=1,\ldots,n}\{1/a_j\}} \tag{3.10}$$

Observe that the denominator is simply the ℓ_∞-norm and it is equivalent to $\max\limits_{j=1,\ldots,n}(1/a_j)$. Therefore the proof is complete since $\frac{1}{\max\limits_{j=1,\ldots,n}\{1/a_j\}}$ which is $\min\limits_{j=1,\ldots,n}\{a_j\}$.

Q.E.D

Coming back to reconstruction error (2.15), without loss of generality, we can try to minimize it only for the winning codebook vector, without taking into consideration its neighborhood. Because, when the update rules are once determined, the surroundings of the "winner" is renewed according to the neighborhood function. Hence, the overall error function becomes

$$E = \int d\left(x, w_j\right) p\left(x\right) dx = \int \min_{j=1,\ldots,n}\left\{d\left(x, w_j\right)\right\} p\left(x\right) d\left(x\right) \tag{3.11}$$

where *d(x,y)* is any distance function.

Using the Lemma 3.1, we can replace the minimization by a limit. Then

$$E = \int \lim_{r \to -\infty} \left[\sum_{j=1}^{n} d\left(x, w_j\right)^r \right]^{1/r} p\left(x\right) dx \tag{3.12}$$

which is equal to

$$\int \lim_{r \to -\infty} A^{1/r} \, p\left(x\right) dx \tag{3.13}$$

with $A = \sum_{j=1}^{n} d\left(x, w_j\right)^r$

The reconstruction error can be minimized subject to one of the codebook vectors w_j, simply by solving the first order optimality conditions, which requires the solution of the equation system

$$\nabla_{w_j} E = \int \lim_{r \to -\infty} \nabla_{w_j} A^{1/r} \, p\left(x\right) dx = 0 \tag{3.14}$$

with respect to w. Let us first compute the gradient of $A^{1/r}$ with respect to w_j:

$$\nabla_{w_j} A^{1/r} = \frac{1}{r} A^{1/r-1} r d\left(x, w_j\right)^{r-1} \nabla_{w_j} d\left(x, w_j\right) \tag{3.15}$$

which is equal to

$$A^{(1/r)-1} d\left(x, w_j\right)^{r-1} \nabla_{w_j} d\left(x, w_j\right). \tag{3.16}$$

Therefore the gradient of the error function becomes

$$\nabla_{w_j} E = \int \lim_{r \to -\infty} \left\{ A^{(1/r)-1} d\left(x, w_j\right)^{r-1} \nabla_{w_j} d\left(x, w_j\right) \right\} p\left(x\right) dx \tag{3.17}$$

or equivalently

$$\int \lim_{r\to-\infty} \left\{ A^{1/r} \frac{1}{A} d\left(x, w_j\right)^{r-1} \nabla_{w_j} d\left(x, w_j\right) \right\} p\left(x\right) dx \tag{3.18}$$

from which

$$\nabla_{w_j} E = \int d\left(x, w_c\right) \lim_{r\to-\infty} \left\{ \frac{d(x, w_j)^{r-1}}{\sum_i d(x, w_i)^r} \nabla_{w_j} d(x, w_j) \right\} p(x) dx \tag{3.19}$$

can be written as a consequence of lemma 3.1. This expression can be reorganized to obtain finally

$$\nabla_{w_j} E = \int \frac{d\left(x, w_c\right)}{d\left(x, w_j\right)} \nabla_{w_j} d\left(x, w_j\right) \lim_{r\to-\infty} \left(\sum_i \frac{d\left(x, w_i\right)^r}{d\left(x, w_j\right)^r} \right)^{-1} p(x) dx \tag{3.20}$$

By reducing the limit expression by simple algebra and replacing it on the error gradient we have

$$\lim_{r\to-\infty} \left[\sum_i \frac{d(x, w_i)^r}{d(x, w_j)^r} \right] = \lim_{r\to\infty} \left[\sum_i \frac{d(x, w_j)^r}{d(x, w_i)^r} \right] = \lim_{r\to\infty} \max_i \left(\frac{d(x, w_j)}{d(x, w_i)}^r \right) \tag{3.21}$$

and thus

$$\nabla_{w_j} E = \int \frac{d\left(x, w_c\right)}{d\left(x, w_j\right)} \nabla_{w_j} d\left(x, w_j\right) \lim_{r\to\infty} \left\{ \max_i \left(\frac{d(x, w_j)}{d(x, w_i)} \right)^r \right\}^{-1} \tag{3.22}$$

Since, c denotes the closest vector among all codebook vectors, the limit equals to 0, except for $c = j$. This can be expressed by using *Kronecker delta* δ_{cj}:

$$\lim_{r\to\infty} \left(\left[\frac{d(x, w_j)}{d(x, w_c)} \right]^r \right)^{-1} = \delta_{cj} = \begin{cases} 1 & \text{if } c = j \\ 0 & \text{otherwise} \end{cases}$$

Finally, we have

$$\nabla_{w_j} E = \int \delta_{cj} \nabla_{w_j} d(x, w_j) p(x) dx \tag{3.23}$$

With this final equation, one can apply Robbins–Monroe stochastic approximation technique [85], and approximate $\nabla_{w_j} E$ by $\nabla_{w_j} E(t)$, with discrete time index t. This approximation is shown by Kohonen [54]. Actually, the exact optimization of (3.23) is an unsolved problem and the best approximation technique is to consider stochastic samples of the function for t=0,1,2,.... Thus:

$$\nabla_{w_j} E \approx \nabla_{w_j} E\,(t) \tag{3.24}$$

and the update rules of $w_i(t)$ are obtained:

Robbins and Monroe proved that this sequence converges with probability one toward the locally optimal value w^* for which E is minimum, if and only if the following conditions are satisfied:

$$\sum_{t=0}^{\infty} \epsilon(t) = \infty \tag{3.25}$$

and

$$\sum_{t=0}^{\infty} \epsilon^2(t) < \infty \tag{3.26}$$

Therefore the sequence

$$w_j\,(t+1) = w_j\,(t) - \epsilon\,(t)\,\delta_{ci}\left[\nabla_{w_j} d\,(x, w_j)\,(t)\right] \tag{3.27}$$

converges to m^{*T} when we choose $\epsilon(t) = k/t$ for example, where k is any constant [54].

As it can be seen the update equation (3.27) depends on the gradient of the distance function used to measure the dissimilarities. Therefore these functions must be differentiable. This is a very useful result because there can be situations where some kind of a priori knowledge about the topology on the data space is embedded into a specific distance function. Then this distance function becomes definetly a better

choice then the Euclidean norm. Also update equations (3.27) generalize SOM updates (2.16) since it is possible to obtain them from (3.27) by replacing the distance function with the square of the Euclidean norm.

In [76] we computed the necessary update rules that minimize the quantization error for different ℓ_r-norms by using this mathematical result. The results are as follows:

Let $\mathbf{x}$ and $\mathbf{w}_j$ be an input signal, and jth weight vector associated to codebook j respectively. Let $\mathbf{x} = \{\xi_i\}$ and $\mathbf{w}_j = \{\mu_{ij}\}$ for $i = 1, \ldots, q$.

1.) $\ell_1 - norm$: $d(\mathbf{x}, \mathbf{w}_j) = |\mathbf{x} - \mathbf{w}_j|$

$$\mu_{ij}(t+1) = \mu_{ij}(t) + \epsilon(t) \operatorname{sgn}(\xi_i - \mu_{ij})$$

2.) $(\ell_2)^2 - norm$: $d(\mathbf{x}, \mathbf{w}_j) = \|\mathbf{x} - \mathbf{w}_j\|^2$; $\|\cdot\|$represents Euclidean norm.

Using equation (3.27)

$$\nabla_{\mathbf{w}_j} d(\mathbf{x}, \mathbf{w}_j) = -2(\mathbf{x} - \mathbf{w}_j)$$

and

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \epsilon(t)[\mathbf{x}(t) - \mathbf{w}_i(t)]$$

which results as the original SOM algorithm.

Please note that, the original SOM algorithm is not obtained when the Euclidean distance is used in the algorithm. It is obtained when we are considering the square of the Euclidean distance as our dissimilarity measure.

3.) $\ell_r - norm$: $d(\mathbf{x}, \mathbf{w}_j) = \|\mathbf{x} - \mathbf{w}_j\|_r = \left[\sum_i |\mathbf{x}_i - \mu_{ij}|^r\right]^{1/r}$

$$\nabla_{\mathbf{w}_j} d(\mathbf{x}, \mathbf{w}_j) = -\|\mathbf{x} - \mathbf{w}_j\|_r^{1-r} \begin{pmatrix} \operatorname{sgn}(\xi_1 - \mu_{1j}) . (\xi_1 - \mu_{1j})^{r-1} \\ \vdots \\ \operatorname{sgn}(\xi_n - \mu_{nj}) . (\xi_n - \mu_{nj})^{r-1} \end{pmatrix}$$

Equivalently, in component-wise updates we have the following formula

$$\mu_{ij}(t+1) = \mu_{ij}(t) + \epsilon(t) \left[\|\mathbf{x} - \mathbf{w}_j\|_r^{1-r}\right] \left[\operatorname{sgn}(\xi_i - \mu_{ij}) (\xi_i - \mu_{ij})^{r-1}\right].$$

From that point on, we will call *Revised Self-Organizing Map (RSOM)* the SOM which uses the general metric function. Especially in MFWP, we expect RSOM to perform superior to SOM. This will become clear in the experimental results section.

3.1.3. Using side information to determine a distance metric

However, there may exist situations where, rather than an already learned metric, some information is available about the distance between input vectors X in the form of a distance matrix. Then we can estimate parameters of an assumed metric form over this distance matrix: we can assume that distances have the form given with (3.4) and solve the least square problem

$$\begin{array}{ll} \min & \sum_{i \neq j} \left(d(X_i, X_j) - a_{ij}\right)^2 \\ s.t. & k, r \geq 1 \end{array} \tag{3.28}$$

with respect to k and r to learn a good metric for the data space. Here a_{ij} are the true distance values given in the distance matrix. This type of distance estimations are frequently implemented in location theory and there are numerous published works. To give some examples we can name the seminal works by Love and Morris [64, 66], the survey by Brimberg and Love [16], and more recent neural approaches by Alpaydın et al. [2], Altınel et al. [3], and Oommen et al. [77].

Another approach has been recently proposed by Xing et al. [99] for clustering with similarity information. They suppose first they have some set of points $\{X_i\}_{i=1}^m \subseteq \mathbb{R}^q$, and are given information that certain pairs of them are *similar*:

$$S = \{(i,j) : \ X_i \text{ and } X_j \text{ are similar}\}, \tag{3.29}$$

and certain pairs of them are dissimilar:

$$D = \{(i,j) : \ X_i \text{ and } X_j \text{ are dissimilar}\}. \tag{3.30}$$

Then they determine the entries of a positive semi-definite matrix A, $A \succeq 0$, which are eventually the unknown parameters of the distance metric of the form

$$d(x,y) = d_A(x,y) = \sqrt{(x-y)^T A (x-y)}. \tag{3.31}$$

by solving the optimization problem

$$\begin{array}{ll} \min & \sum_{(i,j)\in S} \left(d_A(X_i,X_j)\right)^2 \\ s.t. & \sum_{(i,j)\in D} d_A(X_i,X_j) \geq 1 \\ & A \succeq 0 \end{array} \tag{3.32}$$

when A is a diagonal matrix and a full matrix. Finally they use the new learned metric in k–means algorithm and report better clustering of the input data in general. Observe that the objective function demand that pairs of points $(i,j) \in S$ have small squared distance between them. This is trivially satisfied when $A = 0$, which is not useful and they add the inequality to guarantee that A does not collapse the data set into a single point. If dissimilarity information is not explicitly available, then D can be chosen as all pairs not in S. The positive semi-definiteness restriction of A ensures that $d_A(x,y)$ satisfies nonnegativity and triangle inequality, which are necessary metric

properties. In case A is diagonal this optimization problem reduces to

$$\begin{array}{ll} \min & \sum_{(i,j)\in S} \sum_{k=1}^{d} \left(X_{ik} - X_{jk}\right)^2 A_{kk} \\ s.t. & \sum_{(i,j)\in D} \sqrt{\sum_{k=1}^{d} \left(X_{ik} - X_{jk}\right)^2 A_{kk}} \geq 1 \ , \\ & A_{kk} \geq 0 \qquad k = 1, \ldots, d \end{array} \tag{3.33}$$

which can be solved to optimality very efficiently. Here A_{kk} $k = 1, \ldots, d$ denote the diagonal entries of A. However, when A is full, the problem is more difficult and requires more computational effort. Xing et al. [99] proposes algorithms to solve both cases, which we use to determine the entries of matrix A, namely metric (3.31). Once A is learned, (3.31) can be incorporated into RSOM or RVQ. Consequently, at each iteration t of RSOM the winner neuron j^* is first determined by

$$j^* = \arg\min_j \sqrt{\left(X_i - Y_j(t)\right)^T A \left(X_i - Y_j(t)\right)}. \tag{3.34}$$

Then the codebook vectors are adapted according to

$$Y_j(t+1) = Y_j(t) - \Lambda(j, j^*) \frac{1}{\sqrt{\left(X_i - Y_{j^*}(t)\right)^T A \left(X_i - Y_{j^*}(t)\right)}} A \left(X_i - Y_{j^*}(t)\right) \tag{3.35}$$

with the neighborhood function

$$\Lambda(a, b) = K_g e^{\frac{-[d(a,b)]^2}{2\sigma^2}}. \tag{3.36}$$

Here K_g is a normalizing constant, σ specifies the standard deviation of the kernel which is progressively decremented to get the effect of reducing the neighborhood, and $d(a, b)$ is the distance between neurons a and b. In the original Kohonen algorithm, $d(a, b)$ is the Euclidean distance between neurons a and b, i.e., $d(a, b) = \|Y_a - Y_b\|$ where Y_a and Y_b are the coordinates of the neurons on the grid.

3.1.4. Using a dissimilarity measure to determine distance metric

Zhang et al. consider the same problem with a different angle: instead of posing a metric learning problem, they define a dissimilarity measure, which has a favorable property that between-class dissimilarity is always larger than within-class dissimilarity. Then, they perform parametric learning to find a regression mapping from the input space to an Euclidean feature space, such that the dissimilarity between patterns in the input space is approximated by the Euclidean distance between points in the feature space. Once this mapping is completed, clustering is carried out in the Euclidean space. Without given into the details of the approach, we summarize especially the metric definition of Zhang et al, since in the experimentation part we only need that information to compare it with RSOM and Xing et al. approaches.

The prior information about data is assumed to be ready at hand as in the work of Xing et al.; but here, we use a more complete information on the training set such that we know the class labels of all the input signals as well. Using this label information, the similarity between patterns x_i and x_j is defined as:

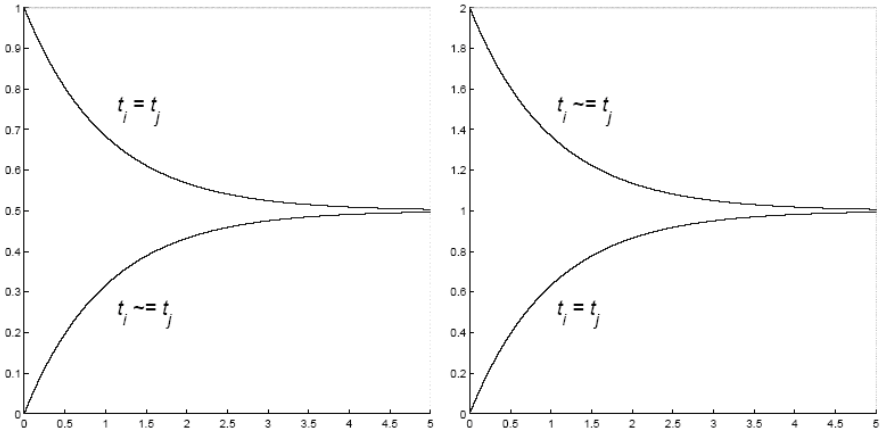
$$s_{ij} = \begin{cases} \frac{1}{2} + \frac{1}{2}\exp\left(-\frac{\|x_i - x_j\|^2}{\beta}\right) & t_i = t_j \\ \frac{1}{2} - \frac{1}{2}\exp\left(-\frac{\|x_i - x_j\|^2}{\beta}\right) & t_i \neq t_j \end{cases} \tag{3.37}$$

where $\|\cdot\|$ denotes the Euclidean norm, and $\beta > 0$ is a width parameter and $t_i = r$ if pattern i belongs to class r. The corresponding dissimilarity δ_{ij} is given by

$$\begin{aligned} \delta_{ij} &= s_{ii} + s_{jj} - 2s_{ij} \\ &= \begin{cases} 1 - \exp\left(-\frac{\|x_i - x_j\|^2}{\beta}\right) & t_i = t_j \\ 1 + \exp\left(-\frac{\|x_i - x_j\|^2}{\beta}\right) & t_i \neq t_j \end{cases} \end{aligned} \tag{3.38}$$

As illustrated in Figure 3.1, this (dis)similarity measure benefits from some nice prop-

erties for pattern discrimination. For example, the (dis)similarity between any two patterns in the same class is always larger (smaller) than the one between any two patterns belonging to different classes. Moreover, the larger the Euclidean distance between the patterns is, the smaller is the within-class similarity while the larger is the between-class similarity.



(a) s_{ij} versus $\|\mathbf{x}_i - \mathbf{x}_j\|^2$ (b) δ_{ij} versus $\|\mathbf{x}_i - \mathbf{x}_j\|^2$

Figure 3.1. Similarity and dissimilarity measures according to Zhang et al.

Besides, Zhang et al. define $d_{ij} = \sqrt{\delta_{ij}}$, and prove that d_{ij} is a metric. Then they compute distance matrix $D = [d_{ij}]_{i,j}$ accordingly for all the training samples. Their approach continues by the subsequent task of finding the embedding: mapping the points $\{\hat{x}_1, ..., \hat{x}_n\} \in^n$ such that the inter-point distance is equal to d_{ij}. They pose the mapping as a regression problem for metric approximation therefore, once solved, it can be used to map unlabeled samples to the Euclidean space where inter-point distance is d_{ij}. Details of this work can be found in [102].

3.2. A method to incorporate statistical information: KNIES

KNIES stands for Kohonen Network Incorporating Explicit Statistics. The primary difference between the SOM and the KNIES is the fact that every iteration in the training phase includes two distinct modules: the attracting module and the dispersing module. As a result of the newly introduced dispersing module the neurons maintain the overall statistical properties of the data points. Thus, although in SOM

the neurons individually find their places both statistically and topologically, in KNIES they collectively maintain their mean to be the mean of the data points which they represent. In this chapter we briefly present the KNIES algorithm, and refer to [7] for the missing details. Instead, in this section, we will bring in a modification step into the dispersion phase of KNIES, and report experimental results later.

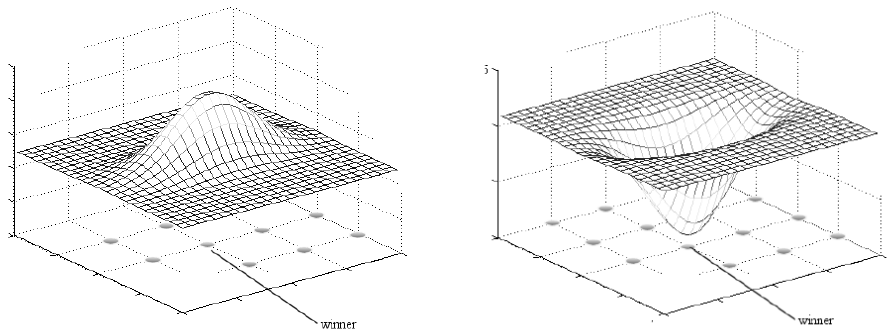
KNIES is based on Kohonen networks, and it is quite similar to original SOM in the beginning. When the data point X_i is presented to the network, the closest neuron and all the neurons within its neighborhood, $B_{j*(t)}$ referred to as "activation bubble", are migrated. This constitutes the attracting module, and in this phase the neurons within the bubble use the local information present in the points as they appear individually and move towards them. As a result of the migration we observe that the mean of all neurons moves away from their initially assigned value. Thus, the dispersing phase now forces the neurons outside the activation bubble to migrate away from their current locations to render the mean of the neurons to continue to be invariant.

First the total migration that has been affected as a result of the attraction module is kept in with the vector $\Delta Y(t) = \sum_{j \in B_{j*(t)}} (Y_j(t+1) - Y_j(t))$, where $Y_j(t)$ is the weight vector associated with the neuron j. This change $\Delta Y(t)$, is distributed among the neurons which did not participate in the attraction as

$$Y_j(t+1) = Y_j(t) - \frac{1}{M - |B_{j*}(t)|}\Delta Y, \qquad \text{if } j \notin B_{j*}(t), \tag{3.39}$$

where $|.|$ represents the cardinality operator, and M is the number of neurons. Notice that this dispersion phase is uniform, and it requires the determination of the neighborhood size in each evaluation. Whenever we use a Gaussian kernel for determining the activation bubble, we need to discretely restraint the neighborhood. In this work we proposed integrating the dispersion with the neighborhood selection by using an inverse Gaussian function. That part is done as follows: the main principle of KNIES is performed without modification and we select the winner and define the neighborhood using the Gaussian kernel. The Gaussian kernel defines the update amount of

the neuron in the neighborhood whether it is high if the neuron is close to the winner, or less, if it is far away. The same but inverted shape can be used for the dispersion as well. This is illustrated in the Figure 3.2. To total displacement is shared using the inverse Gaussian function on the right, where the dispersion amount loaded on a neuron gets increased if it is away from the winner neuron. Function gives 0 for the winner, it does not move.



(a) Update around the winner (b) Dispersion around the winner

Figure 3.2. Use of inverse of the Gaussian kernel for the dispersion phase.

This modification helps us to relax the neighborhood size parameter of the KNIES algorithm, since the attraction and dispersion neighborhoods are determined adaptively. Hence, the performance is not affected by the inefficient setting of the neighborhood size.

3.3. Experimental results

3.3.1. Additional information

We consider three clustering algorithms in our experiments and try to benefit from the background information as much as we can by using a full A matrix:

i. Kohonen's self-organizing map SOM.

ii. K-means using the distance metric with full A learned from S and D.

iii. RSOM with full A learned from S and D.

Matrix A is learned as suggested by Xing et al. In fact, we downloaded and used their MATLAB code available at [100], which implements their projection algorithm [99]. We also adopted their measure of accuracy

$$AC = \sum_{i>j} \frac{I\left\{I\left\{c_i = c_j\right\} = I\left\{\widehat{c}_i = \widehat{c}_j\right\}\right\}}{0.5N(N-1)}, \tag{3.40}$$

to compare the performances of the clustering algorithms. Here $\widehat{c}_i$, $i = 1, \ldots, N$ are the clusters to which data vector X_i is assigned by one of the three clustering algorithms mentioned above, and c_i, $i = 1, \ldots, N$ are the true clusters for data vectors X_i. The indicator function $I\{u, v\} = 1$ if $u = v$ and 0 otherwise. As it can be noticed N is the number of data points. Thus each term in AC is the probability that any input pair (i, j) randomly selected from the same cluster are assigned to the same cluster by the clustering algorithm of concern. In short, larger AC means better performance. We use this measure with the correction Xing et al. suggest [99].

We tested RSOM on 6 data sets from UC Irvine repository [92]. In our experiments the correct clustering is given by the labels in these sets. We duplicated their experimental conditions and ran our experiments using a little side-information and a large amount of side-information. The set of similar points S is generated by picking a random subset of all pairs of points belonging to the same class. For experiments with little side-information, the size of the subset is chosen so that the number of resulting connected components is roughly 90% of the size of the original data set. This becomes 70% for experiments with much side-information. Figure 3.3 summarizes our results. The first three bars are respectively for SOM, K-means with full A, RSOM with full A in case of little side-information. The following three bars are the much side-information version of the first three. N and c respectively denote the number of data vectors and distinct classes in the data set. d indicates the dimensionality of the data. Figure 1 consists of the average accuracies. These values are obtained by averaging the accuracies obtained by running each clustering algorithm for 5 random sets S of similar points, and with 10 multiple restarts for each set, which makes a total of 50 runs for each UCI data set considered.

By comparing the first bar with the third, and the forth bar with the sixth, we can say that, in general, using a learned metric improves the performance of SOM. The only exception has occurred in "protein" data set. In this case the distance metric seems harder to learn and RSOM does not provide any benefit over SOM. An increase in the amount of side-information has slightly more effect on the performance of RSOM than those of K-means. According to our experimental results we can say that RSOM performs better than K-means with full A. The only exceptions are "soybean" data set with much side-information (bars five and six) and "protein" data set with little side-information (bars two and three).

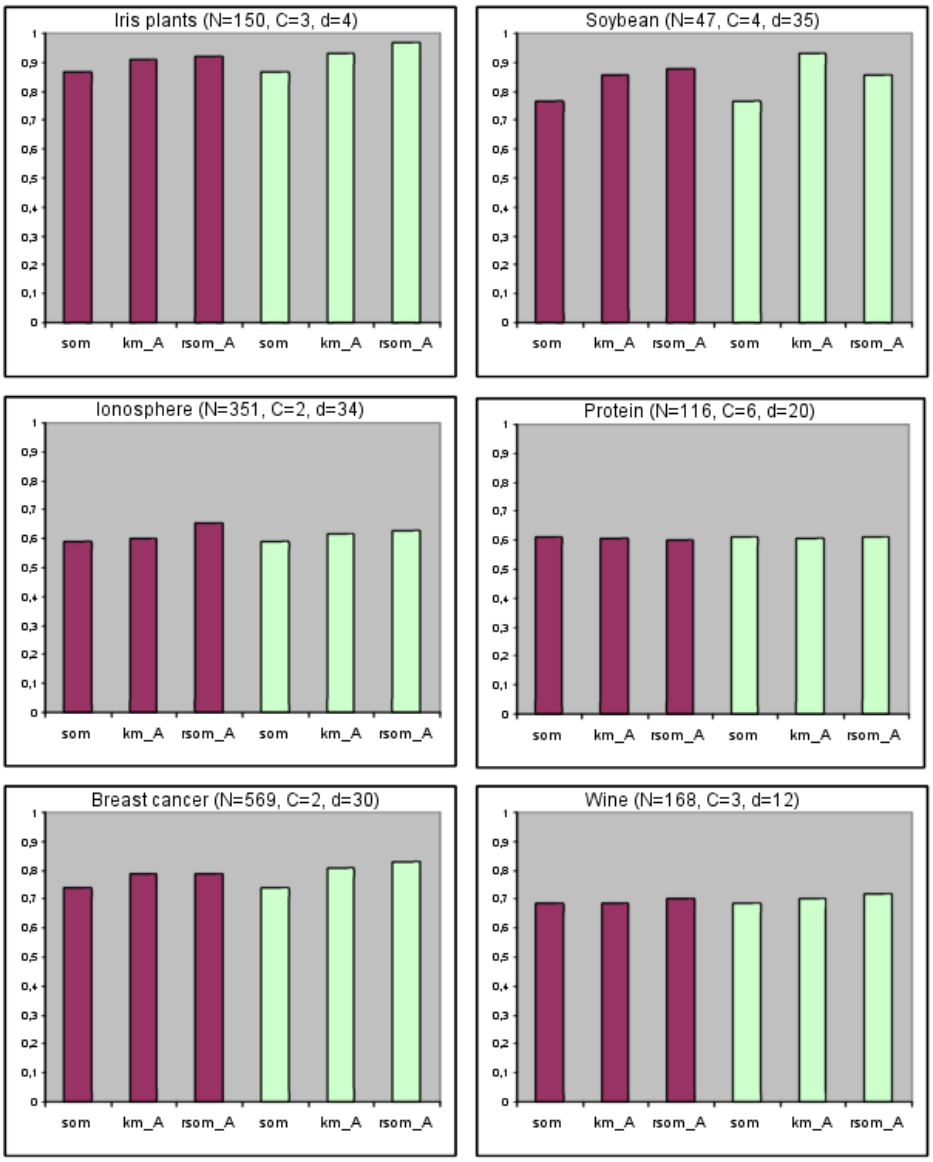


Figure 3.3. Average clustering accuracy on six UCI data sets.

Table 3.1. Performances of different KNIES modifications on UCI datasets

Datasets	**SSOM**	KNIES	KNIES-1	(KNIES-2)
ionosphere	0.83	0.86	0.87	0.83
iris	0.97	0.97	0.97	0.97
pima-indians-diabetes	0.71	0.70	0.70	0.71
protein	0.59	0.63	0.57	0.59
soybean-small	0.85	0.85	0.96	0.96
wdbc	0.91	0.89	0.91	0.91
wine	0.72	0.73	0.73	0.72

3.3.2. Incorporating statistical information

Three types of KNIES based algorithms are used in the experimentations: KNIES with original neighborhood definition (KNIES), with inverse Gaussian kernel (KNIES-1), and with smoothed inverse Gaussian kernel (KNIES-2). The last one (KNIES-2) is actually the same as the KNIES-1 except that the updating amounts are divided by the number of neurons in order to reduce the effect of the dispersion iterations. UCI data sets are used again and clustering is performed using these algorithms. Performances are compared with standard SOM (SSOM) using previously defined accuracy measure (3.40).

Table 3.1 summarizes first part of the experimentations. It seems that KNIES with inverse Gaussian kernel KNIES-1 improves the final clustering performance slightly in some data sets like ionosphere, soybean and wine, but overall quality can be considered as the same. On the other hand there is no much difference between KNIES-1 and KNIES-2, therefore we may conclude that smoothing the dispersion is not necessary if we apply KNIES-1.

In the second part of the experimentation, we studied the effect of KNIES in learning speed. In terms of quality, KNIES does not promise much improvement but, when we follow the sequence of SOM enhanced with KNIES, we noticed that the learning

takes place in early iterations. This is also a result shared by Aras et al. when the method was initially established to solve the Traveling Salesman Problem [7]. To test this claim, we repeated the previous experimentation and reported the state of the accuracies at different stop points. For instance, we solve the clustering problem with one of the algorithms for 500 epochs and report the AC measure every 50 epochs. Eventually, performance of all the algorithms increase as long as we proceed, but the speed of `KNIES` algorithms are higher than the SSOM. We summarize the results in Figure 3.4. Since previous results show that `KNIES-1` outperforms `KNIES` and `KNIES-2`, we only report the comparison of SSOM and `KNIES-1`.

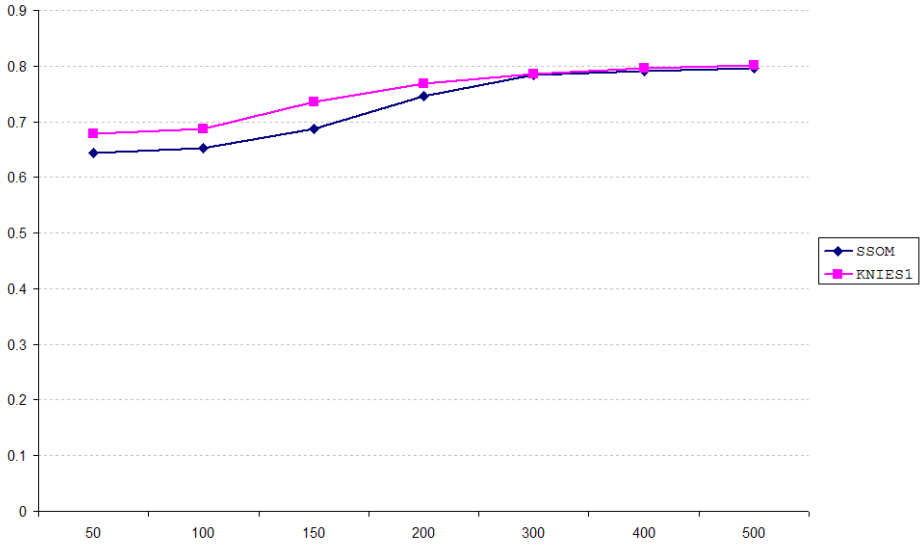


Figure 3.4. Comparison between SSOM and KNIES-1 convergence speed.

Although there exist some counter examples (`wdbc` data set), in most of the data sets, `KNIES-1` leads to better accuracy results than the standard SOM in early stages, but they converge to the same value. We conclude that all SOM algorithms with or without KNIES enhancement converge to the same point ultimately but, use of KNIES modules leads to early convergence. In other terms, we may express this as if we had to stop earlier, KNIES gives more accurate clusters than SSOM.

4. DETERMINISTIC MULTI FACILITY WEBER PROBLEM

Deterministic location-allocation problems are concerned with locating a set of facilities and allocating to them the demands of a set of customers with known locations and demand quantities so that the total distribution cost is minimized. Supply centers such as plants and warehouses may constitute the facilities; retailers and dealers may be considered as customers.

When the facility locations have to be selected from a given set candidate locations, the corresponding location-allocation problem (LAP) becomes a discrete optimization problem. Otherwise, facilities can be located anywhere in the Euclidean plane and we have a continuous LAP. The latter problem is known as the Weber problem [98]. If there exist capacity constraints associated with the facilities, the resulting problem is the capacitated Weber problem. If the facilities are assumed to have infinite capacity, then we have the uncapacitated version of the Weber problem. A final categorization is with respect to the number of facilities to be opened: the problem is called the single-facility Weber problem if the objective is to determine an optimal location for a single facility; it becomes the multi-facility Weber problem when more than one facility have to be located optimally. As it can be easily observed, each customer is served from the nearest facility for uncapacitated problems. However, this result is not valid for the capacitated multi-facility Weber problem.

The mathematical formulation of the uncapacitated multi-facility Weber problem (MFWP in the sequel) can be stated as:

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} u_{ij} d\left(\mathbf{x}_i, \mathbf{a}_j\right) \tag{4.1}$$

s.t.

$$\sum_{i=1}^{m} u_{ij} = h_j \quad j = 1, ..., n \tag{4.2}$$

$$u_{ij} \geq 0 \qquad i = 1, ..., m; j = 1, ..., n \tag{4.3}$$

Here, n is the number of customers and m is the number of facilities to be located. h_j and $\mathbf{a}_j = (a_{j1,} a_{j2})^T$ represent, respectively, the demand and coordinates of customer j. $\mathbf{x}_i = (x_{i1,} x_{i2})^T$ denotes the (unknown) coordinates of facility i to be located. u_{ij} is the amount shipped from facility i to customer j and $d\left(\mathbf{x}_i, \mathbf{a}_j\right)$ is a function which determines the distance between facility i and customer j. The most frequently used distance functions are listed in Table 4.1.

Table 4.1. Most frequently used distance functions.

Euclidean distance	$d\left(\mathbf{x}_i, \mathbf{a}_j\right) = \left(\left\|x_{i1} - a_{j1}\right\|^2 + \left\|x_{i2} - a_{j2}\right\|^2\right)^{1/2}$
Squared Euclidean distance	$d\left(\mathbf{x}_i, \mathbf{a}_j\right) = \left(\left\|x_{i1} - a_{j1}\right\|^2 + \left\|x_{i2} - a_{j2}\right\|^2\right)$
Rectilinear distance	$d\left(\mathbf{x}_i, \mathbf{a}_j\right) = \left\|x_{i1} - a_{j1}\right\| + \left\|x_{i2} - a_{j2}\right\|$
ℓ_p distance	$d\left(\mathbf{x}_i, \mathbf{a}_j\right) = \left(\left\|x_{i1} - a_{j1}\right\|^p + \left\|x_{i2} - a_{j2}\right\|^p\right)^{1/p} \quad p \geq 1$

Notice that rectilinear and Euclidean distances are special cases of the ℓ_p distance. In particular, one can obtain the rectilinear distance when $p = 1$ and the Euclidean distance when $p = 2$. It is worth mentioning that for $p < 1$ the triangle inequality property is violated and thus ℓ_p distance is not a norm. Squared Euclidean distance is not a norm either, because it violates the homogeneity property of norms. Since the objective of the MFWP is to minimize the distribution cost of sending goods from facilities to customers and the distribution cost is proportional to the road distance between locations, it is reasonable to have very accurate estimates of road distances. It has been shown that when ℓ_p distance is used instead of its special cases such as

Euclidean ($p = 2$) or rectilinear ($p = 1$) distances, road distances can be estimated with better accuracy because of the existence of p as a parameter of the distance function [16].

An equivalent formulation of the MFWP can easily be obtained by dividing both sides of equalities (4.2) and inequalities (4.3) by h_j and defining new variables $y_{ij} = u_{ij}/h_j,\ j = 1, ..., n$ which are restricted within the interval $[0, 1]$. Besides, y_{ij}'s are either zero or one at optimality since serving each customer from its nearest facility is an optimal policy for the MFWP. Hence, it is possible to make y_{ij}'s to be binary variables with $y_{ij} = 1$ if customer j is served from facility i and $y_{ij} = 0$ otherwise. Then,

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} h_j y_{ij} d\left(\mathbf{x}_i, \mathbf{a}_j\right) \tag{4.4}$$

s.t.

$$\sum_{i=1}^{m} y_{ij} = 1 \quad j = 1, ..., n \tag{4.5}$$

$$y_{ij} \in \{0, 1\} \quad i = 1, ..., m; j = 1, ..., n \tag{4.6}$$

becomes an alternative formulation for the MFWP. No matter which distance function is used both formulations have non–convex objective functions [26] and solving the MFWP turns out to be a difficult task. The difficulty of the MFWP increases with the problem size, which is determined by the number of customers, n, and the number of facilities to be located, m. This is due to the fact that the number of local minimum solutions grows exponentially with increasing n and m. As an example, we quote some experimental results from Eilon et al. [32]. Table 4 contains these results where the authors solve a 50-customer problem using a heuristic and report the number of local minima as well as the percent deviation of the worst solution from the best solution in a number of trials.

Table 4.2. The difficulty of the MFWP (due to Eilon et al.[32]).

Number of facilities	Number of trials	Number of local minima	Best solution	% Deviation of worst solution
1	2	1	180.13	-
2	230	18	135.52	6.9
3	200	26	105.21	25.8
4	185	37	84.15	15.2
5	200	61	72.14	40.9

4.1. Nontraditional solution methods

There have been both exact and heuristic methods in the literature to solve the MFWP. An excellent review of these methods is given in Brimberg et al. [13] Almost all of the exact methods can solve small sized problems except the one of Krau [58] which computes optimal solutions for a 51-customer problem with the number of facilities in the range (2, 36) and another 287-customer problem with the number of facilities in the range (2, 100). His method integrates column generation with global optimization methods and branch-and-bound. Furthermore, for the rectilinear MFWP, Hansen et al. [41] couple Krau's algorithm with a bundle method due to du Merle et al. [31] to solve problems consisting of up to 1000 customers and 100 facilities. However, both methods need an initial solution which must be very close to an optimal solution. Hence the reason for the requirement of heuristic methods is twofold. First, they are needed to generate initial solutions for exact methods. Second, they are capable of providing good solutions for very large problems.

A large number of heuristics have been suggested in the literature ever since the appearance of Cooper's work [24] where he decomposes the MFWP into two parts referred to as location and allocation phases. Therefore this heuristic is called alternate location-allocation (ALA) heuristic and is based on the following characteristic of the problem: given the location of the facilities it is straightforward to determine the allocation of customers to the facilities in which the demand of a customer is served

from the closest facility. On the other hand, given the allocation of customers to the facilities, it is possible to find an optimal location for each facility because this phase of the heuristic corresponds to a single facility problem which can be solved for different distance measures by a gradient descent method. One of the most widely used descent methods is the Weiszfeld procedure [95] which is shown by Brimberg and Love [17] to converge for l_p distances with $1 \leq p \leq 2$. When the distance measure is the squared Euclidean distance, there is even an analytical formula for the optimal location of each facility, i.e., the weighted centroid of the customer locations where the weights are the demand quantities at the customers. Cooper's heuristic starts from an initial random configuration of the facilities and solves alternatively location and allocation phases until the solution does not change between two consecutive iterations. In order to avoid some inferior local minima, the multi-start version of the heuristic is run with different random configurations.

Brimberg et al. [13] reviewed and compared several heuristics following that of Cooper's can be found in where the authors categorize heuristics as being old, recent, and new. Old heuristics are those which appeared until the projection method developed by Bongartz et al. [11] Among these, the best methods in terms of solution quality turn out to be Cooper's ALA heuristic and Bongartz et al.'s projection method. Recent heuristics consist of Tabu search method by Brimberg and Mladenovic [18], p-median heuristic by Hansen et al. [42], and variable neighborhood search method by Brimberg and Mladenovic [19]. Finally, new heuristics category includes a new genetic algorithm based on the work of Houck et al. [46] and facility relocation heuristics in which the neighborhood of a current solution is constructed by relocating facilities to customer locations which do not coincide already with another facility. This idea is considered new because previous heuristics that perform local search in the neighborhood of a current solution generate the neighborhood by reallocating customers to other facilities. Heuristics from these three categories are compared in Brimberg et al. [13] on the basis of both solution quality and computation time using four benchmark instances. Gamal and Salhi [38] propose a two-phase approach which they call a cellular heuristic. In Phase 1, they apply the multi-start version of Cooper's ALA heuristic to generate several solutions. In Phase 2 they define a grid structure by dividing the continuous

space into well-defined cells. Once the cells are constructed, facility locations found in Phase 1 are assigned to corresponding cells of the grid. Based on two criteria, i.e., frequency of the cells (the number of facilities in each cell) and compatibility between cells (the number of times two cells happen to be in the same solution provided by ALA heuristic), the most attractive cells are selected where the number of the selected cells is equal to the number of facilities to be located. Finally, ALA heuristic is applied with the locations of facilities initialized at the centroids of the selected cells.

Heuristics based on simulated annealing and artificial neural networks have also been developed for solving the MFWP. Liu et al. [62] propose a simulated annealing heuristic for the rectilinear MFWP and compare its performance with the method of Love and Morris [65] which can find the exact solution when rectilinear distances are used and with heuristic H5 introduced in Love and Juel [63]. Based on experimental results for instances with $m = 2, 3, 4, 5$ and $n = 20, 40, 60, 80, 100, 150$ Liu et al. [62] report that in terms of solution quality the performance of the simulated annealing heuristic is comparable to the other methods. In addition, it is better than the exact method of Love and Morris [65] and competes well with H5 of Love and Juel [63] in terms of computation time.

4.1.1. Vector quantization

In this section we will show that the minimization of the quantization error of VQ given in Equation (2.6) is the same as the alternative formulation of the MFWP. Note that VQ aims at determining the optimal values of $\mathbf{w}_k$ by minimizing the quantization error $E(\mathbf{w})$. The optimization problem is then

$$\min \sum_{\mathbf{v}} d(\mathbf{w}_c, \mathbf{v}) = \sum_{\mathbf{v}} \min_k d(\mathbf{w}_k, \mathbf{v}). \tag{4.7}$$

It can be easily seen that the term within the summation, $\min_k d(\mathbf{w}_k, \mathbf{v})$, is equivalent to the following linear programming problem:

$$\min \sum_k \pi_k d\left(\mathbf{w}_k, \mathbf{v}\right) \tag{4.8}$$

s.t.

$$\textstyle\sum_k \pi_k = 1 \tag{4.9}$$

$$\pi_k \geq 0 \tag{4.10}$$

In this problem the variables are π_k and constraints (4.9) and (4.10) describe a simplex whose vertices are unit vectors and the zero vector. As a result the objective function is minimized if $\pi_k = 1$ for $k = \arg\min_l d\left(\mathbf{w}_l, \mathbf{v}\right)$ while $\pi_k = 0$ for all other k's. Hence the nonnegativity constraints $\pi_k \geq 0$ can be replaced by forcing π_k to be binary variables (i.e., $\pi_k \in \{0, 1\}$). As a result, the minimization of the quantization error $E\left(\mathbf{w}\right)$ can be written equivalently as

$$\min \;\; E\left(\mathbf{w}\right) = \sum_{\mathbf{v}} \sum_k \pi_k d\left(\mathbf{w}_k, \mathbf{v}\right) \tag{4.11}$$

s.t.

$$\textstyle\sum_k \pi_k = 1 \tag{4.12}$$

$$\pi_k \in \{0, 1\} \tag{4.13}$$

Note the similarity between formulation (4.11)–(4.13) and the alternative formulation of the MFWP given with (4.4)–(4.6). As a matter of fact, if the demand is the same for all customers, namely it is a constant for all j, then the two formulations turn out to be equivalent. In particular, the following relationship exists between the two problems. The customer locations $\left\{\mathbf{a}_j = \left(a_{j1,} a_{j2}\right)^T : j = 1, \ldots, n\right\}$ of the MFWP correspond to the input vectors $\mathbf{v}$ of VQ and the unknown coordinates for the facilities $\left\{\mathbf{x}_i = \left(x_{i1,} x_{i2}\right)^T : i = 1, \ldots, m\right\}$ correspond to the weight vectors of the output units (codebook vectors).

When customers have different demands, the basic idea of VQ is still applicable for solving the MFWP, but a slight modification in the solution method is necessary. This modification is used by Lozano et al. [67], where instead of presenting the input vectors (customer coordinates) an equal number of times to the network, customers with higher demand are input more frequently than other customers with lower demand values. In particular, the frequency in which customer locations are presented is approximated by the discrete probability distribution of the customer demands. In a forthcoming section where computational results are provided we use this idea to solve the MFWP using VQ. Now we turn our attention to the solution procedure of VQ by competitive neural networks.

In general, no closed-form solution exists to find optimal values for $\mathbf{w}_k$. We can use, however, stochastic gradient descent method as follows [54]: at each instant of time t an input vector $\mathbf{v}(t)$ is presented to the network. The weight vector $\mathbf{w}_c(t)$ of the closest output unit (or closest code–book vector) is updated such that the distance $d(\mathbf{w}_c(t), \mathbf{v})$ is decreased whereas weight vectors of other output units, $\mathbf{w}_k(t)$ with $k \neq c$ are not changed. The update of $\mathbf{w}_c(t)$ is given with the following formula:

$$\Delta \mathbf{w}_c(t) = \mathbf{w}_c(t+1) - \mathbf{w}_c(t) = -\alpha(t) \frac{\partial d(\mathbf{w}_c(t), \mathbf{v})}{\partial \mathbf{w}_c(t)} \tag{4.14}$$

where learning rate (or step length) $\alpha(t)$ assumes monotonically decreasing scalar values with $0 \leq \alpha(t) \leq 1$. Most of the time $\alpha(t)$ goes to zero in time.

When the squared Euclidean distance is used for $d(\mathbf{w}_c(t), \mathbf{v})$, i.e., $d(\mathbf{w}_c(t), \mathbf{v}) = \|\mathbf{v} - \mathbf{w}_c(t)\|^2 = \min_k \|\mathbf{v} - \mathbf{w}_k(t)\|^2$, the reconstruction error becomes the squared error criterion $E(\mathbf{w}) = \sum_{\mathbf{v}} \sum_k \|\mathbf{v} - \mathbf{w}_c(t)\|^2$ and the update formula (4.14) is expressed as:

$$\Delta \mathbf{w}_c(t) = \alpha(t)(\mathbf{v}(t) - \mathbf{w}_c(t)). \tag{4.15}$$

Consequently, the following two equations are used to minimize the quantization error

for the squared Euclidean distance:

$$\begin{aligned} \mathbf{w}_c(t+1) &= \mathbf{w}_c(t) + \alpha(t)(\mathbf{v}(t) - \mathbf{w}_c(t)) \\ \mathbf{w}_k(t+1) &= \mathbf{w}_k(t) \quad k \neq c \end{aligned} \tag{4.16}$$

This is the form of update equations employed in most of the VQ applications [54]. Only the weight vector of the winner of the competition is updated. When a distance measure other than the squared Euclidean distance should be used, not only the winner unit has to be identified based on the specified distance measure, but also the update of the weight vector of the winner unit must be carried out accordingly which leads us to the use of RVQ. Table 4.3 summarizes winner selection and componentwise update formulae for the squared Euclidean, Euclidean, rectilinear and ℓ_p distances. Input vectors are q-dimensional and the updates have to be carried out for $j = 1, ..., q$. The last column gives the amount of update for each component of the weight vector of the winner $\mathbf{w}_c(t)$: $\Delta w_{cj}(t) = w_{cj}(t+1) - w_{cj}(t)$. For notational convenience we drop the index c of the winner and time index t.

Table 4.3. Winner selection and update formulae.

Distance Function	**Winner Selection**	**Winner Update** $\Delta w_j(t), \quad j = 1, \ldots, q$
Euclidean	$\arg\min_k \left\{ \left(\sum_{j=1}^{q} \lvert v_j - w_{kj} \rvert^2 \right)^{1/2} \right\}$	$\alpha(t) \dfrac{(v_j - w_{cj})}{\left(\sum_{j=1}^{q} \lvert v_j - w_{cj} \rvert^2 \right)^{1/2}}$
Squared Eucl.	$\arg\min_k \left\{ \sum_{j=1}^{q} \lvert v_j - w_{kj} \rvert^2 \right\}$	$\alpha(t)(v_j - w_{cj})$
Rectilinear	$\arg\min_k \left\{ \sum_{j=1}^{q} \lvert v_j - w_{kj} \rvert \right\}$	$\alpha(t)\text{sign}(v_j - w_{cj})$
ℓ_p distance	$\arg\min_k \left\{ \left(\sum_{j=1}^{q} \lvert v_j - w_{kj} \rvert^p \right)^{1/p} \right\}$	$\alpha(t) \dfrac{\lvert v_j - w_{cj} \rvert^{p-1}}{\left(\sum_{j=1}^{d} \lvert v_j - w_{cj} \rvert^p \right)^{p/(1-p)}}$

One of inherent shortcomings of a VQ application with a distance function different from the squared Euclidean distance is that the weight vector of the winner unit or equivalently the coordinates of the winner facility may be updated in such a way that its final location falls off the convex hull of the customer locations. This can be easily illustrated by an example for the rectilinear distance. Assume that the

coordinates of a customer which lies in the convex hull are presented to the network. As long as neither of the coordinates of the winner facility coincide with those of the customer, the amount of update of the coordinates of the facility along both axes will be $-\alpha(t)$ or $+\alpha(t)$ because $\text{sign}(v_j - w_j) = \pm 1$. This means that if the current distance along one of the dimensions between the customer and the winner facility is less than $|\alpha(t)|$, then the new coordinates of the facility will move to the other side of the customer location. In other words, the new location of the facility will be outside the convex hull and may be far away from customer locations. Once this occurs for a facility, it becomes possible that this facility will never win until the algorithm terminates and its coordinates will never be updated. The consequence is that this facility may not serve any of the customers. In such a case, the solution obtained by VQ corresponds to a solution of the MFWP with $m-1$ facilities. It should be noted that when the squared Euclidean distance is used, the updated coordinates of the winner facility will always be between the current location of the facility and the customer location. This can be explained as follows. The new coordinate along dimension j is given as $w_j(t+1) = w_j(t) + \alpha(t)(v_j - w_j(t))$. If we rewrite this equation we have $w_j(t+1) = (1-\alpha(t))\, w_j(t) + \alpha(t)\, v_j$. In other words, the new value of the jth coordinate of the facility is found by taking the convex combination of the current value of the jth coordinate of the facility, $w_j(t)$, and the jth coordinate of the customer location, v_j. Hence, the VQ algorithm (using squared Euclidean distance) always terminates with facility locations that lie in the convex hull of the customers.

The complication due to the use of a distance function other than the squared Euclidean can be resolved by the following final adjustment (or post-processing) step added to the end of the VQ algorithm. If there are k facilities which are located outside the convex hull of customers and therefore do not serve any of them, then the goal of this adjustment step is to locate k new facilities within the convex hull in such a way that they serve at least one customer. Note that this step necessarily improves the objective function value of the MFWP because adding more facilities can only decrease the objective value. We locate these k new facilities in a greedy way. We prepare a list by sorting the customers in decreasing order with respect to their contribution to the objective function. This contribution can be formulated as the product of the distance

between a customer and its facility and customer's demand. Then, k new facilities are located at the sites of the first k customers in this list. The adjustment step does not increase the computational time because the required distance information to compute the contribution to the objective function is readily available when the VQ algorithm terminates.

No matter which distance function is used, there is an inherent problem with competitive learning. If the weight vector of an output unit (i.e., coordinates of a facility) is initialized far from any input vector (customer location), it never wins the competition and hence is never updated. Therefore, the choice of initial weight vectors $\mathbf{w}_k(0)$ becomes an important issue, which can be dealt with in several ways. First, weights are initialized from the inputs. For example, if we want to locate m facilities to serve n customers, the initial location of the facilities are randomly assigned to m customer locations rather than arbitrarily locating them anywhere on the plane. Second, rather than updating only the weight vector of the winning output unit, the weight vectors of some of the "loser" output units (i.e., coordinates of the loser facilities) are updated as well with smaller $\alpha(t)$. In other words, the "winner-take-all" approach of competitive learning is modified into "winner-take-some" approach of competitive learning. The so-called self-organizing maps actually perform exactly this idea.

4.1.2. Self-organizing maps

The first paper in which Kohonen self-organizing map (SOM) is employed in solving the MFWP is due to Lozano et al. [67] SOM and its variants are neural network models which are applied to some combinatorial optimization problems including the Euclidean traveling salesman problem (e.g., Aras et al. [7], Aras et al. [8]), Euclidean Hamiltonian Path Problem (Altınel et al. [4]), vehicle routing problem (e.g., Modares et al. [74]), AGV scheduling problem (Soylu et al. [90]). Lozano et al. [67] consider only a single test instance in their experiments. It includes data for 49 customers selected from major towns in Spain and Portugal. This instance is solved with a number of facilities ranging from 2 to 10 by using two variants of SOM. To improve the best solution obtained by either one of these two SOM methods, a fine-tuning step is added

where ALA heuristic of Cooper is performed. It is important to note that the authors apply the SOM approach with the squared Euclidean distances between facilities and customers and mention that the *same* approach can be adopted with no changes in winner selection and weight update rules of the SOM to find an approximate solution for the case with Euclidean distances. The only difference occurs in the ALA process. In the case of squared Euclidean distance the single facility location problems are solved analytically by relocating the facility to the demand-weighted centroid of the customer locations served by that facility whereas iterative Weiszfeld algorithm is used for the Euclidean distance. Although not stated explicitly in this paper, figures illustrating the objective value (distribution cost) as a function of the number of facilities reveal that the improvement provided by the ALA process is more significant when the distance is Euclidean. This observation motivated us to focus on the modifications necessary in the SOM approach to efficiently handle the MFWP with different distance functions.

Recently, in their paper Hsieh and Tien [47] address this issue and solve the rectilinear MFWP by a heuristic method based on SOM. The variant of SOM proposed in this study is the same as the one used by Lozano et al. [67] with the following exceptions. First, a new refinement procedure is employed as a post-processing step instead of the ALA process of Lozano et al. This refinement procedure, called SOFMGR, is a guided search procedure based on the property introduced by Wendell and Hurter [48]; namely the optimal solution of the rectilinear MFWP occurs in those points within the convex hull of customer locations which are found in the intersection of all vertical and horizontal lines passing through these locations. The second difference is in the winner selection rule of the SOM where rectilinear distances are used in measuring the distances between the output units (neurons) of the SOM and customer locations. However, the authors point out that no modification is required in any other step of the SOM other than the winner selection rule. We will show in the sequel that the weight adaptation and winner selection steps in both VQ and SOM are interrelated and one step cannot be considered independent of the other. Incorporation of the type of the distance function *both* in the winner selection rule and in the weight update rule could eliminate the refinement procedure as a postprocessing step following the SOM output. Indeed, computational results obtained with benchmark problems

justify this expectation. The third deviation from Lozano et al.'s work occurs in the way the customer locations are presented to the SOM network. The goal of both approaches is that the frequency in which customer locations are presented to the SOM is approximated by the discrete probability distribution of the customer demands, i.e., $p(h_j) = h_j / \sum_{k=1}^{n} h_k$. This implies that a customer with higher demand is input to the SOM more frequently than another customer whose demand is lower, which results in giving more weight to the former customer. The way how this is accomplished differs between the two papers, and Hsieh and Tien [47] claim that their method of input presentation is faster than the method of Lozano et al. [67]

The following update equation is employed for the revised SOM to solve the MFWP,

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \alpha(t)\,\Lambda_{k,c}(t)\,\nabla_{\mathbf{w}_k} d(\mathbf{v} - \mathbf{w}_k(t)), \tag{4.17}$$

where the neighborhood function $\Lambda_{k,c}(t)$ is maximum when $k = c$ and decreases as the distance between the output unit k and the winner unit c increases. Different distance measures may be used in calculating this distance. In one dimensional map the distance might be simply $|k - c|$ (the absolute value of the difference between the units' indices) whereas in the two-dimensional map, it can be computed as the Euclidean distance between the indices of units k and c, i.e., $\Lambda_{k,c}(t) = \|\mathbf{r}_k(t) - \mathbf{r}_c(t)\|$ where vector $\mathbf{r}_k(t)$ denotes the index vector of output unit k at time t. For better convergence, $\Lambda_{k,c}(t)$ should be a decreasing function of time. That is, at the beginning a large neighborhood is preferred where weight vectors of many output units are updated by moving towards the input vector $\mathbf{v}(t)$. Towards the end of learning, only those output units which are very close to the winner should undergo an update. A suitable form for $\Lambda_{k,c}(t)$, which is at the same time the mostly adopted one is the Gaussian kernel given as

$$\Lambda_{k,c}(t) = \exp\left(\frac{-\|\mathbf{r}_k(t) - \mathbf{r}_c(t)\|^2}{\sigma^2(t)}\right). \tag{4.18}$$

Here $\sigma(t)$ defines the width of the neighborhood at time t. $\sigma(t)$ starts with a large

value which corresponds to a large neighborhood and throughout the iterations its value becomes smaller such that only the winner unit gets updated towards the very end of the iterations.

Now we are ready to perform experiments with VQ and SOM methods for a number rectilinear and Euclidean MFWP test instances. As the last reminder, in VQ and SOM input vectors $\mathbf{v}$ represent the coordinates of customer locations and the weights vectors of output units $\mathbf{w}_k(t)$ represent the coordinates of the facilities.

4.2. Computational Results

The VQ and SOM methods with update equations modified to handle any ℓ_p distance, which we call RVQ and RSOM in the sequel, are implemented for a number of test instances. They are categorized as rectilinear and Euclidean instances and listed in Table 4.4 and Table 4.5, respectively. The first column of both tables includes instance labels. The number of customers, n, and the number of facilities to be located, m, are given in the second and third columns, respectively. The source of each instance is mentioned in the last column. For example, LJ20/100 which consists of 20 customers is a subset of LJ100 that is obtained from Love and Juel [63].

Recall that the objective of this study is twofold. First, we want to show that it is possible to eliminate the refinement procedure proposed by Hsieh and Tien [47] as a post-processing step following the SOM output when solving the rectilinear MFWP. For this purpose we solve rectilinear instances listed in Table 4.4 by RVQ and RSOM methods. Both methods incorporate the rectilinear distance *not only* in the winner selection rule *but also* in the weight update rule as given in Table 4.3. For comparison we also include results obtained with the standard SOM (SSOM) method which uses the squared Euclidean distance in update equations.

We also solve Euclidean MFWPs given in Table 4.5 by RVQ, RSOM as well as SSOM methods. This enables us to justify whether the SSOM can be adopted with no changes in winner selection and weight update rules of the SOM to find an approximate

Table 4.4. Rectilinear instances.

Instance	n	m	**Source**
LJ100	100	2–5	Love and Juel [63]
LJ20/100	20	2–5	subset of LJ100
LJ40/100	40	2–5	subset of LJ100
LJ60/100	60	2–5	subset of LJ100
LJ80/100	80	2–5	subset of LJ100
LM16	16	3	Love and Morris [65]
LM20	20	3	Love and Morris [65]
LM30/100	30	2	subset of LJ100
LM35/100	35	2	subset of LJ100
Liu150	150	2–5	Liu et al. [62] (superset of LJ100)

Table 4.5. Euclidean instances.

Instance	n	m	**Source**
Eil50	50	2–15	Eilon et al. [32]
Bon287	287	2–15	Bongartz et al. [11]
Cooper7 (7 problems)	7	2	Cooper [24]
Rosing15	15	2–6	Rosing [86]
Rosing20	20	2–6	Rosing [86]
Rosing25	25	2–6	Rosing [86]
Rosing30	30	2–5	Rosing [86]
Lozano49	49	2–20	Lozano et al. [67]

solution for the Euclidean MFWPs as suggested by Lozano et al. [67]

All the methods were coded in Microsoft Visual Basic 6.0 and run on a notebook computer equipped with a 1 GHz Celeron processor (411 MFLOPS) and 256 Mbyte RAM. For some problems in Table 4.4 and Table 4.5, such as `Eil50`, demands of all customers are equal to 1, i.e., $h_j = 1$ for $j = 1, \ldots, n$. For these problems, customer coordinates (i.e., input vectors) are presented an equal number of times to the competitive neural network which implements RVQ, RSOM or SSOM. When solving instances where demand quantities are different for customers, we use the approach of Lozano et al. [67] by presenting the customer coordinates in proportion to their demand. For example, if $h_{j_1} = 2h_{j_2}$, then coordinates of customer j_1 are input twice as many times as those of customer j_2. In this fashion, we make sure that the frequency in which customer locations are presented to the neural networks of RVQ, RSOM and SSOM is approximated by the discrete probability distribution of customer demands, i.e., $p(h_j) = h_j / \sum_{j=1}^{n} h_j$. Furthermore, all of the reported values are obtained by appending the adjustment step to the end of RVQ and RSOM. As explained previously, when the solution given by RVQ or RSOM suggests that k facilities be located outside the convex hull of customer locations and do not serve any customers, then this adjustment step moves these facilities back into the convex hull in a greedy way. They are relocated at the locations of those customers whose contribution to the objective function is the largest. Learning rate $\alpha(t)$ is set to 0.9 at the beginning and is reduced geometrically through the epochs, i.e., $\alpha(t_e + 1) = \gamma\alpha(t_e)$, where t_e denotes the number of the epoch. $\gamma \in (0, 1)$ is chosen in such a way that at the end of the learning the value of $\alpha(t)$ becomes approximately zero. The number of epochs is determined to be 1000 for all computations.

4.2.1. Results with Rectilinear instances

In Table 4.6, we compare the results obtained with RVQ, RSOM and SSOM for rectilinear instances. These results are expressed as a percent deviation from the optimal or best known value of the objective function given as $100 \times (z - z^*)/z^*$ where z^* denotes the optimal or best known objective value. Each instance is solved 10 times

by each method. Hence, the average and the best values for each method denote the average and the best objective function values over 10 runs. Similarly, CPU times (in seconds) are averages over 10 values. Each run of RVQ starts with a randomly generated initial facility locations. For RSOM and SSOM, input vectors (i.e., customer coordinates) are presented to the network in a different random sequence. Note, however, that although this sequence is different for each run, the same sequence is used in all epochs of the same run.

The last row of Table 4.6 shows the average quality of the solutions (in terms of percent deviation) provided by the three methods as well as their CPU times over the instances. Based on this last row, we can conclude that RVQ and RSOM outperform SSOM while RSOM is slightly better than RVQ. The average of the best percent deviations are 0.19, 0.28 and 3.88 for RSOM, RVQ and SSOM, respectively. It is also important to note that RSOM provides a better solution than RVQ for some instances while there are also cases where RVQ is better than RSOM. With respect to the CPU time, although RSOM is, on average, faster than both RVQ and SSOM, there is not much significant difference among the methods.

In Table 4.7, we compare the above results with those obtained by the SOFMGR procedure of Hsieh and Tien [47]. Using MatLab Version 5.3 the authors implement SOFMGR on six instances. These are LM16, LM30, LM20, LM35/100, LJ100 and LIU150. Their experiments were run on a desktop computer equipped with a 600 MHz Pentium III processor (243 MFLOPS) and 640 Mbyte RAM. For LM16, LM30, LM20 and LM35/100 the average and best percent deviations as well as CPU times are over 20 runs where each run consists of 1000 epochs. For `LJ100` and `Liu150`, the number of epochs is 2000 while the best and average values are over 25 runs.

The values given in Table 4.7 suggest that for the rectilinear MFWP, RSOM can provide solutions of similar quality as does SOFMGR while reducing the computation effort significantly. On six benchmark instances we use, the reduction in CPU time is about 50-fold. This shows that RSOM does not require a refinement procedure when the winner updates are carried out using the rectilinear distance. Recall that the

Table 4.6. Comparison of methods for rectilinear instances.

Instance	m	**Percent Deviation**						**CPU Time (s)**		
		RVQ		RSOM		SSOM		RVQ	RSOM	SSOM
		Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Avg.	Avg.
LJ100	2	0.55	0.52	0.55	0.51	2.60	2.28	2.510	1.632	1.655
	3	0.42	0.35	0.40	0.34	2.25	2.17	2.518	2.101	2.133
	4	0.71	0.69	1.22	0.69	4.04	3.13	2.576	2.578	2.603
	5	2.30	0.78	0.87	0.75	3.69	2.68	2.620	3.044	3.066
LJ20/100	2	0.25	0.01	0.12	0.02	6.32	4.70	0.614	0.389	0.393
	3	0.08	0.03	0.08	0.02	17.59	9.11	0.625	0.484	0.487
	4	2.78	2.56	1.61	0.10	10.21	7.39	0.607	0.586	0.583
	5	2.56	0.09	0.26	0.02	7.92	6.93	0.608	0.679	0.678
LJ40/100	2	0.10	0.02	0.13	0.01	1.59	1.10	1.083	0.704	0.704
	3	0.08	0.02	0.08	0.03	1.38	0.96	1.094	0.888	0.900
	4	0.37	0.09	0.40	0.06	6.63	6.27	1.093	1.080	1.089
	5	2.10	0.65	1.11	0.68	7.34	6.13	1.106	1.265	1.277
LJ60/100	2	0.25	0.02	0.36	0.02	1.69	1.29	1.556	1.009	1.026
	3	0.05	0.02	0.07	0.03	2.66	2.34	1.557	1.297	1.301
	4	1.30	0.07	0.15	0.05	3.47	3.31	1.586	1.582	1.590
	5	3.52	0.07	0.22	0.09	5.15	4.90	1.618	1.855	1.870
LJ80/100	2	0.05	0.02	0.06	0.03	1.75	1.53	2.036	1.321	1.340
	3	0.76	0.06	0.22	0.06	2.28	1.84	2.072	1.698	1.716
	4	0.66	0.04	0.90	0.03	3.02	1.91	2.077	2.075	2.102
	5	1.40	0.08	0.17	0.05	1.85	1.71	2.120	2.455	2.467
LM16	3	1.05	0.05	0.21	0.05	10.43	8.56	0.507	0.411	0.421
LM20	3	1.27	0.06	0.66	0.04	9.52	9.32	0.615	0.483	0.498
LM30/100	2	0.04	0.01	0.06	0.02	4.87	4.65	0.843	0.542	0.557
LM35/100	2	0.09	0.01	0.26	0.03	3.27	3.20	0.970	0.625	0.626
Liu150	2	0.38	0.36	0.38	0.36	3.13	2.74	3.707	2.423	2.463
	3	0.48	0.46	0.50	0.47	4.00	3.86	3.756	3.138	3.173
	4	0.71	0.35	0.68	0.35	3.13	2.90	3.860	3.860	3.880
	5	0.94	0.47	0.51	0.47	1.83	1.78	3.898	4.549	4.584
Avg.		**0.90**	**0.28**	**0.44**	**0.19**	**4.77**	**3.88**	**1.780**	**1.598**	**1.614**

Table 4.7. Comparison of RSOM with SOFMGR for rectilinear instances.

Problem	m	**Percent Deviation**				**CPU Time (s)**	
		RSOM		SOFMGR		RSOM	SOFMGR
		Avg.	Best	Avg.	Best	Avg.	Avg.
LM16	3	0.21	0.05	0.00	0.00	0.411	13.80
LM20	3	0.66	0.04	0.00	0.00	0.483	12.75
LM30/100	2	0.06	0.02	0.00	0.00	0.542	19.61
LM35/100	2	0.26	0.03	0.00	0.00	0.625	23.25
LJ100	2	0.55	0.51	0.50	0.50	1.632	53.54
	3	0.40	0.34	0.32	0.32	2.101	64.67
	4	1.22	0.69	0.67	0.67	2.578	87.94
	5	0.87	0.75	0.72	0.71	3.044	140.13
Liu150	2	0.38	0.36	0.35	0.35	2.423	103.20
	3	0.50	0.47	0.45	0.45	3.138	78.25
	4	0.68	0.35	0.35	0.32	3.860	127.38
	5	0.51	0.47	0.44	0.43	4.549	161.87
Avg.		**0.52**	**0.34**	**0.32**	**0.31**	**1.598**	**73.87**

updates of Hsieh and Tien [47] rely on the squared Euclidean distance. In fact, the importance of the distance function becomes apparent when we compare the results of RSOM with those of SSOM given in Table 4.6.

4.2.2. Results with Euclidean instances

In Table 4.8 and Table 4.9 we present the results obtained for Euclidean instances. To the best of our knowledge, there is no published work solving the Euclidean MFWP by a SOM method. The only exception is the paper by Lozano et al. [67] In fact, they apply the SSOM method to solve a *squared* Euclidean distance MFWP instance with 49 customers corresponding to major towns in Spain and Portugal. We refer to this problem as Lozano49. The authors point out that SSOM can also be used to generate an approximate solution to the Euclidean MFWP and solve Lozano49 as a Euclidean

instance with this approach.

Our main objective in this subsection is to demonstrate whether RVQ and RSOM may improve the results obtained for the Euclidean MFWP by SSOM. To this end we make use of benchmark instances listed in Table 4.5. Table 4.8 includes results for relatively small instances where we report the average and best percent deviations from optimal objective values over 10 runs. In terms of the solution quality it is possible to conclude that RVQ and RSOM exhibit quite similar performances. The average of the best percent deviations are respectively 0.03 and 0.08 for RSOM and RVQ. The average of the average percent deviations are also very close for RVQ and RSOM: they are 0.70 and 0.72, respectively. We observe that SSOM is dominated by both RVQ and RSOM while the average CPU times of all methods are approximately the same. It is interesting to notice that even for very small instances such as `Cooper7A`–`7G` which consist of seven customers and two facilities, SSOM shows an inferior performance with respect to RVQ and RSOM.

Table 4.9 presents the best and average percent deviations obtained on two relatively large instances, `Eil50` and `Bon287`, over 20 runs. The optimal solutions for these problems are given in Brimberg et al. [13]. Results in this table reveal several important characteristics of the methods. First of all, similar to the results of Table 4.8, SSOM turns out to be the worst method in terms of solution quality while RSOM shows the best performance. For example, the best solution of SSOM has 10 percent deviation from optimal value when solving `Eil50` with $m = 15$ whereas RSOM yields a percent deviation of 0.13. Another conclusion we can draw is that for all methods the average and best percent deviation from the optimal solution become larger as the number of facilities to be located increases. This effect is more noticeable for `Bon287`, which is not unexpected since the number of local minima for the MFWP is an increasing function of *both* the number of customers *and* the number of facilities to be located. Thus the possibility of getting trapped in a local optimum goes up with higher values of n and m. This fact also explains the difference between the average and best objective values found by the methods. For smaller problems such as the ones of Table 4.7, the difference is relatively smaller. The computational effort as measured

Table 4.8. Comparison of methods for small Euclidean instances.

Problem	m	**Percent Deviation**						**CPU Time (s)**		
		RVQ		RSOM		SSOM		RVQ	RSOM	SSOM
		Avg	Best	Avg.	Best	Avg.	Best	Avg.	Avg.	Avg.
Cooper7A	2	1.02	0.02	1.62	0.02	5.61	3.00	0.292	0.231	0.228
Cooper7B	2	0.07	0.02	0.04	0.02	2.14	1.66	0.291	0.246	0.205
Cooper7C	2	0.19	0.03	0.20	0.03	1.30	0.51	0.286	0.219	0.247
Cooper7D	2	0.08	0.04	0.04	0.01	0.33	0.23	0.293	0.247	0.203
Cooper7E	2	1.42	1.38	1.04	0.04	2.82	2.26	0.292	0.219	0.240
Cooper7F	2	2.91	0.01	2.57	0.00	7.94	2.66	0.290	0.218	0.234
Cooper7G	2	2.57	0.02	2.55	0.02	3.92	3.43	0.288	0.257	0.216
Rosing15	2	0.21	0.01	0.03	0.01	2.55	2.26	0.526	0.384	0.357
	3	0.14	0.04	0.13	0.03	1.19	0.96	0.517	0.485	0.426
	4	0.10	0.03	1.32	0.04	2.92	1.53	0.559	0.584	0.513
	5	0.82	0.05	0.52	0.06	2.59	2.01	0.554	0.679	0.595
	6	1.63	0.02	2.04	0.04	3.51	3.39	0.571	0.773	0.683
Rosing20	2	0.25	0.02	0.24	0.01	0.95	0.80	0.664	0.475	0.431
	3	0.06	0.02	0.03	0.02	1.94	1.50	0.662	0.617	0.541
	4	0.78	0.03	1.33	0.03	5.83	5.07	0.705	0.744	0.656
	5	1.02	0.05	1.14	0.07	6.28	5.51	0.703	0.879	0.770
	6	1.60	0.05	1.32	0.16	8.60	5.45	0.724	1.005	0.881
Rosing25	2	0.07	0.01	0.03	0.01	0.62	0.49	0.794	0.580	0.518
	3	0.04	0.01	0.07	0.01	1.54	1.32	0.789	0.741	0.664
	4	0.54	0.00	0.13	0.00	3.65	2.89	0.854	0.914	0.794
	5	0.05	0.03	0.06	0.02	6.57	2.85	0.848	1.076	0.939
	6	0.80	0.04	0.95	0.05	7.29	4.92	0.877	1.242	1.078
Rosing30	2	0.19	0.01	0.12	0.02	0.58	0.54	0.919	0.670	0.613
	3	0.11	0.02	0.07	0.02	0.99	0.78	0.934	0.876	0.775
	4	0.29	0.03	0.31	0.02	3.42	2.74	0.999	0.936	1.077
	5	1.23	0.02	0.81	0.04	4.50	3.52	1.006	1.273	1.103
Avg.		**0.70**	**0.08**	**0.72**	**0.03**	**3.45**	**2.39**	**0.624**	**0.643**	**0.571**

Table 4.9. Comparison of methods for large Euclidean instances.

Problem	m	**Percent Deviation**						**CPU Time (s)**		
		RVQ		RSOM		SSOM		RVQ	RSOM	SSOM
		Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Avg.	Avg.
Eil50	2	0.19	0.00	0.02	0.00	1.73	0.40	1.817	1.546	1.277
	3	0.35	0.01	0.11	0.00	1.89	1.61	1.993	2.006	1.693
	4	0.05	0.01	0.06	0.02	1.99	1.90	2.002	2.538	2.112
	5	0.55	0.03	0.52	0.01	3.23	1.23	2.093	3.003	2.487
	6	0.08	0.02	0.20	0.03	3.92	3.74	2.190	3.565	2.931
	7	0.34	0.02	0.29	0.02	4.30	2.44	2.272	3.993	3.286
	8	0.97	0.05	0.95	0.04	4.10	3.11	2.361	4.621	3.734
	9	1.71	0.07	1.52	0.03	3.92	2.98	2.484	5.109	4.225
	10	4.11	0.11	1.30	0.05	5.36	2.56	2.561	5.600	4.564
	11	6.16	0.11	2.52	0.08	8.33	3.86	2.650	6.405	4.907
	12	6.51	3.10	2.77	0.09	11.51	5.46	2.898	6.574	5.343
	13	7.27	2.65	2.06	0.08	9.57	6.40	2.812	6.996	5.671
	14	10.60	2.94	3.77	0.12	18.94	13.11	2.914	7.599	6.174
	15	11.50	3.01	3.36	0.13	19.91	10.12	3.043	8.057	6.548
Bon287	2	0.29	0.01	0.23	0.02	5.74	5.41	7.115	5.161	4.709
	3	6.34	0.14	6.16	0.06	14.73	7.27	7.379	6.882	6.156
	4	7.52	0.13	7.52	4.25	12.16	10.89	7.684	8.639	7.664
	5	3.24	0.09	1.09	0.07	12.90	12.79	7.987	10.340	9.096
	6	4.56	0.16	4.17	0.12	21.11	20.94	8.304	12.156	10.609
	7	4.55	0.19	2.52	0.12	11.10	10.81	8.638	13.810	12.084
	8	8.38	0.99	4.88	0.64	14.74	10.39	8.923	15.622	13.617
	9	10.31	4.05	7.21	1.77	16.55	14.49	9.230	17.278	14.968
	10	7.97	2.93	7.25	1.86	18.26	14.81	9.587	19.107	16.536
	11	9.89	2.83	5.32	0.23	18.27	16.99	9.884	20.723	17.940
	12	9.67	4.78	8.33	4.00	22.49	19.04	10.177	22.619	19.506
	13	11.20	3.39	7.84	2.97	20.04	16.80	10.546	23.610	20.686
	14	11.44	4.57	8.93	3.50	23.65	19.12	10.564	25.386	21.898
	15	11.78	6.17	11.74	5.93	24.40	19.35	10.825	26.985	23.252

by CPU time is also increasing in n and m. In this respect, RVQ seems to be more time-efficient than RSOM and SSOM due to the fewer number of updates carried out in vector quantization. Recall that at each iteration both RSOM and SSOM update not only the coordinates of the winner facility, but also the coordinates of the facilities in the neighborhood of the winner. For a large number of facilities to be located the number of updates grows as well.

Table 4.10. Comparison of methods for Lozano49.

m	**Best Solution Found**			**% Dev. of SSOM**	**Avg. CPU Time (s)**		
	RVQ	RSOM	SSOM	**from RSOM**	RVQ	RSOM	SSOM
2	0.2702	0.2702	0.2718	0.58	1.378	1.001	0.898
3	0.2049	0.2049	0.2058	0.44	1.446	1.327	1.228
4	0.1707	0.1707	0.1745	2.23	1.467	1.656	1.435
5	0.1398	0.1399	0.1422	1.69	1.526	1.982	1.702
6	0.1148	0.1148	0.1172	2.09	1.585	2.310	1.970
7	0.1015	0.1015	0.1048	3.25	1.642	2.631	2.233
8	0.0945	0.0945	0.0981	3.83	1.700	2.960	2.506
9	0.0892	0.0877	0.0915	4.30	1.759	3.290	2.776
10	0.0836	0.0825	0.0856	3.77	1.819	3.608	3.040
12	0.0748	0.0728	0.0761	4.59	1.927	4.249	3.578
14	0.0663	0.0654	0.0687	5.14	2.045	4.897	4.108
16	0.0593	0.0579	0.0627	8.38	2.171	5.545	4.645
18	0.0519	0.0507	0.0549	8.41	2.276	6.185	5.174
20	0.0484	0.0451	0.0510	13.13	2.403	6.828	5.707

Finally, we report in Table 4.10 the performances of the methods on `Lozano49`. As the optimal objective values are not known, we compare the methods on the basis of the best solution they provide over 10 runs. The last three columns show the average CPU times. As was the case for `Eil50` and `Bon287`, there is an increase in CPU time as a function of m. Also, RSOM and SSOM require more computational effort than RVQ does. More interesting, however, are the percent deviation of SSOM with

respect to RSOM. The percent deviations calculated as $100 \times (z_{\mathrm{SSOM}} - z_{\mathrm{RSOM}}) / z_{\mathrm{RSOM}}$ are increasing in the number of facilities to be located. This means that when SSOM is used to generate an approximate solution to the Euclidean MFWP, this solution can be far away from the optimal one, particularly when m is large.

5. PROBABILISTIC MULTI FACILITY WEBER PROBLEM

The deterministic multi-facility Weber problem assumes that customer locations are known, which is not satisfied in most of the real world applications. A typical example is the determination of emergency station locations. Since we can never exactly know the location of emergency calls, a reasonable approach is to assume that they have random locations. We will refer to this probabilistic version of the problem, where customer locations are two-dimensional random vectors distributed according to some bivariate probability density function, as the probabilistic Multi-facility Weber Problem (PMFWP). This can be illustrated in Figure 5.1. In this example customer locations are not fixed and the only information available about their location is their probability distribution. where customers are distributed randomly.

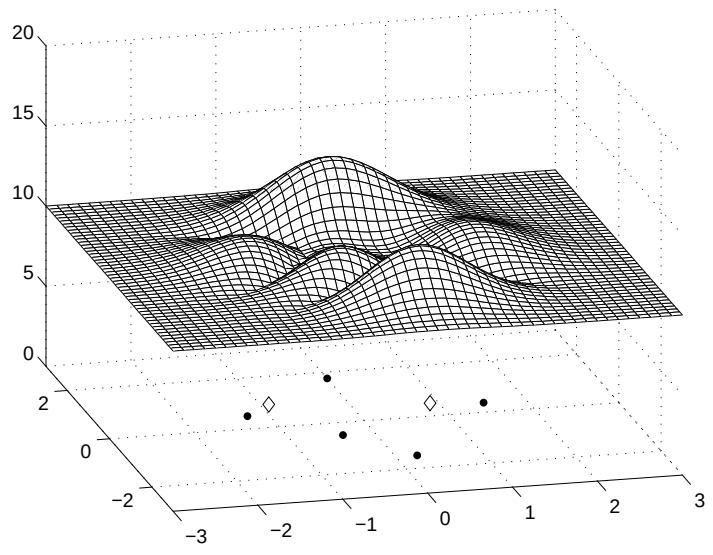


Figure 5.1. Illustration of probabilistic customer locations

Several attempts have been made to solve different classes of this problem. The earliest ones are due to Cooper [28], and Katz and Cooper [52]; they consider the single-facility probabilistic Weber problem, or simply the probabilistic Weber problem, with the Euclidean distance function. They assume that customer locations are in-

dependently distributed according to a bivariate normal distribution and propose an algorithm to minimize total expected transportation cost. Later on, Katz and Cooper [53] extend their previous results to also consider the cases where customer locations follow bivariate exponential or bivariate symmetric exponential distributions. The paper by Wesolowsky [97] is also on the probabilistic Weber problem, but involves the rectilinear distance function, and three different probability distributions for customer locations: bivariate normal, bivariate symmetric exponential and bivariate uniform. Although there are works on the multi-facility (pure) location problem with probabilistic customer locations [5, 82], none of them actually deals with the PMFWP. The only exception is due to Cooper [29] where he proposes a solution procedure for the probabilistic multi-facility location-allocation problem in which facilities have limited capacities using his [28], and Katz and Cooper's [52] earlier results. This method requires the complete enumeration of all extreme points of the transportation polytope and hence it can be used only for small problems.

In this chapter we consider PMFWP and propose first new local improvement methods using Cooper's alternate location-allocation (ALA) idea [25]. The remainder of the chapter we first give the definition of the probabilistic MFWP then explain the new ALA-type two-phase heuristic and its extensions. We then present several strategies for handling the probabilistic expression, namely the expected distance between facilities and customers locations. Solution techniques and experimental results are given in following chapters.

5.1. Modeling of stochastic decision variables

In modeling a real–world decision problem, Morris [75] suggests that three alterantives are available. The problem can be modeled as a decision under assumed certainty, a decision under risk or a decision under uncertainty. Most of the researches on location problems have concentrated on modeling the problem as a decision under assumed certainty. Thus deterministic approaches were taken, where either the total travel cost was minimized (minisum criterion), or the maximum travel cost was minimized (minimax criterion).

In a decision risk it is assumed that the probability distributions are known for all random variables. Also, a number of alternate principles of choice are possible. Sengupta and Portillo-Cambell [87] suggest four possible optimization criteria under conditions of risk:

- expected–value criterion
- portfolio criterion
- aspiration–level criterion
- fractile criterion

The expected-value criterion involves the determination of the location of the new facilities such that an appropriately defined expected-cost function is minimized. The portfolio criterion seeks the location which minimizes the variance of cost, subject to a constraint on the expected cost produced. An aspiration–level criterion is used when the facilities are to be located such that the probability of cost being less than some specified value is maximized. The fractile criterion involves the location of the new facilities so that one minimizes the value on the cumulative distribution function of total cost which represents the acceptable probability of cost exceeding that value. Mathematically, the fractile criterion can be stated as:

$$\text{minimize} \qquad z \tag{5.1}$$

$$\text{subject to} \qquad P\left(f(x) \leq z\right) \geq \alpha \tag{5.2}$$

where $f(x)$ is the total cost resulting from $\mathbf{x}$, the vector of coordinate locations on the new facilities, and z is the cost below which total cost occurs with at least a probability of α.

Among those criteria, we will be concentrated on the expected–value criterion in our problem definitions in the sequel of this work and we search for the facility coordinates that minimizes total random travel cost. For other stochastic models proposed for the probabilistic customer locations we may refer to Seppala et al. [88] and Charnes and Cooper [20] and Drezner et al. [30].

5.2. Mathematical formulation

Now, we would like to consider the PMFWP. In the new formulation, customer coordinates $\mathbf{a}_j = (a_{j1}, a_{j2})^T$ are no longer given values; they are independent random variables with known probability distributions. Then, the PMFWP we wish to solve consists of finding the facility locations that minimize the expected cost

$$\min \; E[z] = \sum_{i=1}^{m} \sum_{j=1}^{n} y_{ij} h_j E[d(\mathbf{x}_i, \mathbf{a}_j)], \tag{5.3}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} y_{ij} = 1 \quad j = 1, ..., n \tag{5.4}$$

$$y_{ij} \in \{0, 1\} \qquad i = 1, ..., m; j = 1, ..., n. \tag{5.5}$$

Here, n is the number of customers and m is the number of facilities to be located. h_j and $\mathbf{a}_j = (a_{j1}, a_{j2})^T$ represent respectively, the demand and coordinates of customer j. $\mathbf{x}_i = (x_{i1}, x_{i2})^T$ denotes the (unknown) coordinates of facility i to be located, and $d(\mathbf{x}_i, \mathbf{a}_j)$ is a function which measures the distance between facility i and customer j. Notice that if demand h_j is random, new and existing methods explained in the sequel can be applied without further modification by after replacing h_j with its expected value $E[h_j]$ as long as demand and customer locations are independent random variables.

Like MFWP, the PMFWP formulation (5.3)-(5.5) with the Euclidean distance function can also be seen as the enumeration of Voronoi partitions of the customer set, which is shown to be NP-Hard [72].

5.3. Alternating Location-Allocation Heuristics

Since y_{ij}'s are either zero or one at an optimal solution of the PMFWP, it is possible to solve it by enumerating all feasible zero-one assignments of y_{ij} variables and calculating associated objective function values. Once an assignment of customers

to facilities is given, the problem reduces to the solution of m probabilistic single-facility location, or simply probabilistic Weber problems, consisting of the minimization of

$$E\left[z_i\right] = \sum_{j=1}^{n_i} h_j E\left[d\left(\mathbf{x}_i, \mathbf{a}_j\right)\right], \tag{5.6}$$

each for one of the facilities $i = 1, \ldots, m$, with n_i denoting the number of customers served by facility i. In other words n_i is the size of the set $\{j : y_{ij} = 1; j = 1, \ldots, n\}$. Clearly, $\sum_{i=1}^{m} n_i = n$. Notice that the expected cost $E\left[z_i\right]$ is a function of the coordinates $\mathbf{x}_i = (x_{i1}, x_{i2})^T$ of facility i. In short, when a zero-one assignment of y_{ij} variables is given, the problem reduces to optimally locating single facilities with respect to n_i customers with randomly distributed locations and fixed demands. Consequently, it is possible to tailor Cooper's ALA heuristic [25] to produce a reasonable solution for the PMFWP. We give a formal outline of the probabilistic ALA (PALA) below.

Algorithm 1. *PALA heuristic*

1. *Locate the facilities at arbitrarily selected points* $\mathbf{x}_i = (x_{i1}, x_{i2})^T$, $i = 1, \ldots, m$.
2. *Set* $y_{ij} = 0$ *for* $i = 1, \ldots, m$ *and* $j = 1, \ldots, n$.
3. *Calculate the expected distances* $\gamma_{ij} = E[d(\mathbf{x}_i, \mathbf{a}_j)]$ *between customers and facilities.*
4. *For each customer* j *determine the nearest facility* $i^* = \arg\min_{i=1,\ldots,m} \{\gamma_{ij}\}$, *and set* $y_{i^*j} = 1$, *and* $y_{ij} = 0$ *for* $i \neq i^*$.
5. *Using allocations* y_{ij} *determine customer subsets for each facility* i.
6. *Solve* m *probabilistic Weber problems (5.6) to relocate* m *facilities.*
7. *Repeat steps (2) – (6) until either facility locations* $\mathbf{x}_i = (x_{i1}, x_{i2})^T$ *or allocations* y_{ij} *for* $i = 1, \ldots, m$ *and* $j = 1, \ldots, n$ *remain unchanged.*

As it can be observed, the determination of the expected distance

$$E[d\left(\mathbf{x}_i, \mathbf{a}_j\right)] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} d\left(\mathbf{x}_i, \mathbf{a}_j\right) f_j(\mathbf{a}_j) da_{j1} da_{j2} \tag{5.7}$$

as a function of unknown facility coordinates is crucial for the solution of the PMFWP, which may not be possible because of a couple of reasons. First of all the bivariate probability density function $f_j(\mathbf{a}_j)$ of customer locations may be unavailable. Second, even if such information is available it can still be impossible to evaluate the double integral analytically or numerically for the assumed form of the distance function $d(\mathbf{x}_i, \mathbf{a}_j)$. Besides, even if it is possible to evaluate the double integral and obtain a closed form expression for $E[d(\mathbf{x}_i, \mathbf{a}_j)]$, it may still be extremely difficult to solve PMFWP because of the nonconvexity of the objective function.

5.4. Specific cases with an analytical expression for expected distance

In their work on the probabilistic Euclidean multi-facility (pure) location problem, assuming that customer coordinates are independent and follow symmetric bivariate normal distributions, Aly and White [5] develop a closed form expression for the expected distance $E[d(\mathbf{x}_i, \mathbf{a}_j)]$. In other words coordinates, when a_{j_1} and a_{j_2} are independent normal random variables with mean μ_{j_1} and μ_{j_2}, and standard deviations $\sigma_{j_1} = \sigma_{j_2} = \sigma_j$ for $j = 1, \ldots, n$, the expected distance (5.7) can explicitly be written as

$$E[d(\mathbf{x}_i, \mathbf{a}_j)] = \sqrt{\frac{\pi}{2}}\sigma_j \mathrm{H}\left(-\frac{1}{2}, 1, -\frac{\lambda_{ij}^2}{2\sigma_j^2}\right). \tag{5.8}$$

Here, $\lambda_{ij} = \sqrt{(x_{i1}-\mu_{j_1})^2 + (x_{i2}-\mu_{j_2})^2}$ is the Euclidean distance between facility i and the mean location of customer j, and $H(\cdot,\cdot,\cdot)$ is the confluent hypergeometric function

$$H(a, b, t) = \sum_{k=0}^{\infty} \frac{\Gamma(a+k)}{\Gamma(a)} \frac{\Gamma(b)}{\Gamma(b+k)} \frac{t^k}{k!} \tag{5.9}$$

where $\Gamma(\cdot)$ is the gamma function. Then the objective function (5.4) of the PMFWP can be expressed as

$$E[z] = \sum_{i=1}^{m}\sum_{j=1}^{n} y_{ij} h_j \sqrt{\frac{\pi}{2}}\sigma_j \mathrm{H}\left(-\frac{1}{2}, 1, -\frac{\lambda_{ij}^2}{2\sigma_j^2}\right), \tag{5.10}$$

and the PMWFP becomes the minimization of (5.10) subject to (5.4) and (5.5), for which the only known exact solution procedure is the brute-force approach based on the evaluation of $E[z]$ for every feasible zero-one assignment of y_{ij} variables. This is how we solve small instances to optimality in Section 5. However, it is also possible to approximately solve large instances by using PALA since the probabilistic Weber problem stated as the minimization of

$$E[z_i] = \sqrt{\frac{\pi}{2}} \sum_{j=1}^{n_i} h_j \sigma_j \mathrm{H}\left(-\frac{1}{2}, 1, -\frac{\lambda_{ij}^2}{2\sigma_j^2}\right), \tag{5.11}$$

can be solved for each facility i.

We remark that the objective function of the probabilistic Weber problem has always a positive lower bound. This is the smallest value

$$\sqrt{\frac{\pi}{2}} h_j \sigma_j \tag{5.12}$$

one can obtain from (5.11) for $m = n_i = 1$. Here, h_j and σ_j are respectively the demand and the variance of the location of customer j. Then, the location of facility i converges exactly to the mean location of customer j. This implies that $\lambda_{ij} = 0$ and $H\left(-\frac{1}{2}, 1, 0\right) = 1$. Clearly, this lower bound becomes

$$\sqrt{\frac{\pi}{2}} \sum_{j=1}^{n} h_j \sigma_j \tag{5.13}$$

for the objective function (5.3) of the PMFWP. As can be observed this lower bound is zero in the deterministic case which is obtained when $m = n$ and each facility is located exactly on the customer it serves.

It is also possible to show that the objective function $E[z_i]$ is convex in the facility

coordinates and has partial derivatives

$$\frac{\partial E\left[z_i\right]}{\partial x_{ik}} = \frac{1}{2}\sqrt{\frac{\pi}{2}}\sum_{j=1}^{n_i} h_j \left(\frac{x_{ik}-\mu_{a_{jk}}}{\sigma_j}\right) \mathrm{H}\left(\frac{1}{2}, 2, -\frac{\lambda_{ij}^2}{2\sigma_j^2}\right) \qquad k = 1, 2 \tag{5.14}$$

by using again Aly and White's results [5]. Hence the location phase can be accomplished by using the steepest descent method whose update equations are

$$x_{ik}(t+1) = x_{ik}(t) - \alpha(t)\frac{\partial E\left[z_i\right]}{\partial x_{ik}} \qquad k = 1, 2 \tag{5.15}$$

with $\alpha(t)$ denoting the step length at step t for facilities $i = 1, \ldots, m$. Yet another approach is to repeat

$$x_{ik}^{(t+1)} = \frac{\sum\limits_{j=1}^{n_i} \frac{h_j}{\sigma_j}\mu_{j_k}\mathrm{H}\left(\frac{1}{2}, 2, -\frac{\left(\lambda_{ij}^{(t)}\right)^2}{2\sigma_j^2}\right)}{\sum\limits_{j=1}^{n_i} \frac{h_j}{\sigma_j}\mathrm{H}\left(\frac{1}{2}, 2, -\frac{\left(\lambda_{ij}^{(t)}\right)^2}{2\sigma_j^2}\right)} \qquad k = 1, 2 \tag{5.16}$$

obtained by combining the generalization of Weiszfeld's procedure [95] suggested by Cooper [28] and Katz and Cooper [52], [53] with Aly and White's results [5].

Recall that in the allocation phase of PALA we have to assign customers to facilities with the smallest expected distance. In other words, facility i^* serves customer j if

$$i^* = \arg\min_{i=1,\ldots,m} \{E[d\left(\mathbf{x}_i, \mathbf{a}_j\right)]\}. \tag{5.17}$$

This is equivalent to setting

$$i^* = \arg\min_{i=1,\ldots,m} \{\lambda_{ij}^2\}, \tag{5.18}$$

for this particular situation since $E[d\left(\mathbf{x}_i, \mathbf{a}_j\right)]$ is an increasing function of λ_{ij}, which can be demonstrated by using the definition of the confluent hypergeometric function

$\mathrm{H}\left(\frac{1}{2}, 2, -\frac{\lambda_{ij}^2}{\sigma_j^2}\right)$ [1]. The new allocation is then determined by repeating (5.18) for all customers, which completes the allocation phase.

An interesting result occurs when the squared Euclidean metric is used to measure the distances between facility locations and customers. Under the same probabilistic assumptions, namely when $a_{j1} \sim N(\mu_{j_1}, \sigma_j)$, and $a_{j2} \sim N(\mu_{j_2}, \sigma_j)$ it has been shown that $E[d(\mathbf{x}_i, \mathbf{a}_j)] = 2\sigma_j^2 + \lambda_{ij}^2$ for facilities $j = 1, ..., m$ [5]. Then the objective function (5.11) of the probabilistic Weber problem becomes

$$E[z_i] = \sum_{j=1}^{n_i} h_j(2\sigma_j^2 + \lambda_{ij}^2) \tag{5.19}$$

for facility i. $E[z_i]$ is a strictly convex function of $\mathbf{x} = (x_{i1}, x_{i2})^T$, whose partial derivatives with respect to x_{i1} and x_{i2} are

$$\frac{\partial E[z_i]}{\partial x_{ik}} = 2\sum_{j=1}^{n_i} h_j\left(x_{ik} - \mu_{j_k}\right) \qquad k = 1, 2. \tag{5.20}$$

and the unique optimal location occurs at

$$x_{ik}^* = \frac{\sum_{j=1}^{n_i} h_j \mu_{j_k}}{\sum_{j=1}^{n_i} h_j} \qquad k = 1, 2. \tag{5.21}$$

In short, for the squared Euclidean distance the solution of the probabilistic Weber problem is equivalent to its deterministic version where customer coordinates are replaced with their means. However, this is not true for the Euclidean distance probabilistic Weber problem [28].

Finally, we should mention that although most of the existing works in location theory dealing with probabilistic customer locations consider the Euclidean distance and bivariate symmetric normal distribution. There are also some exceptions, but unfortunately they are very few. Wesolowsky [97] considers the rectilinear distance instead of the Euclidean. There exist results as well where customer locations follow bivariate exponential and shifted bivariate exponential [53], and bivariate uniform [97]

distributions. We can also use them to exactly solve PMWFP by brute-force approach (for only small instances), or approximately by PALA.

5.5. Approximating expected distance using the mean, variance and covariance

There may exist situations where explicit bivariate probability distributions for customer coordinates are known. However, it may still be impossible to derive a closed form expression for the expectation $E[d(\mathbf{x}_i, \mathbf{a}_j)]$. The following approximation provides a solution for this problem. It is a generalization of the idea used to approximate the expected value of the square root of a single random variable by Tan et al. [91].

5.5.1. Approximating the expected value of a function of random variables

Here we provide a general approximation formula for twice differentiable functions of p dimensional random vectors. Note that $p = 2$ for the PMWFP in particular since it is defined in $\mathbb{R}^2$.

Let $\phi(\mathbf{x}, \mathbf{a})$ be a two-vector function where $\mathbf{a}$ is random in $\mathbb{R}^p$ with known mean vector $\boldsymbol{\mu} = \left(\mu_1, \mu_2, \ldots, \mu_p\right)^T$, variance vector $\sigma^2 = \left(\sigma_1^2, \sigma_2^2, \ldots, \sigma_p^2\right)^T$, and/or covariance matrix $\boldsymbol{\rho} = [\rho_{ij} : \quad i, j = 1, \ldots, n]$. The second order approximation of $\phi(\mathbf{x}, \mathbf{a})$ with respect to $\mathbf{a}$ around $\boldsymbol{\mu}$ is

$$\phi(\mathbf{x}, \mathbf{a}) \approx \phi(\mathbf{x}, \boldsymbol{\mu}) + \nabla_a \phi(\mathbf{x}, \boldsymbol{\mu}) (\mathbf{a} - \boldsymbol{\mu}) + \frac{1}{2} (\mathbf{a} - \boldsymbol{\mu})^T \nabla_a^2 \phi(\mathbf{x}, \boldsymbol{\mu}) (\mathbf{a} - \boldsymbol{\mu}) . \tag{5.22}$$

Here $\nabla_a \phi(\mathbf{x}, \boldsymbol{\mu})$ and $\nabla_a^2 \phi(\mathbf{x}, \boldsymbol{\mu})$ denote respectively the gradient vector and Hessian matrix of $\phi(\mathbf{x}, \mathbf{a})$ both with respect to $\mathbf{a}$ and evaluated at the mean vector $\boldsymbol{\mu}$. Note that (5.22) can equivalently be expressed as

$$\phi(\mathbf{x}, \mathbf{a}) \approx \phi(\mathbf{x}, \boldsymbol{\mu}) + \sum_{j=1}^{p} \frac{\partial \phi(\mathbf{x}, \boldsymbol{\mu})}{\partial a_j} (a_j - \boldsymbol{\mu}_j) + \frac{1}{2} \sum_{i=1}^{p} \sum_{j=1}^{p} \frac{\partial^2 \phi(\mathbf{x}, \boldsymbol{\mu})}{\partial a_i \partial a_j} (a_i - \mu_i) (a_j - \mu_j) , \tag{5.23}$$

and becomes

$$E\left[\phi(\mathbf{x},\mathbf{a})\right] \approx \phi(\mathbf{x},\boldsymbol{\mu}) + \sum_{j=1}^{p} \frac{\partial \phi(\mathbf{x},\boldsymbol{\mu})}{\partial a_j}\left(E\left[\mathbf{a}_j\right] - \boldsymbol{\mu}_j\right) + \frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{p} \frac{\partial^2 \phi(\mathbf{x},\boldsymbol{\mu})}{\partial a_i \partial a_j} E\left[\left(a_i - \mu_i\right)\left(a_j - \mu_j\right)\right] \tag{5.24}$$

after taking the expectation of both sides of (5.23) with respect to a. As can be observed, $\phi(\mathbf{x},\boldsymbol{\mu})$ is a constant and the second term is equal to zero since $E\left[a_j\right] = \mu_j$. In the last term, $E\left[\left(a_i - \mu_i\right)\left(a_j - \mu_j\right)\right] = \sigma_i^2$ when $i = j$ and $E\left[\left(a_i - \mu_i\right)\left(a_j - \mu_j\right)\right] = \rho_{ij}$ otherwise. As a result (5.24) becomes

$$E\left[\phi(\mathbf{x},\mathbf{a})\right] \approx \phi(\mathbf{x},\boldsymbol{\mu}) + \frac{1}{2}\sum_{\imath=1}^{p} \sigma_i^2 \frac{\partial^2 \phi(\mathbf{x},\boldsymbol{\mu})}{\partial a_i^2} + \frac{1}{2}\sum_{i=1}^{p}\sum_{\substack{j=1 \\ i\neq j}}^{p} \rho_{ij} \frac{\partial^2 \phi(\mathbf{x},\boldsymbol{\mu})}{\partial a_i \partial a_j}, \tag{5.25}$$

which reduces to

$$E\left[\phi(\mathbf{x},\mathbf{a})\right] \approx \phi(\mathbf{x},\boldsymbol{\mu}) + \frac{1}{2}\sum_{i=1}^{p} \sigma_i^2 \frac{\partial^2 \phi(\mathbf{x},\boldsymbol{\mu})}{\partial a_i^2} \tag{5.26}$$

when random variables a_j $j = 1, \ldots, n$ are independent.

5.5.2. Approximating expected distance

We can use (5.25) or (5.26) to approximate $E[d\left(\mathbf{x}_i,\mathbf{a}_j\right)]$ by replacing $\phi(\mathbf{x},\mathbf{a})$ with $d\left(\mathbf{x}_i,\mathbf{a}_j\right)$ provided that the mean vector $\boldsymbol{\mu}_j = \left(\mu_{j1},\mu_{j2}\right)^T$, the variance vector $\sigma_j^2 = \left(\sigma_{j1}^2,\sigma_{j2}^2\right)$, and/or the 2×2 covariance matrix $\rho_j = \left[\rho_{j1j2} : \quad j_1, j_2 = 1,2\right]$, and the 2×2 Hessian matrix of $d\left(\mathbf{x}_i,\mathbf{a}_j\right)$ with respect to customer coordinates $\mathbf{a}_j = \left(a_{j1},a_{j2}\right)^T$ can be derived. Under the assumption that coordinates are independent, symmetric random variables, namely $\rho_{j1j2} = 0$ for $j_1 \neq, j_2$, and $\sigma_{j1}^2 = \sigma_{j2}^2 = \sigma_j^2$ for customer j, expression (5.26) becomes

$$E\left[d(\mathbf{x}_i,\mathbf{a}_j)\right] \approx d(\mathbf{x}_i,\boldsymbol{\mu}_j) + \frac{1}{2}\frac{\sigma_j^2}{d(\mathbf{x}_i,\boldsymbol{\mu}_j)} \tag{5.27}$$

for $d(\mathbf{x}_i, \mathbf{a}_j)$ is the Euclidean distance between facility i and customer j. Hence,

$$E[z_i] \approx \sum_{j=1}^{n_i} h_j \left\{ d(\mathbf{x}_i, \boldsymbol{\mu}_j) + \frac{1}{2} \frac{\sigma_j^2}{d(\mathbf{x}_i, \boldsymbol{\mu}_j)} \right\} \tag{5.28}$$

can be used to approximate (5.11). Notice that this approximation is undefined for $\mathbf{x}_i = \boldsymbol{\mu}_j$ for some j. In other words, the use of approximation (5.27) in the solution of the probabilistic Weber problems gives very poor results in the neighborhood of μ_j decreasing the accuracy of the heuristics for instances whose optimal solution has at least one facility located at one of the mean customer coordinates.

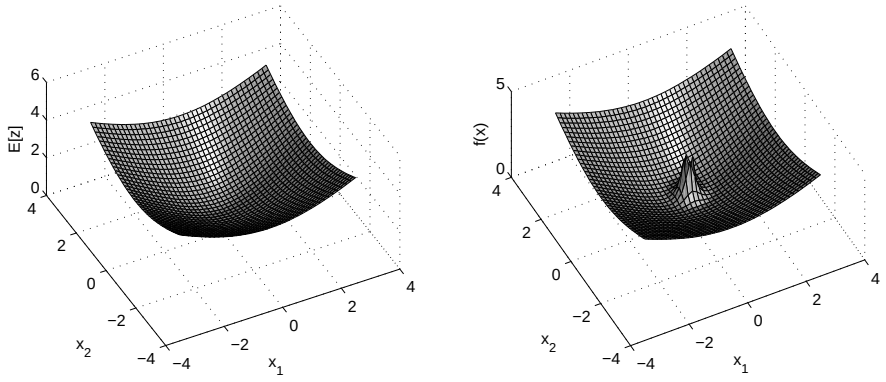


Figure 5.2. Illustrations for exact and approximate expectations (29) and (28)

In order to develop a reasonable strategy to deal with this problem, let us drop facility index i and customer index j in (5.27) to simplify the notation and analyze the behavior of function

$$f(\mathbf{x}) = d(\mathbf{x}, \boldsymbol{\mu}) + \frac{1}{2} \frac{\sigma^2}{d(\mathbf{x}, \boldsymbol{\mu})} \tag{5.29}$$

in the two-dimensional real space $\mathbb{R}^2$ in particular, which can be generalized for higher dimensions easily. Recall that $d(\mathbf{x}, \boldsymbol{\mu}) = [(x_1 - \mu_1)^2 + (x_2 - \mu_2)^2]^{1/2}$. It is not a convex function and has an interesting shape. It is illustrated at Figure 5.2 for $\mu = \mathbf{0}$. First order partial derivatives of $f(\mathbf{x})$ are

$$\frac{\partial f(\mathbf{x})}{\partial x_k} = \frac{x_k - \mu_k}{d(\mathbf{x}, \boldsymbol{\mu})} \left(1 - \frac{1}{2} \left[\frac{\sigma}{d(\mathbf{x}, \boldsymbol{\mu})} \right]^2 \right) \text{ for } k = 1, 2$$

and they become zero for one of the following four cases:

i) $\mathbf{x} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}$,

ii) $x_1 = \mu_1$ and $1 - \frac{1}{2}\left[\frac{\sigma}{d(\mathbf{x},\boldsymbol{\mu})}\right]^2 = 0$, implying $x_2 = \mu_2 \pm \frac{\sigma}{\sqrt{2}}$

iii) $x_2 = \mu_2$ and $1 - \frac{1}{2}\left[\frac{\sigma}{d(\mathbf{x},\boldsymbol{\mu})}\right]^2 = 0$, implying $x_1 = \mu_1 \pm \frac{\sigma}{\sqrt{2}}$

iv) $\mathbf{x} \in \mathbb{R}^2$ such that $d(\mathbf{x}, \boldsymbol{\mu}) = \frac{\sigma}{\sqrt{2}}$.

Since $f(\mathbf{x})$ becomes arbitrarily large as $\mathbf{x}$ becomes arbitrarily close to $\boldsymbol{\mu}$, the first stationary point cannot be a local minimum of $f(\mathbf{x})$. Cases (ii) and (iii) are included in case (iv) and it gives the set of local minima as $\left\{\mathbf{x} \in \mathbb{R}^2 : d(\mathbf{x}, \boldsymbol{\mu}) = \frac{\sigma}{\sqrt{2}}\right\}$. They are infinitely many with minimum objective value $\sqrt{2}\sigma$. This can also be observed from Figure 5.3. We therefore propose to approximate $f(\mathbf{x})$ as

$$g(\mathbf{x}) = \begin{cases} d(\mathbf{x}, \boldsymbol{\mu}) + \frac{1}{2}\frac{\sigma^2}{d(\mathbf{x},\boldsymbol{\mu})} & \text{if } d(\mathbf{x}, \boldsymbol{\mu}) > \frac{\sigma}{\sqrt{2}} \\ \\ \sqrt{2}\sigma & \text{if } d(\mathbf{x}, \boldsymbol{\mu}) \leq \frac{\sigma}{\sqrt{2}} \end{cases} . \tag{5.30}$$

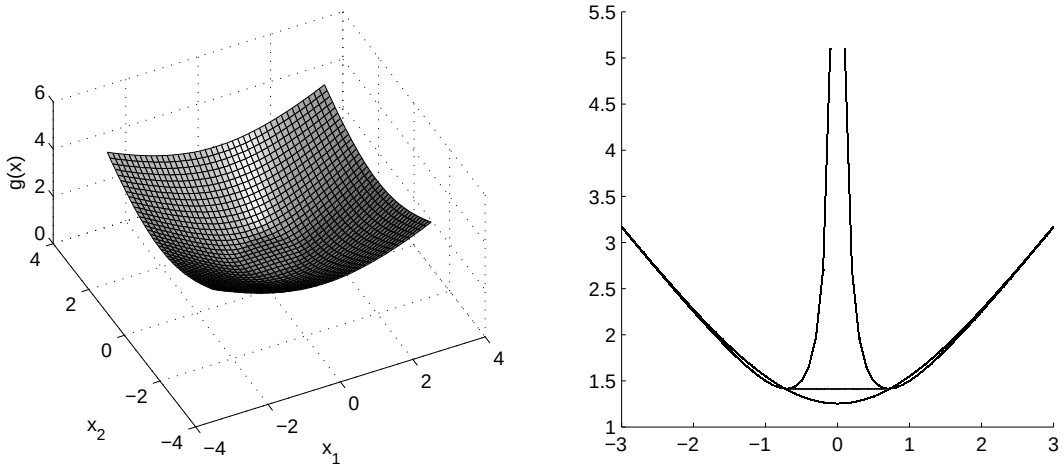


Figure 5.3. Illustration of (5.30)

It can be shown that $g(\mathbf{x})$ is convex whenever $d(\cdot)$ is a non–decreasing convex function. For this aim, we use the following propositions.

Lemma 5.1. *Let $U : \mathbb{R}^n \longrightarrow \mathbb{R}$ be a convex function and $G : \mathbb{R} \longrightarrow \mathbb{R}$ be a non–decreasing and convex function. Then $F(x) = G(U(x))$ is a convex function for all $x \in \mathbb{R}^n$.*

Proof. Let a, b be two points of $\mathbb{R}^n$. By definition of F, we have

$$F(\lambda a + (1-\lambda)b) = G(U(\lambda a + (1-\lambda)b))$$

Now, since U is convex, we have

$$U(\lambda a + (1-\lambda)b) \leq \lambda U(a) + (1-\lambda)U(b)$$

Further, since G is non–decreasing, for $v \leq w$, we have $G(v) \leq G(w)$. Therefore,

$$G(U(\lambda a + (1-\lambda)b)) \leq G(\lambda U(a) + (1-\lambda)U(b))$$

But now, by convexity of G,

$$\begin{aligned} G(U(\lambda a + (1-\lambda)b)) &\leq \lambda G(U(a)) + (1-\lambda)G(U(b)) && (5.31) \\ &\leq \lambda F(a) + (1-\lambda)F(b) && (5.32) \end{aligned}$$

we have the convexity of F. □

Lemma 5.2. *Let $h : \mathbb{R} \longrightarrow \mathbb{R}$ such that $h(u) = u + \frac{a}{u}$. Then $h(u)$ is a non–decreasing convex function for $a \geq 0$ and $u > \sqrt{a}$.*

Proof. It is sufficient to show that $h^{'}$ and $h^{''}$ are both non–negative.

$$\begin{aligned} h\,(u) &= \frac{u^2 + a}{u} && (5.33) \\ h^{'}\,(u) &= \frac{2u^2 - u^2 - a}{u^2} = \frac{u^2 - a}{u^2} && (5.34) \end{aligned}$$

Since $u > \sqrt{a}$ we have $h^{'}$ non–negative within this interval. Therefore h is a non–decreasing function. Further,

$$h^{''}(u) = \frac{(2u)\,u^2 - 2u\,(u^2 - a)}{u^4} = \frac{2au}{u^4} = \frac{2a}{u^3}$$

Since $a \geq 0$ and $u \geq \sqrt{a} \geq 0$, $h^{''}(u) \geq 0$ which implies that h is a non–decreasing convex function. □

Lemma 5.3. *If $d(\boldsymbol{x}, \boldsymbol{\mu})$ is a convex non–decreasing distance function then $g(x)$ in (5.30) is convex.*

Proof. When $d\,(\mathbf{x}, \boldsymbol{\mu}) \leq \frac{\sigma}{\sqrt{2}}$, $g(\mathbf{x})$ is a constant function, then it is sufficient to consider the case $d\,(\mathbf{x}, \boldsymbol{\mu}) > \frac{\sigma}{\sqrt{2}}$ for showing the convexity of $g(\mathbf{x})$. Now, using Lemma (5.1) and Lemma (5.2) and by setting U in Lemma (5.1) equal to $d\,(\mathbf{x}, \boldsymbol{\mu})$, G in Lemma (5.1) to h in Lemma (5.2) and letting a in Lemma (5.2) be equal to $\frac{\sigma^2}{2}$ we obtain the convexity of $g(\boldsymbol{x})$. □

We illustrate $g(\mathbf{x})$ and the exact expectation for bivariate normal distribution with Euclidean distance

$$\sqrt{\frac{\pi}{2}}\sigma \mathrm{H}\left(-\frac{1}{2}, 1, -\frac{1}{2}\left[\frac{d(\mathbf{x}, \boldsymbol{\mu})}{\sigma_j}\right]^2\right) \tag{5.35}$$

for $\boldsymbol{\mu} = 0$ in Figure 5.4. The exact expectation is a convex function [5] and reaches its minimum value$\sqrt{\frac{\pi}{2}}\sigma$ for $\mathbf{x} = \boldsymbol{\mu}$. Using this information we may compute an upper and lower bound to the approximation error for bivariate normal distribution with Euclidean distance case when $d(\boldsymbol{x}, \boldsymbol{\mu}) < \sigma/\sqrt{2}$. The approximation error (AE) can be defined as the difference between the exact and approximated expectations, i.e.

$$AE(\boldsymbol{x}) = \sqrt{\frac{\pi}{2}}\sigma \mathrm{H}\left(-\frac{1}{2}, 1, -\frac{1}{2}\left[\frac{d(\mathbf{x}, \boldsymbol{\mu})}{\sigma}\right]^2\right) - \left(d(\mathbf{x}, \boldsymbol{\mu}) + \frac{\sigma}{2d(\mathbf{x}, \boldsymbol{\mu})}\right). \tag{5.36}$$

Then, we can restrict the approximation error between two bounded values as

$$\begin{array}{lcccl} \sqrt{\frac{\pi}{2}}\sigma - \sigma\sqrt{2} & \leq & AE & \leq & \sqrt{\frac{\pi}{2}}\sigma \mathrm{H}\left(-\frac{1}{2}, 1, -\frac{1}{4}\right) - \sigma\sqrt{2} \\ \sigma\sqrt{2}\left(\frac{\sqrt{\pi}}{2} - 1\right) & \leq & & \leq & \sigma\sqrt{2}\left[\frac{\sqrt{\pi}}{2}\mathrm{H}\left(-\frac{1}{2}, 1, -\frac{1}{4}\right) - 1\right] \\ -0.1609\sigma & \leq & & \leq & -0.0089\sigma \end{array} \tag{5.37}$$

Hence, for $d(\boldsymbol{x}, \boldsymbol{\mu}) < \sigma/\sqrt{2}$ the approximation error lies within the interval

$$\left\{\sqrt{2}\sigma\left(\frac{\sqrt{\pi}}{2} - 1\right), \sqrt{2}\sigma\left[\frac{\sqrt{\pi}}{2}\mathrm{H}\left(-\frac{1}{2}, 1, -\frac{1}{4}\right) - 1\right]\right\}, .$$

However for $d(\boldsymbol{x}, \boldsymbol{\mu}) \geq \sigma/\sqrt{2}$, finding the bounds is much more difficult. In this case

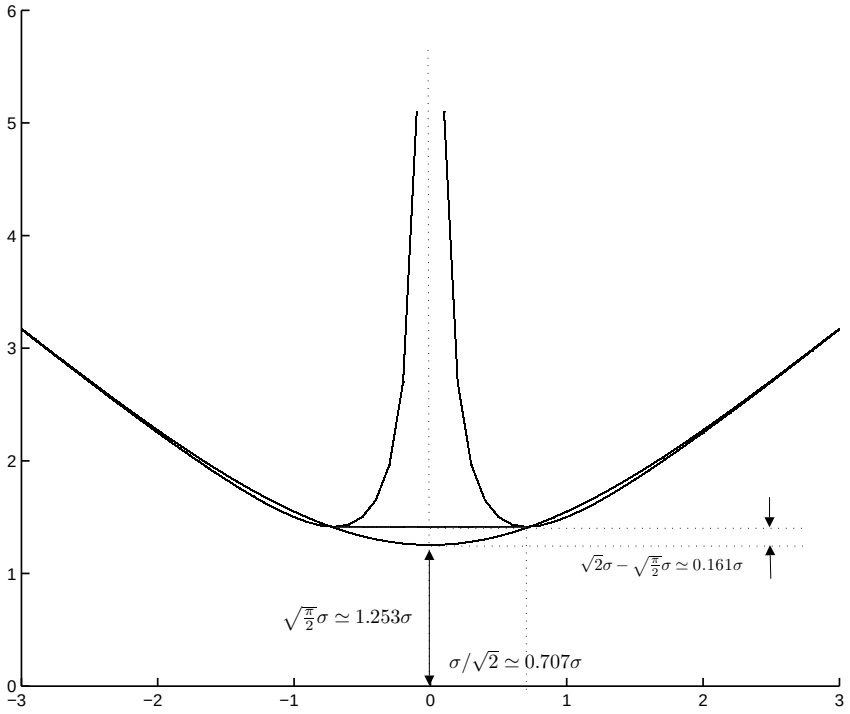


Figure 5.4. Error committed when using approximation for bivariate normal distribution and Euclidean distance case

when we take the first derivative of AE with respect to x in order to find the extremum

values of AE, we obtain

$$\frac{\partial AE}{\partial \boldsymbol{x}} = (x_k - \mu_k)\left[\frac{\sqrt{\pi}}{2\sqrt{2}\sigma}\mathrm{H}\left(\frac{1}{2}, 2, -\frac{1}{2}\left(\frac{d(\boldsymbol{x},\boldsymbol{\mu})}{\sigma}\right)^2\right) - \frac{1}{d(\boldsymbol{x},\boldsymbol{\mu})}\left(1 - \frac{1}{2}\left(\frac{\sigma}{d(\boldsymbol{x},\boldsymbol{\mu})}\right)^2\right)\right].$$

However, setting $\frac{\partial AE(x)}{\partial x_k}$ equal to 0 and solving the corresponding system of equation is not easy because of the nature of the confluent hypergeometric function. In figure (5.5), we illustrate the plot of $AE(x)$ and compute by numerical evaluations upper and lower bounds when $\sigma = 1$.

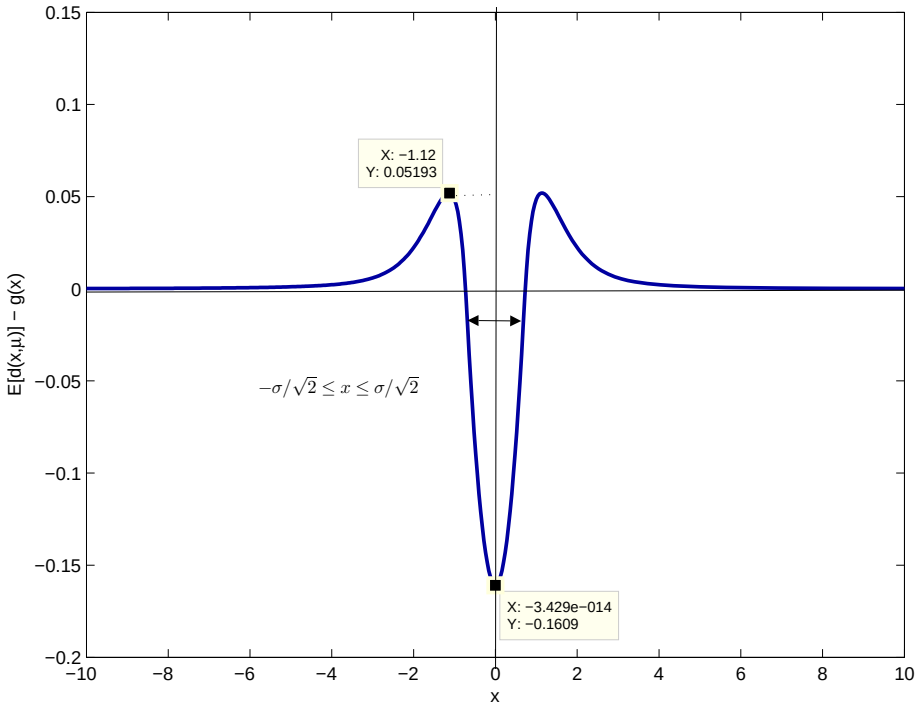


Figure 5.5. Upper and lower bounds for the approximation error

An interesting situation occurs when $d(\mathbf{x}_i, \mathbf{a}_j)$ is the squared Euclidean distance. Under the same probabilistic assumptions for customer coordinate distribution the approximation (5.26) becomes $E[d(\mathbf{x}_i, \mathbf{a}_j)] \approx d(\mathbf{x}_i, \boldsymbol{\mu}_j) + p\sigma_j$, where p is the dimension of the subspace. It is $p = 2$ for our problem. In short, when squared Euclidean distance is used to measure the distance between facility i and customer j, approximation becomes perfect.

5.5.3. Approximating expected distance using the average

Let us assume that we have a large enough random sample of location vectors for customers, and let S_j denote the set of sample vectors for customer j. This is in fact the case in most of the realistic applications: we rather have sample vectors instead of multivariate probability distributions of customer locations. Then we can use the simple average

$$\frac{1}{|S_j|} \sum_{\mathbf{a}_j \in S_j} d\left(\mathbf{x}_i, \mathbf{a}_j\right) \tag{5.38}$$

as an estimate for the expected distance. Such a sample is given either directly or as a distribution function from which we can generate the customer locations. Replacing $E[d\left(\mathbf{x}_i, \mathbf{a}_j\right)]$ in (5.3) with (5.38) results in the new objective function

$$E\left[z\right] = \sum_{i=1}^{m} \sum_{j=1}^{n} h_j y_{ij} \left(\frac{1}{|S_j|} \sum_{\mathbf{a}_j \in S_j} d\left(\mathbf{x}_i, \mathbf{a}_j\right) \right), \tag{5.39}$$

which is to be minimized subject to (5.4) and (5.5). Clearly,

$$E\left[z_i\right] = \sum_{j=1}^{n_i} \frac{h_j}{|S_j|} \sum_{\mathbf{a}_j \in S_j} d\left(\mathbf{x}_i, \mathbf{a}_j\right) \tag{5.40}$$

is the approximate probabilistic Weber problem which has to be solved during the location phase of PALA in order to locate facility i. Notice that it is equivalent to a (deterministic) Weber problem with customer set $\bigcup_{j=1}^{n_i} S_j$ in which facility i has to be located optimally with respect to $\sum_{j=1}^{n_i} |S_j|$ customers. This can be done by repeating formulae

$$x_{ik}^{(t+1)} = \frac{\sum_{j=1}^{n_i} \sum_{\mathbf{a}_j \in S_j} \frac{h_j}{|S_j|} \frac{a_{jk}}{d(\mathbf{x}^{(t)}, \mathbf{a}_j)}}{\sum_{j=1}^{n_i} \sum_{\mathbf{a}_j \in S_j} \frac{h_j}{|S_j|} \frac{1}{d(\mathbf{x}^{(t)}, \mathbf{a}_j)}} \qquad k = 1, 2 \tag{5.41}$$

when $1 \leq p \leq 2$. This is a straightforward extension of Weiszfeld's algorithm and its generalizations [14],[17],[33]. When there is only one sample vector available for every customer, i.e., $|S_j| = 1$ for $j = 1, \ldots n$, formula (5.40) reduces to the original Weiszfeld's formula. In other words, the location phase of PALA becomes equivalent to that of ALA.

Then, the allocation phase of PALA involves determining facility i^* for each customer j as follows:

$$i^* = \arg \min_{i=1,\ldots,m} \left\{ \frac{1}{|S_j|} \sum_{\mathbf{a}_j \in S_j} d\left(\mathbf{x}_i, \mathbf{a}_j\right) \right\}. \tag{5.42}$$

Observe that, this can be equivalently written as

$$i^* = \arg \min_{i=1,\ldots,m} \left\{ \sum_{\mathbf{a}_j \in S_j} d\left(\mathbf{x}_i, \mathbf{a}_j\right) \right\}. \tag{5.43}$$

6. A VARIABLE NEIGHBORHOOD SEARCH ALGORITHM FOR SOLVING PROBABILISTIC MULTI-FACILITY WEBER PROBLEM

6.1. Variable neighborhood search

Variable neighborhood search (VNS) is a metaheuristic that combines local search with a systematic way of exploring the solution space [43]. VNS works with a set of predefined neighborhood structures. If the current solution is locally optimal with respect to a given neighborhood, then the neighborhood is augmented to examine a larger region of the solution space. Although there are some variations and extensions of VNS [39], we adopt in this chapter the version that is successfully applied by Brimberg et al. [15] for the solution of the MFWP (also referred to as the multi-source Weber problem) with the Euclidean distance. However, we modify it by replacing the step of the algorithm which performs local search. Namely, instead of ALA we employ the probabilistic version (PALA) as outlined below.

Algorithm 2. *VNS heuristic for the PMFWP*

1. *(Initialization): Specify a starting solution and run PALA to obtain a local minimum, say* y_c*, set* $k = 1$.
2. *(Neighborhood search): Randomly select* b *points in the* k*-neighborhood of* y_c*. For each of these points run PALA to obtain local minima* y_l*,* $l = 1, \ldots, b$*. These solutions are not necessarily all unique. Retain the best solution,* $y^* \in \{y_l, l = 1, \ldots, b\}$*. If* y^* *is a better solution than* y_c*, set* $y_c = y^*$*,* $k = 1$ *and go to 2; otherwise go to 3.*
3. *(Augmenting the neighborhood):Set* $k = k+1$*. If* $k \leq k_{\max}$ *then go to 2; otherwise STOP and declare* y_c *as the solution.*

The only parameters to be specified in VNS are b and $k_{\max}$. The selection of the neighborhood structure also plays a crucial role in the performance of the VNS

heuristic. There are two types of neighborhood structures considered in the literature: reallocation and relocation neighborhoods [15]. In the reallocation neighborhood, the k-neighborhood of a given solution is defined as the set of all possible solutions that can be obtained by changing the allocations or assignments of k customers. When the number of facilities to be located is m and the number of customers is n, then the size of the k-neighborhood is given as $\binom{n}{k}(m-1)^k$. The earliest VNS heuristic proposed for the solution of the MFWP is based on this neighborhood structure [19]. Later on, Brimberg et al. [15] propose the relocation neighborhood where new solutions are generated from the current one by changing the locations of facilities. From the results on benchmark instances of the MFWP, the authors conclude that the quality of the solutions obtained with VNS is improved significantly when relocation neighborhood is used.

There exist different moves (strategies that lead from one solution to another one) that can be used to perform facility relocations. All of them consist of removing one or more facilities from their sites and inserting them back at unoccupied customer locations. In this work we implement exchange (EX) (also known as swap or interchange) and drop-and-add (DA) moves. The k-neighborhood of a solution with respect to EX move is obtained by removing k facilities from their locations in the solution and reinserting them at k unoccupied customer locations. The size of the neighborhood becomes then $\binom{m}{k}\binom{n}{k}$. If $k = 1$, then the location of only one facility is exchanged, which makes the neighborhood size equal to mn. In a k-DA move first k facilities are deleted according to some criterion. Then they are inserted at unoccupied customer locations using another criterion. Depending on the criterion selected for facility deletion and insertion, it is possible to obtain a multitude of DA moves. It is found by Brimberg et al. [15] that the best performing criteria for dropping a facility are "Drop least useful", "Drop by second closest", and "Drop by minimum potential", while "Add least useful" and "Add by second closest" are their counterparts for inserting a facility. Considering both the solution quality and running time performance of these criteria reported in [15], we choose "Drop by second closest" (DSC) and "Add by second closest" (ASC) rules for the DA moves.

The DSC rule for a single facility works as follows. First, the second closest facility is determined for each customer. For this purpose, we use customers' mean location vector. Then facility i is removed temporarily. The contribution of i to the current objective value, Φ_i, is computed. The demand-weighted sum of distances between customers without a facility and their second closest facility,Ψ_i, is also computed. Note that $\Psi_i \geq \Phi_i$. After Φ_i and Ψ_i are determined for each facility $i = 1, \ldots, m$, the facility to be deleted is decided as the one for which $\Psi_i - \Phi_i$ is minimum. In other words, the facility whose deletion causes the smallest increase in the objective value when the customers it serves are reassigned to their second closest facilities is the one to be deleted.

The underlying principal of ASC rule is quite similar. Deleted facility is inserted back at an unoccupied customer location j. Those customers that are now closer to the newly inserted facility other than to their current facility are reassigned. The new objective value is denoted as Φ_j. This facility insertion procedure is repeated for all unoccupied customer locations, and the one with the smallest Φ_j becomes the location of the inserted facility. The extension to k facilities is straightforward.

Consequently, the DA move with DSC and ASC criteria (referred to as DASC in the sequel) and EX move are used in the relocation neighborhood structure that constitutes Step 2 of our VNS algorithm. As will be noticed in the next section, the relocation neighborhood performs much better than the reallocation neighborhood and improves the accuracy of VNS considerably.

6.2. Computational Results

We first consider the situation where it is possible to obtain an exact analytical expression for the expected distance (5.6) and report results with PALA and VNS. We then try to assess the quality of the two approaches proposed for approximating the expectation according to their influence on VNS heuristics.

Confluent hypergeometric distribution does not have a closed form formula and

should be evaluated numerically. Although this can be done by using one of the available functions, its computational evaluation is expensive and decreases overall efficiency drastically. A quick remedy is to create a table of confluent hypergeometric probabilities and look it up whenever a confluent hypergeometric probability is required. Missing values can be calculated through linear interpollation. Based on our previous experiences we can say that the error due to table look-ups is insignificant.

In the implementation of VNS we use the relocation neighborhood structure. In an earlier version of this work the neighborhood was generated by reallocating customers. The performance of the reallocation neighborhood structure was inferior. Thus we decided to use the relocation neighborhood structure in our VNS implementations. Earlier results by Brimberg et al. [15] supports also this choice. As mentioned before, DASC and EX move strategies are implemented to generate neighboring solutions within the VNS framework. Consequently, each test instance is solved by PALA, VNS with DASC moves (VNS-DASC), and VNS with EX moves (VNS-EX). VNS parameters are set $b = 1$ and $k_{\max} = n$ for small instances and $b = 1$ and $k_{\max} = m$ for medium and large instances.

A test problem library including optimal objective values is not available for the PMFWP. Therefore we tried to solve our test instances by using a brute-force approach. We first enumerated all possible customer-to-facility assignments. We then computed optimal solution of the corresponding probabilistic (single facility) Weber problems for each facility. Although finding a global optimum is guaranteed by this exhaustive scheme, it cannot be implemented for medium and large instances since the number of possible assignments grows exponentially with the increasing number of facilities and/or customers. As a result we could only find optimal solutions of the instances based on Cooper's (`cooper7`, seven instances) and the two smallest of Rosing's (`ros10` and `ros15`) test problems, and we report percent deviations from optimal objective values for these small instances. They are calculated according to formulae $100\times\left(z_{\text{PALA}} - z^*\right)/z^*$ and $100\times\left(z_{\text{VNS}} - z^*\right)/z^*$, where z_{PALA} and z_{VNS} correspond to the objective values obtained by PALA and two VNS heuristics, and z^* is the optimal objective value computed by complete enumeration. For medium

and large instances we compare percent improvements introduced by VNS heuristics over PALA, which is calculated using formulae $100 \times (z_{\text{VNS-DASC}} - z_{\text{PALA}}) / z_{\text{PALA}}$ and $100 \times (z_{\text{VNS-EX}} - z_{\text{PALA}}) / z_{\text{PALA}}$. Each instance is solved 10 times by PALA, VNS-DASC and VNS-EX. Each run is randomly initialized and the same random initial conditions are used for all of three heuristics. The objective value for each run is computed using analytical expression (5.10) with the facility coordinates produced by the heuristic. In other words, objective function (5.10) is used as a performance metric.

To make a fair comparison among the three heuristics in terms of the computation time, we adapted the policy suggested in [15]. Each instance is solved by PALA 10 times, each run starting with a different random initial condition. Total CPU time of this 10 runs is then used as a limit to terminate the iterations of a single VNS-DASC or VNS-EX run. In other words, the maximum time devoted for each run of the VNS heuristics is determined by the computation time required for 10 PALA runs. It is possible that a VNS run is completed before the time limit is reached. As a result, we do not report CPU time statistics.

6.2.1. Test bed

No standard test library for the PMFWP is available. Therefore, we created a test bed by modifying some of the existing Euclidean and rectilinear MFWP instances. They are listed in the first column of Table 6.1. Remaining columns indicate the number of facilities to locate (m), the number of customers (n), and the source of the base MFWP instance. Instances based on Cooper [24], Eilon et al. [32], Rosing [86] and TSPLIB [84] have unit demands, which is not the case for the remaining ones.

In order to make these instances suitable for our experiments we set the mean values of the confluent hypergeometric distribution equal to coordinates given in the reference. Variances along both axis were chosen equal and set to $p\%$ of the absolute difference between the smallest and largest customer coordinates. We call p the "variance parameter" and set $p = 10\%$ during the generation of our test instances.

Table 6.1. Test instances

Instance	m	n	Source
cooper7(7 problems)	2	7	Cooper [24]
ros10	2	10	Rosing [86]
ros15	2	15	Rosing [86]
ros20	2–6	20	Rosing [86]
ros25	2–6	25	Rosing [86]
ros30	2–5	30	Rosing [86]
lj100	2–10	100	Love and Juel [63]
lj20/100	2–10	20	lj100
lj40/100	2–10	40	lj100
lj60/100	2–10	60	lj100
lj80/100	2–10	80	lj100
eil50	2–10, 15, 20, 25, 30	50	Eilon et al. [32]
bon287	2–10, 20, 30, 40,50	287	Bongartz et al. [11]
p654	2–10, 20, 30, 40,50	654	TSPLIB [84]
u1060	2–10, 20, 30, 40, 50	1060	TSPLIB [84]

6.2.2. Results with exact expectation

We use the descent method whose iterations are given with expression (5.15) with gradient (5.14) to solve probabilistic Weber problem, namely minimize (5.11).

6.2.2.1. Small instances. The first three columns of Table 6.1 include information about the instances: data set it is based, number of facilities and optimal objective value calculated by complete enumeration. We report the average of 10 relative deviations obtained at the end of 10 runs on each test instance.

Column averages provided in the last row indicate that the accuracy of VNS is higher than that of PALA as expected, and DASC is slightly more accurate then EX as move strategy. The average computation times over all instances are 4.82, 0.06 and 0.43 seconds for PALA, VNS-DASC and VNS-EX respectively. However, complete

Table 6.2. Percent deviations from the optimal objective values.

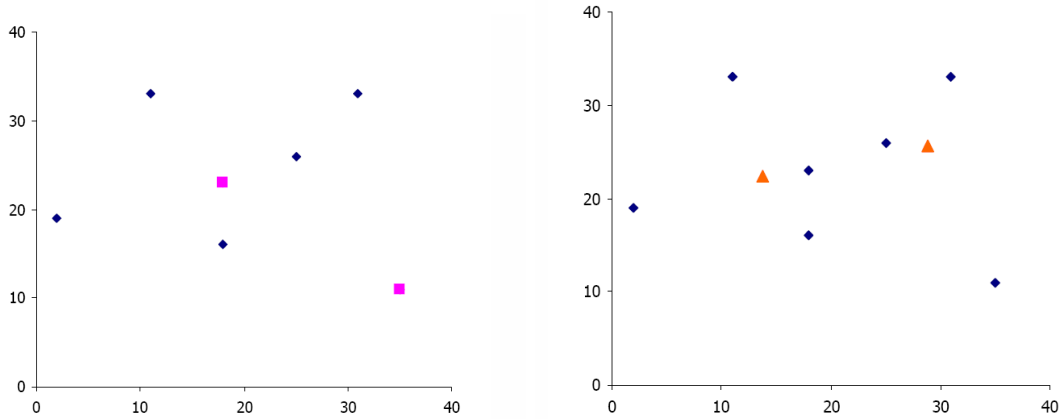
Dataset	m	Optimal	PALA (%)	VNS-DASC (%)	VNS-EX (%)
cooper7A	2	56.5023	3.62	0.07	0.65
cooper7B	2	79.7259	6.02	0.00	1.43
cooper7C	2	47.7226	0.00	0.00	0.00
cooper7D	2	53.0093	0.00	0.00	0.00
cooper7E	2	43.5922	0.00	2.02	0.91
cooper7F	2	68.4067	3.84	1.91	1.18
cooper7G	2	71.0697	0.23	0.79	0.39
ros10	2	115.4125	0.00	0.00	0.00
ros15	2	234.905	2.67	0.54	0.88
Average			1.82	0.59	0.60

enumeration takes 2657.59 seconds on the average. Since the instances are very small, the computational effort spent on complete enumeration is not prohibitive.

At this point, we also would like to remark that final solutions obtained for probabilistic problem PMFWP should be expected to be different than the ones obtained for the deterministic problem MFWP. This fact is illustrated in Figure 6.1 for `cooper7F`. "♦" represents customer locations in (a) and mean customer locations in (b). "■" and "▲" represent optimal facility locations $(18, 23)^T$ and $(35, 11)^T$ for the MFWP with an objective value of 59.72 [24], and $(13.74, 22.42)^T$ and $(28.86, 25.68)^T$ with and objective value of 68.41 for the PMFWP.

6.2.2.2. Medium and large instances. In this section we solve the remaining test instances given in Table 6.1 using PALA and VNS heuristics and compare their results. The instances are categorized as medium and large based on the number of customers, n. An instance, which is not in the set of small instances, is medium if $n \leq 50$, and large if $n > 50$. Figure 6.2 visualizes results for medium instances in terms of percent improvement VNS-DASC and VNS-EX introduces over PALA. Labels on the x-axis are the name of the instance and the number of facilities to be located. For example the first two values are obtained by averaging 10 values computed with 10 randomly

initilized runs of VNS-DASC and VNS-EX on `eil50` based instance with $m = 2$. For each heuristic, facility location calculated at the end of each run is evaluated with (5.10) and relative improvement introduced by the heuristics are calculated. Then the average of these relative improvements is represented in Figure 6.2.



(a) MFWP (b) PMFWP

Figure 6.1. Optimal Facility Locations for cooper7F

Clearly, there is an improvement for all instances as expected; since VNS extends the capabilities of local search anyhow. Moreover, VNS-DASC outperforms VNS-EX. The improvements are particularly noticeable when the number of facilities increases for a dataset, i.e, for the same mean and variance of customer locations, and customer demands. For some of the instances, e.g. `lj20/100` and `lj40/100`, the percent improvements become as high as 30%. The average improvements by VNS-DASC and VNS-EX, namely the overall average of the averages computed by running a heuristic 10 times, are respectively 9.0% and 12.87%.

Figure 6.3 is a replication of Figure 6.2 for large instances. As can be observed the performances of the VNS heuristics are higher for large instances except for `bon287`. Improvements up to 40% and 47% are reached for `lj100` ($m = 8$) and `p654` ($m = 10$). VNS-DASC again performs better than VNS-EX. Overall average percent improvements, for both medium and large instances, over PALA are 25.1% for VNS-DASC and 21.3% for VNS-EX. If we treat results on `bon287` as outliers, these improvements go up to 21.3 % and 25.1%.

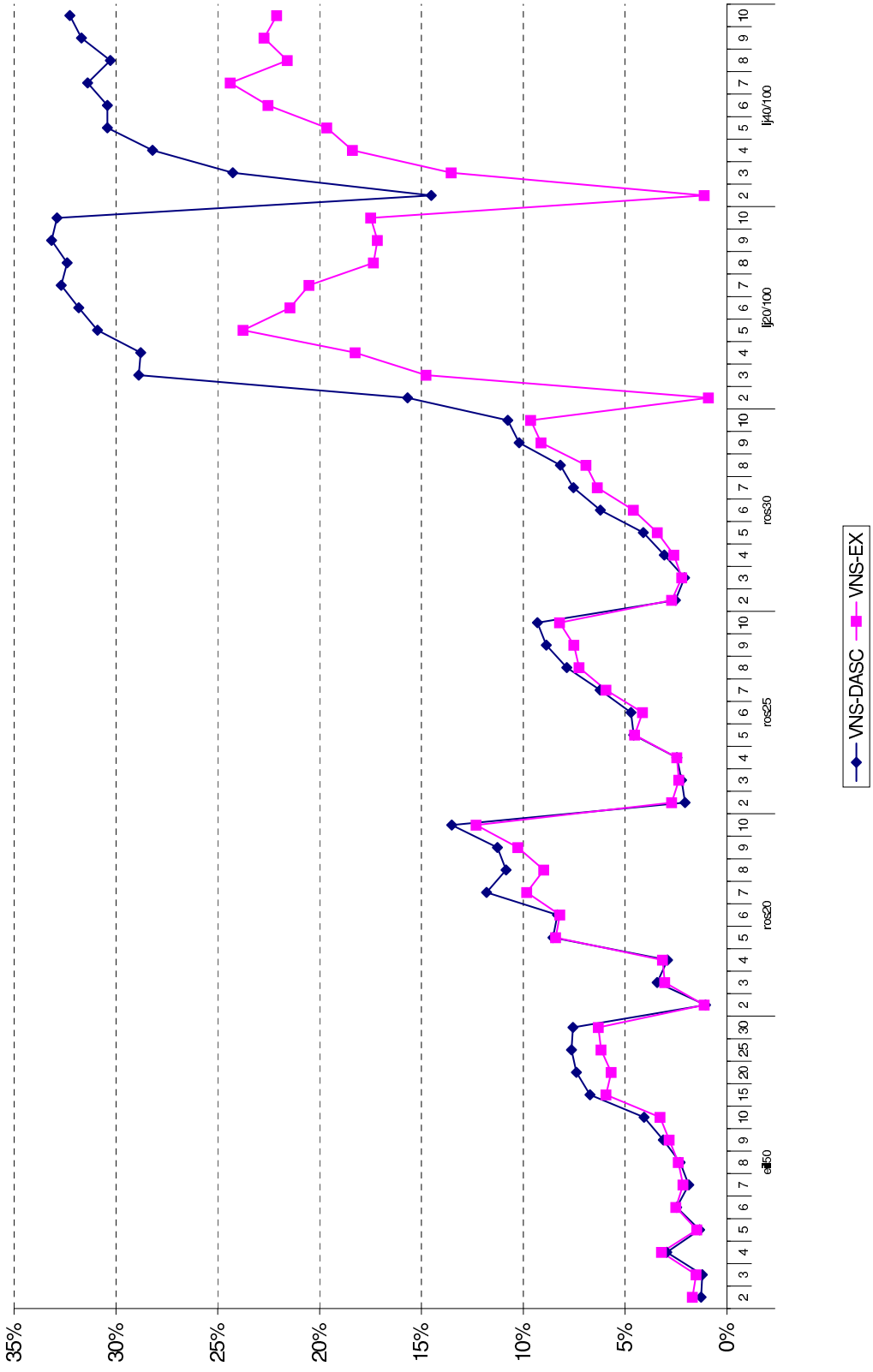


Figure 6.2. Performance of VNS heuristics on medium instances

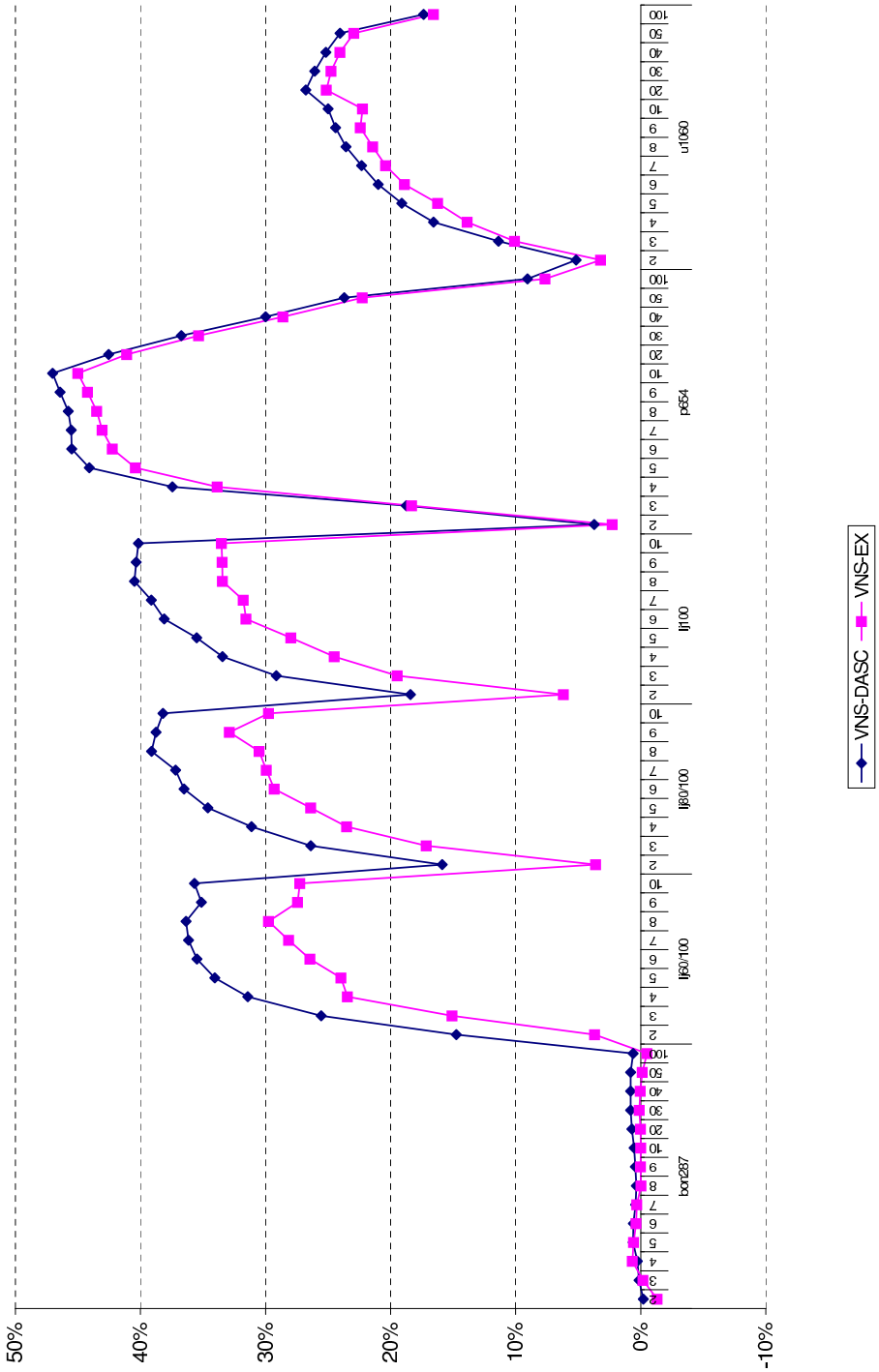


Figure 6.3. Performance of VNS heuristics on large instances

6.2.3. The quality of the approximations

We note that heuristics are derived under the assumption that customer coordinates are independent symmetric bivariate normal random variables. Therefore, they evaluate (5.8) in their updates. Now, we leave the assumption that a closed form expression for the expected distance is not possible to derive and use two approaches proposed previously to approximate (5.7). First one approximates $E[d(\mathbf{x}_i, \mathbf{a}_j)]$ using the mean, variance and covariances of customer coordinates a_j. The second one, on the other hand, assumes that a sample of customer coordinates is available and suggests to replace $E[d(\mathbf{x}_i, \mathbf{a}_j)]$ with the average distance. We let VNS-DASC1 and VNS-EX1 denote the versions of VNS heuristics which use the first approximation, and VNS-DASC2 and VNS-EX2 denote the versions with the second approximation. We compare the performance with the ones of VNS-DASC and VNS-EX to assess the quality of the approximations.

6.2.3.1. Small instances. Table 6.3 includes the average of 10 percent relative deviations from optimal. Descent method (5.15) is used to determine heuristic solution during the calculation of the deviations. Recall that VNS-DASC and VNS-EX performs very well on these instances (see Table 6.2). When $E[d(\mathbf{x}_i, \mathbf{a}_j)]$ is approximated, deviations from the optimal tend to increase, as expected. They go up to 13.8% with VNS-DASC1 (`cooper7D`) and 20.46% with VNS-EX1 (`cooper7F`). However, they become very close to zero, for VNS-DASC2 and VNS-EX2. Hence, we conclude that the second approximation performs better. This can also be observed in the last row of Table 6.3: overall averages for VNS-DASC1 and VNS-EX1 are respectively 6.51 % and 11.32 %, which become 1.75 % and 4.5 % for VNS-DASC2 and VNS-EX2.

6.2.3.2. Medium and large instances. Similar to the previous section, we use averages of 10 runs obtained with PALA as benchmark, and we report percent deviations from them. We keep the same test bed except `u1060` based instance is excluded, just because the computation time it requires is prohibitive for our environment. We keep the same values for the VNS parameters and use the same strategy to limit running times of the

Table 6.3. Quality of the approximations: relative deviations from optimal objective values

Dataset	m	Optimal	VNS-DASC1 (%)	VNS-EX1 (%)	VNS-DASC2 (%)	VNS-EX2 (%)
cooper7A	2	56.5023	6.44	12.91	0.69	3.06
cooper7B	2	79.7259	2.43	11.44	2.45	7.21
cooper7C	2	47.7226	6.68	16.13	0.82	2.52
cooper7D	2	53.0093	13.80	4.29	2.57	6.17
cooper7E	2	43.5922	10.45	14.36	3.80	8.19
cooper7F	2	68.4067	4.64	20.46	2.30	4.15
cooper7G	2	71.0697	4.36	17.58	1.39	3.80
ros10	2	115.4125	7.07	1.98	0.89	3.48
ros15	2	234.905	2.76	2.69	0.82	1.94
Average			6.51	11.32	1.75	4.50

heuristics.

We summarize final results obtained with VNS heuristics using approximations in Figure 6.4 and Figure 6.5. Results are given as percent deviations between the final objective values calculated with VNS heuristics and PALA. There are four plots representing respectively VNS-DASC1, VNS-EX1, VNS-DASC2 and VNS-EX2 respectively. As can be oberved, both approximations improve on PALA on most of the test problems. For some of them, such as `lj` data sets and `p654`, improvements are as high as the ones obtained with exact expectation; they go up to 30%. Improvements usually increase with increasing number of facilities as before. However, for data sets like `bon287`, `eil50` and `ros`, the performance of the first approximation is poor and worsens with increasing number of facilities on the contrary; the poorest results are obtained for `bon287`. We present Figure 6.6 and Figure 6.7 in order to give a better idea on the quality of the expected distance approximations. Figure 6.6 illustrates the differences between the average percent improvements over PALA obtained by VNS heuristics (VNS-DASC1 and VNS-EX1) employed with approximated expected distances and those obtained by VNS heuristics (VNS- DASC and VNS-EX) executed with exact

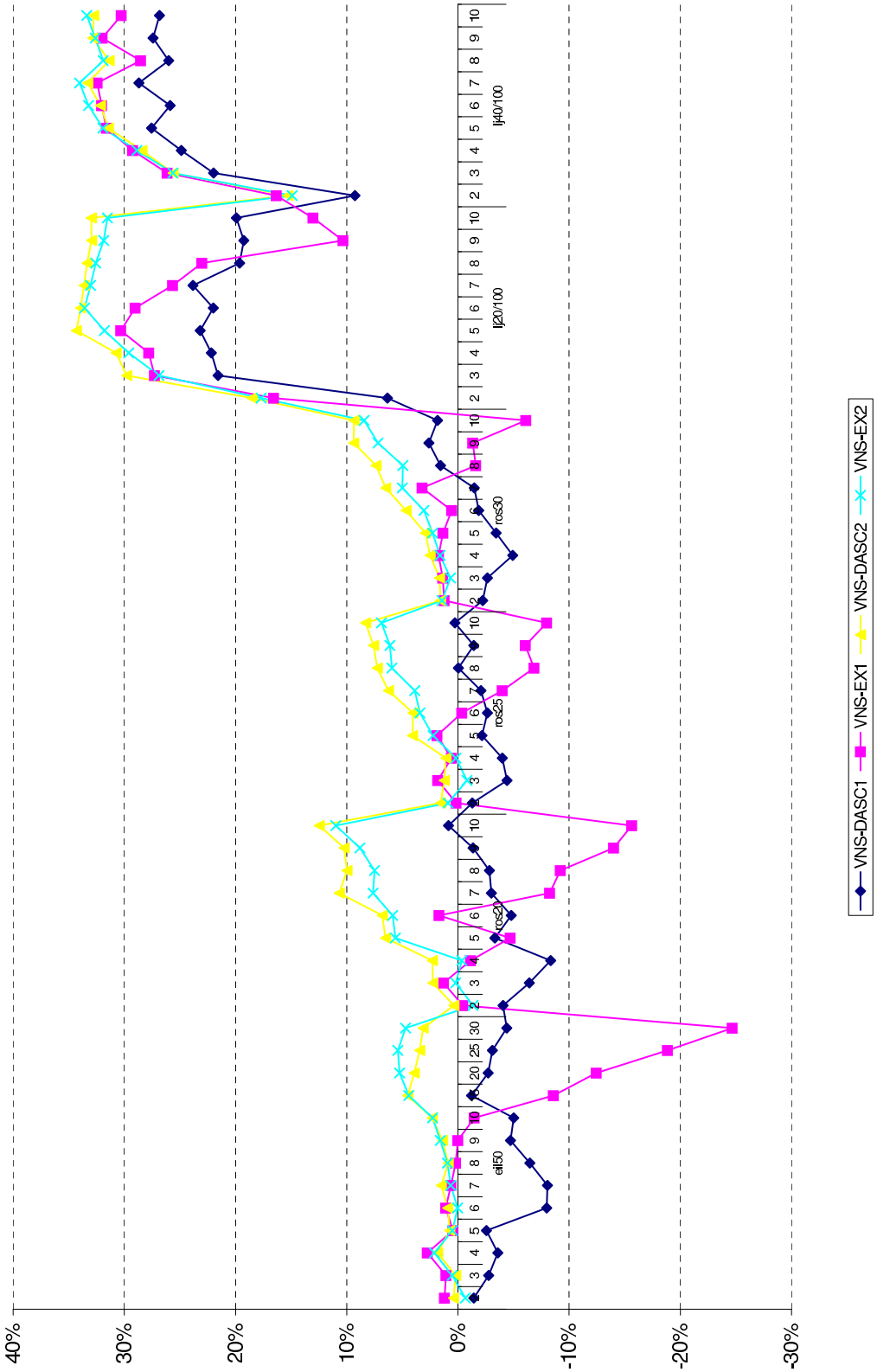


Figure 6.4. Performance of VNS heuristics with approximations on medium instances

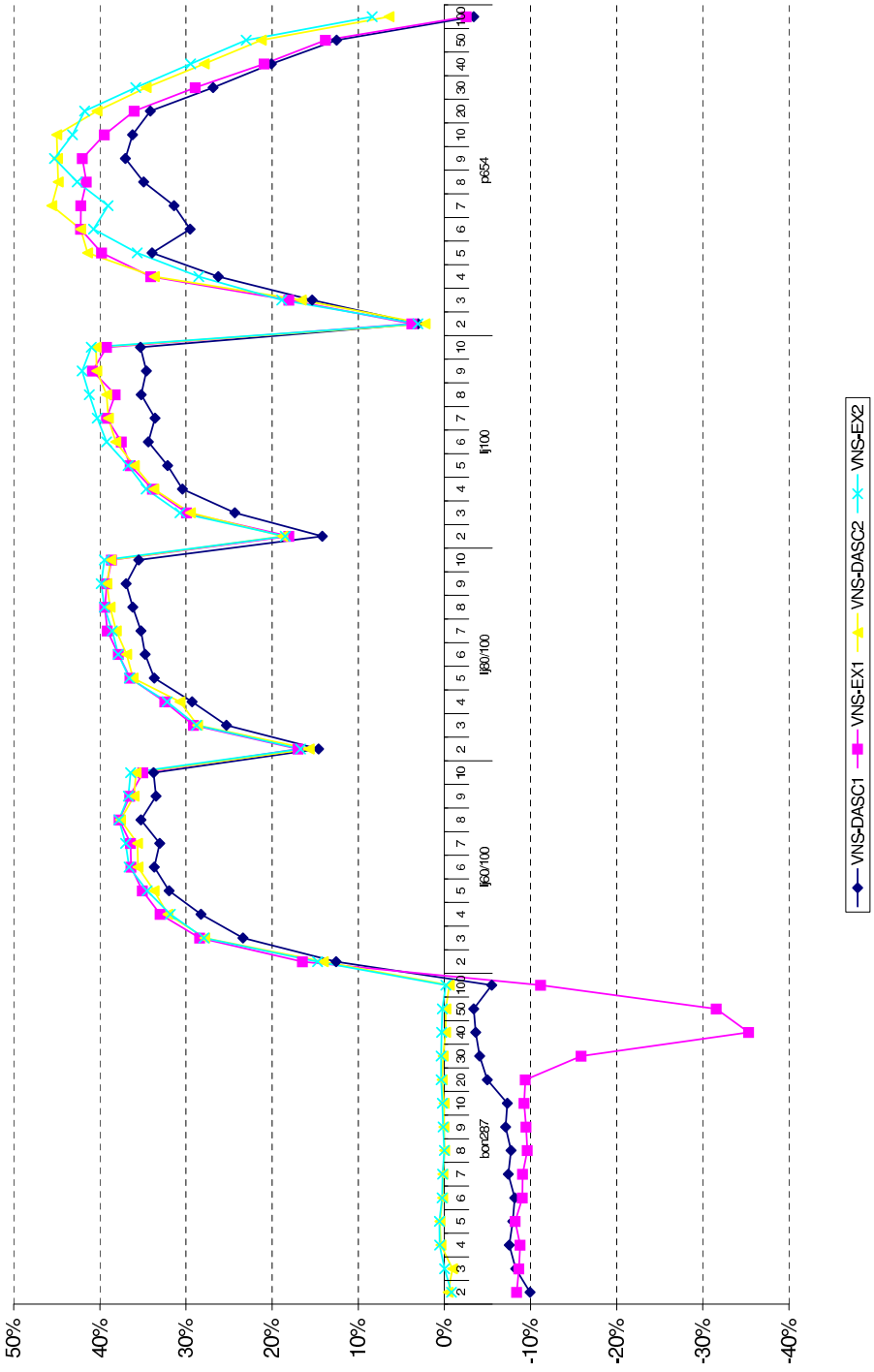


Figure 6.5. Performance of VNS heuristics with approximations on large instances

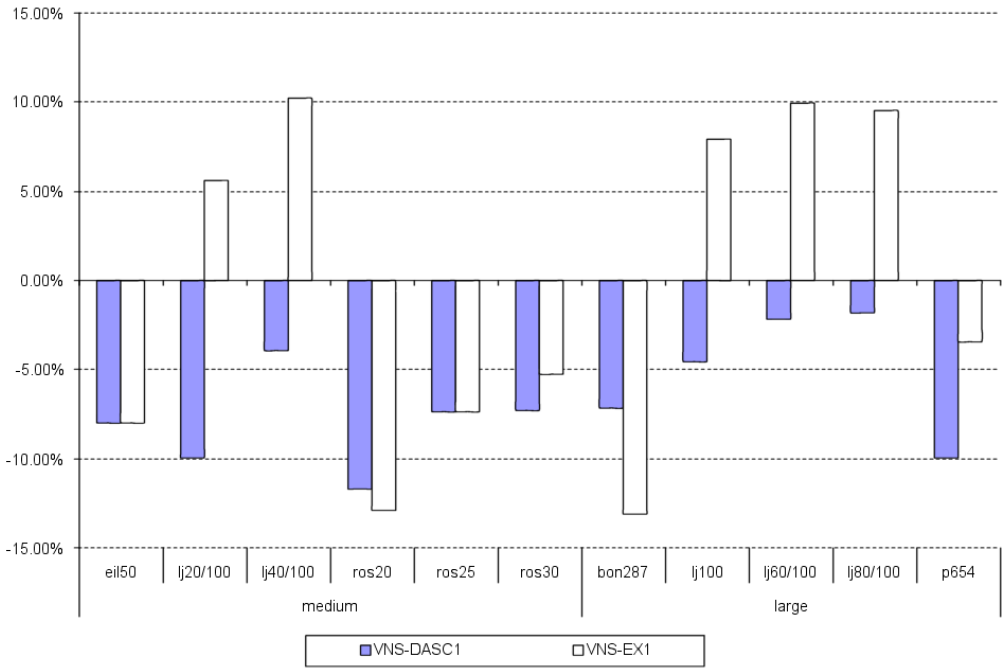


Figure 6.6. Quality of the first approximation

expected distances. Note that a negative difference shows that the approximation deteriorates the objective value while a positive difference is an indication for an improved objective value. Furthermore, for each instance we consider the average percent improvements with and without approximation are calculated over all the problems with dierent number of facilities. For example, instance eil50 consists of 13 dierent problems with $2 \leq m \leq 30$. A closer look at the figures reveals that VNS-DASC1 is worse than VNS-DASC on all instances. However, VNS-EX1 performs better than VNS-EX on *lj* data sets where improvements up to 10% higher can be obtained with approximation on lj40/100, lj60/100 and lj80. We present Figure 6.7 to assess the quality of the second approximation in which expected distances are replaced by the average distances calculated using a sample for customer locations. Performances of VNS-DASC2 and VNS-EX2 are closer to the ones of VNS-DASC and VNS-EX. For both VNS-DASC2 and VNS-EX2, the gap is below 2% on most data sets. An improvement of 10% is obtained for `lj` data set for example. We report the overall averages (average on all test instances) of the average improvements obtained with VNS heuristics over PALA in Table 6.4. The first column identifies the method used to evaluate expected distances. The highest improvements are obtained with exact evaluation of $E[d(x_i; a_j)]$,

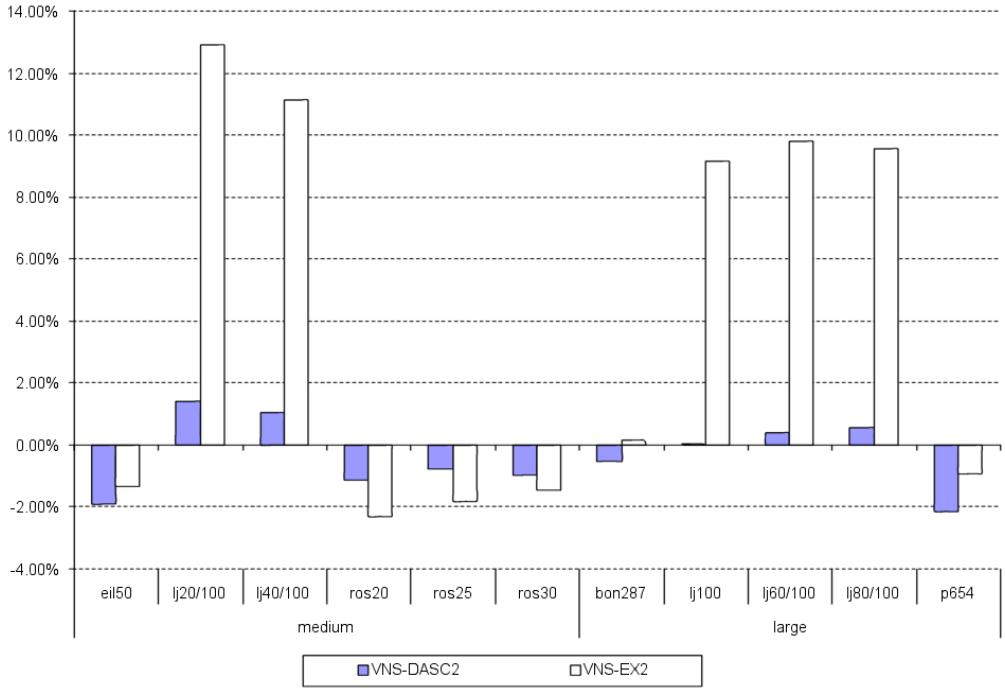


Figure 6.7. Quality of the second approximation

as expected: average improvement over PALA is 18.83%. We obtain a similar performance with the second approximation. Moreover, as mentioned previously, we remark higher performance of VNS-EX2 on the average. We also would like to remark that VNS-DASC1 performs the poorest, with an average decrease of -6.91%. This means that, on the average, VNS-DASC1 performs 6.91% worse than VNS-DASC. When the same approach is used for VNS-EX1 a decrease of -1.54% occurs. These values are -0.51% for VNS-DASC2 and 3.50% for VNS-EX2.

For DASC move strategy the performance of the first approximation is worse than the second one. For EX move strategy it is surprising to see that one of the approximations (second one) gives a lower average deviation than exact evaluation does. This can be explained as the positive effect of the implicit "noise" introduced by the approximation: it creates a chance for an uphill move to a better local minimum. The use of random noise is essential in noising methods for solving combinatorial optimization problems [21]. When $E[d(\mathbf{x}_i, \mathbf{a}_j)]$ is approximated, VNS can have a chance of moving from a local optimal solution easier for some data sets and the approximation creates an implicit noise resource. However, noise is introduced explicitly in the noising

Table 6.4. Overall average of improvements over PALA

	VNS-DASC (%)	VNS-EX (%)
Exact	18.83	14.52
First Approx.	11.92	12.98
Second Approx.	18.32	18.02

methods. This also explain the reason why our approximations performs better than exact evaluations on some of the test instances.

7. VECTOR QUANTIZATION FOR SOLVING THE PROBABILISTIC MULTI-FACILITY WEBER PROBLEM

VQ and PMFWP are closely related. Note that, similar to the deterministic case, y_{ij}'s are either zero or one at optimality since serving each customer from the facility with the smallest expected distance is an optimal policy, and therefore the PMFWP can be formulated equivalently as

$$\min_{\mathbf{x}_1,\ldots,\mathbf{x}_m} \sum_{j=1}^{n} h_j \min_{i=1,\ldots,m} \left\{ E\left[d\left(\mathbf{x}_i, \mathbf{a}_j \right) \right] \right\}. \tag{7.1}$$

Moreover, it is possible to compute the minimum of the expected distances between a given random customer location and facilities in 7.1 as similar to what we have in the objective function of vector quantization. This is shown in the following lemma.

Lemma 7.1. *For an arbitrary customer j, computing the minimum of expected distances in (7.1) is the same as computing the expected distance of the closest facility for the VQ problem. In other terms,*

$$\min_{i=1,\ldots,m} \left\{ E\left[d\left(\mathbf{x}_i, \mathbf{a}_j \right) \right] \right\} = E\left[\min_{i=1,\ldots,m} \left\{ d\left(\mathbf{x}_i, \mathbf{a}_j \right) \right\} \right]$$

Proof. We denote the closest facility by the index c similarly to the notations of the previous chapters. Then for $i \neq c, d\left(\mathbf{x}_c, \mathbf{a}_j \right) \leq d\left(\mathbf{x}_i, \mathbf{a}_j \right)$. Note that $\min_{i=1,\ldots,m} \left\{ E\left[d\left(\mathbf{x}_i, \mathbf{a}_j \right) \right] \right\}$ is equal to

$$\min \left\{ E\left[d\left(\mathbf{x}_1, \mathbf{a}_j \right) \right], E\left[d\left(\mathbf{x}_2, \mathbf{a}_j \right) \right], \ldots, E\left[d\left(\mathbf{x}_m, \mathbf{a}_j \right) \right] \right\}. \tag{7.2}$$

Let us denote by index k, the minimum of these expected values. Therefore,

$$E\left[d\left(\mathbf{x}_k, \mathbf{a}_j \right) \right] \leq E\left[d\left(\mathbf{x}_i, \mathbf{a}_j \right) \right] \quad \text{for } i \neq k \tag{7.3}$$

Now assume that $k \neq c$. Then, the minimum of these expected distances should be

even smaller than the expectation of the distance between the closest facility and the arbitrary customer j. However this is a contradiction. Since

$$d(\mathbf{x}_c, \mathbf{a}_j) \leq d(\mathbf{x}_i, \mathbf{a}_j) \quad \text{for } i \neq c \tag{7.4}$$

we have

$$d(\mathbf{x}_c, \mathbf{a}_j) \leq d(\mathbf{x}_k, \mathbf{a}_j) \tag{7.5}$$

Integrating both sides and using the mathematical definition of expected value of an arbitrary function of a random variable, we may conclude that

$$\int d(\mathbf{x}_c, \mathbf{a}_j) f(\mathbf{a}_j) d\mathbf{a}_j \leq \int d(\mathbf{x}_k, \mathbf{a}_j) f(\mathbf{a}_j) d\mathbf{a}_j \tag{7.6}$$

$$E[d(\mathbf{x}_c, \mathbf{a}_j)] \leq E[d(\mathbf{x}_k, \mathbf{a}_j)] \tag{7.7}$$

where $f(\mathbf{a}_j)$ is the probability density function of the location vector of customer j. Hence, we should have $k = c$. □

Therefore (7.1) can also be stated as

$$\min_{\mathbf{x}_1,\ldots,\mathbf{x}_m} \sum_{j=1}^{n} h_j \left\{ E\left[\min_{i=1,\ldots,m} \{d(\mathbf{x}_i, \mathbf{a}_j)\} \right] \right\}. \tag{7.8}$$

In particular, the following analogy exists between the two problems: the customer locations $\left\{ \mathbf{a}_j = (a_{j1}, a_{j2})^T : j = 1, \ldots, n \right\}$ of the PMFWP correspond to the input vectors $\mathbf{v}$ of VQ and the unknown coordinates of the facilities

$$\left\{ \mathbf{x}_i = (x_{i1}, x_{i2})^T : i = 1, \ldots, m \right\}$$

correspond to the unknown coordinates of the codebook vectors. Then $d(\mathbf{w}_c, \mathbf{v})$ becomes $d(\mathbf{x}_c, \mathbf{a}_j)$, which is exactly $\min_{i=1,2,\ldots,m} d(\mathbf{x}_i, \mathbf{a}_j)$. We exploit this relation while

proposing the new VQ algorithms. Each algorithm is devised for a different level of information about customer locations.

7.1. Unlabeled data

First we assume that an unlabeled sample set of customer locations is available. In other words, it is completely unknown which of the available location vectors belongs to which customer. We therefore assume that any location vector $\mathbf{a}$ is sampled from some unknown probability density function $f(\mathbf{a})$. Let us consider the situation where customers have equal demands. In this case the expected value of the minimum distance can be written as

$$E[\min_{i=1,\ldots,m} \{d(\mathbf{x}_i, \mathbf{a})\}] = \int_{\mathbf{a}} \min_{i=1,\ldots,m} \{d(\mathbf{x}_i, \mathbf{a})\} f(\mathbf{a}) d\mathbf{a}. \tag{7.9}$$

As a result, optimization problem (7.8) reduces to

$$\min \int_{\mathbf{a}} \min_{i=1,\ldots,m} \{d(\mathbf{x}_i, \mathbf{a})\} f(\mathbf{a}) d\mathbf{a}, \tag{7.10}$$

where the outer minimization is on facility coordinates. Since $f(\mathbf{a})$ is not known and there is a sample set S of vector $\mathbf{a}$, we replace the above integral with the average to obtain

$$\min \frac{1}{|S|} \sum_{\text{For all } \mathbf{a}} \min_{i=1,\ldots,m} \{d(\mathbf{x}_i, \mathbf{a})\} \tag{7.11}$$

as the approximating optimization problem. Note that $\frac{1}{|S|}$ is a constant and can be dropped off, and the problem becomes equivalent to the two-dimensional version of VQ and the algorithm defined for the MFWP in Chapter 4 could be used to conclude a solution. This revised VQ algorithm is modified to obtain the following update formula when a distance function other than the squared Euclidean distance is considered in

expression (7.11):

$$\begin{aligned} \mathbf{x}_c\,(t+1) &= \mathbf{x}_c\,(t) - \alpha\,(t)\,\nabla_x d\,(\mathbf{x}_c\,(t)\,,\mathbf{v}) \\ \mathbf{x}_i\,(t+1) &= \mathbf{x}_i\,(t) \quad\quad i \neq c \end{aligned} \tag{7.12}$$

The winner must also be selected according to the new distance function. Table 4.3 summarizes winner selection and componentwise update formulae for the Euclidean, squared Euclidean, rectilinear and the most general ℓ_p distances.

7.2. Labeled data

In this case we assume that sample location vectors are labeled with respect to customers. Hence it is possible to partition the sample set S into n disjoint subsets S_j: $j = 1, \ldots, n$, with S_j denoting the subset of vectors sampled for the location of customer j. We therefore implicitly assume that vector $\mathbf{a}_j$ is sampled from some unknown joint probability density function $f(\mathbf{a}_j)$. Then, the expected distance for customer j becomes

$$E[d\,(\mathbf{x}_i, \mathbf{a}_j)] = \int_{\mathbf{a}_j} d\,(\mathbf{x}_i, \mathbf{a}_j)\, f(\mathbf{a}_j) d\mathbf{a}_j. \tag{7.13}$$

Note that the PMFWP aims at determining the optimal facility locations by minimizing the minimum of the expected distances and thus formulation (7.8) can be equivalently written as

$$\min \sum_{j=1}^{n} h_j \min_{i=1,\ldots,m} \int_{\mathbf{a}_j} d\,(\mathbf{x}_i, \mathbf{a}_j)\, f(\mathbf{a}_j) d\mathbf{a}_j. \tag{7.14}$$

Since $f(\mathbf{a}_j)$ is unknown and there is a sample set S_j of vector $\mathbf{a}_j$, we approximate the above integral with the average distance to obtain

$$\min \sum_{j=1}^{n} h_j \min_{i=1,\ldots,m} \frac{1}{|S_j|} \sum_{\mathbf{a}_j \in S_j} d\,(\mathbf{x}_i, \mathbf{a}_j) \tag{7.15}$$

as the approximating optimization problem, which is similar to the formulation of VQ when customers have equal demand. The major difference is in the function used to measure the distance between the input and codebook vectors: average distances between codebooks (facilities) and class of input vectors (customer locations for each customer) are used. We therefore set the winner facility c as the one with the smallest average distance, namely

$$c = \arg \min_{i=1,\ldots,m} \left\{ \frac{1}{|S_j|} \sum_{\mathbf{a}_j \in S_j} d\left(\mathbf{x}_i, \mathbf{a}_j\right) \right\}. \tag{7.16}$$

Also the update rule (7.12) becomes

$$\begin{aligned} \mathbf{x}_c\left(t+1\right) &= \mathbf{x}_c\left(t\right) - \alpha\left(t\right) \tfrac{1}{|S_j|} \sum_{\mathbf{a}_j \in S_j} \nabla_{\mathbf{x}} d\left(\mathbf{x}_c\left(t\right), \mathbf{a}_j\right) \\ \mathbf{x}_i\left(t+1\right) &= \mathbf{x}_i\left(t\right) \quad i \neq c \end{aligned}, \tag{7.17}$$

by taking the gradient of the average distance, instead of the distance between the winner facility and customer j.

When customers have different demands, we can apply the earlier strategy: instead of presenting the input vectors (customer coordinates) an equal number of times to the neural network, samples belonging to customers with higher demand values can be input more frequently.

7.3. Computational Experiments

In this section we evaluate the performance of the new VQ methods. We would like to remind that in the case of unlabeled data PMFWP and MFWP become equivalent and we can say that VQ is very successful based on the results reported in Chapter 4. We therefore direct any interested reader to that work for an assessment of the performance of the new VQ algorithms for the PMFWP with unlabeled customer location data and focus on the labeled case.

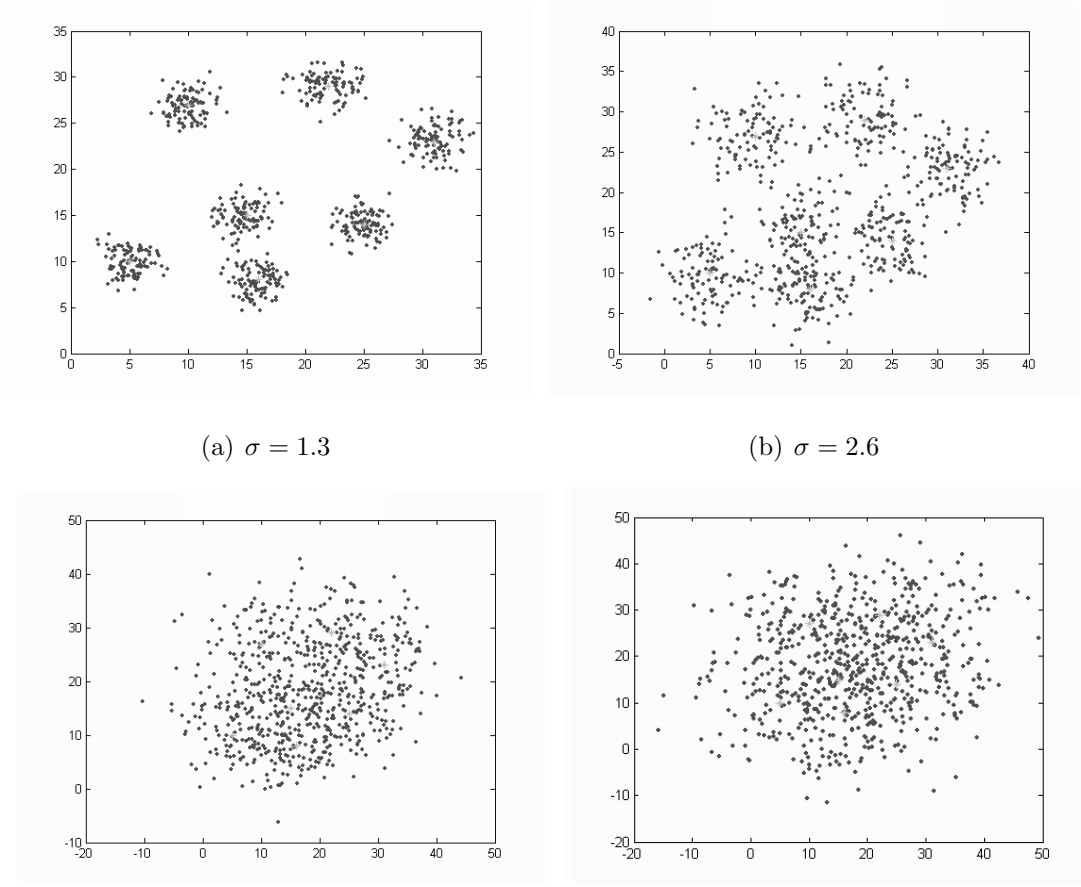
7.3.1. Test Bed

Since there is no available standard test problem library for the PMFWP, we again modify a selected subset of commonly used Euclidean and rectilinear instances in the deterministic facility location literature in order to have comparable results on a test bed. They are listed in the first column of Table 7.1. The values in the remaining columns are respectively the number of customers, n, the number of facilities to locate, m and finally the source of the test problem. In their original form, customer demands and two-dimensional coordinates are given for these instances. Customers in Cooper's [24], Eilon et al.'s [32] and Rosing's [86] data sets have unit demands, which is not the case for the remaining instances.

Table 7.1. Facility location data sets used in this research

Small Instances	n	m	**Source**
`cooper7` (7 problems)	7	2	Cooper [24]
`lj20/100`	20	2–10	subset of `LJ100`
`lj40/100`	40	2–10	subset of `LJ100`
`ros10`	10	2	Rosing [86]
`ros15`	15	2	Rosing
`ros25`	25	2–10	Rosing
`ros30`	30	2–10	Rosing
`eil50`	50	2–10	Eilon et al. [32]
`lj100`	100	2–10	Love and Juel [63]
`lj60/100`	60	2–10	subset of `lj100`
`ljJ80/100`	80	2–10	subset of `lj100`

In order to make these data sets suitable for the PMFWP we assume that customer locations follow bivariate normal distributions with means set to the given coordinate values in the deterministic data sets. To satisfy the assumption of our probabilistic model, the variances along both dimensions are set to the same value. As a matter of fact, we set this value to 5%, 10%, 20%, 30% and 40% of the maximum range of the given coordinate values for each problem instance. Clearly, the higher is the variance, the more the problem differs from the deterministic version. A scatter plot showing



(a) $\sigma = 1.3$ (b) $\sigma = 2.6$

(c) $\sigma = 5.2$ (d) $\sigma = 7.8$

Figure 7.1. The effect of variance: symmetric bivariate normal distribution

customer distributions for different values of variance parameter on `cooper7A` data set is given in Figure 7.1, where 100 sample location vectors ($\bullet$) are generated with seven mean location vectors ($*$).

7.3.2. Results on small instances

We started our experimentation with very small instances to see how VQ heuristics work. Since there is no optimally solved test instance available in the literature, we had to create reference values to judge the accuracy of the new methods. These are the seven instances generated based on Cooper's test problems and two instances based on `ros10` and `ros15` problems. For this purpose, we first enumerated all possible facility allocations, and then solved corresponding probabilistic Weber problems (i.e. minimization of (5.11) for every facility) for each of the allocations by using the steepest descent method explained in Section 5.3. One can surely calculate the global optimum

of the problem by using this brute force approach. Unfortunately, this is only possible for small instances since the number of possible facility allocations is exponential.

Results are reported in Table 7.2. We estimate $E[d(\mathbf{x}_i, \mathbf{a}_j)]$ by using average distances and set the variance parameter to 10% of the maximum range of customer locations. We employed three different sample sizes while generating customer locations: 10, 100 and 1000. VQ's performance is expressed as a percent deviation from the optimal value given as $100 \times (z_{VQ} - z^*)/z^*$ where z^* is the optimal objective value, and z_{VQ} is the objective value calculated by a VQ heuristic. Each instance is solved 10 times by the VQ heuristics. Hence, the average and the best values for each method denote the average and the best objective function values of 10 runs. Each run of VQ starts with a randomly generated initial facility locations.

Table 7.2. Percent deviations of VQ results from the optimal – Small instances

Data set	**m**	**Optimal**	**VQ for PMFWP**					
			10 Samples		**100 Samples**		**1000 Samples**	
			Avg.	**Best**	**Avg.**	**Best**	**Avg.**	**Best**
cooper7A	2	56.5023	2.48	0.35	0.14	0.01	0.08	0.00
cooper7B	2	79.7259	3.24	0.13	1.05	0.01	0.00	0.00
cooper7C	2	47.7226	1.15	0.18	0.10	0.02	0.00	0.00
cooper7D	2	53.0093	0.49	0.11	0.04	0.01	0.00	0.00
cooper7E	2	43.5922	2.00	0.13	0.06	0.01	0.00	0.00
cooper7F	2	68.4067	1.06	0.45	1.03	0.02	1.04	0.00
cooper7G	2	71.0697	3.83	0.10	2.44	0.01	2.38	0.00
ros10	2	115.4125	0.51	0.05	0.05	0.03	0.00	0.00
ros15	2	234.905	1.09	0.08	0.06	0.00	0.00	0.00
Average			1.76	0.18	0.55	0.01	0.39	0.00

Regarding the results of Table 7.2, we first remark that the performance of VQ increases as the sample size increases, which is expected. The last row of Table 7.2 shows the average quality of the solutions obtained by VQ. Based on these values, we can conclude that VQ is able to find the optimal value at least once in one of the

10 runs for the 1000-sample case. We should point out that a reasonable amount of computational time is required to obtain the results.

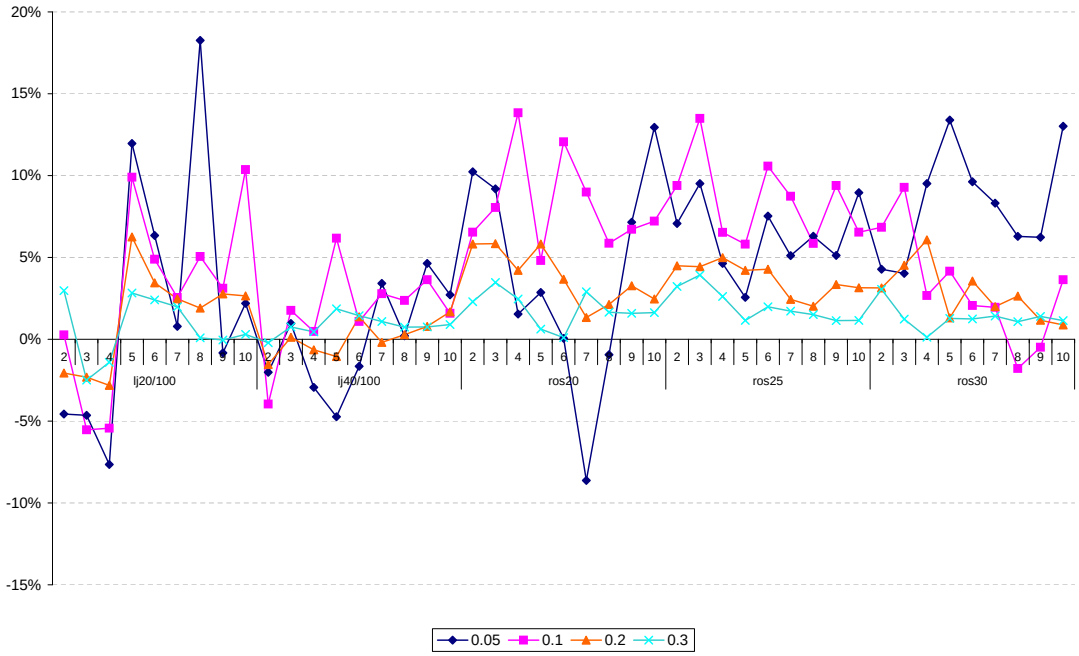
7.3.3. Results on medium and large instances

In the following we assess the performance of VQ on problem instances which are too large to be solved by brute force approach. Therefore we calculate a good solution of the PMFWP using PALA to be used as a reference for assessing the performance of VQ heuristics. Both algorithms start from the same initial facility locations. As mentioned previously, we implement both algorithms for labeled data.

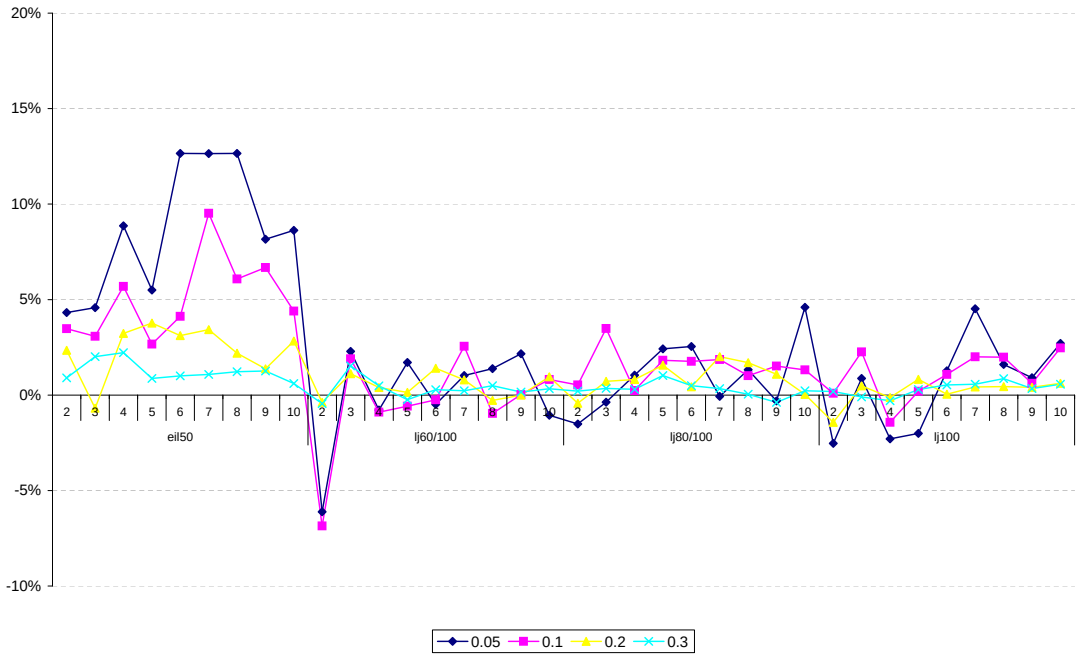
In our experiments we run each heuristic for 10 times and report the average objective value of these 10 runs. The exact objective value given as the analytical expression (5.10) is used to determine the minimum and average of these 10 values. Namely, we use (5.10) as the metric to compare heuristics performances, and evaluate it with the locations calculated with an heuristic. The instances are categorized as medium and large based on the number of customers. An instance is considered to be medium when $n \leq 50$, otherwise it is designated as large. As the results of Table 7.2 suggest that generation of 100 samples for approximating $E[d(\mathbf{x}_i, \mathbf{a}_j)]$ seems satisfactory, we only present results for 100 samples.

First, we focus on the average performance gain of VQ over PALA when each instance solved 10 times. In Figure 7.2 we compare the results for small instances in terms of percent improvement over the average objective value of PALA, i.e., $100 \times (z_{\text{PALA}} - z_{\text{VQ}})/z_{\text{PALA}}$. The labels on the x-axis are the names of the data set and the number of facilities to be located. Results obtained for different values of the variance parameter are also plotted in the same figure to illustrate the effect of the variance; there are four plots corresponding to 5, 10, 20 and 30% of the maximum coordinate range.

Figure 7.2 reveals that there is an improvement for most of the instances, which implies the superiority of VQ over PALA. For some of the instances, e.g. `lj20/100`



(a) `lj20/100`, `lj40/100`, `ros20`, `ros25` and `ros30` data sets



(b) `eil50`, `lj60`, `lj80` and `lj100` data sets

Figure 7.2. Comparison of VQ and PALA for labeled data

and `lj40/100`, the percent improvements become as high as 20%. The improvements are particularly noticeable when the number of facilities increases for the same data set, i.e, the same mean and variance of customer locations and demands. When the variance increases, the performance gain decays and VQ and PALA results overlap. In summary, we have carried out a total of 4050 runs for labeled problems. In 2882 out of 4050 values, VQ resulted in lower final objective value than PALA (71.2%).

The average improvements by VQ over all instances versus the variance parameters set to 5, 10, 20 and 30% are found to be 3.47, 3.51, 1.75 and 1.00% respectively (Figure 7.3). Hence, we can say that the performance of PALA approaches to the performance of VQ as the variance of customer locations increases.

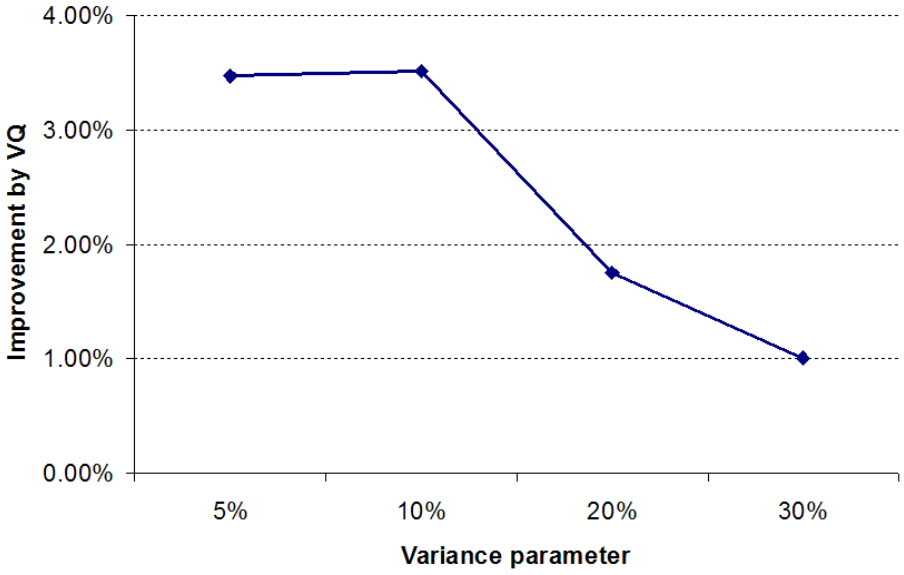


Figure 7.3. VQ vs. PALA: the effect of increasing variance

7.3.4. Value of information

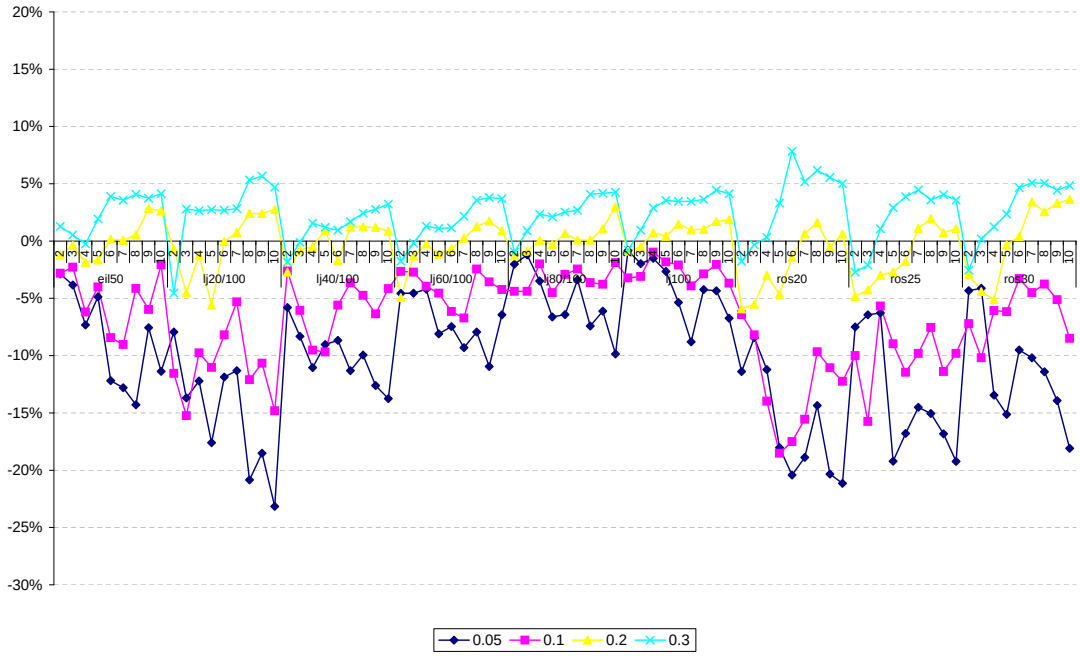
Now we turn our attention to the modelling properties we impose for analyzing the PMFWP, and consider the value generated by using the labeling information. To this end, we are solving labeled and unlabeled versions of a problem instance and compare labeled results with the unlabeled ones. As for the previous stages of the experimentation, we present our results attained by the heuristics over 10 runs.

Percent improvements in the average objective values, computed using the comparison metric (5.10), obtained with PALA for labeled data over PALA for unlabeled data are illustrated in Figure 7.4(a), which is replicated for VQ in Figure 7.4(b). The percent deviation used in this experimentation is defined as $100\times(z_{\texttt{UNLABELED}} - z_{\texttt{LABELED}})/z_{\texttt{LABELED}}$, which represents the value of the labeling information. A negative value represents that solution obtained with unlabeled data gives a lower objective function value, compared to the solution obtained with labeled data. Again, each plot is for a different variance value.

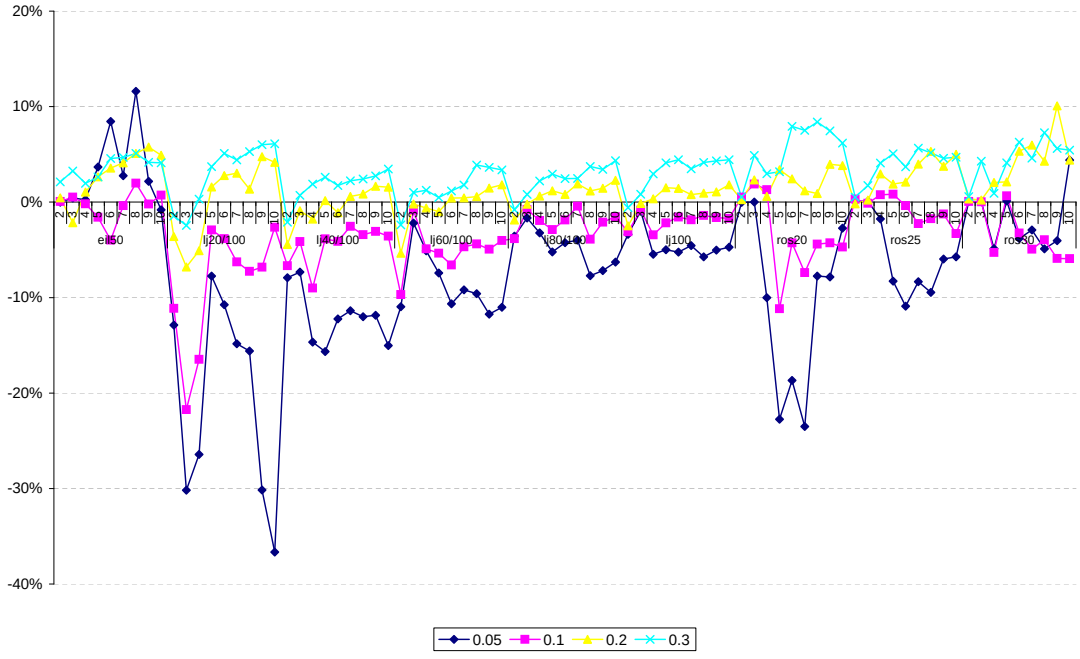
When we consider the results in Figure 7.4(a) or Figure 7.4(b), we remark that percent deviation is mostly negative for lower variance (all observations, except `eil50` with VQ heuristic) whenever the variance parameter is set to 5 or 10% of the maximum coordinate range. It is observed to be as low as -23.16% for `lj20/100` when 10 facilities are considered with PALA. On the other hand, it reaches to -36.65% with VQ for the same settings. However, as the variance increases, the problem becomes more nondeterministic, and the label information becomes more valuable. This is true for both of the algorithms and when the variance is set to 30% of the maximum range of a data set, information value can go up to 10% (Figure 7.4(b), `ros20`, 9 facilities). For instance, when the variance parameter is set to 30%, deviation for `ros20` with 8 facilities is 8.4% and 6.2% with VQ and PALA heuristics respectively.

Hence we conclude according to these results that, the value of label information increases with increasing variance in customer locations. We should also remark that the value of label information increases with the number of facilities for higher variance. When variance is set to 30% of the maximum coordinate range, the slope of the deviation is positive for both PALA and VQ heuristics (Figure 7.4).

Finally, in Figure 7.5, we compare PALA and VQ heuristics with respect to the value of label information. On the figure, average percent value gains with PALA and VQ for different values of the variance are plotted. As can be observed, VQ algorithm outperforms PALA on the average, especially for increasing values of the variance. However, PALA catches up VQ with increasing variance. Based on the



(a) PALA



(b) VQ

Figure 7.4. Value of label information

experimentations, when variance parameter is set to 10%, VQ is 2.95% better than PALA on the average. This superiority decreases down to 1.46, 0.67 and 0.25% when the variance is set respectively to 10, 20 and 30% of the maximum coordinate range.

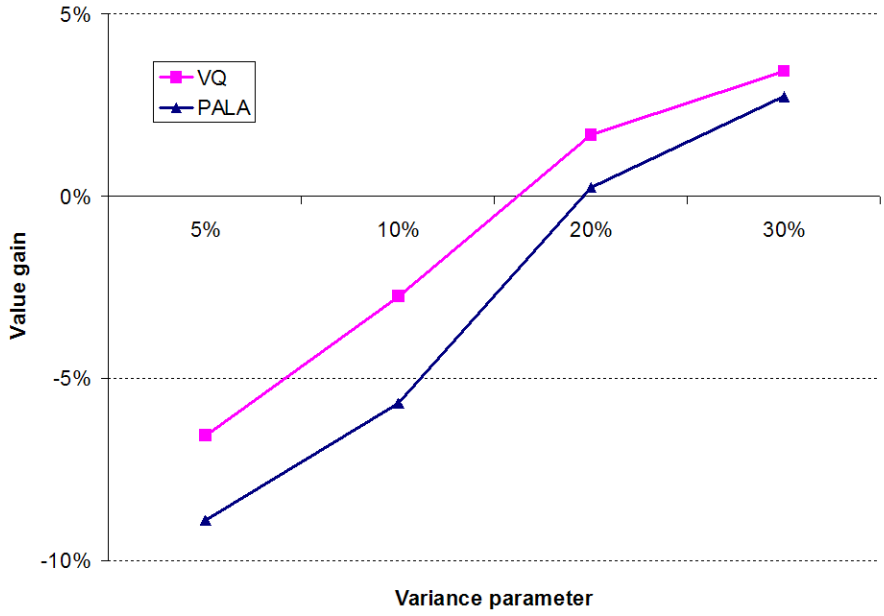


Figure 7.5. Comparison of algorithms according to their profit of label information

8. CONCLUSIONS

In this research we have investigated the relation between Kohonen type competitive learning methods and a class of location–allocation problems. We proposed methods based on RVQ and RSOM, which are variants of standard vector quantization and self-organizing map approaches with the difference that RVQ and RSOM employ distance function information in their winner selection and update rules.

We applied these variants first in solving rectilinear and Euclidean multi-facility Weber problems (MFWPs). Based on the experimental results obtained on benchmark instances we have drawn the conclusion that when dealing with the rectilinear MFWP, RVQ and RSOM obviates the need for a guided refining procedure. Moreover, RSOM can yield solutions of similar quality with much less computational effort. Additionally, the standard SOM approach, which can be employed for solving the squared Euclidean MFWP can produce inferior results when used for solving the Euclidean MFWP. As a consequence, a vector quantization or self-organizing map based method solving the MFWP can provide better results and eliminate the need for a possibly expensive final refinement procedure if it is tailored (i.e., their winner selection and update rules are modified) to the distance function used in the objective function of the problem. On different benchmark instances it is possible to see the declining performance of RVQ and RSOM as the problem size increases.

Then we have considered the probabilistic multi-facility Weber problem. It is a more realistic extension of the deterministic multi-facility Weber problem where customer coordinates are distributed according to some multivariate probability distribution. We have proposed new heuristic solution procedures. One of them is a local search method and generalizes the alternate location-allocation heuristic originally suggested for the deterministic problem. Others incorporate this local search heuristic within a variable neighborhood search scheme and improves its capabilities through different neighborhood structures and search strategies.

The existence of a closed form expression for the expected distance is crucial in the use of these methods. This depends on the form of the distance function and bivariate probability distribution and unfortunately it is possible for only very few cases. We have therefore suggested two approximation strategies; they can be used for any distance function and customer location distribution. They give very good results, which are even better than the ones of exact evaluation in some cases due to the implicit uphill move capabilities they introduce.

We also investigated new heuristics using the principles of VQ for solving probabilistic version of the uncapacitated multi-facility Weber problem. Having theoretical advantage over scalar quantization which enables exploitation of the dependencies between neighboring samples, vector quantization is applied for the first time to this class of problems and based on the computational experiments, it gave comparable or better results than allocation–location type heuristics. From the computational point of view, fast parallel implementations of VQ are also promising for larger problems.

We also note that algorithms proposed in this research do not require a specific probability distribution since the expectation is approximated either using average distances or an approximation scheme based on statistical information about input distribution. Hence we do not need to use computationally demanding integral evaluations within the updates. Modification is necessary only if the problem is defined for a specific metric function. In such a case the winner selection rule and the update equations of VQ are modified. Moreover, VQ type algorithms are better in incorporating the information than the PALA algorithms.

We noted that similar to the deterministic case of the problem, it is possible to see the declining performance of VQ as the problem size increases on different benchmark instances of the probabilistic multi–facility Weber problem. This occurs particularly when there is a large number of facilities to be located, which creates many local optimal solutions for the problem to be solved, both in deterministic and the probabilistic scenarios.

REFERENCES

1. Abramowitz, M., and I.A. Stegun (editors), *Handbook of mathematical functions*, Applied Mathematics Series, National Bureau of Standards, 1968.

2. Alpaydın, E., İ.K. Altınel, and N. Aras, "Parametric distance functions versus nonparametric neural networks for estimating road travel distances", *European Journal of Operational Research* **92**: 230–243, 1996.

3. Altınel, İ.K., J. Oommen, and N. Aras, "Vector quantization for arbitrary distance function estimation.", *INFORMS Journal on Computing* **9**: 439–451, 1998.

4. Altınel İ.K., N. Aras, and B.J. Oommen, "Fast, efficient and accurate solutions to the Hamiltonian path problem using neural approaches", *Comput Operations Research* **27:** 461–494, 2000.

5. Aly, A.A. and J.A. White, "Probabilistic formulations of the multifacility Weber problem", *Naval Research Logistics Quarterly* **25** 531–547, 1978.

6. Angéniol, B., C. Vaubois and J.Y. LeTexier, "Self-organizing feature maps and the travelling salesman problem", *Neural Networks* **1**: 289–293, 1988.

7. Aras, N., B.J. Oommen, and İ.K. Altınel, "The Kohonen network incorporating explicit statistics and its application to the travelling salesman problem", *Neural Networks* **12**: 1273–1284, 1999.

8. Aras, N., İ.K. Altınel, and B.J. Oommen, "A Kohonen-like decomposition method for the Euclidean traveling salesman problem—KNIES_Decompose", *IEEE Transactions on Neural Networks* **14**: 869–890, 2003.

9. Bauer H.U. and K. Pawelzik. , "Quantifying the neighborhood preservation of selforganizing feature maps", *IEEE Transactions on Neural Networks* 3(4):570-

579, 1992.

10. Bishop, C., *Neural Networks for Pattern Recognition* Oxford University Press, 1998

11. Bongartz, I., P.H. Calamai, and A.R. Conn, "A projection method for l_p norm location-allocation problems", *Math Program* **66**: 283–312, 1994.

12. Bottou L. and Y. Bengio, "Convergence Properties of the K-Means Algorithms", *Advances in Neural Information Processing Systems*, The MIT Press , 7, p.585-592, 1995.

13. Brimberg J., P. Hansen, N. Mladenovic, and E.D. Taillard ,"Improvements and comparison of heuristics for solving the uncapacitated multisource weber problem", *Operations Research* **48**: 444–460, 2000.

14. Brimberg, J., R. Chen, and D. Chen, "Accelarating convergence in the Fermat–Weber location problem", *Operations Research Letters* **22** 151–157, 1998.

15. Brimberg, J., P. Hansen, N. Mladenovic, and T.D. Taillard, "Improvements and comparison of heuristics for solving the uncapacitated multisource Weber problem'", *Operations Research* **48** 444–460, 2000.

16. Brimberg J., and R.F. Love ,"Estimating distances in facility location: survey and applications" in Drezner Z (ed), *Facility Location: A Survey of Applications and Methods*. Springer Series in Operations Research: New York, pp. 9–32, 1995.

17. Brimberg J., and R.F. Love, "Global convergence of a generalized iterative procedure for the minisum location problem with l_p distances", *Operations Research* **41**: 1153–1163, 1993.

18. Brimberg J., and N. Mladenovic, "Solving the continuous location-allocation problem with tabu search", *Stud Loc Anal* **8**: 23–32, 1996.

19. Brimberg J., and N. Mladenovic, "A variable neighborhood algorithm for solving

the continuous location-allocation problem", *Stud Loc Anal* **10**: 1–12, 1996.

20. Charnes, A. and W. W. Cooper, "Chance constrained programming" *Management Sci.*, 6, 7379. 1959

21. Charon I., O. Hudry, "The Noising Methods: A Generalization of Some Metaheuristics", *European Journal of Operational Research* **135** (2001)86–101, 2001.

22. Hansen, P. and B. Jaumard, "Cluster Analysis and Mathematical Programming", *Mathematical Programming*, 79, 191-215, 1997.

23. Conn, A. R. and G. Cornuéjols, "A projection method for the uncapacitated facility location problem", *Mathematical Programming*, V46, 273298, 1990.

24. Cooper, L., "Location-allocation problems", *Operations Research* **11**: 331–343, 1963.

25. Cooper, L., "Heuristic methods for location-allocation problems", *SIAM Review* **6** 37–53, 1964.

26. Cooper, L., "Solutions of generalized location equilibrium models", *J Regional Sci* **7:** 1–18, 1967.

27. Cooper, L., "The transportation location problem", *Operations Research* **20** (1972)94–108, 1972.

28. Cooper, L., "A random locational equilibrium problem", *Journal of Regional Sciences* **14** (1974)47–54, 1974.

29. Cooper, L., "The stochatic transportation-location problem", *Computational Mathmatics with Applications* **4** (1978)265–275., 1978.

30. Drezner, T. and Z. Drezner, "Retail facility location under changing market conditions" IMA Journal of Management Mathematics 13(4):283-302, 2002.

31. du Merle, O., D. Villeneuve, J. Desrosiers, and P. Hansen, *Stabilization dans le cadre de la génération de colonnes*, Les Cahiers du GERAD, University of Montreal, 1997.

32. Eilon, S., C.D.T. Watson-Gandy, and N. Christofides, *Distribution Management: Mathematical modelling and practical analysis*, Charles Griffin & Company Limited London, 1973.

33. Frenk J.B.G., M.T. Melo, and S. Zhang, "A Weiszfeld method for a generalized l_p distance minisum location model in continuous space", *Location Science* **2** 111–117, 1994.

34. Fritzke, B. "Growing cell structures – a self-organizing network for unsupervised and supervised learning.", *Neural Networks*, 7 (9), 1441–1460, 1994.

35. Fritzke, B. "Fast learning with incremental RBF networks", Neural Processing Letters, 1:2-5, 1994.

36. Fritzke, B. , "A growing neural gas network learns topologies", in G. Tesauro, D. S. Touretzky, and T. K. Leen, (eds.), *Advances in Neural Information Processing Systems*, p.625-632. MIT Press, Cambridge MA, 1995.

37. Fritzke, B. *Some competitive learning methods*, http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/JavaPaper/

38. Gamal M.D.H., and S. Salhi, "A cellular heuristic for the multisource Weber problem", *Computers and Operations Research* **30**: 1609–1624, 2003.

39. Hansen, P., and N. Mladenović, "Variable neigborhood search", in F. Glover, G. Kochenberger (eds.), *Handbook of Metaheuristics*, 145–184, 2003.

40. Gray, R. M. "Vector quantization", *IEEE ASSP Magazine*, 1:4-29, 1984.

41. Hansen, P., B. Jaumard, S. Krau, and O. du Merle, *A column generation algo-*

rithm for the multisource Weber problem, Les Cahiers du GERAD, University of Montreal, 1997.

42. Hansen, P., N. Mladenovic, and E.D. Taillard, "Heuristic solution of the multisource Weber problem as a p-median problem" *Operations Research Letters* **22**: 55–62, 1998.

43. Hansen, P., and N. Maldenović, "Variable neighborhood search: Principles and applications", *European Journal of Operational Research* **130** 449–467, 2001.

44. Hansen, P., and N. Mladenovic, "Variable Neighbourhood Search", in P. Pardalos and M. Resende (eds.), Handbook of Applied Optimization, pp. 221-234, Oxford University Press, New York, 2002.

45. Honelka, T., S. Kaski, T. Kohonen, and K. Lagus, "Self-Organizing Maps of Very Large Document Collections" *Classification, Data Analysis, and Data Highways*, Proc. 21st Ann. Conf. of the GFKL, Springer, p. 245-252, 1998

46. Houck, C.R., J.A. Joines, and M.G. Kay, "Comparison of genetic algorithms, random restart and two-opt switching for solving large location-allocation problems. *Computers and Operations Research* **23:** 587–596, 1996.

47. Hsieh, K.H., and F.C. Tien, "Self-organizing feature maps for solving location-allocation problems with rectilinear distances", *Computers and Operations Research* **31**: 1017–1031, 2004.

48. Hurter A. P., M. K. Schaefer and R.E. Wendell, "Solutions of Constrained Location Problems, *Management Science*, 22, 51-56, 1975.

49. Jain, A. K., M. N. Murty and P. J. Flynn "Data clustering: a review" *ACM Computing Surveys* Volume 31 , Issue 3 P264 – 323

50. Kaski S. and J. Sinkkonen, "Principle of learning metrics for data analysis", The Journal of VLSI Signal *Processing-Systems for Signal, Image, and Video Technol-*

ogy, special issue on Data Mining and Biomedical Applications of Neural Networks, (to appear).

51. Katz, I. N., and L. Cooper, "Facility Locations in the Presence of Forbidden Regions", *European Journal of Operations Research*, 6, 166-173, 1981.

52. Katz, I.N. , L. Cooper, "An always convergent numerical scheme for a random locational equilibrium problem", *SIAM Journal of Numerical Analysis* **11** 683–692, 1974.

53. Katz, I.N., L. Cooper, "Normally and exponentially distributed locational equilibrium problems", *Journal of Research of the National Bureau of Standards* Section:B 53–73, 1976.

54. Kohonen, T., *Self-Organizing Maps* Berlin: Springer-Verlag, 1995

55. Kohonen, T., "The Self-Organizing Map", *Proceedings of the IEEE*, Vol. 78, No. 9, pp. 1464-1478, 1990.

56. Kohonen, T., "Self-Organizing Maps: Optimization Approaches", in T. Kohonen, K. Makisara, O. Simula and J. Kangas (Eds.), *Artificial Neural Networks*, pp. 981-990, Elsevier Science Puplishers, 1991.

57. Köerkel, M. "On the exact solution of large-scale simple plant location problems" *European Journal of Operational Research*, Vol. 39, no. 2, pp. 157–173, 1989.

58. Krau S. *Extensions du problems de Weber*. PhD Thesis, Ecole Polytechnique de Montreal, Canada, 1997.

59. Cherkassky, V., and F. Mulier, *Learning From Data* , John Wiley and Sons, 1998.

60. Levin Y. and Adi Ben-Israel, "A Heuristic Method for Large-Scale Multi-Facility Location Problems", Computers and Operations Research, 31, p.257-272, 2004

61. Linde, Y., A. Buzo, and R. M. Gray. "An algorithm for vector quantizer design", *IEEE Transactions on Communication*, COM-28, p.84-95, 1980.

62. Liu C.M., R.L. Kao and A.H. Wang, "Solving location-allocation problems with rectilinear distances by simulated annealing", *J Opl Res Soc* **45**: 1304–1315, 1994.

63. Love, R.F., and H. Juel, Properties and solution methods for large location-allocation problems, *Journal of the Operational Research Society* **33** 443–452, 1982.

64. Love, R.F. and J.G. Morris, "Modeling inter-city road distances by mathematical functions," *Operational Research Quarterly*, 23, 61–71, 1972.

65. Love RF and J.G. Morris, "A computation procedure for the exact solution of location-allocation problems with rectangular distances", *Nav Res Logist Q* **22**: 441–453, 1975.

66. Love, R.F. , and J.G. Morris, "Mathematical models of road travel distances", *Management Sciences* **25**: 130–139, 1979.

67. Lozano, S., F. Guerrero, L. Onieva and J. Larraneta, "Kohonen maps for solving a class of location-allocation problems", *European Journal of Operational Research* **108**: 106–117, 1998.

68. Luenberger, D. G., "Linear and Nonlinear Programming" , Addison-Wesley, 1984.

69. MacQueen, J. "Some methods for classification and analysis of multivariate observations", *Proceedings of the Fifth Berkeley Symposium on Mathematical statistics and probability*, p.281-297, Berkeley, 1967.

70. Mangiamelli, P., S. K. Chen, and D. West, "A comparison of SOM neural network and hierarchical clustering methods", *European Journal of Operational Research* **93**: 402–417, 1996.

71. Martinetz, T., and K. Schulten "A Neural Gas Network Learns Topologies", *Arti-*

ficial Neural Networks Proceedings of the ICANN91. Elsevier Science

72. Megiddo, N., and K.J. Supowit, "On the complexity of some common geometric location problems", *SIAM Journal on Computing* **13** 182–196, 1984.

73. Mirkin, B. G. *Mathematical classification and clustering* , Boston - Kluwer Academic Pubishers, 1996.

74. Modares, A., S. Somhom and T. Enkawa, "A self-organizing neural network approach for multiple traveling salesman and vehicle routing problems", *Int T Opl Res* **6**: 591–606, 1999.

75. Morris, W.T. "On the art of modeling", Management Science, 13 707–717 1967

76. Özkısacık, K. "A Self Organizing Map Algorithm in the General Metric", Ms. Thesis, Boğaziçi University, Istanbul.

77. Oommen, J., I.K. Altinel, and N. Aras, "Discrete vector quantization for arbitrary distance function estimations", *IEEE Trans. Systems, Man, and Cybernetics–Part B: Cybernetics* **28**: 496–510, 1998.

78. Osman, I.H., and G. Laporte, "Metaheuristics: A bibliography. Ann. Oper. Res. 63, 513628, 1996.

79. Owen, S. H. and M.S. Daskin,"Strategic Facility Location: A Review", *European Journal of Operational Research*, Vol. 111, pp. 423-447, 1998.

80. Pal, N.R., J. C. Bezdek, and E.C.K. Tsao, "Generalized clustering networks and Kohonen's self-organizing scheme", *IEEE Transactions on Neural Networks* **4**: 549–557, 1993.

81. Preparata, F.P., and M.I. Shamos, "Computational geometry: an introduction" - Springer-Verlag New York, USA, 1985.

82. Rao, J.R. , N.H. Varma, "Stochastic multi-facility minisum location problem involving Euclidean distances", *OPSEARCH* **22** 232–240, 1985.

83. Reeves, C. (editor), *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell,Oxford, 1993.

84. Reinelt, G., "TSLIB—a traveling salesman library", *ORSA J. on Computing* **3** 376–384, 1991.

85. Robbins, H. and S. Monroe, "A Stochastic Approximation Method", *Annuaire Mathematics and Statistics*, 22, p.400-407, 1951

86. Rosing, K.E., "An optimal method for solving the (generalized) multi-Weber problem", *European Journal of Operational Research* **58**: 414–426, 1992.

87. Sengupta J.K. and J.H. Portillo-Campbell,"A Fractile Approach to Linear Programming Under Risk", *Management Science*, 16, 298-308, 1970.

88. Seppala Y. "On a Stochastic Multi-Facility Location Problem", *AIIE Transactions7*, 56-62, 1975.

89. Kaski, S., J. Kangas and T. Kohonen, *Bibliography of Self-Organizing Map (SOM) Papers*, http://www.cis.hut.fi/research/refs/, 1998.

90. Soylu M, N.E. Özdemirel and S. Kayalıgil, A self-organizing neural network approach for the single AGV routing problem. *European Journal of Operational Research* **121**: 24–137, 2000.

91. Tan, T., R. Güllü, N. Erkip, "Modelling imperfect advance demand information and analysis of optimal inventory policies", *European Journal of Operational Research*, 2006, (To appear).

92. UCI Repository, http://www.ics.uci.edu/ ˜mlearn/mlrepository.html, (last accessed May, 2004).

93. Villmann, T. , R. Der, M. Herrmann, and T. Martinetz. "Topology presevation in self-organizing feature maps: exact definition and measurement", IEEE TAW, 1994.

94. Watson-Gandy, C.D.T, "The Solution of Distance Constrained Mini-Sum Location Problem", *Operations Research*, 33, 784-802, 1985.

95. Weiszfeld E, "Sur le point lequel la somme des distances de n points donné est minimum", *Tôhoku Math. Journal* **43**: 355–386, 1937.

96. Wendell, R.E. and A.P. Hurter, "Location theory, dominance and convexity", *Operations Research* **21**: 314–320, 1973.

97. G.O. Wesolowsky, "The Weber problem with rectangular distances and randomly distributed destinations", *Journal of Regional Sciences* **17** 53–60, 1977.

98. Wesolowsky, G., "The Weber problem: history and perspectives", *Location Sci* **1**: 5–23, 1993.

99. Xing, E.P, Ng, A.Y., Jordan, M.I. & Russell, S., "Distance metric learning with applications to clustering with side information", *Proc. Advances in Neural Information Processing Systems, NIPS'03.*

100. Xing, E.P., http://www.cs.berkeley.edu/~epxing/paper/code/_metric/_online.tar.gz., (last accessed May, 2004).

101. Zador, P.L., "Asymptotic quantization error of continuous signals and the quantization dimension", *IEEE Trans. Information Theory* **28**: 139–149, 1982.

102. Zhang Z, J. T. Kwok, and D.Y. Yeung, "Parametric Distance Metric Learning With Label Information", Technical Report HKUST-CS03-02 January 18, 2003.