Chapter 5

Learning Variable Quantisation Thresholds

The research presented in this Chapter has been previously published in Moran et al. (2013b).

5.1 Introduction

In Chapter 4, I introduced a new multi-threshold quantisation algorithm for hashing-based approximate nearest neighbour (ANN) search. This semi-supervised quantisation model optimised the position of one or more thresholds along each projected dimension by fusing affinity information on data-point pairs arising from both the high dimensional input feature space and the lower-dimensional projected feature space. The intuitive objective of the algorithm was to position the quantisation thresholds so that the projections for related data-points end up in the same thresholded regions and the projections of dissimilar data-points fall within different regions. Each thresholded region was associated with a unique bitcode from a single or multi-bit binary codebook. To quantise the projection of a data-point I compared the projected value to the quantisation thresholds to pinpoint the appropriate thresholded region and assigned the corresponding bitcode of that region to the data-point. By repeating this procedure for the remaining projected dimensions I was able to build up a K-bit binary hashcode for each of the data-points. The experimental analysis demonstrated that a significant increase in retrieval effectiveness could be achieved from both optimising the positioning of the thresholds to preserve as many must-link and cannot-link relationships between the data-points as possible, and additionally for certain projection functions (such as

PCA), allocating more than a single threshold per projected dimension. The experimental results also suggested that the optimal threshold allocation ($T \in \mathbb{Z}_+$) *varied* according to the specific hash function responsible for generating the low-dimensional projections. For example, LSH was generally shown to prefer a single threshold allocation ($T = 1$) to each projected dimension whereas PCA projections benefited significantly from a multiple-threshold allocation in which, for example, three thresholds ($T = 3$) were allocated to each dimension.

In the previous chapter the quantity of thresholds T allocated per projected dimension was decided *a-priori* and remained the same for all K projected dimensions arising from a specific projection function. In this chapter I expand upon the finding that the threshold allocation is projection function specific by relaxing assumption A_2 outlined in Chapter 1. To this end I argue that the allocation of thresholds should *vary per projected dimension*, rather than being kept uniform across projected dimensions. I further contend that the optimal variable allocation can effectively be found by computing a measure of the *neighbourhood preserving quality* of each projection and assigning more thresholds to those projected dimensions that better conserve the pairwise relationship between data-points in the low-dimensional projected space. In other words I argue that retrieval effectiveness can be further increased by learning a variable allocation of thresholds $T_k \in \mathbb{Z}_+$ for each projected dimension $\mathbf{y}^k \in \mathbb{R}^N$ where the allocation is informed by the quality of the projection. In this chapter I keep with the general theme of this thesis and advocate a data-driven approach to learning a variable threshold allocation subject to a specified threshold budget. To the best of my knowledge the research described in this chapter was the first to introduce and formulate the variable quantisation threshold learning problem in the context of hashing-based ANN search (Moran et al. (2013b)).

The remainder of this chapter is structured as follows: in Section 5.2 I introduce two data-driven algorithms that learn a variable allocation of quantisation thresholds across projected dimensions for a specific projection function. In Section 5.3 I evaluate the two variable threshold quantisation models on their ability to generate effective binary hashcodes for image retrieval. Finally, in Section 5.4 I summarise the main contributions of this chapter and present conclusions arising from the experimental evaluation.

5.2 Variable Quantisation Threshold Allocation

5.2.1 Problem Definition

The problem definition in this chapter is broadly similar to the definition given in the previous chapter (Chapter 4, Section 4.2.1) but with the additional requirement to learn an appropriate number of thresholds for each projected dimension. Concretely the objective in this chapter is to *learn* a set of thresholds $\mathbf{t}_k = [t_{k1}, t_{k2}, \ldots, t_{kT_k}]$ where $t_{ki} \in \mathbb{R}$ and $t_{k1} < t_{k2} \ldots < t_{kT_k}$ for each of the K projected dimensions $\left\{\mathbf{y}^k \in \mathbb{R}^N\right\}_{k=1}^K$ while also finding the optimal allocation of thresholds $\{T_k \in \mathbb{Z}_+\}_{k=1}^K$ to each of the K projected dimensions, subject to a total threshold budget $\sum_{k=1}^K \log_2(T_k+1) = K$ and $T_k \in \{0,1,3,7,15\}$. A threshold budget, or equivalently a bit budget, is required because we wish to extract the maximum retrieval effectiveness from as short a hashcode as possible. Short hashcodes ($<$ 128 bits) are particularly useful because they save both time and memory when retrieving nearest neighbours, as compared to much longer hashcodes ($\gg$ 128 bits) or the original high-dimensional feature vectors. In a similar manner to Chapter 4 the quality of the resulting quantisation will be judged by applying the computed hashcodes to the task of query-by-example image retrieval. This threshold allocation problem is computationally difficult given the upper bound of T_{max}^K possible allocations of thresholds, where $T_{max} \in \mathbb{Z}_+$ is the maximum number of thesholds that can be allocated to any given projected dimension. This space of threshold allocations is impossible to search exhaustively therefore necessitating the introduction of more efficient search algorithms to solve the variable threshold allocation problem. It is the specification and evaluation of these algorithms which forms the focus of Section 5.2.2.

5.2.2 Algorithms for Variable Threshold Allocation

In this section I introduce two novel algorithms for learning an allocation of thresholds for each projected dimension based on a numerical measure of the quality of a projected dimension. In Section 5.2.2.1, I describe how this novel quality measure is based on counting the number of pairwise constraints between data-points that are conserved in a projected dimension thresholded with T_k thresholds. This quality measure is the F_β-measure computed on the data-point adjacency matrix $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$ and constitutes a minor adaptation of my multi-threshold quantisation objective function first introduced in Chapter 4. Having defined and motivated this measure of projected

dimension quality I then introduce two algorithms, dubbed Variable Bit Quantisation (VBQ), for efficiently solving the combinatorial search problem of finding the optimal threshold allocation across projected dimensions. The first algorithm I propose is framed as a Binary Integer Linear Program (BILP) (Section 5.2.2.3) that seeks to allocate thresholds in such a way so as to maximise the cumulative F_β-measure across all projected dimensions subject to an upper bound on the total permissible number of thresholds that can be allocated. The BILP is solved using standard branch-and-bound search. My alternative threshold allocation algorithm is a greedy approach that reallocates quantisation thresholds (Section 5.2.2.4) from lower quality (i.e. low F_β-measure) projected dimensions to projected dimensions that are deemed to be of a higher quality (i.e. higher F_β-measure). This greedy algorithm has a similar effect to the branch-and-bound solution in seeking a threshold allocation which maximises the cumulative F_β-measure. I compare the effectiveness and efficiency of both threshold allocation algorithms in my experimental evaluation in Section 5.3.

5.2.2.1 Judging Projection Quality: F_β-Measure Scoring Function

In this section I argue that counting the number of true nearest neighbours that fall within the same thresholded regions (true positives, TPs), the number of non-nearest neighbours that fall within the same regions (false positives, FPs) and the number of true nearest neighbours that fall within different thresholded regions (false negatives, FNs) is an effective measure of projected dimension quality[1]. In a similar manner to the multi-threshold quantisation algorithm introduced in Chapter 4, the TPs, FPs and FNs are derived from data-point adjacency graph $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$, with $S_{ij} = 1$ if data-points $\mathbf{x}_i, \mathbf{x}_j$ are true nearest neighbours, and $S_{ij} = 0$ otherwise. By quality I refer to the *locality preserving* ability of a projected dimension which is simply the ability to faithfully preserve the neighbourhood relationship between the data-points. For example, if two data-points are close by in the original feature space then their projections should ideally remain within close proximity along the projected dimension, and vice-versa for more distant data-points. If the projections of nearest neighbours are close by then they are more likely to fall within the same thresholded region of the projected dimension and therefore receive the same bit(s). I contend that this hypothetical projected dimension should be deemed to be of high quality as the bits generated from

[1] An unsupervised measure such as variance would also be a possible candidate for measuring the quality of a projected dimension. My threshold allocation algorithms presented in Sections 5.2.2.3-5.2.2.4 could also conceivably be used with variance as the allocation signal. I leave investigation of variance to future work.

the dimension are likely to be similar for many nearest neighbours and dissimilar for non-nearest neighbours, exactly the criteria for building effective hashcodes for image retrieval.

I propose in this section to combine the TPs, FPs and FN counts using the well-known F_β-measure metric from the field of Information Retrieval (IR). I previously touched on the F_β-measure in Chapter 3, Section 3.6.1 in the context of evaluating unranked sets of retrieved images. The F_β-measure is the weighted harmonic mean of recall (R) and precision (P) (Rijsbergen (1979)) which I present again in Equation 5.1 for reading convenience

$$\begin{aligned} F_\beta &= \frac{(1+\beta^2)PR}{\beta^2 P + R} \\ &= \frac{(1+\beta^2)TP}{(1+\beta^2)TP + \beta^2 FN + FP} \end{aligned} \tag{5.1}$$

The parameter $\beta \in \mathbb{R}_+$ specifies the contribution from the precision and recall. Setting $\beta < 1$ in Equation 5.1 weights precision higher than recall, and vice-versa for for a setting of $\beta > 1$. I find in my experimental evaluation in Section 5.3 that it is important to correctly set β when computing a threshold allocation from the F_β-measure scores. The TPs, FPs and FNs for Equation 5.1 are computed in an identical fashion to Chapter 4. To recapitulate, I first build an indicator matrix $\mathbf{P}^k \in \{0,1\}^{N_{trd} \times N_{trd}}$ for projected dimension $\mathbf{y}^k \in \mathbb{R}^N$ that specifies the data-point pairs $(\mathbf{x}_i, \mathbf{x}_j)$ that fall within the same thresholded regions of the projected dimension $\mathbf{y}^k \in \mathbb{R}^N$ (Equation 5.2).

$$P_{ij}^k = \begin{cases} 1, & \text{if} \quad \exists_\gamma \quad s.t. \quad t_{k\gamma} \leq (y_i^k, y_j^k) < t_{k(\gamma+1)} \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

Recall that $[t_{k1} \ldots t_{kT}]$ denotes the T thresholds partitioning the k^{th} projected dimension. The *true positives* (TP), *false negatives* (FN) and *false positives* (FP) are computed using Equations 5.3-5.5.

$$TP = \frac{1}{2}\sum_{ij} P_{ij} S_{ij} = \frac{1}{2}\|\mathbf{P} \circ \mathbf{S}\|_1 \tag{5.3}$$

$$FN = \frac{1}{2}\sum_{ij} S_{ij} - TP = \frac{1}{2}\|\mathbf{S}\|_1 - TP \tag{5.4}$$

$$FP = \frac{1}{2}\sum_{ij} P_{ij} - TP = \frac{1}{2}\|\mathbf{P}\|_1 - TP \tag{5.5}$$

With these definitions the F_β-measure for a particular thresholding of an arbitrary projected dimension $\mathbf{y}^k$ can be specified in matricial form as given in Equation 5.6

$$F_\beta(\mathbf{t}_k) = \frac{(1+\beta^2)\|\mathbf{P}\circ\mathbf{S}\|_1}{\beta^2\|\mathbf{S}\|_1 + \|\mathbf{P}\|_1} \tag{5.6}$$

In Chapter 4, Section 4.3.3.2 I found that it is optimal to set the interpolation parameter between the supervised (F_β-measure) and unsupervised terms to $\alpha = 1$, for hash-code lengths up to 128 bits, when using the Manhattan hashing quantisation codebook. In this chapter I therefore do not interpolate the F_β-measure with an unsupervised term as I did in Equation 4.6 in Chapter 4. Investigating the benefit, or otherwise, of interpolation with an unsupervised signal in the context of variable threshold allocation is left to future work.

In this chapter my objective is to find both, the optimal number of thresholds to allocate to each projected dimension, and the optimal placement of those thresholds. The possible quantity of thresholds for each dimension T_k is constrained by the Manhattan quantisation codebook to be within the finite set $T_k \in [0,1,3,7,15]$. I argue that projected dimensions with a higher locality preserving quality should be given a larger allocation of the available thresholds so that the neighbourhood structure captured by that dimension can be maximally exploited. To grade the quality of a threshold allocation I find the optimal position of the thresholds by maximising Equation 5.6 using evolutionary algorithms (EA). The resulting F_β-measure, relating to the optimal position of the T_k thresholds as found by the EA, is then used as the indicator of projected dimension quality. Importantly once the optimal threshold positions are computed they do not have to be re-optimised in order to perform the threshold allocation. This offline training procedure will yield five F_β-measure scores per projected dimension, one for each threshold quantity $T_k \in [0,1,3,7,15]$.

The F_β-measure scores are then used by my variable threshold allocation algorithm to compute the threshold allocation that yields the highest *cumulative* F_β-measure subject to a fixed threshold allocation budget. In doing my contention is that the F_β-measure varies widely for each possible threshold quantity $T_k \in [0,1,3,7,15]$ and that there is a unique quantity of thresholds that provides an overall greatest F_β-measure

for a given projected dimension. In addition I argue that the value of the maximum F_β-measure is higher for projected dimensions that are more effective at preserving the pairwise constraints between data-points in the adjacency matrix $\mathbf{S}$. If this hypothesis is correct then the optimised F_β-measure will most likely provide an effective signal for threshold allocation in a way that assigns a greater proportion of the thresholds to those projected dimensions that respect the locality structure encoded in the adjacency graph $\mathbf{S}$.

Figure 5.1 provides a further intuition as to why assigning a projected dimension a threshold quantity T_k that maximises Equation 5.6 might be expected to lead to an effective multi-threshold quantisation. In this toy example I show a two-dimensional (2D) input feature space in which data-points are represented by coloured shapes. Those data-points with the same shape and colour are nearest neighbours in the 2D plane. Two hyperplanes $\mathbf{h}_1 \in \mathbb{R}^2, \mathbf{h}_2 \in \mathbb{R}^2$ are shown partitioning the space into four regions. The hyperplanes $\mathbf{h}_1 \in \mathbb{R}^2, \mathbf{h}_2 \in \mathbb{R}^2$ have normal vectors $\mathbf{w}_1 \in \mathbb{R}^2, \mathbf{w}_2 \in \mathbb{R}^2$, respectively. On the bottom diagram in Figure 5.1, I show the resulting projected dimensions $\mathbf{y}^1 \in \mathbb{R}^2, \mathbf{y}^2 \in \mathbb{R}^2$ obtained by projecting the data-points onto the normal vectors. The value of Equation 5.6 for projected dimension $\mathbf{y}^2$ with $\beta = 1$ is at the maximum value ($F_1 = 1$). Observing the quantised regions we see that all nearest neighbours are located beside each other and are not partitioned by any threshold. This can be deemed a perfect quantisation of the projected dimension because all nearest neighbours will be assigned the same hashcode. This fact is highlighted by the high value of Equation 5.6. Furthermore we observe that three thresholds is the minimal quantity of thresholds in this contrived example that will lead to a perfect quantisation, with either more ($T > 3$) or less thresholds ($T < 3$) receiving a lower F_β-measure score. The opposite is the case for projected dimension $\mathbf{y}^1$. Given the high mixing of the data-points *no* thresholds do just as well in this situation as one or more thresholds. The inability of this hyperplane to clump together related data-points along the projected dimension is highlighted by a low F_β-measure. In this case the corresponding hyperplane ($\mathbf{h}_1$) should ideally be pruned and the allocation of thresholds distributed to the more effective hyperplane ($\mathbf{h}_2$).

5.2.2.2 A Link to the Laplacian Score

My application of the F_β-measure to allocate thresholds to projected dimensions is in some senses acting as a filter-based feature selection score that has been constructed specifically for hashing-based ANN search in which multiple thresholds are used for

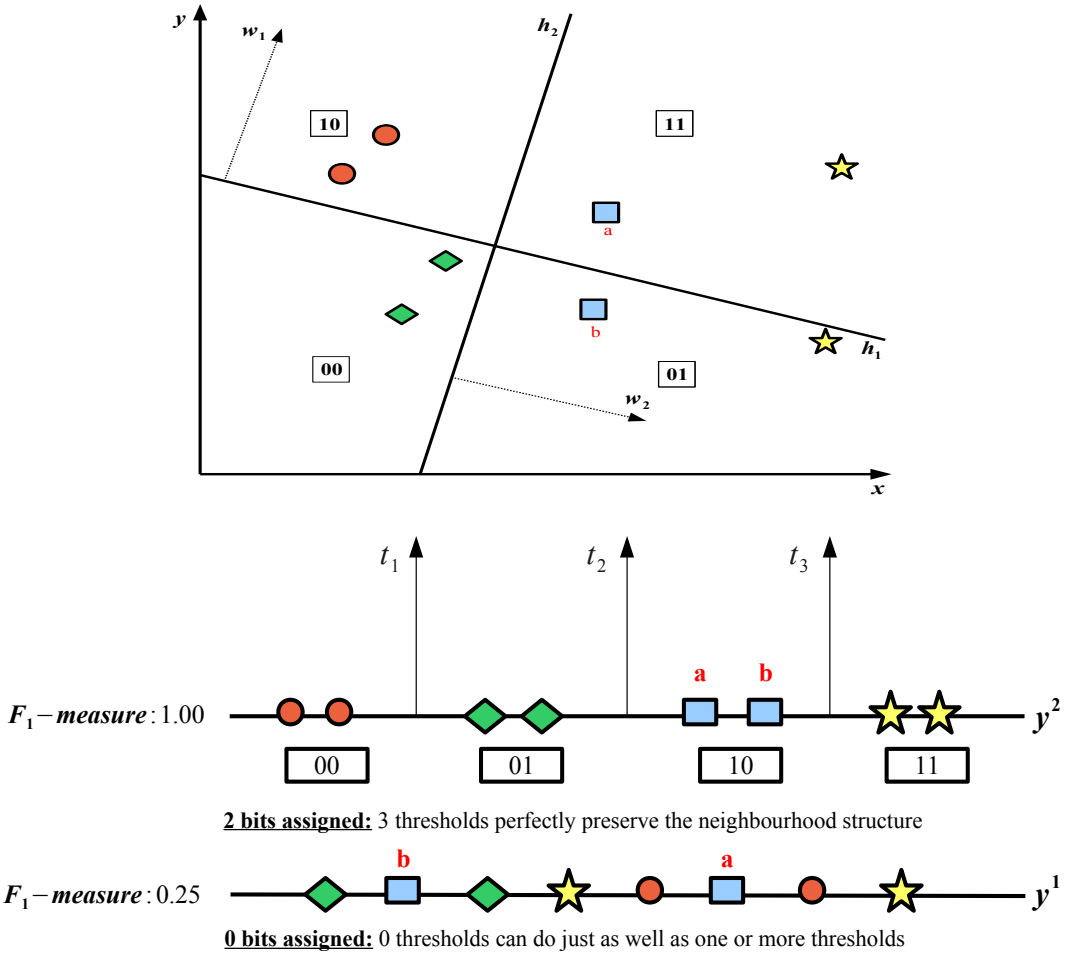


Figure 5.1: Intuition behind the application of the F_β-measure as a means of grading the quality of a quantisation resulting from an assignment of a given number of thresholds. I set $\beta = 1$ for the purposes of this example. The top diagram illustrates a 2D plane in which data-points (indicated by the coloured shapes) are embedded. The diagram below illustrates the projection of the data-points onto the normal vectors $\mathbf{w}_1, \mathbf{w}_2$. Projected dimension $\mathbf{y}^2$ can be perfectly quantised with 3 thresholds. In the case of data-points a, b both end up within the same region. This is not the case for projected dimension $\mathbf{y}^1$ which is much more difficult to quantise due to related data-points ending up much farther away from each other along the dimension. The reader is guided to Section 5.2.2.1 for a further and fuller description.

quantisation. By viewing my algorithm under the lens of feature selection I can draw an interesting parallel to the Laplacian score (He et al. (2005)), a popular supervised feature selection score which originated in the field of Machine Learning. The Laplacian score is a filter-based feature selection metric which can be applied independent of a classifier, the latter bond being a requirement of wrapper-based algorithms. Briefly

Laplacian Score is reminiscent of the Laplacian Eigenmap which I discussed in detail in Chapter 2, Section 2.6.4.4 in the context of the Self Taught Hashing (STH) model. The Laplacian Eigenmap seeks a low dimensional projection of a set of data-points in a way that preserves the pairwise relationships between those data-points as encoded in the adjacency graph $\mathbf{S}$. Data-points that are neighbours according to the adjacency graph ($S_{ij} = 1$) should end up close together when projected into the low-dimensional space. Laplacian Score further develops this key idea into a feature selection metric that assigns lower (lower is better) scores to features that are related ($S_{ij} = 1$) and which are also closer together along the projected dimension, that is the distance $(y_i^k - y_j^k)^2$ between the projections of data-points $\mathbf{x}_i, \mathbf{x}_j$ is low (Equation 5.7)

$$L_k = \sum_{ij} \frac{(y_i^k - y_j^k)^2 S_{ij}}{var(\mathbf{y}^k)} \tag{5.7}$$

where L_k is the Laplacian score of the k^{th} dimension and $var(\mathbf{y}^k)$ computes the variance of the dimension $\mathbf{y}^k \in \mathbb{R}^N$, that is $var(\mathbf{y}^k) = 1/N_{trd} \sum_{i=1}^{N_{trd}} (y_i^k - \mu)^2$, and μ denotes the mean of the projected dimension. The Laplacian score is minimised for features that respect the graph structure, while also having a large variance and therefore presumably high representational power. In a different manner to the Laplacian Score, my application of the F_β-measure explicitly takes into account how many true pairs end up within the same thresholded regions but does not account for the closeness of their corresponding projections.

5.2.2.3 Threshold Allocation via Branch-and-Bound

Having defined the metric I use to grade the quality of a projected dimension in Section 5.2.2.1 I will now introduce my first algorithm that leverages the resulting quality scores to learn an effective distribution of thresholds across K projected dimensions. As I argued in Section 5.2.2.1 the F_β-measure scores per hyperplane ($\mathbf{h}_k$), per threshold count ($T_k \in [0, 1, 3, 7, 15]$) are an effective signal for threshold allocation as more informative hyperplanes tend to have higher F_β-measures for a higher number of thresholds. I hypothesise that seeking the threshold allocation that maximises the total F_β-measure as accumulated across all K projected dimensions, subject to a threshold budget, is a suitable objective for optimisation. Unfortunately this combinatorial optimisation problem is NP-hard which can be immediately deduced with analogy to the binary *knapsack problem*, a classic problem in combinatorial optimisation (Dantzig (1957)). The binary knapsack problem involves selecting a number of items to place into a

knapsack that maximises the total value of the items while adhering to a limit on the total weight. An item can only be chosen for inclusion once, hence the binary nature of the problem (0 is exclude from knapsack, 1 is include in knapsack). The inherent difficulty of this problem stems from this integrality constraint. This is exactly my threshold allocation problem in which an "item" is one or more thresholds for a dimension, the "weight" is the threshold quantity for that dimension and the value is the F_β-measure for that number of thresholds when positioned optimally along the projected dimension. By drawing a parallel to the task in this way I can benefit from the rich literature that has already been established on approximately solving the binary knapsack problem (Martello and Toth (1990)). My first solution has indeed been inspired in this manner and involves framing the learning objective as a *binary integer linear program* (BILP).

To construct a BILP appropriate for my purposes I firstly collate the F_β-measure scores per hyperplane, per threshold count in a matrix $\mathbf{F} \in \mathbb{R}^{(B_{max}+1)\times K}$ with elements F_{bk} which represent the accuracy that results from allocating $2^b - 1$ thresholds to dimension k. In this case $b \in \{0,\ldots,B_{max}\}$ indexes the rows, with B_{max} being the maximum number of bits allowable for any given hyperplane, and $k \in \{1\ldots,K\}$ indexes the columns of the F_β-measure matrix. Note the equivalence between thresholds and bits: if we have T_k thresholds for a particular projected dimension this equates to an allocation of $B_k = \log_2(T+1)$ bits for that dimension. The BILP uses $\mathbf{F}$ to find the threshold allocation that maximises the cumulative F_β-measure across the K hyperplanes (Equation 5.8)

$$\begin{aligned} \max \quad & \|\mathbf{F}\circ\mathbf{Z}\| \\ \text{subject to } & \|\mathbf{Z}_c\| = 1 \qquad c \in \{1\ldots K\} \\ & \|\mathbf{Z}\circ\mathbf{D}\| \leq K \\ & \mathbf{Z} \quad \text{is binary} \end{aligned} \tag{5.8}$$

where $\|.\|$ denotes the Frobenius L_1 norm, $\circ$ the Hadamard (elementwise) product and $\mathbf{D} \in \mathbb{Z}_+^{(B_{max}+1)\times K}$ is a constraint matrix, with $D_{bh} = b-1$, ensuring that the threshold allocation remains within the bit budget B. I now give an intuitive overview of each term in Equation 5.8:

- $\|\mathbf{F}\circ\mathbf{Z}\|$: This term computes the cumulative F_β-measure score for bit allocation specified by $\mathbf{Z}$

- $\|\mathbf{Z}_c\| = 1$: This constraint ensures that a specific number of bits selected from the set $b \in [0,1,2,3,4]$ is allocated to each hyperplane, with $B_{max} = 4$. This constraint also ensures that more than one value from this set cannot be allocated to any single hyperplane.

- $\|\mathbf{Z} \circ \mathbf{D}\|$: This constraint enforces the bit allocation $\mathbf{Z}$ to be less than or equal to the available bit budget K.

The BILP is solved using the off-the-shelf *branch and bound* optimisation algorithm (Land and Doig (1960))[2]. The branch and bound algorithm is a common workhorse for solving integer programming problems. Branch and bound enumerates the entire space of candidate solutions but maintains tractability by discarding large portions of the search space based on estimated lower and upper bounds on the quantity being optimised. Describing the specifics of this solver is beyond the scope of this thesis, however the reader is pointed to Brassard and Bratley (1996) for an introductory overview. The output from the branch and bound BILP solver is an indicator matrix $\mathbf{Z} \in \{0,1\}^{(B_{max}+1)\times K}$ whose columns specify the optimal threshold allocation for a given hyperplane, that is, $Z_{bk} = 1$ if the BILP decided to allocate b bits (equivalently $2^b - 1$ thresholds) for the k^{th} hyperplane, and zero otherwise. I compute the quality score of a zero bit allocation by computing the F_β-measure on a projected dimension with an allocation of zero thresholds. Example input and output from the branch and bound algorithm for the toy problem in Figure 5.1 are given in matrices $\mathbf{F}, \mathbf{D}, \mathbf{Z}$ below (in this example, $B_{max} = 2$ and $K = 2$). The indicator matrix $\mathbf{Z}$ is output by the branch and bound solver as the best feasible solution found subject to the problem constraints.

$$\begin{array}{cccc} \mathbf{F} & k_1 & k_2 & \mathbf{D} \\ b_0 & 0.25 & 0.25 & \\ b_1 & 0.35 & 0.50 & \\ b_2 & 0.40 & 1.00 & \end{array} \quad \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 2 & 2 \end{pmatrix} \quad \mathbf{Z} \quad \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$$

Notice how the indicator matrix $\mathbf{Z}$ specifies an assignment of 0 bits (0 thresholds) for hyperplane $\mathbf{h}_1$ and 2 bits (3 thresholds) for hyperplane $\mathbf{h}_2$ as this yields the highest cumulative F_1-measure (1.25) across the K projected dimensions while also meeting the required threshold budget. This result accords with our intuition in how the thresholds should be allocated in the toy example of Figure 5.1. The BILP as framed in

[2]Specifically I use the `bintprog` Matlab solver with default parameters.

Equation 5.8 is therefore a principled method for both selecting a discriminative subset and allocating an appropriate quantity of thresholds to those hyperplanes subject to a fixed overall threshold budget. The computational time complexity of solving the BILP using branch and bound has exponential ($O(2^{(B_{max}+1)K})$ time complexity in the worst-case (Lawler and Wood (1966)), although in practice the tree search is actually very efficient. In general I find that the branch and bound cost is negligible in comparison to the dominant training time cost of computing the $(B_{max}+1)K$ F_β-measure scores in the matrix $\mathbf{F}$ which is $O(T_{max}N_{trd}^2F + KN_{trd}^2)$, with N_{trd} denoting the number of training data-points, F the number of objective function evaluations (Equation 5.6) and $T_{max} = K\sum_{b=1}^{B_{max}} 2^b - 1$. I will now introduce an alternative bit allocation algorithm that improves upon the computational time complexity of the branch-and-bound solution by eliminating the dependence on B_{max}.

5.2.2.4 Greedy Threshold Allocation Algorithm

The branch-and-bound solution presented in Section 5.2.2.3 is dominated by the computation of the $(B_{max}+1)K$ F_β-measure scores taking $O(T_{max}N_{trd}^2F + KN_{trd}^2)$ time. In my second contribution of this chapter I introduce a novel greedy algorithm for threshold allocation that reduces this computational time complexity to $O(T_{gdy}N_{trd}^2F + KN_{trd}^2)$, where $T_{gdy} \ll T_{max}$ and is independent of B_{max}. This algorithm performs the same task as the BILP but is greedy and hence it is expected to be faster and simpler. This algorithm is based on a simple and intuitive idea, namely the redistribution of thresholds at each iteration from lower quality (low F_β-measure scores) hyperplanes to higher quality (higher F_β-measure scores) hyperplanes while adhering to the specified threshold budget. I use a new notation in this section to indicate the number of bits assigned per projected dimension. Rather than use an indicator matrix $\mathbf{Z}$ as in Section 5.2.2.3, I unroll $\mathbf{Z}$ as a vector of bit counts $\mathbf{z} \in \mathbb{Z}_+^K$, which will make the exposition of the algorithm somewhat easier. For example $z_h = 3$ if we have allocated 3 bits ($2^3 - 1 = 7$ thresholds) to the h^{th} hyperplane. I initialise $z_h = 1, \forall_h$ so at the start of the algorithm we have allocated one threshold per hyperplane. The matrix of F_β-measures $\mathbf{F} \in \mathbb{R}^{(B_{max}+1)\times K}$ is computed in the same manner as for the binary integer linear program (BILP) based solution outlined in Section 5.2.2.3. To recapitulate, to compute F_{bh} I position $2^b - 1$ thresholds along the h^{th} projected dimension to maximise Equation 5.6. The quantisation threshold positioning is found using evolutionary algorithms as outlined in Chapter 4, Section 4.2.3.2. Element F_{bh} of matrix $\mathbf{F}$ is then set to the maximum F_β-measure arising from the optimal threshold configuration dis-

covered by the stochastic search. Crucially, however, I do not compute all $(B_{max}+1)K$ F_β-measure scores at once as for the branch-and-bound solution. Rather the greedy algorithm permits a more efficient *on-demand* computation of the F_β-measure scores.

I initialise the $\mathbf{F}$ matrix by computing the F_β-measure score for $B = 0, 1, 2$ bits, requiring $O(3K)$ computations. Having initialised $\mathbf{F}$, I learn the optimal bit allocation $\mathbf{z}$ through *threshold redistribution*. In the first iteration this procedure involves finding a projected dimension that would "most benefit" (largest delta in F_β-measure) from having its bit allocation increased by one unit and the projected dimension that would "suffer least" (smallest delta in F_β-measure) from having a bit subtracted from its bit allocation. To determine the projected dimension that should have a bit added to its allocation I simply search for the projected dimension h_{max} where $h_{max} = \text{argmax}_h(F_{2,h} - F_{1,h})$. Intuitively the projected dimension that has the greatest *increase* in F_β-measure between its F_β-measure for its current allocation of 1 bit and its F_β-measure for a potential allocation of 2 bits, should be assigned that additional 1 bit. Having allocated an additional bit to projected dimension h_{max} we have now allocated $K+1$ bits, 1 bit over the maximum bit quota of K. To respect the quota I remove a bit from the hyperplane which has the *lowest difference* between its F_β-measure at a bit allocation of 1 and its F_β-measure for a bit allocation of 0 bits.

More formally we wish to find hyperplane h_{min} where $h_{min} = \text{argmin}_h(F_{1,h} - F_{0,h})$. In allocating 0 bits to h_{min} we have effectively discarded that hyperplane and eliminated it from further consideration. Note that once a hyperplane has 0 bits or B_{max} bits assigned the count for that hyperplane forever remains fixed at those values. By adding a bit to the hyperplane that contributes the greatest increase in F_β-measure while removing a bit from the hyperplane that loses the least F_β-measure I greedily approximate a maximisation of the cumulative F_β-measure across all K hyperplanes. The algorithm proceeds in this manner until we reach a point where the maximum increase in F_β-measure is lower than the minimum decrease in F_β-measure, which indicates that there is no more benefit from redistributing thresholds amongst the K projected dimensions. On each of these subsequent steps post initialisation of the $\mathbf{F}$ matrix, the computationally expensive operation of computing the F_β-measure needs only to be computed *once per step* namely for that hyperplane that was recently promoted to have the maximum current number of bits. A pseudocode version of the algorithm is presented in Algorithm 8.

This greedy threshold allocation algorithm has a training time complexity characterised by $O(T_{gdy}N_{trd}^2F + KN_{trd}^2)$ and a threshold allocation time complexity of $O(1)$,

Algorithm 8: GREEDY THRESHOLD ALLOCATION ALGORITHM

Input: Projected dimensions $\left\{\mathbf{y}^k \in \mathbb{R}^N\right\}_{k=1}^K$, the *maximum* number of bits per dimension $B_{max} \in [1,2,3,4]$, F_β-measure scores $\mathbf{F} \in \mathbb{R}^{(B_{max}+1)\times K}$

Output: Optimal allocation of thresholds to projected dimensions $\mathbf{z} \in \mathbb{Z}_+^K$

Set $F_\beta^{max} = \infty$, $F_\beta^{min} = 0$
Set $\mathbf{z} = \mathbf{1}^K$ `// Initalise bit counts to 1`
while $F_\beta^{max} > F_\beta^{min}$ **do**
  $h_{min} = \text{argmin}_h(F_{z_h h} - F_{z_h - 1 h})$
  $F_\beta^{min} = F_{z_{h_{min}} h_{min}} - F_{z_{h_{min}} - 1 h_{min}}$
  $h_{max} = \text{argmax}_h(F_{z_h + 1 h} - F_{z_h h})$
  $F_\beta^{max} = F_{z_{h_{max}} + 1 h_{max}} - F_{z_{h_{max}} h_{max}}$
  if $F_\beta^{max} > F_\beta^{min}$ **then**
    $z_{h_{max}} = z_{h_{max}} + 1$ `// Increment bit count for hyperplane` h_{max}
    if $z_{h_{max}} = B_{max}$ **then**
      $\mathbf{F}_{\bullet h_{max}} = NaN$ `//` h_{max} `cannot be chosen again`
    end
    $z_{h_{min}} = z_{h_{min}} - 1$ `// Decrement bit count for hyperplane` h_{min}
    if $z_{h_{min}} = 0$ **then**
      $\mathbf{F}_{\bullet h_{min}} = NaN$ `//` h_{min} `cannot be chosen again`
    end
  end
end
return z

where $T_{gdy} \ll T_{max}$ and $T_{max} = K\sum_{b=1}^{B_{max}} 2^b - 1$. The training time complexity is independent of B_{max}, and so the greedy threshold learning algorithm is expected to be faster at learning than the branch-and-bound solution presented in Section 5.2.2.3. The reasoning for this is as follows: prior to the execution of the algorithm only the first three rows of matrix $\mathbf{F} \in \mathbb{R}^{(B_{max}+1)\times K}$ need be initialised, relating to a bit allocation of 0, 1 and 2 bits for each of the K available hyperplanes. The total number of F_β-measure computations for this particular initialisation is of $O(3K)$. I contend that only $O(K)$ additional F_β-measure computations are then required for the learning step. The reason for this is that VBQ$_{bound}$ needs only to compute the F_β-measure once per subsequent iteration. This F_β-measure is computed for the hyperplane that was most recently (i.e.

at the previous iteration) promoted to have two or more bits, as this will be the only hyperplane for which the F_β-measure is unknown for an additional $(+1)$ bit over its current allocation. Furthermore, the algorithm needs only to perform this computation less than K times as there are only K available bits. The greedy algorithm therefore needs only $O(K)$ F_β-measure computations, independent of the B_{max} term, and we can therefore expect a lower learning time compared to the branch-and-bound algorithm introduced in Section 5.2.2.3. Having just scored the hyperplanes for differing threshold quantities in the learning step, the threshold allocation step is straightforward as we simply use the allocation specified at the end of the final iteration. The threshold allocation time complexity of this algorithm is therefore $O(1)$. A comparison between the learning and allocation runtimes of my proposed greedy and branch-and-bound algorithms are presented in Section 5.3.3.4.

5.3 Experimental Evaluation

5.3.1 Experimental Configuration

In this section I will experimentally test my two variable threshold quantisation algorithms introduced in Section 5.2. The experimental setup will be almost identical to the literature standard configuration used to evaluate my multi-threshold quantisation algorithm in Chapter 4, Section 4.2 (Kong et al. (2012); Kong and Li (2012a,b); Kulis and Darrell (2009); Raginsky and Lazebnik (2009); Gong and Lazebnik (2011)). Specifically my task will be query-by-example image retrieval with the experimental parameters shown in Table 5.1. As a baseline I will compare against my multi-threshold quantisation model (NPQ) introduced in Chapter 4. The NPQ model was shown to significantly outperform competing scalar quantisation models in the literature (MHQ, DBQ, SBQ) that assign a uniform quantity of thresholds across projected dimensions.

I structure the experiments in this chapter to answer the following two main hypotheses:

- H_1: *Allocating a* variable *number of thresholds can yield a higher retrieval effectiveness than a* uniform allocation *of thresholds (NPQ).*
- H_2: *Branch-and-bound (VBQ_{bound}) finds a threshold allocation that leads to a significantly higher retrieval effectiveness than the greedy threshold allocation algorithm (VBQ_{greedy}).*

Parameter	Setting	Chapter Reference
Groundtruth Definition	ε-NN	Chapter 3, Section 3.3
Evaluation Metric	AUPRC	Chapter 3, Section 3.6.3
Evaluation Paradigm	Hamming Ranking	Chapter 3, Section 3.4
Random Partitions	10	Chapter 3, Section 3.5
Number of Bits (K)	16-128	Chapter 2, Section 2.4

Table 5.1: Configuration of the main experimental parameters for the results presented in this chapter.

Method	# Thresholds/dim	Encoding	Ranking Strategy
VBQ	Variable	NBC	Manhattan distance
NPQ	3	NBC	Manhattan distance

Table 5.2: Parametrisation of the quantisation models studied in this chapter.

Hypothesis H_1 will examine whether a variable threshold allocation can yield a higher retrieval effectiveness than a uniform assignment. Hypothesis H_2 will confirm whether the sub-optimal search strategy of the greedy approach results in an inferior threshold allocation compared to branch-and-bound.

In order to abstract from the effect of the codebook and the hashcode ranking strategy I parametrise all quantisation models in this chapter to use the Manhattan Hashing Quantisation (MHQ) codebook with the Manhattan distance as the hashcode ranking strategy (Table 5.2)[3]. Furthermore to ensure easy replication of my results by interested researchers the experiments in this chapter will be carried out on the standard LabelMe, CIFAR-10 and NUS-WIDE image collections as described in Chapter 3, Section 3.2.1. In a similar manner to my evaluation in Chapter 4, I follow previously accepted procedure in the literature (Kong et al. (2012), Kong and Li (2012a)) and randomly select $N_{teq} = 1,000$ data points as testing queries ($\mathbf{X}_{teq} \in \mathbb{R}^{N_{teq} \times D}$), with the remaining points ($\mathbf{X}_{db} \in \mathbb{R}^{N_{db} \times D}$) being used as the database upon which to learn and test the hash functions according to either the literature standard or improved dataset splitting strategy. A further breakdown on the specific dataset splits I use in this chapter are shown in Tables 5.3-5.4.

[3]The reader is guided to Chapter 2, Section 2.5.4 for an overview of this codebook and hashcode ranking strategy.

Partition	LABELME	CIFAR-10	NUS-WIDE
Test queries (N_{teq})	1,000	1,000	1,000
Validation queries (N_{vaq})	1,000	1,000	1,000
Validation database (N_{vad})	10,000	10,000	10,000
Training database (N_{trd})	2,000	2,000	10,000
Test database (N_{ted})	8,000	46,000	247,648

Table 5.3: Improved splitting strategy partition sizes for the experiments in this chapter. This breakdown is based on the splitting strategy introduced in Chapter 3, Section 3.5.2. There is no overlap between the data-points across partitions.

Partition	LABELME	CIFAR-10	NUS-WIDE
Test queries (N_{teq})	1,000	1,000	1,000
Validation queries (N_{vaq})	1,000	1,000	1,000
Validation database (N_{vad})	10,000	10,000	10,000
Training database (N_{trd})	2,000	2,000	10,000
Test database (N_{ted})	21,000	59,000	268,648

Table 5.4: Literature standard splitting strategy partition sizes for the experiments in this chapter. This breakdown is based on the splitting strategy introduced in Chapter 3, Section 3.5.1.

5.3.2 Parameter Optimisation

To set the hyperparameters of my model, I conduct a grid search for the best fitting β, B_{max} hyperparameters over the the values $\beta \in \{0.5, 1, 2, 5, 10\}$ and $B_{max} \in \{1, 2, 3, 4\}$. This grid search is performed on the held out validation dataset ($\mathbf{X}_{vaq} \in \mathbb{R}^{N_{vaq} \times D}$, $\mathbf{X}_{vdb} \in \mathbb{R}^{N_{vdb} \times D}$), selecting the parameter combination that maximises validation dataset AUPRC. As the NPQ model introduced in Chapter 4 has a free parameter β for the F_β-measure term I also optimise this parameter for NPQ on the validation portion of the dataset. I refer to the optimised NPQ model as NPQ$_{opt}$ in my result tables. For VBQ, in all experiments the thresholds and threshold allocation are learnt on the training database ($\mathbf{X}_{trd} \in \mathbb{R}^{N_{trd} \times D}$). The learnt thresholds are then subsequently used to quantise the testing dataset projections ($\mathbf{X}_{teq} \in \mathbb{R}^{N_{teq} \times D}$, $\mathbf{X}_{tdb} \in \mathbb{R}^{N_{ted} \times D}$). The same procedure is used to learn the thresholds for the NPQ and NPQ$_{opt}$ baselines. The reported AUPRC figures for the test dataset are the average over ten random dataset

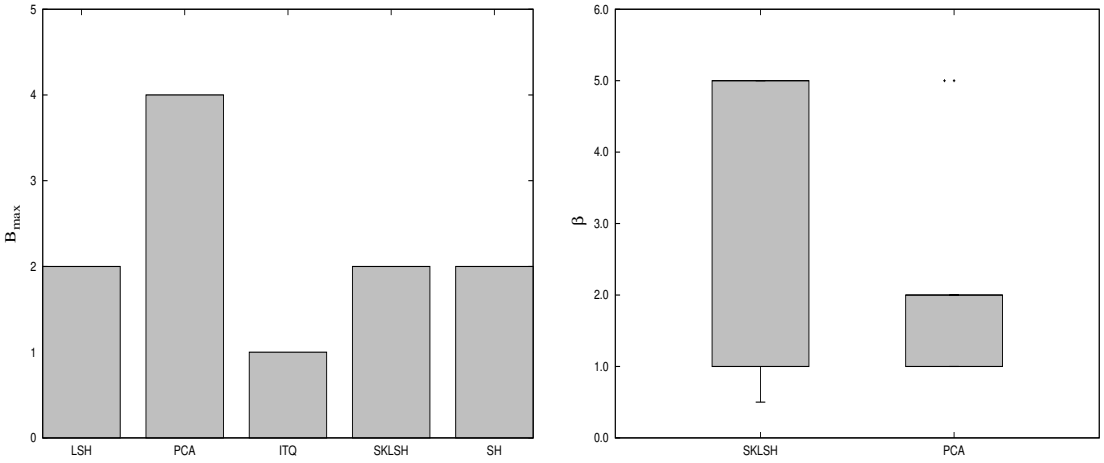
splits as explained in Chapter 3, Section 3.5. The Wilcoxon signed rank test (Wilcoxon (1945)) is used to determine the statistical significance of the retrieval results. In this case, when comparing system A to system B, a pair of AUPRC values arising from a retrieval run on the current fold forms the unit of the significance test. In all presented result tables, ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over NPQ_{opt}, while ▲/▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.05$) over NPQ_{opt}.

5.3.3 Experimental Results

5.3.3.1 Experiment I: Effect of the β and B_{max} hyperparameters

The VBQ_{bound} and VBQ_{greedy} models have two hyperparameters that need to be set on a held-out validation dataset: $\beta \in \{0.5, 1, 2, 5, 10\}$ that determines the importance given to precision versus recall in Equation 5.6, and $B_{max} \in \{1, 2, 3, 4\}$ that indicates the maximum number of bits that can be allocated per projected dimension. Figure 5.2 (a) plots the optimal value of B_{max} at 32 bits on CIFAR-10 for five data-independent and dependent projections functions: LSH, PCA, SKLSH, SH and ITQ. The optimal value of the B_{max} parameter varies between each of these different projection functions. Projection functions such as PCA, which produce hyperplanes of widely different neighbourhood preserving quality, benefit a greater maximum number of bits that can be allocated to the small subset of high quality hyperplanes, thereby boosting their contribution to the construction of the hashcodes. ITQ on the other hand attempts to make the quality of the K hyperplanes equal by rotating the input feature space such that the variance captured by each hyperplane is approximately balanced. In this case there is no subset of very high quality hyperplanes that should attract the majority of the available bit budget and therefore the B_{max} parameter in this case is quite intuitively set to $B_{max} = 1$.

Figure 5.2 (b) shows the variation in β for SKLSH and PCA projections across each random dataset split. I observe that β has a larger variance for SKLSH projections with the preferred setting falling between $\beta \in [0.5, 5]$, while for PCA projections β appears more stable with $\beta \in [1, 2]$ generally appearing optimal across most random dataset splits. This latter observation is a consequence of the widely differing partitions induced by the randomly sampled SKLSH hyperplanes compared to the deterministic setting of the PCA hyperplanes across each random dataset split. The variance in β, particularly for SKLSH, suggests that for some partitionings of the input feature space



(a) B_{max} **parameter** (b) β **parameter**

Figure 5.2: Figure (a) shows the optimal value of the B_{max} parameter per projection function. Figure (b) shows the variability of the β parameter per random dataset split for LSH and PCA projections. Results were obtained for the CIFAR-10 dataset at 32 bits.

it is more detrimental to separate pairs of similar images in different quantised regions ($\beta > 1$), compared to placing dissimilar images in the same region, and vice-versa for $\beta < 1$.

5.3.3.2 Experiment II: Variable versus Uniform Threshold Allocation

In this experiment I will study the primary hypothesis (H_1) of this chapter, namely that a variable threshold allocation adapted to specific hyperplanes will yield a higher retrieval effectiveness for nearest neighbour search compared to a uniform allocation of thresholds. To examine this hypothesis I will compare both variable threshold allocation algorithms (VBQ_{bound}, VBQ_{greedy}) against the strong baseline of NPQ, my own multi-threshold quantisation model (NPQ) introduced in Chapter 4. NPQ assigned a uniform number of thresholds to every projected dimension irrespective of the differing locality preserving qualities of the corresponding hyperplanes. I also explore the effect of tuning the NPQ $\beta \in \{0.5, 1, 2, 5, 10\}$ parameter in this chapter as it could conceivably result in an added boost in retrieval effectiveness. This optimised variant is referred to as NPQ_{opt} in all experiments. Comparing directly to NPQ will therefore tease apart the effect of an informed variable allocation of thresholds. To ensure a meaningful comparison VBQ, NPQ and NPQ_{opt} are constrained to have an identical amount of supervision, namely $N_{trd} = 2,000$ for CIFAR-10 and LabelMe and $N_{trd} = 10,000$ for

	Quantisation Model			
	VBQ$_{bound}$	**VBQ**$_{greedy}$	**NPQ**	**NPQ**$_{opt}$
LSH	0.2035 (0.2105)	**0.2176** (0.2169)▲	0.1621 (0.1662)	0.1742 (0.1744)
ITQ	0.3278 (0.3229)	0.3354 (0.3349)▼▼	0.3917 (0.3898)	**0.3947** (0.3943)
SH	**0.2549** (0.2546)▲▲	0.2299 (0.2284)	0.1834 (0.1871)	0.2044 (0.2038)
PCA	0.2712 (0.2707)	**0.2739** (0.2728)▲▲	0.1660 (0.1683)	0.1833 (0.1834)
SKLSH	0.1830 (0.1827)	**0.1832** (0.1831)▲▲	0.1063 (0.1088)	0.1070 (0.1179)

Table 5.5: AUPRC on the CIFAR-10 dataset with a hashcode length of 32 bits. The quantisation algorithms listed on the first row are used to quantise the projections from the hash functions in the first column. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over NPQ$_{opt}$. ▲/▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.05$) over NPQ$_{opt}$.

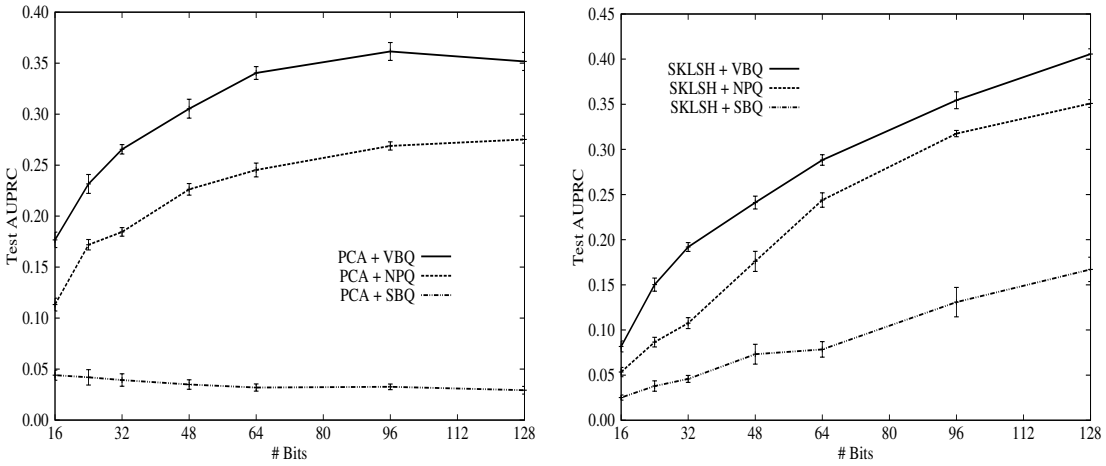
the larger NUS-WIDE dataset[4].

(a) LSH, PCA, SH, SKLSH Projections Quantised with Variable Thresholds

In this section I will examine the retrieval results obtained from quantising LSH, PCA, SH and SKLSH projections with a variable threshold allocation. The variable threshold quantisation results for ITQ projections are presented in the next section. The query-by-example image retrieval results for this section are presented in Tables 5.5-5.7 for a hashcode length of 32 bits on the CIFAR-10, LabelMe and NUS-WIDE image datasets. In addition, Figures 5.3-5.5 present the retrieval results for hashcodes of length 16-128 bits for both PCA and SKLSH projections across the three considered datasets.

The retrieval results presented in Tables 5.5-5.7 suggest that a variable allocation of thresholds (VBQ$_{bound}$, VBQ$_{greedy}$) generally outperforms a uniform allocation (NPQ, NPQ$_{opt}$) for the four considered projection functions. For example, for SKLSH projections on CIFAR-10, VBQ$_{bound}$ achieves a 71% relative increase in AUPRC compared to NPQ$_{opt}$. This increase in retrieval effectiveness is statistically significant based on a Wilcoxon signed rank test ($p < 0.01$). The results from quantising PCA projections with variable thresholds are also particularly encouraging. For example, on the CIFAR-10 image dataset VBQ$_{bound}$ obtains a statistically significant 48% relative increase at

[4]These settings of N_{trd} for CIFAR-10 and NUS-WIDE were found to be optimal in Chapter 4, Section 4.3.3.1.



(a) **PCA projections** (b) **SKLSH projections**

Figure 5.3: AUPRC versus hashcode length on CIFAR-10 for PCA (Figure (a)) and SKLSH (Figure (b)) projections. Retrieval results are shown for VBQ_{bound} and NPQ_{opt}. Bars represent the standard error of the mean.

32 bits compared to the uniform threshold allocation of NPQ_{opt} for PCA projections. The gain in retrieval effectiveness for SKLSH and PCA projections are also observed on the LabelMe (Table 5.6) and NUS-WIDE (Table 5.7) datasets. Furthermore, in Figures 5.3-5.5 it is readily apparent that this boost in retrieval effectiveness for PCA and SKLSH projections is maintained for both shorter ($<$ 32 bits) and longer ($>$ 32 bits) hashcodes. The same observation is made for PCA and SKLSH projections on the LabelMe (Figure 5.4) and NUS-WIDE datasets (Figure 5.5).

In contrast, the retrieval results for SH and LSH projections are less consistent across the three image datasets. Significant gains in retrieval effectiveness are found for LSH across the CIFAR-10 and LabelMe datasets, but not for the NUS-WIDE dataset. Similarity, for SH I observe a statistically significant increase in effectiveness for CIFAR-10 and NUS-WIDE, but not for the LabelMe dataset. This result suggests that a variable allocation of thresholds is not always guaranteed to give a performance boost, and in some cases appears to be projection and dataset specific. The gain in AUPRC for Spectral Hashing (SH) is perhaps slightly surprising given that this projection function itself allocates more bits to high variance directions in the feature space[5]. The fact that VBQ_{bound} and VBQ_{greedy} can, in some cases, extract a further gain in retrieval performance suggests that there is additional scope for improvement in retrieval effectiveness over and above the variable bit allocation mechanism of SH.

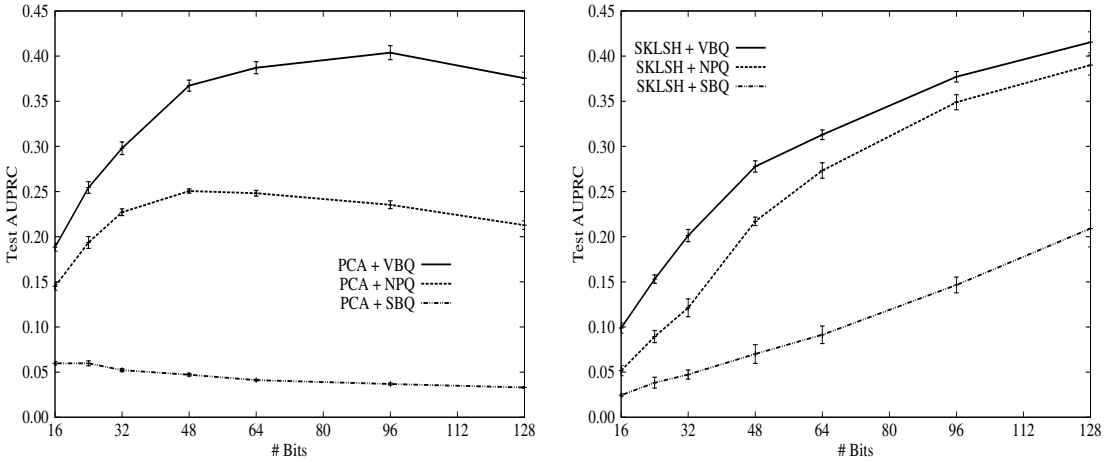
[5]The reader is referred to Chapter 2, Section 2.6.3.2 for an overview of Spectral Hashing (SH).

	Quantisation Model				
	VBQ$_{bound}$	**VBQ$_{greedy}$**	**NPQ**	**NPQ$_{opt}$**	**SBQ**
LSH	**0.2155▲**	0.2118	0.1810	0.1787	0.1574
ITQ	0.3077▼▼	0.3014	0.3658	**0.3821**	0.2822
SH	0.2500	0.2487	0.2471	**0.2581**	0.0901
PCA	0.2975	**0.3043▲▲**	0.2151	0.2306	0.0515
SKLSH	**0.2108▲▲**	0.2043	0.1311	0.1443	0.0355

Table 5.6: AUPRC on the LabelMe dataset with a hashcode length of 32 bits. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over NPQ$_{opt}$. ▲/▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.05$) over NPQ$_{opt}$

Lastly, comparing the optimised variant of my NPQ model (NPQ$_{opt}$) to the unoptimised variant (NPQ) in Tables 5.5-5.7 I note that there is a small boost in retrieval effectiveness across the three considered datasets. This result suggests that $\beta = 1$ is generally a suitable setting for NPQ, and varying β is only necessary for VBQ when using the F_β-measure as the basis for computing an optimal threshold allocation across projected dimensions. Furthermore, the results in Table 5.5 indicate that there is no significant difference between the AUPRC scores resulting from the literature standard and improved splitting strategies outlined in Chapter 3, Section 3.5. This latter observation agrees with the wealth of similar findings presented in Chapter 4, Section 4.3.3.5.

In conclusion, based on these results, I can confirm my primary hypothesis (H_1) that a variable allocation of thresholds can indeed be beneficial to the effectiveness of nearest neighbour search using many different projection functions, both of the data-dependent and independent variety. The results in this section strongly suggest that my supervised scoring metric (the F_β-measure) is effective at downweighting the contribution of ineffective hyperplanes to the computation of the hashcode ranking metric (Manhattan distance). The downweighting is achieved by allocating a low number of thresholds (or none) to these more ineffective projected dimensions, which in turn will reduce the amount of bits contributed from these dimensions to the hashcode for a data-point.



(a) **PCA projections** (b) **SKLSH projections**

Figure 5.4: AUPRC versus hashcode length on LabelMe for PCA (Figure (a)) and SKLSH (Figure (b)) projections. Retrieval results are shown for VBQ_{bound} and NPQ_{opt}. Bars represent the standard error of the mean.

(b) ITQ Projections Quantised with Variable Thresholds

ITQ is the only projection function on which a variable allocation of thresholds fails to achieve an increase in retrieval effectiveness over a uniform allocation with my multi-threshold quantisation model (NPQ) presented in Chapter 4. In fact, when compared to NPQ, both VBQ_{bound} and VBQ_{greedy} appear to hurt retrieval effectiveness when quantising ITQ projections (18% decrease in AUPRC versus NPQ_{opt}). Recall from my review in Chapter 2, Section 2.6.3.3 that ITQ rotates the input feature space to balance the variance captured by the K hyperplanes. The assumption made by ITQ is that hyperplanes capturing a low amount of the variance in the input feature space generate poor quality bits as they generally fail to clump related data-points close together along a projected dimension. By rotating the input feature space the variance captured becomes more balanced across the K hyperplanes and each corresponding bit therefore generally has an equal quality. I conjecture that the rotation performed by ITQ is removing part of the signal relied upon by my variable threshold allocation algorithms, namely a high degree of difference between the locality preserving qualities of the K hyperplanes. Recall that my variable threshold allocation algorithms specifically seek out informative hyperplanes in order to allocate those hyperplanes proportionally more thresholds than the more uninformative hyperplanes. With this signal evidently all but removed the variable threshold allocation struggles to improve beyond a uniform multi-threshold allocation (NPQ). The fact that VBQ_{bound} and VBQ_{greedy} exhibit

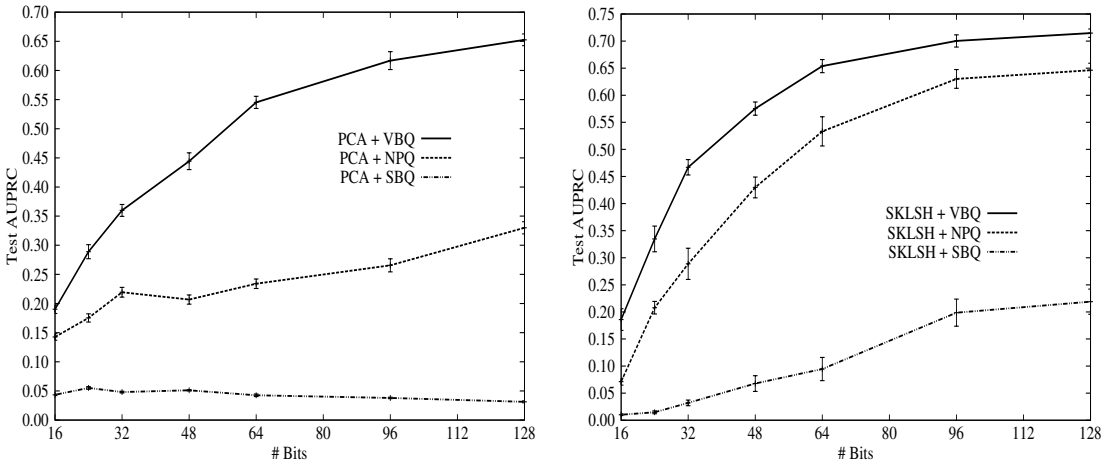
	Quantisation Model				
	VBQ$_{bound}$	**VBQ$_{greedy}$**	**NPQ**	**NPQ$_{opt}$**	**SBQ**
LSH	**0.5119**	0.4970	0.4238	0.4507	0.2961
ITQ	0.4438▼▼	0.4438	0.5130	**0.5226**	0.4842
SH	**0.3323▲▲**	0.3251	0.1965	0.1970	0.0232
PCA	**0.3741▲▲**	0.3621	0.2178	0.2340	0.0516
SKLSH	0.4505	**0.4675▲▲**	0.2650	0.2798	0.0310

Table 5.7: AUPRC on the NUS-WIDE dataset with a hashcode length of 32 bits. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over NPQ$_{opt}$.

a lower AUPRC than NPQ can be explained by the different allocations of thresholds made by the algorithms. In the case of NPQ I manually specified an allocation of three thresholds (2 bits) per projected dimension and therefore $K/2$ of the available hyperplanes were discarded. On the other hand, VBQ$_{bound}$ and VBQ$_{greedy}$ chose to allocate only one threshold (1 bit) per projected dimension and therefore retained all K of the available hyperplanes, which is clearly not optimal in this case. Allocation of thresholds based of F_β-measure maximisation is therefore not perfectly correlated with maximisation of AUPRC.

(c) Examining the Bit Allocations of VBQ$_{bound}$

Finally, it is interesting to examine the specific allocation of bits (thresholds) made by VBQ$_{bound}$. I show in Figure 5.6 the bit allocation assigned by VBQ$_{bound}$ to each of the 32 hyperplanes generated by the PCA and SKLSH projection functions. In Figure 5.6a I rank the PCA projected dimensions in descending order of variance, so that the first projected dimension has the highest overall variance, the second projected dimension exhibits the second highest variance and so forth. It is clear that the higher variance PCA hyperplanes (1-10) garner the vast proportion of the available bit budget (equivalently thresholds). This result suggests that those hyperplanes that capture the highest variance in the input feature space are also effective at preserving the pairwise relationships between data-points encoded in the adjacency matrix $\mathbf{S}$. Nevertheless, I also note that lower variance hyperplanes also attract a proportion of the available bits (12, 14, 19, 22), which suggests that variance is not perfectly correlated with neighbourhood preservation. This latter fact is not surprising given that variance is a quantity computed in an unsupervised manner, independent of any knowledge on



(a) **PCA projections** (b) **SKLSH projections**

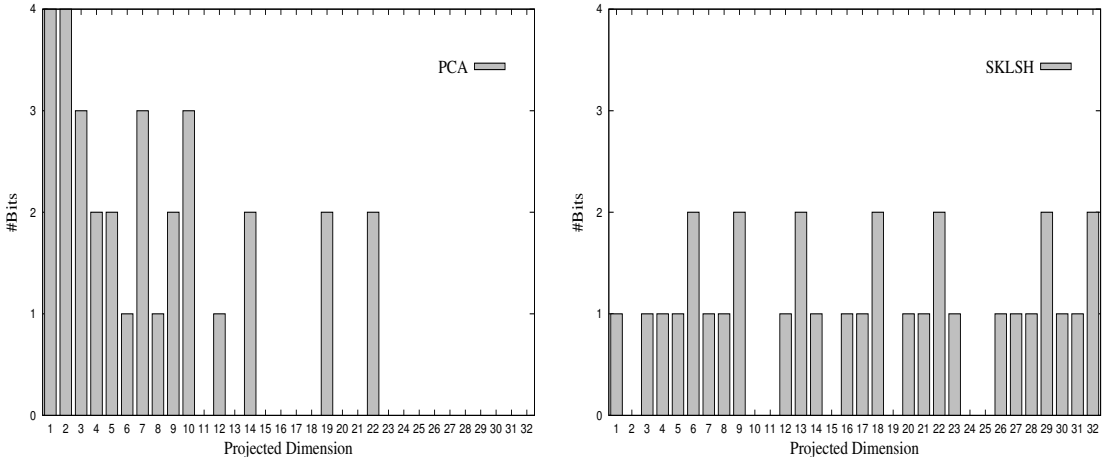
Figure 5.5: AUPRC versus hashcode length on NUSWIDE for PCA (Figure (a)) and SKLSH (Figure (b)) projections. Retrieval results are shown for VBQ_{bound} and NPQ_{opt}. Bars represent the standard error of the mean.

the true data-point relationships, whereas my F_β-measure explicitly seeks out the most locality preserving hyperplanes using supervision from the adjacency matrix $\mathbf{S}$. In Figure 5.6b I show the allocation for the SKLSH projected dimensions. In contrast to PCA, the bit budget is more evenly allocated across the 32 projected dimensions with a large proportion of the dimensions attracting one bit each with a smaller number of hyperplanes attracting two bits compared to PCA. This result is quite intuitive if we consider that SKLSH draws the hyperplanes randomly within the input feature space.

An attractive property of my variable threshold algorithms is the pruning that is performed on hyperplanes that exhibit a low locality preservation. Notice in Figure 5.6a that 18 out of 32 (56%) of the available PCA hyperplanes have been discarded (assigned 0 bits). This sparse solution is a form of dimensionality reduction that may be particularly advantageous to end-applications where there is limited memory to store potentially high dimensional and dense hyperplane normal vectors (for example, in embedded systems).

5.3.3.3 Experiment III: Branch-and-Bound versus Greedy Allocation

In this section I examine the second and final hypothesis of this chapter, namely that finding a threshold allocation using a branch-and-bound search (VBQ_{bound}) leads to a higher retrieval effectiveness than the greedy threshold redistribution algorithm VBQ_{greedy}. The retrieval results on CIFAR, LabelMe and NUS-WIDE are shown in



(a) **PCA projections** (b) **SKLSH projections**

Figure 5.6: The bit allocation assigned by VBQ$_{bound}$ for PCA (Figure (a)) and SKLSH projections (Figure (b)).

Tables 5.5-5.7. I conduct a Wilcoxon signed rank test to determine whether the difference in the AUPRC values for VBQ$_{bound}$ and VBQ$_{greedy}$ obtained over ten random dataset splits is significant. On the CIFAR-10 dataset for all considered projection functions (ITQ, PCA, LSH, SH, SKLSH) I find that the difference between the AUPRC achieved by both algorithms is *not* significant with $p < 0.01$. I find a similar result on the LabelMe and NUS-WIDE datasets for all considered projection functions. Taken together these results suggest that I *cannot* refute the null hypothesis and therefore I am not able to confirm hypothesis H_2 based on the experiments conducted in this section. From an efficiency standpoint this is an encouraging finding because it means we can benefit from the substantially faster training and allocation time of the VBQ$_{greedy}$ model while attaining a retrieval effectiveness that is statistically indistinguishable to the more computationally expensive VBQ$_{bound}$.

5.3.3.4 Experiment IV: Evaluation of Training Time

In this last experiment I will examine the training time of my variable threshold allocation models and compare the computational cost to the multi-threshold quantisation algorithm introduced in Chapter 4. The timing results in seconds are given in Table 5.8. I break the computation time into the time taken to *learn the thresholds* and the time taken to compute the *threshold allocation*. The multi-threshold quantisation algorithm (NPQ) introduced in Chapter 4 attains the lowest training time, which is not unexpected given it only learns thresholds for one particular threshold quantity (T) and

	Threshold Learning	Time (s)	Threshold Allocation	Time (s)
VBQ$_{bound}$	$O(T_{max}N_{trd}^2F + KN_{trd}^2)$	42.6	$O(2^{(B_{max}+1)K})$	0.134
VBQ$_{greedy}$	$O(T_{gdy}N_{trd}^2F + KN_{trd}^2)$	30.0	$O(1)$	0.001
NPQ	$O(K^{'}TN_{trd}^2F)$	3.60	–	–

Table 5.8: Threshold learning and allocation timings for the proposed models against NPQ. Time (seconds) taken at 32 bits (CIFAR-10, LSH) to grade the hyperplanes (*learning thresholds*) and the time then taken to solve the threshold allocation problem (*threshold allocation*). $T_{max} = K\sum_{b=1}^{B_{max}} 2^b - 1$ is an expression for the total number of thresholds, with $K^{'}T \ll T_{gdy} \ll T_{max}$, where the factor T_{gdy} depends on the specific choices made by the greedy algorithm during a particular run. $K^{'} = \lfloor K/B \rfloor$ denotes the number of hyperplanes used to generate K-bits for NPQ, with a constant allocation of B bits per projected dimension. N_{trd} denotes the number of training data-points, F the number of objective function evaluations (Equation 5.6). The timing results were recorded on an otherwise idle Intel 2.7GHz, 16Gb RAM machine and averaged over 10 random dataset partitions. All models are implemented in the same software stack (Matlab).

has no need to learn the allocation. In terms of threshold allocation time the VBQ$_{greedy}$ threshold allocation algorithm (Section 5.2.2.4) is *two orders* of magnitude faster than the binary integer linear program (VBQ$_{bound}$). Threshold learning is 30% faster for VBQ$_{greedy}$ compared to VBQ$_{bound}$ due to the greedy on-demand computation of the F_β-measure scores which ensures that there is no wasted computation.

5.4 Conclusions

In this chapter I have proposed two new algorithms for learning a *variable allocation* of quantisation thresholds per projected dimension. The intuition behind these quantisation algorithms was that the locality preserving quality of the hashing hyperplanes tends to vary widely within the input feature space. My central argument was that those hyperplanes that are better at maintaining the neighbourhood structure between the data-points should attract a higher quantity of the available thresholds, and vice-versa for lower quality hyperplanes. The quantity of thresholds assigned to a given projected dimension dictates how finely that dimension is quantised, with a greater allocation of thresholds contributing to a much less granular partitioning of the pro-

jected dimension. I argued in this chapter that the level of granularity, or equivalently the number of thresholds assigned per projected dimension, was an important factor in the ultimate retrieval effectiveness of the generated hashcodes. Quantising a projected dimension with a high locality preserving power with too few thresholds will run the risk of grouping together many unrelated data-points in the same quantised regions, therefore leading to a high number of false positives. To capitalise on the additional structure available in the higher quality projected dimensions, a higher number of thresholds should ideally be assigned to that dimension so that as many related data-points fall within the same quantised regions while minimising the number of unrelated data-points in the same regions. However, a quantisation of too fine a granularity may also result in a high degree of false negatives, that is many true nearest neighbours falling within different quantised regions. My contention was that there is a *sweet spot* for the number of thresholds that minimises the number of false positives and false negatives for a given projected dimension.

To learn the optimal number of thresholds I made two key contributions in this chapter: firstly in Section 5.2.2.1, I proposed the F_β-measure as a metric for grading the neighbourhood preserving quality of the projected dimensions and, secondly, in Sections 5.2.2.3-5.2.2.4 I introduced two new algorithms that used this scoring function to compute an allocation of thresholds subject to a total threshold budget. The variable threshold allocation algorithms tackled this NP-hard optimisation problem using two different strategies. My first proposed algorithm formulated the allocation problem as a binary integer linear programme (BILP) which was solved using branch and bound (Section 5.2.2.3). My alternative allocation algorithm greedily redistributed thresholds from the lowest quality hyperplanes to the highest quality hyperplanes (Section 5.2.2.4). To the best of my knowledge the research in this chapter is the first to introduce and solve the variable threshold allocation problem in the context of hashing-based ANN search.

I evaluated the proposed variable threshold quantisation algorithms in an extensive set of image retrieval experiments against the state-of-the-art baseline model introduced in Chapter 4. The key findings from the experimental evaluation were threefold:

- Allocating a variable number of quantisation thresholds per projected dimension using the F_β-measure as an indicator of hyperplane quality was shown, for certain projection functions, to lead to higher quality hashcodes and a significantly higher retrieval effectiveness compared to a uniform allocation of thresholds.

Section 5.3.3.2 and Tables 5.5-5.7 present the experimental results that support this claim.

- There is no significant difference (Wilcoxon signed rank test, $p < 0.01$) in retrieval effectiveness between a branch-and-bound based algorithm for solving the threshold allocation and a greedy threshold redistribution algorithm. The latter allocation algorithm is however *two orders* of magnitude faster at threshold allocation time and 28% *faster* at threshold learning. The quantitative results supporting this claim are presented in Section 5.3.3.4 and Table 5.8.

- There is no significant difference (Wilcoxon signed rank test, $p < 0.01$) between the retrieval results originating from the literature standard and improved dataset splitting strategies first outlined in Chapter 3, Section 3.5. This result accords with similar findings from my experiments in Chapter 4. Section 5.3.3.2 and Table 5.5 presents the results that validate this claim.

To the Computer Vision practitioner, the findings of this chapter hint at the following recommendations, dependent on whether an eigendecomposition is possible on the dataset of interest:

- If it is computationally tractable to compute a PCA projection on the dataset then a rotation of the feature space (ITQ) followed by the uniform multiple threshold quantisation algorithm of Chapter 4 leads to the overall highest retrieval effectiveness.

- If computationally constrained to other projection functions, such as LSH/SKLSH, then a variable threshold quantisation is highly advantageous to retrieval effectiveness compared to a uniform allocation. It is most likely intractable to compute PCA on the large high-dimensional datasets prevalent in real-world nearest neighbour search scenarios, and so this use-case of being constrained to random projections is perhaps the most common. To learn the thresholds and allocation the greedy approach (VBQ_{greedy}) is the most efficient while still maintaining attractive accuracy.

In this chapter and the previous chapter, I have studied the process of scalar quantisation for hashing-based ANN search in considerable detail, proposing a set of new quantisation algorithms that were found to significantly improve hashcode quality and the resulting effectiveness of query-by-example image retrieval. In Chapter 6 I will

now turn my attention to the related problem of *projection function learning* which fractures the input feature space with a set of K hyperplanes in a way that attempts to maximise the number of true nearest neighbours that fall within the same polytope-shaped regions of the space.