Exploring Qualitative Probabilities for Image Understanding

by

Jennifer Listgarten

A thesis submitted in conformity with the requirements
for the degree of Master's of Science
Graduate Department of Computer Science
University of Toronto

0-612-53388-3

Abstract

Exploring Qualitative Probabilities for Image Understanding

Jennifer Listgarten

Master's of Science

Graduate Department of Computer Science

University of Toronto

2000

In this thesis we explore and work with a particular probabilistic framework for image interpretation called Qualitative Probabilities. Introduced by Jepson and Mann, Qualitative Probabilities formalize the notion of non-accidentalness, a cornerstone of object recognition.

First we examine the search space associated with Qualitative Probabilities. We also experimentally verify one of the underlying principles of the theory, the asymptotic rate of 'accidents'. Then we incorporate Qualitative Probabilities into a relatively simple search which we find to be efficient and effective. Comparing search for interpretations using Qualitative Probabilities to search using a more standard 'cover' measure, we find that the former is far superior both in terms of efficiency and quality of block models found. Lastly, we design and test a new search algorithm, called Cascade search, that uses Qualitative Probabilities.

Acknowledgements

First and foremost I wish to thank my supervisor, Allan Jepson. His insight, patience, and his love of computer vision make him an invaluable mentor. Additionally, I would like to thank him for use of his lisp code which formed the core of the code used for this thesis, and for use of several of his pictures which appear in this thesis. I would also like to thank Richard Mann for helping to lead me down the right path for this Master's thesis, and Sven Dickinson for making helpful suggestions in his reading of my thesis.

I also wish to thank many of the students in Computer Science at the University of Toronto who have made graduate studies a pleasure. Without them I never would have lasted. In particular, I wish to thank my former office mates Natasa and Gerard, the guys in the lab, Chakra, John, Sean, Kiam, Faisal and Tom, as well as John W. and Iannis T.. I also thank Anna for her friendship, emotional support, encouragement and good cooking.

I wish to thank my parents for their constant support and love.

And Jude, I thank you for all.

Contents

Chapter 1

Introduction

One goal of computational vision is to get machines to arrive at interpretations of the world that are similar to the ones human observers would have arrived at. given the same visual stimuli. Photons incident on the retinas spawn a series of synaptic activities whose pathways and purposes are little understood. though heavily studied. The end result is that humans look around and know with great certainty what are coherent objects. how they are positioned relative to one another. how they are moving relative to one another. and very often. their name and purpose. Our perception of the world is near perfect despite noisy and missing data. occlusion of objects and varied lighting sources. In fact. our visual perception is so finely tuned that most people would be hard pressed to describe why the task of vision is difficult at all.

The task is amazingly daunting for the computational vision researcher. The specific problem of object recognition. a major component of the entire vision problem. is far from being solved.

David Lowe. one of the pioneers of modern day object recognition. nicely defines the problem of object recognition in his oft-cited. 1985 book on perceptual organization and visual recognition:

"Recognition implies that a correspondence has been found between elements of the image and a prior representation of objects in the world." [22]

Thus there are two essential elements involved in computational recognition:

1. sensor data which makes up an image
2. some previously defined notion of which objects exist. in a format suitable to being matched up with the sensor data. or derivatives thereof.

Some object recognition algorithms use laser range-finder data. while others use intensity images. Sometimes data are processed to form. say. edge-images. or 'line-drawings', which are then used to index into the model database. There are myriad combinations used of image data. data processing and types of models. each with unique advantages and difficulties with respect to the recognition problem.

The object model may be two-dimensional. three-dimensional. it may be purely geometric. or it may also include colour and other properties. A typical categorization of object models is i) CAD-based (*i.e.* geometric). ii) view-based (*e.g.* eigenspace). iii) primitive based (*e.g.* geon-based). We discuss each of these at greater length in the next section.

While some object recognition research is oriented toward fairly specific goals. such as face recognition. or detection of vehicles on a road. the work presented in this thesis is oriented toward "generic object-recognition". that is, recognition of previously unseen objects such as a new coffee cup. or a tree. This thesis explores object recognition from line drawings derived from real world images of simple objects. That the objects used are simple is a reflection of the state of the art in this area of research.

1.1 Contributions

In [20], Jepson and Mann introduce 'Qualitative Probabilities' for image understanding. They work in the domain of edge images derived from real images, formalizing the notion of non-accidentalness, a cornerstone of object recognition. This thesis builds on this work in several ways:

- By examining the search space associated with this particular probabilistic framework from the current literature (Qualitative Probabilities/QP [20]), we provide a deeper understanding of the framework. In doing so, we are also able to experimentally verify one of the underlying principles of the theory, the asymptotic rate of so-called 'accidents'.

- We compare search for interpretations using Qualitative Probabilities to search using a more standard 'cover' measure, and show that Qualitative Probabilities far outperform the cover measure. This further motivates and justifies use of the Qualitative Probabilities for the purpose of object recognition.

- We introduce a new search algorithm which uses the Qualitative Probabilities. Earlier searches using the Qualitative Probabilities required some preset, hard thresholds. This new search algorithm is more self-adapting, and no longer requires these thresholds.

1.2 Thesis Outline

- **Chapter 2** This chapter presents relevant background material.

- **Chapter 3** This chapter provides a thorough explanation of the Qualitative Probabilities presented in [20].

- **Chapter 4** This chapter explores the search space associated with Qualitative Probabilities in the context of searching for blocks from edge images. We also experimentally verify the asymptotic nature of the Qualitative Probabilities discussed in Chapter 3.

- **Chapter 5** This chapter explains how depth-first search is used to hypothesize interpretations for line drawings, given a model. Its purpose is to prepare the reader for Chapter 6.
- **Chapter 6** This chapter begins by showing that Qualitative Probabilities can be incorporated into a search algorithm in an effective, efficient way. We then compare how QP compares to a more standard 'cover' measure which is introduced.
- **Chapter 7** In this chapter, we discuss a new algorithm that uses QP to search for interpretations in edge images.
- **Chapter 8** This chapter contains a conclusion of the work presented as well as some future directions.

Chapter 2

Background

2.1 View-Based Approaches

In the introduction we framed the object recognition problem as one where image data are assigned to object models. For some approaches, usually called view-based approaches, the object models are derived from a set of training images. For example, eigenspace approaches approximate the training images with a linear 'object-space'. Then unknown images are projected into this object space to determine which object forms the closest match.

Limitation of these methods are that they are sensitive to changes in lighting conditions, and to translation, scaling and rotations [5]. Also, it is often assumed that the object has been segmented ahead of time, a key problem in generic object recognition. Lastly, these approaches often produce little or no semantic breakdown of the object being recognized. However, for tasks that are quite specific in nature, such as face recognition, eigenspace methods have proven to be very powerful.

For the purpose of this thesis, we leave these methods aside, and assume that our models are known ahead of time and are specified in terms of simple image features such as lines.

2.2 The Search Problem

Given that we know which models are to be used in trying to semantically parse an image and that these models specify the presence of particular image features, object recognition can be viewed, fundamentally, as a search problem. We have a set of image features, and a set of models, and we must search through a hypothesis space to find the right interpretation.

Sometimes this is posed as a labeling problem [16]. The interpretation space consists of every possible assignment of image feature to model, in every possible way. Suppose we have only ten objects in our database, and that each is modeled by describing the relative position and orientation of only five lines. Suppose that there are only 50 lines in total in the image. This gives a space of cardinality greater than $(10*5)^{50}$, a number so huge that the problem is already intractable in even such a simplified universe. Clearly some way of getting to the correct answer, without searching this whole space is necessary. Note also that the problems of imperfect data due to noise, occlusion, varied illumination, unknown viewpoint *etc.* have not even been mentioned yet.

Grimson and Lozano-Perez's 1987 paper [16] sheds some light on some of the problems and possible solutions to the search problem in this context. Grimson and Lozano-Perez try to identify and locate overlapping objects by modeling objects as polyhedra of up to six sides. The algorithm consists of a generate-and-test loop. Hypotheses are generated, tested for consistency based on geometric constraints, and discarded if found to be inconsistent. Thus they use a depth-first search, pruning entire subtrees when geometric inconsistencies are found. Extraneous data can be assigned to a null model. Finding this technique to be unacceptable in terms of time, they introduce several heuristics, including the following two:

1. *Hough Clustering* – A generalized Hough transform is used to reduce the number of poses that need to be examined, and thus a good portion of the search space

is pruned. Hough transforms work by collecting 'votes' for different poses based on local geometric constraints. Poses with the most votes are considered the most likely candidates. The authors find this crucial in reducing the search time, but note that it causes the algorithm to miss some correct interpretations occasionally. A theoretical analysis is given in [15, 14], where it is shown that this method is not reliable in noisy or cluttered scenes, or those scenes that have much occlusion.

A related method to Hough clustering is Geometric Hashing. Geometric Hashing works by storing transformation invariant properties of objects in a hash table. By calculating invariants from the image data, the table is then indexed to find possible corresponding object models. Those models that are indexed most often are the most likely candidates. Overviews of the technique are given in [14, 30]. Geometric Hashing suffers from the same types of problems as Hough transforms. In [2], Beis and Lowe try to improve on these indexing schemes by using a modified k-d tree search algorithm instead of a hash table. In [1], they propose learning a probabilistic indexing function to help disambiguate among indexed hypotheses.

Overall, it is not clear that these types of subspace localization methods based on voting for local image features scale up well with the number of models. This is due to the fact that the feature primitives used to index are relatively simple; simple features will potentially correspond to many objects. As alluded to in [10], more complex primitives are required for generic object recognition. More complex primitives make for less complex object models, and fewer matches to these models. However, these richer primitives are far more difficult to extract from the original data. The work in this thesis is a step toward being able to robustly and efficiently recover sufficiently complex primitives.

2. *Early termination* – When an interpretation is deemed sufficiently 'good', the search is terminated. Grimson and Lozano-Perez conclude that this is crucial to an efficient

solution. Goodness is judged by the total length of image edges in the interpretation. In fact, the key idea to take away from Grimson and Lozano-Perez's work is that using a rudimentary and *ad hoc* measure of 'goodness of interpretations', we can more effectively prune the search tree than with consistency alone. This leads us to believe that with a more rigorous and theoretically sound measure of goodness, we might be able to do much better. Qualitative Probabilities [20], which are explained in detail in Chapter 3, are such a measure.

2.3 Perceptual Grouping

The number of subsets of image features that can be formed is responsible for the combinatorial explosion of the search space. This in turn is a result of the number of image features, or in our case, the number of lines. This number could be reduced, if, for instance, we first grouped all collinear line segments that 'belonged together' into larger lines, and then only used these to index into the model database. Alternatively, we could group together lines that formed V's. This idea of grouping the more primitive features into higher order features is called perceptual grouping or perceptual organization and is acknowledged to be a necessary step in the recognition process [25]. Essentially its purpose is to uncover causal relationships between real world objects and image features, that is, to find image features that come from the same object in the real world. In the context of generic object recognition, we would use perceptual grouping to help us pick out complex primitives.

Finding these higher order features reduces the search in two different ways: i) By reducing the size of the total search space. ii) By allowing us to abandon certain paths (and thus subtrees) earlier on, because a more complex feature will match fewer models. A third reason for performing perceptual grouping is to help overcome noisy data and occluded objects. If we can correctly determine which image lines belong together, then given some prior knowledge about the world such as expectations of continuity, we may fill in the gaps in intelligent ways. Additionally, perceptual organization allows us to do some grouping that is independent of which particular object we are looking for.

Much work has gone into finding out what cues are good for perceptual grouping and how to reliably and efficiently detect them in images. We will next introduce this area of work with the Gestalt psychologists.

2.3.1 Gestalt Psychology

In the early twentieth century, before its use in machine vision, perceptual grouping was studied substantially by the Gestalt psychologists. Their main contribution was the design and execution of a large number of experiments dealing with grouping phenomena in humans. Their work forms the basis and motivation for much of the work in grouping by computer vision researchers. The upshot of the Gestaltists work was that grouping can be broken into six main classes [22]:

1. *Proximity* - elements that are closer together tend to be grouped together
2. *Similarity* - elements that are similar in physical attributes, such as color, orientation or size are grouped together
3. *Continuation* - elements that lie along a common line or smooth curve are grouped together
4. *Closure* - there is a tendency for curves to be completed so that they form enclosed regions.
5. *Symmetry* - any elements that are bilaterally symmetric about some axis are grouped together
6. *Familiarity* - elements are grouped together if we are used to seeing them together.

It is clear what an impact the Gestaltist's studies have made in the area of computational vision, as reference is made to them throughout the computer science grouping literature, [17, 22, 18, 26, 11, 29] to name a few. We should be careful not to attribute too special a meaning to these particular features. Though important, they are most useful as a guide rather than as necessary and sufficient properties for perceptual organization.

2.3.2 Non-Accidentalness

According to Saund [26], "The challenge facing the modern computational study of Perceptual Organization is to formalize and extend the gestaltists' intuitive insights in terms of testable theories and implementable programs." One way of formalizing the Gestalt laws is in terms of their non-accidental nature, which is closely related to notions of genericity and viewpoint invariance.

According to Lowe and many others, the most important property for a feature is non-accidentalness, that is, the property that the feature is unlikely to have arisen by accident. For example, we might wish to group image lines together if they are parallel, as the Gestaltists suggest that humans do. Intuitively we may reason that this is a good thing to do because if two lines in an image are parallel, they must either be parallel in the real world, or else we must be viewing the object from one particular view, that is, from a non-generic viewpoint, which is highly unlikely. One can argue in this way for each property the Gestaltists mention.

An equivalent way of stating this idea is in terms of viewpoint invariance. Lowe suggests that only features that are mostly viewpoint invariant should be looked at. Because a viewpoint invariant structure projects to the same set of image features, these features are more likely to occur, and when they do, are more likely to have arisen by some real 3D structure than by accident.

However, as shown formally by Jepson *et al* in [19], non-accidentalness alone will not lead to reliable inferences about the real world. One further criterion is that the feature needs to have a non-zero *a priori* probability of occurring in the given context. This falls out of the Bayesian formulation of the problem.

In [12, 13], Feldman proposes a formal, logical framework for capturing non-accidentalness. He calls it 'regularity-based' grouping. 'Grouping interpretations' are logical expressions, and a logical inference theory is presented to work with these expressions. Parse trees store the degrees of genericity present, and those interpretations with the highest genericity are considered the most plausible. These ideas are closely linked with Jepson and Mann's Qualitative Probabilities [20]. An entire chapter is devoted to this (Chapter 3), so it is not discussed further here.

2.3.3 Other Work in Perceptual Grouping

In Lowe's SCERPO system [22], straight lines are grouped together using collinearity, proximity of endpoints and parallelism. Only lines that are close enough together have the possibility of being grouped. A potential group is assigned significance inversely proportional to the likelihood that it is accidental in origin. This likelihood contains little information and is simply the ratio of i) the separation of the lines involved, to ii) the length of the shortest line segment involved.

Jacobs' GROPER system [17] estimates the probability that two convex contours come from the same object. The estimate is based on the distance separating the two contours, d, as well as on their relative orientation, t. Let $O_1 = O_2$ indicate that the objects that produced groups one and two are the same. Jacobs calculates $p(O_1 = O_2|d, t)$ through use of Baye's rule. By making certain assumptions and massaging equations, he in the end needs only to calculate four fairly simple probabilities (*i.e.* simpler than the ones required by Baye's rule alone). An approximation to these probability distributions is obtained by generating random polygons, and calculating several statistics.

Jacobs pursues his convex grouping further in [18]. He contends that convex collections of line segments, in which a large fraction of the convex hull of these segments is covered, are salient cues for detection of underlying structure. Later in this thesis, we show that coverage is actually not a particularly useful property on its own.

2.3.4 Primitive-Based Recognition

One class of approaches to the recognition/grouping problem that falls under the heading of primitive-based recognition is motivated by Biederman's Recognition-by-Components theory [4]. Biederman contends that "*primal access*, the first contact of a perceptual input from an isolated, unanticipated object to a representation in memory" is edge-based. He posits that humans understand images by parsing objects into their component shapes, 'geons', and conducts studies to back his theory. The geons are simple, parameterized, 3D shapes that he postulates are recognized by their 2D edges alone, based on qualitative measures, such as curved versus straight. He reasons that surface characteristics of objects play only a secondary role because they are generally less efficient for indexing into the human model base. He even goes so far as to say that the parsing into visual primitives "does not appear to depend on our familiarity with the particular object being identified", that is, it does not depend on context.

Some of the attempts to put this theory into practice can be found in [3, 7, 10], and a discussion of the successes and problems encountered in some of these attempts are given in [9]. In this panel discussion paper, the consensus is that Biederman's idea of parsing images into a finite number of visual primitives is extremely valuable. However, it is argued that many parts of the real world cannot be represented by geons alone. Also, the problem of extracting good line drawings from real images is considered to be extremely difficult, and many of the attempts at implementing Recognition-By-Components start with images that are completely noise-free.

Primitive-based object recognition is perhaps the method best suited to the task of generic object recognition, as it allows for very loosely defined classes of objects. An object is defined as consisting of several, of a finite group, of generic primitives, as well as a qualitative description of how these are interconnected. Though geons themselves might not be the basis required to represent prototypical objects, work in trying to recover them from images and ensuing object recognition has been enlightening, especially in terms of the types of algorithms used, which are mostly independent from the geons themselves (especially for the latter task of 'gluing' the primitives together).

In this thesis, we are on one level preoccupied with searching for blocks, however, the underlying goal and incentive is to provide a robust and efficient method of extracting complex primitives from real images, pushing us toward the goal of generic object recognition.

Chapter 3

Qualitative Probabilities for Image Interpretation

In the previous chapter it was shown that the property of non-accidentalness plays a central role in object recognition. It helps one deduce which image features are likely to belong to the same object in the real world, and therefore helps to make the search in object recognition more tractable. In Jepson and Mann's *Qualitative Probabilities for Image Interpretation* [20], the notion of non-accidentalness is formalized from a Bayesian point of view. The qualitative probabilities (QP) act as a measure of goodness for hypothesized image interpretations.

The theory is introduced in a 'card-world' domain, that is, line images are interpreted as a combination of sticks and convex, opaque polygons. For example, Figure 3.1a shows the input image of straight line segments, while (b), (c), and (d) show different global interpretations of the image. Using the QP metric, it is found that of the three hypothesized interpretations, (b) is preferred. This interpretation also corresponds to what we would intuitively describe as a correct interpretation of the scene, a property clearly desired from any metric in this context.

In this chapter we review the details of these Qualitative Probabilities and consider their application to simple images.

3.1 Bayes Theorem and Model Comparison

From a Bayesian point of view, in order to deduce correct interpretations for an image, one wants to be able to calculate the probability of an interpretation or scene model given the image data, $p(M|I)$, where M is a given interpretation or scene model and I is the image data. Models that produce a high value for this expression should correspond to the 'good' models. Examples of different M for the same I can be seen in Figure 3.1, where in (b), M describes the lines as a triangle in front of a quadrilateral, as well as a stick, whereas in (d), M describes each line as a simple stick, independent from the other line segments.

For the purpose of calculation, we must appeal to Bayes theorem:

$$p(M|I) = \frac{p(I|M)p(M)}{p(I)} \tag{3.1}$$

where, $p(M|I)$ is termed the *posterior*, $p(I|M)$, the *likelihood* of the model, and $p(M)$, the *prior* for the model.

Often, one is comparing different models for the same set of data, I, and thus the denominator is not needed. This is a lucky thing since

$$p(I) = \sum_{M} p(I|M)p(M), \tag{3.2}$$

that is, the probability of the image is equal to the summation over the probability of the data, given *every* possible model in the universe, a usually intractable calculation. When this term can be ignored, one computes instead the *unnormalized posterior*,

$$q(M|I) = p(I|M)p(M). \tag{3.3}$$

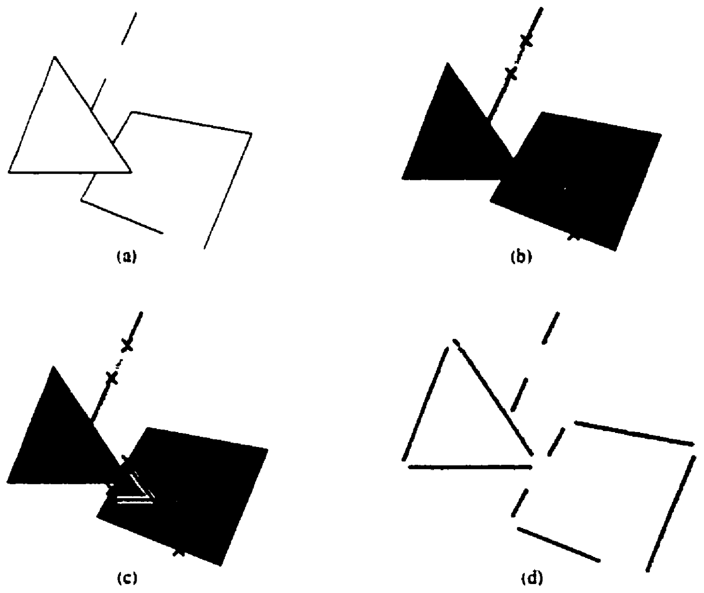


Figure 3.1: Example of image edges and three hypothesized 'card-world' models *Legend*: *Thin black lines* – image segments, *Thick grey lines* – sticks, *Shaded grey regions* – opaque polygonal card, *Crosses* – breakpoints in the image segments for sticks and polygon edges. (a) image edges (b) model consists of a triangle in front of a quadrilateral, and a stick (c) model consists of a triangle behind a quadrilateral, and a stick (d) model consists of 10 different sticks. Using the Qualitative Probabilities introduced in this chapter, we find that (b) is the preferred interpretation. (Figure courtesy of Allan Jepson [20])

Before explaining the details of QP, it would be nice to understand this equation on an intuitive level. We will do so by means of an example.

Suppose for instance that the image data we are looking at is (a) in Figure 3.1, and that the underlying true scene is shown in (b), a triangle in front of a quadrilateral, and a stick. As a first stab at finding the correct model for this scene, we might try to maximize the match between scene model and image data, that is, we would choose the model which maximizes the likelihood of the model, $P(I|M)$. Of all possible models, the one which would be selected in this case is the one in Figure 3.1d – a perfect match. But we know that cameras produce images with missing and noisy data and that edge detectors are imperfect. What is in the image is not a mirror reflection of reality. If we were to select a model based on the maximum likelihood criterion, we would need to select a very complex model, in fact, an overly complex model that could account for all of the noise. What we really need is a way of balancing the fit of the data to the model, with the complexity of the model. This is exactly what $p(M)$ in equation 3.1 does. It is the *prior* of the model, and reflects how probable the model is in its own right. Under normal circumstances, the more complex the model, the more unlikely it is, a principle sometimes referred to as *Okham's Razor*. This concept is closely related to ideas in coding theory, such as minimum description length. [1]

3.2 Qualitative Prior Probabilities

A common criticism of Bayesian statistics is that it is impossible to choose the correct prior. However, in practice the prior need not be specified perfectly. Priors that are 'guesses' and that may only be correct to within an order of magnitude still prove to be extremely useful. In their *Qualitative Probabilities* paper [20], Jepson and Mann define a wide equivalence class of prior probability *densities*, instead of selecting particular quantitative prior probabilities. From this class of densities, they are able to asymptotically analyze prior probabilities, resulting in a qualitative prior probability. They also show how to compute a qualitative likelihood to use with this prior. The qualitative probabilities presented are both intuitively pleasing and easy to compute.

First they set out the conditions imposed on the prior probability density for the occurrence of a single line segment. From this all prior probabilities for models in the specified domain follow.

[1]Curve fitting real, noisy data is similar. If we fit the curve that best matches the data, that is, the least squares solution, then we may end up with a very bad estimate of the underlying function because we will have chosen a curve of very high order to ensure that it goes through every point. We need to balance the fit of the data with the complexity of the curve in order to get a truly meaningful model of the data.

3.2.1 Prior Probabilities from the Probability Density

Let $p(L(\vec{x}_1, \vec{x}_2))$ denote the prior probability density for a particular line segment with independent endpoints $\vec{x}_1$ and $\vec{x}_2$. The conditions imposed on this density are that it is bounded away from zero and that it is bounded from above. That is, for some d_0 and d_1:

$$0 < d_0 \leq p(L(\vec{x}_1, \vec{x}_2)) \leq d_1 \tag{3.4}$$

Since the $\vec{x}_i$'s are continuous and since the density is bounded from above, integrating over any one set of line endpoints gives a probability of zero:

$$\int_{S_0} p(L(\vec{x}_1, \vec{x}_2))\, d\vec{x}_1\, d\vec{x}_2 = 0 \tag{3.5}$$

where S_0 denotes a particular pair of endpoints, $(\vec{x}_1, \vec{x}_2) = (\vec{x}_A, \vec{x}_B)$. In other words, the prior probability of a line whose endpoints have been specified to infinite precision is zero. To get a non-zero probability for a particular line, one can only specify it to some finite precision. Furthermore, because the density is bounded away from zero, *any* line segment specified with finite precision has non-zero probability of occurring. Both of these results satisfy our intuition about the problem.

To find out the prior probability of a line segment specified with finite precision, one integrates over the region of uncertainty. Suppose that one specifies the precision of a line segment with ϵ, that is, the position of each of the endpoints is known to within a radius ϵ. Then the prior probability of this line, given this resolution, follows from the bounds imposed on the density in Equation 3.4 (the bracketed expressions are squared because the uncertainty of each of the two endpoints must be integrated over):

$$0 < (\pi\epsilon^2 d_0)^2 \leq \int_{S_0^\epsilon} p(L(\vec{x}_1, \vec{x}_2))\, d\vec{x}_1\, d\vec{x}_2 \leq (\pi\epsilon^2 d_1)^2 \tag{3.6}$$

where S_0^ϵ denotes the set of line endpoints $(\vec{x}_1, \vec{x}_2)$ such that $\vec{x}_1$ and $\vec{x}_2$ lie within a disk of radius ϵ centered on $\vec{x}_A$ and $\vec{x}_B$ respectively (see next page)

(*i.e.* $S_0^\epsilon = \{(\vec{x}_1, \vec{x}_2) \mid (\|\vec{x}_1 - \vec{x}_A\| \leq \epsilon\|)$,
and $(\|\vec{x}_2 - \vec{x}_B\| \leq \epsilon\|)\}$).

Let us denote a model, M, specified to resolution ϵ to be M^ϵ, and the prior probability of this model, obtained by integration as in Equation 3.6, to be $p(M^\epsilon)$. In particular, denote the prior probability of a line segment, specified to precision ϵ to be $p(L^\epsilon(\vec{x}_1, \vec{x}_2))$, or simply $p(L^\epsilon)$. When no power of epsilon is shown, it will mean that the expression is a probability density rather than a probability. In this notation Equation 3.6 becomes:

$$0 < (\pi\epsilon^2 d_0)^2 \leq p(L^\epsilon) \leq (\pi\epsilon^2 d_1)^2 \tag{3.7}$$

According to the standard terminology in Computer Science [6], Equation 3.7 indicates that $p(L^\epsilon)$ is tightly bound by ϵ^4:

$$p(L^\epsilon) \in \Theta(\epsilon^4) \tag{3.8}$$

Similarly, one can show that the prior probability of a single endpoint, specified to resolution ϵ, is tightly bound by ϵ^2.

For the remainder of this thesis, we say that events that have probabilities tightly bound by some power of epsilon are "of order" that power of epsilon. For example, we will say that the prior probability of a line segment is of order ϵ^4. This makes for less cumbersome phrasing. Similarly, we will often omit the epsilon in $p(M^\epsilon)$ when it is obvious from the context, and instead write only $p(M)$.

They key ideas needed in deriving the prior probability for a single line segment were i) the bounds placed on the probability density and ii) integrating over each area of uncertainty. As presented, each area of uncertainty actually corresponds to two degrees of freedom in the model, each having the same resolution (proportional to ϵ). Each point in a plane has two degrees of freedom. Thus two independent points specifying a line have four degrees of freedom. This is intuitively very satisfying since our epsilon order estimates were just these numbers. Furthermore, this indicates that these QP's might be extended to more complex objects. This is done by keeping the same bounds on the probability densities, but integrating over the appropriate number of degrees of freedom.

To specify a convex n-gon to resolution ϵ, C_n^ϵ, one must specify n points in the plane at this same resolution. Following the analysis described above, it is found that the prior probability for such an n-gon, $p(C_n^\epsilon) \in \Theta(\epsilon^{2n})$.

What about complex objects, such as the projection of a block, where the number of degrees of freedom does not correspond to the number of vertices as is the case in a polygonal world? The orthographic projection of a 3D block is fully specified by four points in the plane. These four points are not unique, but this is unimportant for the QP analysis which tells us that the prior probability for a block, $p(B^\epsilon) \in \Theta(\epsilon^{4*2}) = \Theta(\epsilon^8)$. Again, one need only integrate over each degree of freedom, regardless of whether we know where specifically in the image they are located.

For a scene model consisting of multiple objects, Jepson and Mann take the shape and position of each object to be independent. Thus simply multiplying the prior probabilities of each object together gives the 'composite' prior probability for the entire scene model. For example, in Figure 3.1b, the prior for the entire scene is calculated by multiplying together the the prior for the triangle, $p(T^\epsilon) \in \Theta(\epsilon^6)$, the quadrilateral, $p(Q^\epsilon) \in \Theta(\epsilon^8)$, and the stick, $p(L^\epsilon) \in \Theta(\epsilon^4)$, producing a result of $p(T^\epsilon Q^\epsilon L^\epsilon) \in \Theta(\epsilon^{18})$.

The issue of depth ordering of objects in the scene has so far been ignored. Since objects in the specified domain are opaque and convex, depth layerings are reduced to binary choices between pairs of overlapping objects. Suppose we had a scene consisting of n objects, with m possible depth orderings. Suppose that the prior, before we take these depth possibilities into account is $p(M)$. Then the prior reflecting the number of depth orderings is $p(M)/m$, assuming that every depth ordering is equally likely. But since m is independent of the resolution, the ϵ order estimate remains unchanged.

Thus, from the simple assumption that the prior probability density of a line segment is bounded from above, and from below, away from zero, we obtain a simple but elegant qualitative expression for the prior probability of any scene model in the given domain. In the next section we will describe how these qualitative probabilities are extended so that the second term in Equation 3.1, the likelihood, can be computed, and thus the posterior.

3.3 Posterior Probabilities and Likelihood

In order to assess the likelihood of a scene model, $p(I|M)$, an *imaging model* [20] is needed. The imaging model should embody the types of errors that are typically made by either the camera or the edge detector, or both. Obviously one could have an imaging model at any level of detail. One might choose to physically model the camera's lens for example. However, in the given domain, the imaging process can be treated very simply. In fact, so that the likelihood and the prior may be combined in a meaningful way, one needs to make sure that their descriptions are of the same currency. Thus it would prove fruitless to have a more quantitative model than is set out in the prior probability calculations.

One of the errors accounted for in the imaging model presented in [20] is that various segments may be missing entirely, or in part, from the output of the edge detector. Jepson and Mann refer to these missing pieces as 'drop-outs'. Analogously to the prior calculations, the endpoint(s) of the drop-out must be specified. If an entire segment is missing, then, no additional points need to be specified (Figure 3.2b). If a single, middle segment is missing, two additional points need to be specified, each at a cost of ϵ^1 (since each is constrained to lie on the line) for a total of ϵ^2. If an end of a segment is missing, one additional point needs to be specified at a cost of ϵ^1 (Figure 3.2c).

The probability of a drop-out error is not only dependent on the probability of the endpoints, but also on the image contrast needed by the edge detector, which Jepson and Mann call δ. For every missing section of a line, the probability of that line is decreased

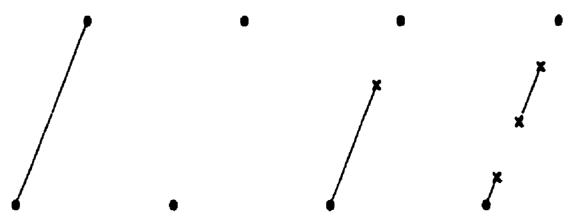


Figure 3.2: (a) (b) (c) (d)

Use of the imaging model to account for noise. (a) Perfectly imaged line: $p(I|M) = \epsilon^0\delta^0$, $p(M) = \epsilon^4$, $p(M|I) = \epsilon^4$ (b) Endpoints present, but entire line missing: $p(I|M) = \epsilon^0\delta^1$, $p(M) = \epsilon^4$, $p(M|I) = \epsilon^4\delta^1$ (c) End of segment missing: $p(I|M) = \epsilon^1\delta^1$, $p(M) = \epsilon^4$, $p(M|I) = \epsilon^5\delta^1$ (d) Middle and end of segment missing: $p(I|M) = \epsilon^3\delta^2$, $p(M) = \epsilon^4$, $p(M|I) = \epsilon^7\delta^2$. (Figure courtesy of Allan Jepson[20])

by a factor δ. Note that δ is independent of the length of the missing segment. Intuitively this makes sense since if an image edge does not have enough contrast over some length of the segment, it it likely due to shadow or poor lighting. There is no reason to believe that up to a certain size, the probability of a large shadow has a different asymptotic order than that of a smaller one. One can argue similarly for poor lighting. However, to get two drop-outs along the same line segment, one would need two processes, such as two shadows, which should be less likely than a single shadow. Thus each further drop-out becomes less likely. See Figure 3.2 for a few simple examples.

Another error accounted for by the imaging model is that there is only limited resolution. Because of this fact, two image lines resulting from say two different blocks, may by accident be extremely close and nearly collinear (see Figure 3.3). Thus the image line-finder will identify them as a single line segment, and we need to be able to 'use' only part of a line segment. A part of a line segment not being used to explain a particular model (when the rest of that segment is being used) is called a *tail*. In Section 5.5 we will show an example that discusses how tails affect computation of probabilities.

Lastly, errors in position and orientation are accounted for by the concept of a *cover*.

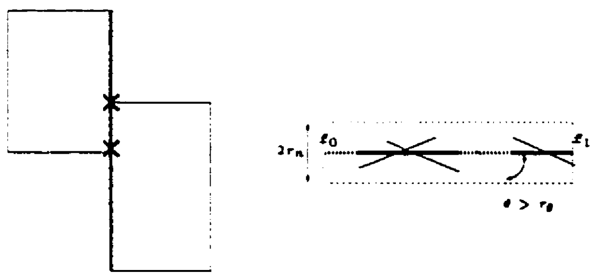


Figure 3.3: (a) (b)

(a) Two blocks abut, their edges forming a single image edge. (b) An example of a 'cover': angular tolerance is dictated by τ_θ, while tolerance of perpendicular extent is governed by τ_n. The baseline is the middle horizontal line. (Picture courtesy of Allan Jepson [20])

A cover is defined to be a rectangular box, of any length, and some pre-specified width. Associated with the cover is a baseline (see Figure 3.3b). A cover is a model for a real-world straight line; the model 'covers' the various imaged line segments that make up the true line (or *vice versa*). If several image line segments can fit inside of a cover, and the angle they form with the cover baseline is not larger than some maximum allowed angle, τ_θ, then they are considered to be a single line. This allows us to robustly group together image segments into single lines, tolerating errors in position and orientation. Note also that this definition allows parallel, or nearly parallel line segments to be grouped together, provided they are sufficiently close together.

An important and nice feature that comes out of the definition of covers is the following: no matter what order image segments are added to a cover, we will obtain the same result - yes they form a cover, or no they do not form a cover. Thus the 'growing' of covers, through the inclusion of additional image segments, is transitive.

3.4 Can QP be used for Search?

As alluded to at the start of this chapter, scene models with high QP values, correspond well to intuitively 'correct' models. Thus what QP has provided so far is a measure of the 'quality' of different *global* scene models for a particular set of image data. To see which model best fits the data, one needs to find, of *all* possible scene models, the one which maximizes the posterior.

But how can such a measure help in the search for the correct interpretation? As presented, one would first need to generate all possible scene models in order to find the maximal values of the posterior and the corresponding models. This is exactly the problem that must be avoided, since these hypothesis spaces are combinatorial, and a complete search through them is intractable. What is required is a means of evaluating *partial hypotheses*. Given such a mechanism, one could build hypotheses bottom-up, keeping around only the better interpretations to explore further.

3.4.1 Evaluating Partial Hypotheses

Can QP be directly applied to partial hypotheses to produce a meaningful metric? Suppose there is a line image in which it is desired to find all block interpretations (for example, see Figure 4.9). Building hypotheses bottom-up, one will at some point encounter two hypothesized interpretations, each consisting of a different set of lines. Let us call the two sets of lines, I_1 and I_2, where $I_1, I_2 \subseteq I$. If applied directly, QP tells us to compute the posterior as in Equation 3.1 for each of the two hypotheses. Since the two sets of image lines are not the same, $p(I_1)$ and $p(I_2)$ must each be computed as in Equation 3.2. These are unwieldy and practically speaking, impossible calculations. To overcome this hurdle, Jepson and Mann 'normalize' the posterior by other means. Strictly speaking it is not a normalization, but its desired functionality is the same, the desired functionality being that it allows one to compare different hypotheses of different data 'fairly'. How fairly is another question, one which is addressed for the specific case of a blocks world in Section 3.4.3.

Instead of taking the ratio of the unnormalized posterior to the probability of the data (*i.e.* the posterior), they take the ratio of the unnormalized posterior to *the unnormalized* *posterior of the same subset of data, I_1, under the most simple of scene interpretations - one which considers each image line to be a 'stick'.* The value of this ratio corresponds to the following: how much more likely is it that this subset of image data, I_1, was generated by a scene, M, as compared to a scene consisting entirely of sticks, S?

$$Odds(M) = \frac{p(M|I_1)}{p(S|I_1)} = \frac{p(I_1|M)p(M)}{p(I_1)} \frac{p(I_1)}{p(I_1|S)p(S)} = \frac{p(I_1|M)p(M)}{p(I_1|S)p(S)} \tag{3.9}$$

Toward the end of this section, we will use a detailed example in the blocks domain to demonstrate what a full computation might look like.

If the odds were an ideal metric, the most intuitively plausible interpretation would be the one with the highest odds. However, QP does not model all of reality, and consists of only order estimates. Furthermore, the odds are an approximation made to overcome the difficulties of calculating the posterior. In particular, they consider only one subset of image data, I_1, and not any further evidence that might be obtained from the remainder of the image data. Thus in practice, one cannot expect that the odds be ideal. Instead, it is important to note that if the intuitively correct model lies in the top percentage of all models, as ordered by QP odds, then a good heuristic will have been found. If this is not the case, all hope can be abandoned for this approach. In their experiments in a blocks world domain, Jepson and Mann find that the correct block interpretations do normally lie in the top one percent of block hypotheses [20]. This issue is also explored further in Chapter 6 and Chapter 7.

Given a small set of interpretations in which the correct one is known to exist, further quality checks can then be used to select the best among these top interpretations. Since the combinatorial explosion of the search space will have been battled and won, one can then spend the time and resources on more quantitative models. The beauty of QP is that it is simple, general, elegant and easy to compute.

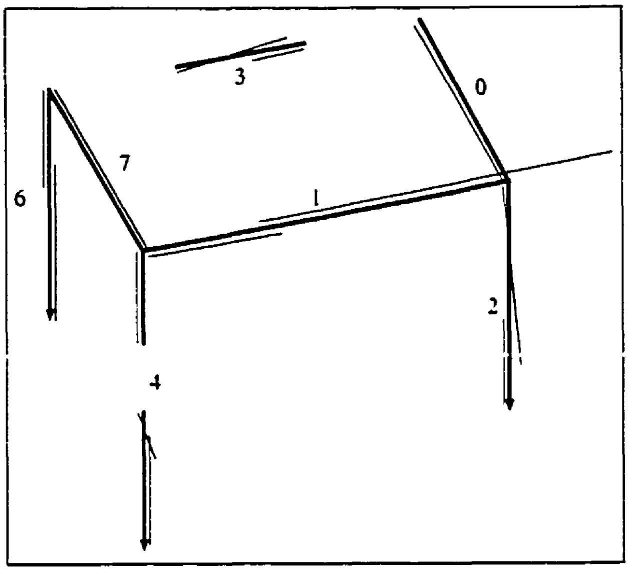


Figure 3.4: Example of image edges and a hypothesized block model *Legend*: *Solid bold line* - Block model (*i.e.* the baseline of the cover for each edge is shown). *Solid line* - Actual image edge *Arrow* - Indicates that the end position of that model edge is not known (*i.e.* it is not constrained, it is a free endpoint)

3.4.2 Example Computation of QP Odds for a Block

We will now show a full computation of the odds of a particular block model and particular image line segments: these are shown in Figure 3.4.

First we will calculate the asymptotic unnormalized posterior probability of a block model, given the image data, then we will calculate the asymptotic unnormalized posterior probability of an all sticks model, and, finally, the odds for the block model. In what follows we will call asymptotic probabilities just plain probabilities.

To obtain the prior probability of the block model, we note that there is one free parameter in the model shown - downward length, and that a fully specified block model has 8 known parameters. Thus the prior probability, $p(B)$ is given by:

$$p(B) \in \Theta(\epsilon^{8-1}) = \Theta(\epsilon^7) \tag{3.10}$$

Now we need to use the imaging model to find the likelihood of this block model, $p(I|B)$.

How many gaps are there? There is one at the end of each of model edges 2 and 6, and 2 gaps on edges 4 and 3. Thus there are 6 gaps, contributing a factor of δ^6 to the likelihood. Furthermore, each of the gaps in edge 3 each have one endpoint that needs to be specified, costing ϵ^1 each, as do the 'end' gaps on edges 2, 4, and 6. This adds a cost of ϵ^5, while the gap in edge four needs two endpoints specified, adding ϵ^2. Thus the likelihood for the model that we have calculated so far is:

$$p(I|B) \in \Theta(\epsilon^7\delta^6) \tag{3.11}$$

and the unnormalized posterior,

$$q(B|I) = p(I|B)p(B) \in \Theta(\epsilon^{14}\delta^6) \tag{3.12}$$

However, note that one of the image edges projects beyond the end of model edge 1 - the model edge has what we call a tail. How do we account for this tail in our computation? We must add something to our model so that the tail is accounted for. Thus we must add a term to both our prior and our likelihood. In other words, we need to multiply the unnormalized posterior in equation 3.12 by the unnormalized posterior for a 'tail model'.

Going back to our polygonal world, we could explain the tail by saying that it came from a stick with an uncertain left endpoint, lying along the image line in the sub-segment covered by the baseline. This is the worst case/highest cost scenario (barring gaps in the tail), and the unnormalized posterior would be of order ϵ^3 – we would 'charge' ϵ^2 for the known endpoint,and ϵ^1 for the uncertain endpoint. Alternatively, the tail could be explained by it belonging to some polygon. Since a polygon with n sides has unnormalized posterior ϵ^{2n}, the prorated unnormalized posterior of any one side of a polygon can be said to be ϵ^2. Yet another alternative is that the tail can be explained by it belonging to part of another block model, or some other kind of model entirely, unspecified and unknown. The higher the ratio of model edges to degrees of freedom of this model, the higher the prorated unnormalized posterior of a single edge, and the less of a 'penalty' incurred.

Clearly the problem is intractable - we do not know what to charge for the tail. The best we can do is take a guess at an appropriate choice. For the images we are using, not very many spurious edges (those not actually belonging to a true underlying block) are present. Since most edges in the images we use belong to a block, the prorated cost of an edge is, on average, $(\epsilon^8)^{\frac{1}{9}}$, because a block has 9 model edges. Thus, throughout this thesis, we will use a tail cost of ϵ^1.

Our final unnormalized posterior probability for the block model then, is,

$$q(B|I) \in \Theta(\epsilon^{15}\delta^6) \tag{3.13}$$

One last computation needed before we can obtain the odds, is the unnormalized posterior probability of the 'sticks' model, where each image edge is explained by a different stick. Using this model, each image edge has unnormalized posterior ϵ^4, and the whole image has unnormalized posterior $\epsilon^{4 \times number\ of\ edges}$. However, using this formulation, the probability of the stick model is relatively small when many image edges are present, and thus the odds of a block model can be relatively high for images where in fact little block structure is observed. The problem is that collinear image line segments, by themselves, are providing evidence of blocks, because there are no other linear processes in the domain. However, intuitively, we know that while a few collinear line segments provide some evidence of a block, this evidence should be at most incrementally more than the evidence provided by a single image line segment.

To remedy this situation, we introduce a second linear process. We do this by revamping our definition of "sticks". Sticks are now a new type of scene object, modeled in the same way as other scene objects. Rather than treating each individual image line segment as a stick, we now allow several collinear (or close and parallel) line segments to model a stick, accounting for gaps with the imaging model previously described. Some examples are shown in Figure 3.3.

To calculate our odds then, we use the *best*, all sticks model for the denominator in Equation 3.9. By best we mean the model that has the highest, asymptotic probability. This results in each set of collinear line segments being treated as a stick object, possibly with interior gaps (gaps that do not extend to an endpoint of the stick object). In turn, this means that any set of collinear line segments, on their own, provide no evidence for the existence of a block.

In our particular example, there are seven sets of collinear lines, two of which have interior gaps, thus the unnormalized posterior for the best all sticks interpretation is, $q(S|I) \in \Theta(\epsilon^{4\times 7}\epsilon^{2}\delta) = \Theta(\epsilon^{30}\delta)$.

Thus, the equation we've been waiting for, the odds of the block model shown in Figure 3.4, for the image edges shown is:

$$Odds(B) = \frac{p(B|I)}{p(S|I)} = \frac{q(B|I)}{q(S|I)} \in \Theta\left(\frac{\epsilon^{15}\delta^{6}}{\epsilon^{30}\delta}\right) = \Theta(\epsilon^{-15}\delta^{5}) \qquad (3.14)$$

Now that we have our odds, how can we use them? What meaning does an $\epsilon\delta$ term have? In order to use the odds, we must be able to compare different odds and say which one is bigger. A simple way of doing this is to say that for some constant, $q \geq 0$, as $\epsilon \to 0$:

$$\epsilon = \Theta(\delta^{q}) \qquad (3.15)$$

As it turns out, the specific value of q turns out not to matter in practice [20]. Thus for the remainder of this thesis, we treat $\epsilon \ll \delta$ (since the power of ϵ is negative in almost all cases, and since both ϵ and δ are less than one). In this thesis, we will actually ignore δ altogether. Also, since almost all block models will have odds of zero or lower, we will drop the negative sign on the epsilon odds throughout the remainder of this thesis. With this notation, the higher the power of epsilon, the more evidence we have for a block.

3.4.3 Odds for Single Block Interpretations

At the start of Section 3.4.1 it was mentioned that calculating the odds of the probability of some model to the probability of a sticks model serves as a sort of normalizing factor,

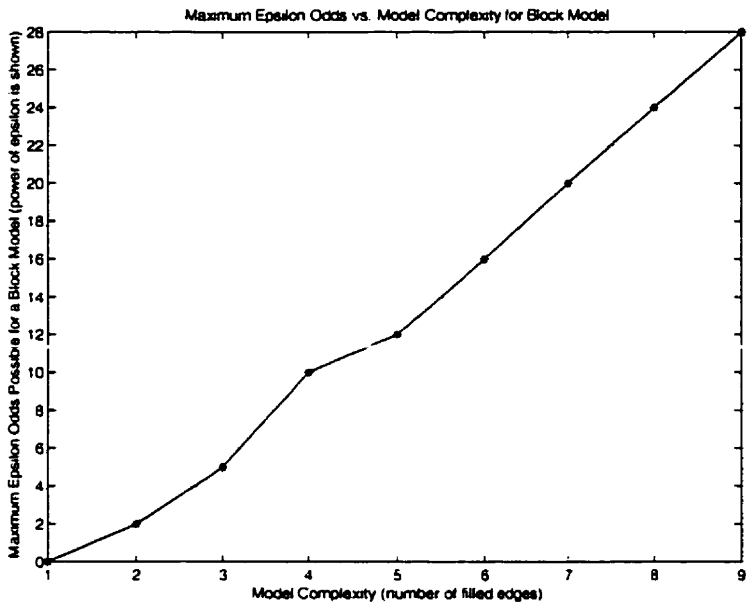


Figure 3.5: The maximum possible odds (power of epsilon) is shown, for blocks instantiated to different degrees, that is, one edge only, two edges only ...).

allowing for comparisons between different hypotheses of different data, in what is hoped, a fair way. Let us look a little more closely at how the odds behave for a blocks world to see just how fair this measure is.

Figure 3.5 shows the maximum possible epsilon odds for single, partial block hypotheses instantiated to different degrees, that is, for blocks that have only one model edge filled, two model edges filled *etc.* (A description of how to calculate the maximum possible odds is provided in Section 3.4.4). Immediately it is obvious that this 'normalization' is not fair. Clearly one cannot compare the quality of two hypotheses if they do not have the same number of block edges filled. Based on the definition of the odds, this intuitively makes sense: a hypothesis that consists of only one or two block edges can never be as convincing as the best one with five or six, no matter how good a two-edge hypothesis is present. This imposes some restrictions on how a search for true interpretations can proceed, which is discussed at greater length in Chapter 4.

3.4.4 Calculating the Maximum Possible Block Odds

The maximum odds in Figure 3.5 can be calculated in two ways, one by hand, the other automatically. To calculate the maximum odds by hand, we must realize that the maximum odds for a block model with a given number of edges occurs when the block is perfectly imaged. Thus it has no tails or gaps and $p(I|M) \in \theta(1)$ and all junctions are perfectly co-terminating. Given these facts, the problem of maximizing the odds is reduced to maximizing $p(M)$ (with a specified number of edges). To maximize $p(M)$, one needs to reduce the number of free ends, and have as non-generic a set of edges as possible. Thus by coming up with block models (*i.e.* hand drawing some 'image edges' in particular locations and orientations) that satisfy these constraints, one can determine the maximum possible odds. Alternatively, one can give an exhaustive search algorithm an image consisting of a perfect block, such as the one in Figure 4.1, run the algorithm, and then extract the highest QP odds blocks at each level of filled edges, noting the QP odds.

For this thesis, we calculated the odds by hand, and the double-checked the values using the search algorithm method.

Chapter 4

Exploring the QP Blocks World

From Jepson and Mann's results [20] it is clear that the Bayesian framework of qualitative probabilities can be used to efficiently explore the huge space of all hypothesized block models. Different algorithms can be used to traverse this space. In order to use QP to its potential and discover how large is this full potential, it would be instructive to take a look at the model space from the point of view of QP. In this chapter we do just that by contrasting the QP block hypothesis space for random line images, to the space for images containing blocks. In this framework, we are also able to experimentally prove the asymptotic nature of QP.

4.1 Basic Vocabulary

In this section we will introduce some basic vocabulary in order to facilitate later discussion.

Let I_N denote an input image consisting of N line segments (for example, Figure 4.9, or Figure 4.2). Let a subset of this input image be l_n, that is, a subset of n line segments. Let a block model, b_{l_n}, be an eight parameter model of a block (fully or partially constrained), which is a least-squares fit to line segments l_n, for $n \geq 1$. The line segments associated with the model must be consistent with a block. [1] A block model also specifies which edges in the model have been instantiated (since a block model may be only partial), and assigns labels to these edges. A formal definition of the labels appears later in this section. Figure 4.1 shows a fully instantiated block model with labels for each edges. Denote the set of all unique block models for image I_N as $\mathcal{B}_I$. Sometimes we will call this the model space for an image.

In order to understand what constitutes a unique model in this context we need to understand the symmetries that exist in a projected block model. Looking at Figure 4.1 which shows the edges of a block model as directed line segments, we see that there are essentially three classes of edges: those edges that point into a triple junction where the two other edges point away ($\{0, 3, 6\}$), those whose edges point into a triple junction where the two other edges also point into this junction ($\{1, 4, 7\}$), and those whose edges point into a double junction, where the other edge points out of the junction ($\{2, 5, 8\}$). These are the single-edge symmetry classes of directed block edges. Analogously, we see that pairs of edges may be formed into symmetry classes. For example, the pair $\{0, 2\}$ is equivalent to the pairs $\{3, 5\}$ and $\{6, 8\}$. In fact, all of the symmetry is captured by the labelling presented in Figure 4.1. In the following we will formally define symmetry.

Definition of Symmetric Block Labellings

A block labelling is an assignment of numbers and directions to image edges, as in Figure 4.1, and is represented mathematically as a function.
L : *image edge* $\rightarrow$ *(edge label, direction)*. For better clarity in the definition to follow, we break down the labelling function into two functions, L_l : *image edge* $\rightarrow$ *edge label*, and L_d : *image edge* $\rightarrow$ *direction*. The domain of a labelling is the set of image edges

[1]Consistency is determined by criteria such as Robert's criterion, which states that the three edges forming the triple junction in the orthographic projection of a block must all form angles which are obtuse or right with one another[24]. The other constraints are collinearity of line segments along a given edge, up to some tolerance, as described in Section 3.3, and that the least squares fit is sufficiently good.

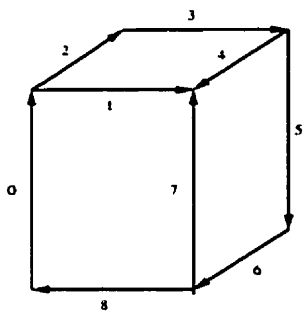


Figure 4.1: Single-edge symmetry classes of directed block edges: $\{0,3,6\}$, $\{1,4,7\}$, $\{2,5,8\}$

which it labels, and the range of L_l is an integer between 0 and 8, while the range of L_d is $\{0,1\}$. Two block labellings, L_1, and L_2, with edge domains $\{e_{11}, e_{12}, \ldots, e_{1n}\}$ and $\{e_{21}, e_{22}, \ldots, e_{2m}\}$ respectively, are said to be symmetric iff i) their domains are identical, and ii) $\exists i \in \{0,3,6\}$ such that $\forall j \leq n$,

$$((L_{1l}(e_{1_j}) + i) \bmod 9) = L_{2l}(e_{2_j}) \qquad \textit{and} \qquad L_{1d}(e_{1_j}) = L_{2d}(e_{2_j}) \tag{4.1}$$

Without loss of generality we have assumed in ii) that the domains are the same and that the orders of the edges in the domains of the two functions are identical. Note that the labelling function, L, is not one-to-one: several image edges may map to the same edge label.

Uniqueness of Block Models

Two block models are the same if they have symmetric block labellings. Thus if two block models are the same, they have the same set of labelled image edges, and these edges are either labelled in exactly the same way, or in a way that is symmetric with respect to each other. A block model is unique if it is not the same as any other block model.

4.2 The QP Hypothesis Space

A rigorous, experimental method to explore how QP behaves is, for a given image I, to conduct a full depth-first search through β_{I_t}, thereby encountering every possible model for this set of images. In the first part of this section we will report on the results from such an experiment, using ten randomly generated line images (see Section 4.2.1) such as the one shown in Figure 4.2. During the course of the depth-first search through β_{I_t}, the number of image lines added to a model increases with the depth of the search tree. At each depth step, another line segment is added. To be more precise, at each depth step, *each* remaining line segment is added in *every* possible way consistent with a block model. Thus an image edge could be added up to 18 (9 *edges* × 2 *directions*) different ways to a given block model, depending on how many of these ways were consistent with a block model. Tolerances on the errors allowed were the same as was used in [20].

What is important to understand at this point is not so much the details of this search algorithm, which is in no way proposed as an efficient search. Rather, we would like to emphasize that the end result of the search in this section is *a list of every possible block model*, (which, recall, includes both fully and partially instantiated blocks) for the image line segments on which the search was conducted. We use this complete set of block models to investigate how QP breaks it down according to different criteria through the use of histograms and graphs. We hope that by doing this we will gain a better understanding and intuition of QP in this domain, and perhaps this information will motivate an efficient search. A more detailed explanation of the search in this section can be found in Chapter 5, while a more sophisticated search is described in Chapters 6 and 7.

All results reported in this section are from aggregate data (*i.e.* summing the data from the set of ten random images).

4.2.1 Random Line Images and QP

The random lines were generated by first selecting a single endpoint, uniformly, at random over the image area. Then an orientation was selected uniformly over 360 degrees. Lengths of image lines were selected from a uniform distribution of lengths from 20 to 220 pixels (similar to the lengths of the lines in [20]). If the randomly selected length took the line outside of the image area, the line was not used. The size of the image was the same as those of the block images in [20], namely, 640x480 pixels. Each image contained 30 line segments, again, a number comparable to the images found in [20].

Figure 4.3 shows a bird's eye view of the full 3D β_l-space in several formats, on a normal scale as well as a $\log_{10}$ scale. Figure 4.4 shows slices through the 3D space. Figure 4.5 shows the same data yet again, this time collapsed along the epsilon odds axis so that the number of models at each number of filled edges can be seen.

4.2.2 Observations and Implications

A few observations should be noted about the plots just described:

1. From Figure 4.3 we see that the data is consistent with Figure 3.5 which shows the maximum possible epsilon odds for each number of filled edges in a model.

2. No real blocks are present in the random images used, yet there are still 218,899 models found, with epsilon odds going up to a power of 7. This indicates that accidents *do* happen. QP formalizes the notion of non-accidentalness, and tries to filter 'good' from 'bad' based on this property, but accidents will happen no matter how good a measure of non-accidentalness is used. This is not a problem specific to QP, rather one that is inherent in the property we are measuring.

3. Figure 4.5 and Figure 4.4 show that the most block interpretations occur when the number of edges filled is 2, 3 or 4. With fewer than 2 edges, the combinatorics of

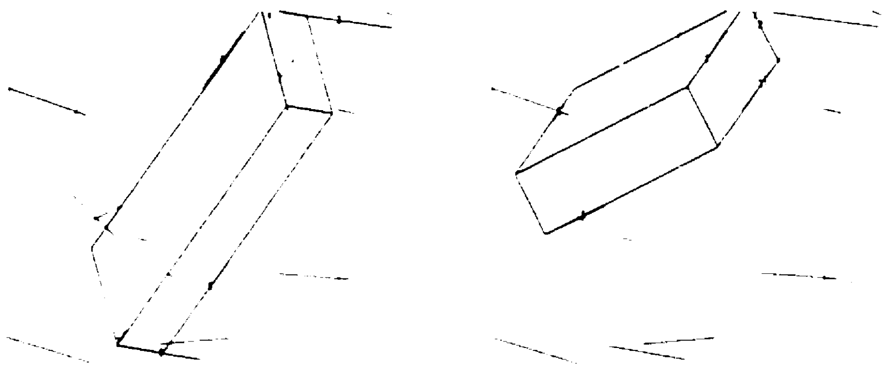


Figure 4.2: (a) (b)

Example of one of the random line input images. Shown are two different block hypotheses which clearly do not correspond to true blocks. *Legend*: Green lines are input data consisting of line segments. Red lines are those lines that have been assigned to a partial block hypothesis (some are covered in blue). Blue lines correspond to the portion of the hypothesized block model that is constrained. (Note that Roberts criterion (Section 4.1) did not need to be met exactly: the angle constraints were enforced, up to some predefined tolerance.)

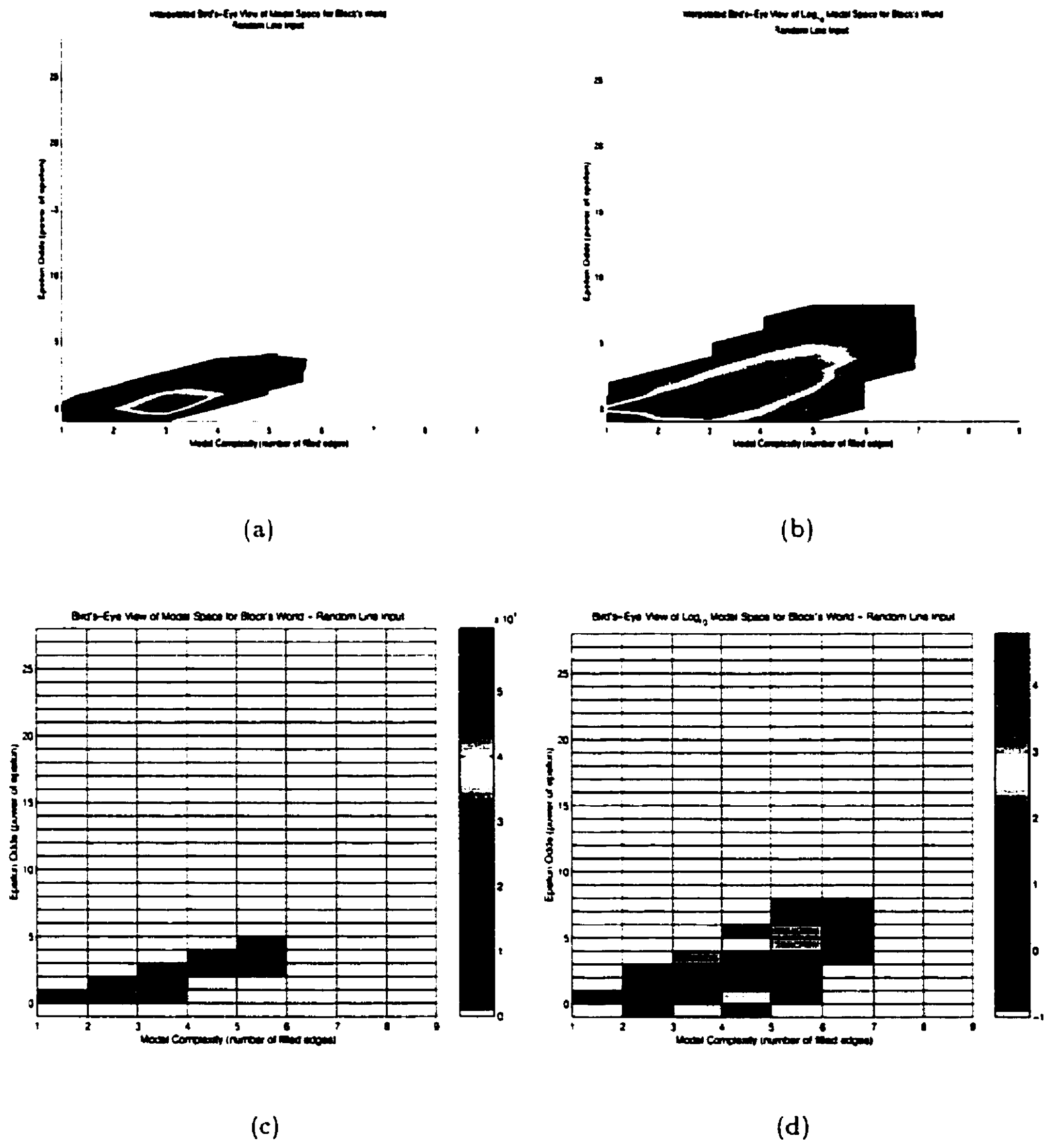


Figure 4.3: Number of models in β_{I_t} for aggregate data of ten random line images, broken down according to number of edges filled and epsilon odds. The top two images show interpolated versions, while the bottom two show the actual discrete data. The right most plots are $\log_{10}$ versions. Their bins have been forced to a minimum of -1.

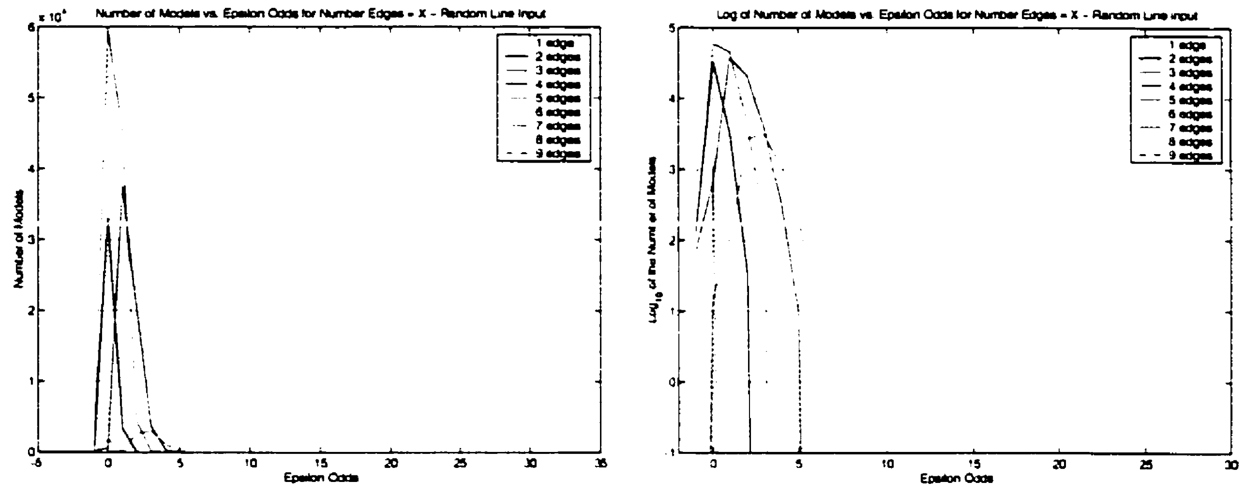


Figure 4.4: Same data as in Figure 4.3, but shown as slices of the 3D space.

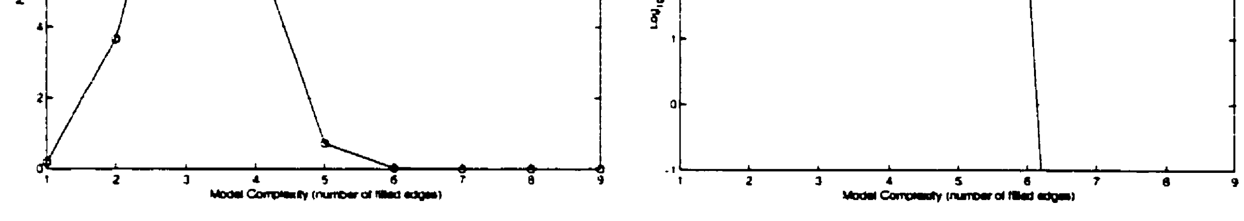


Figure 4.5: Same data as in 4.3, but collapsed along the epsilon odds axis.

selecting k image edges don't come into full swing, while with more than 5 edges, it becomes extremely difficult to find a large collection of lines that are consistent with a block model. In fact, no collections of size greater than 6 are found.

4. The test images used had only 30 lines. The number of accidents that occur should scale up in a very nasty way. We can see this by looking at the combinatorics. If we double the number of lines in the image from 30 to 60, the number of three-edge subsets goes from ${}_{30}C_3 = 4,060$ to ${}_{60}C_3 = 34,220$, [2] increasing by a factor of 8. The number of four-edge subsets changes from ${}_{30}C_4 = 27,405$ to ${}_{60}C_4 = 487,640$, increasing by a factor of 17. Since all three-edge sets and a large percentage of four-edge sets are consistent with a block model, this means that the number of interpretations should scale in a similar way to the number of subsets - an extremely worrisome prospect.

5. When we do the same analysis on images that actually contain blocks, we will see that there are still, relatively speaking, many accidental models with 2, 3, and 4 edges, and thus these are quite difficult to sift through. However, we also see that the high-odds tails of the curves in the right-most plot of Figure 4.4 extend further to the right since true blocks should score higher epsilon odds.

One might ask how sensitive these result are to changes in the tolerance of the block consistency check. This issue will be addressed in Section 4.3.

4.2.3 Block Images and QP

Using the six block images from [20] (shown in Figure 4.10), we ran the same experiment as in Section 4.2.1. An example of one of the input images along with two block hypotheses is shown in Figure 4.9. Figures 4.6, 4.7 and 4.8 are analogous to the plots presented Section 4.2.1.

[2]Here ${}_N C_r$ is the standard notation for 'N choose R', where ${}_N C_r = \frac{n!}{r!(n-r)!}$.

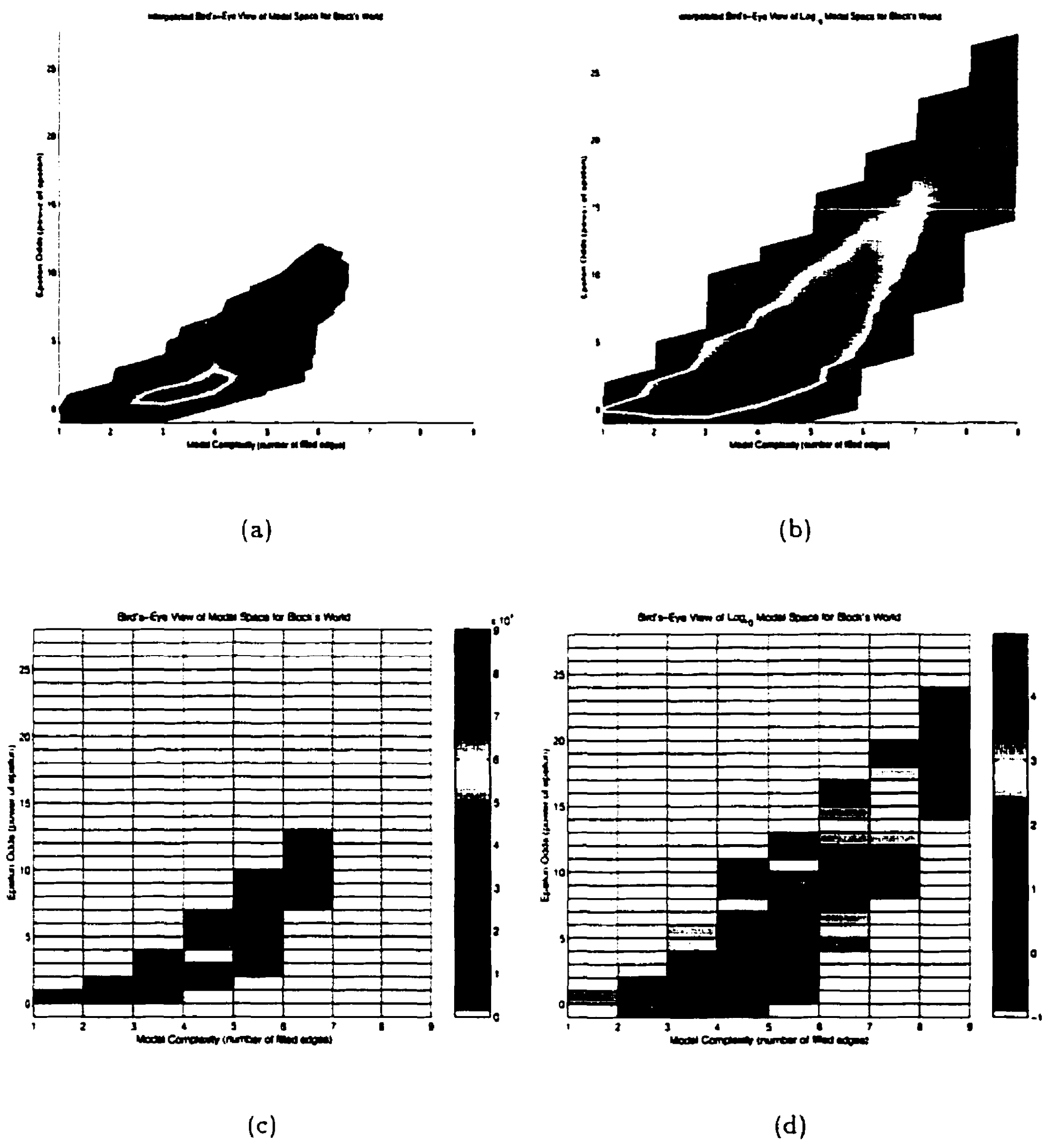


Figure 4.6: Number of models in $\mathfrak{Z}_{I_t}$ for aggregate data from six images from [20]. Data is broken down according to number of edges filled and epsilon odds. The top two images show interpolated versions, while the bottom two show the actual discrete data. The right most plots are $\log_{10}$ versions. Their bins have been forced to a minimum of -1.

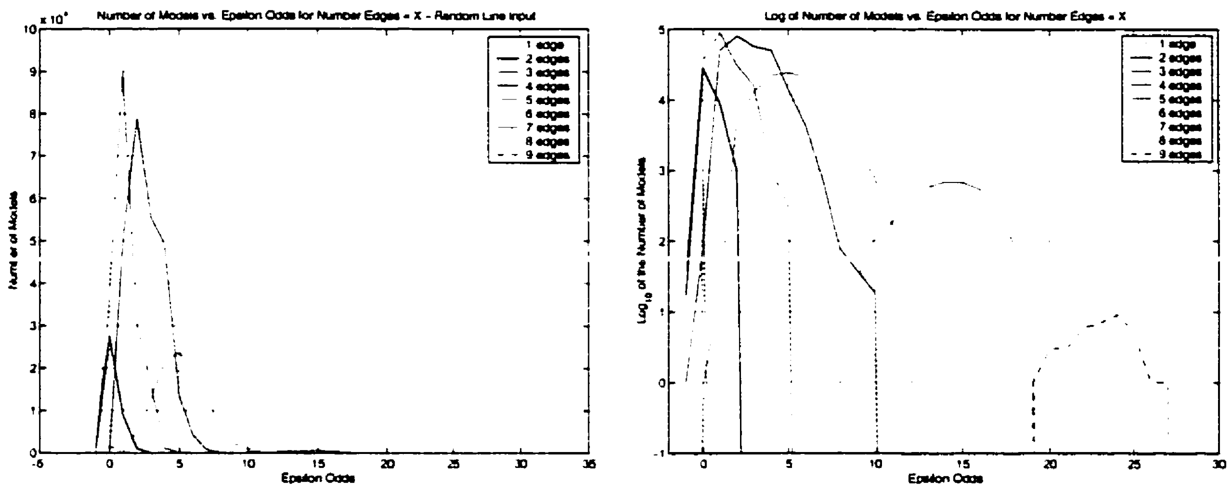


Figure 4.7: Same data as in 4.6, but collapsed along the epsilon odds axis.

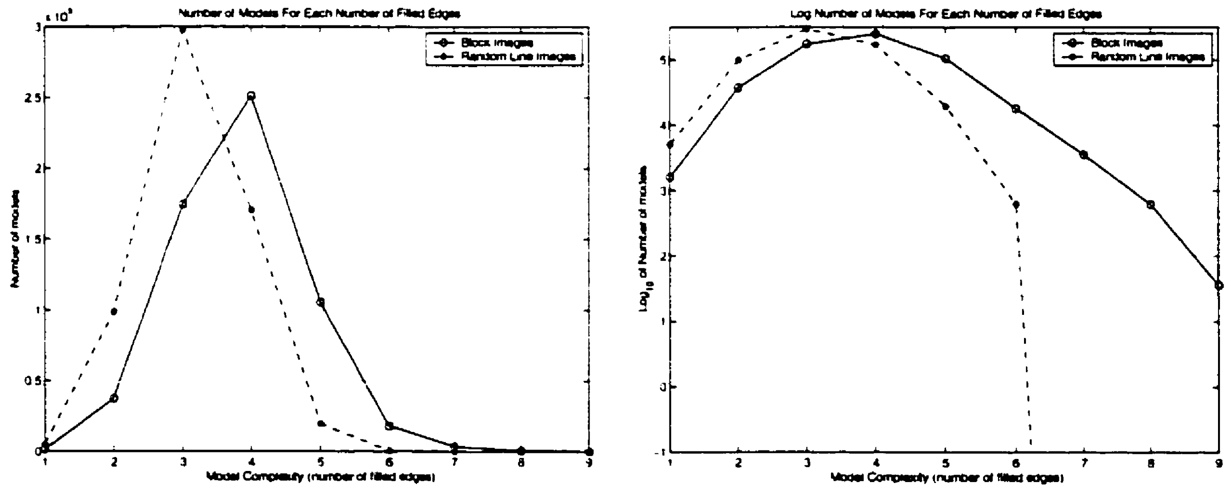


Figure 4.8: Same data as in Figures 4.6, but collapsed along the epsilon odds axis. Also shown for comparison is the data for random input images (Figure 4.5), which has been normalized so that the total number of models found is the same for both plots.

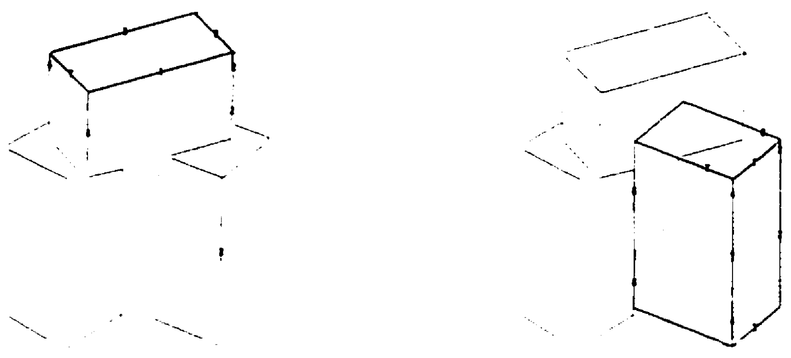


Figure 4.9: Examples of block line images from [20]. *Legend*: Green lines are input data consisting of line segments. Red lines are those lines that have been assigned to a partial block hypothesis (some are covered in blue). Blue lines correspond to the portion of the hypothesized block model that is constrained. Note that each of the blocks models has a 'correct' subset of block lines, as well as one or more that do not belong to the true block. Nevertheless, both sets of lines are consistent with a block model.

4.2.4 Observations and Implications

1. Again, the epsilon odds are consistent with Figure 3.5, which shows the maximum possible epsilon odds for each number of filled edges in a model.

2. We see in Figure 4.8 that there is an enormous concentration of models with 3, 4 and 5 edges, as opposed to 2, 3 and 4 in Figure 4.5. Relatively speaking within each plot, there are now far more models with more than 5 edges. This is simply an indication that there are now some true underlying blocks present in the images.

3. As predicted, the tails in the right-most plot of Figure 4.7 are far longer than in Figure 4.4. This is promising since these high odds interpretations must be a result of having real blocks in the image. Thus we have evidence to suggest that QP is working correctly in that it assigns high odds to a significant portion of correct block models, and that it does not assign very high odds to spurious block models. Furthermore, it is likely that small variations of the true block models produce

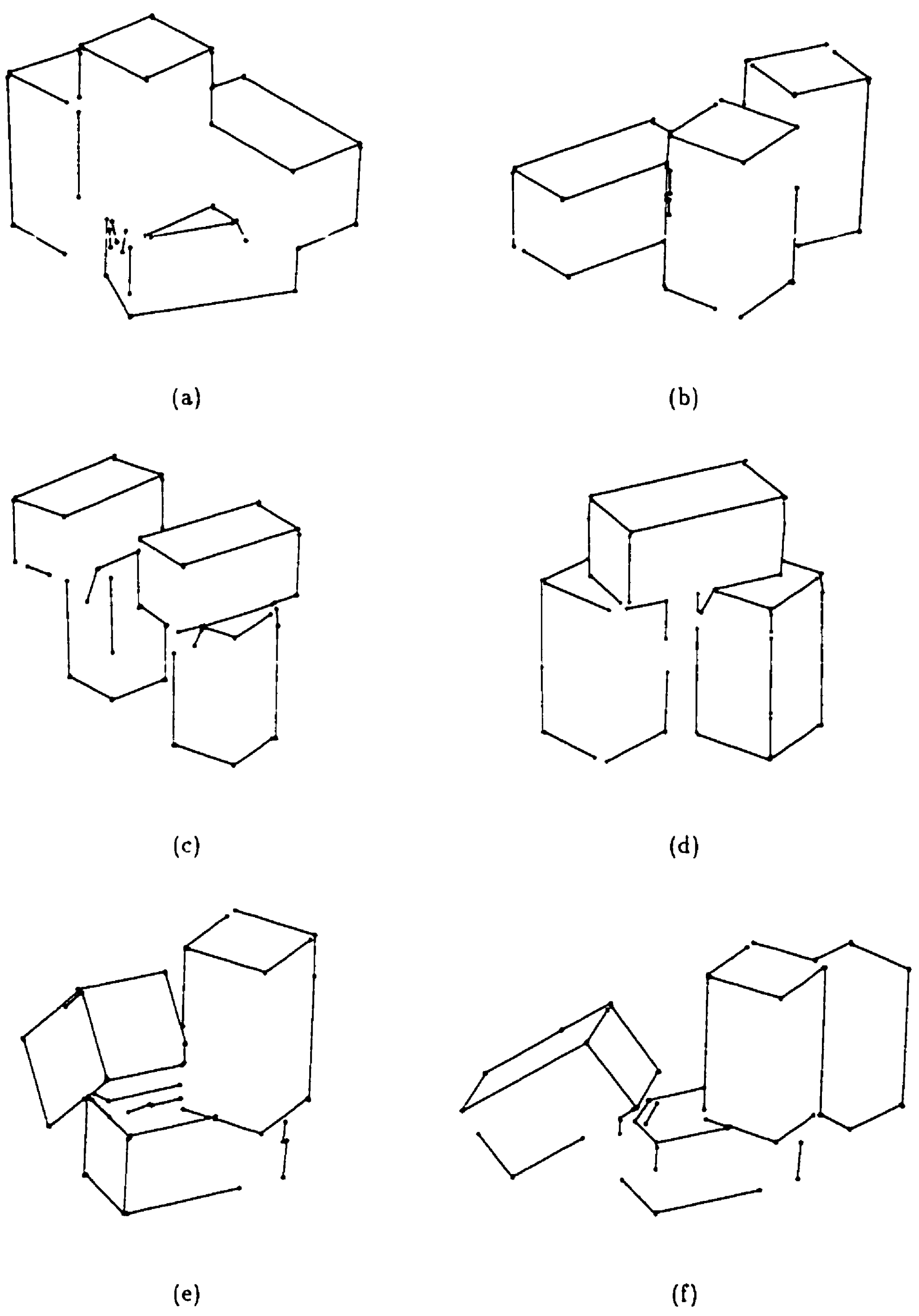


Figure 4.10: Edge images used in [20] and in this thesis. (a) im4, (b) im5, (c) im7, (d) im9, (e) imd, (f) ime

lower odds models, along with the 'accidents', thus making for many more models overall.

4. In Figure 4.7, it is apparent that the number of edges in a block model dictates what kind of epsilon odds it can have. Distinct windows of odds for each number of edges is present in the right-most plot. In particular, every window has a distinct lower bound for the epsilon odds. Why is this? The larger the set of lines that is consistent with a block model, the less likely this set is to be completely spurious, and thus the higher the odds. Suppose that we have three edges that instantiate a block model, and that they are neither parallel, nor co-terminating with one another (any three lines are consistent with a block model). Then there are no 'accidents' occurring between these lines; they are completely generic. If we add a fourth block edge to this set, and the resulting set is still consistent with a block model, then it is not possible that all four lines are generic. There *must* be some accident, such as that two of them are parallel or co-terminating. Because of this new non-degeneracy, the odds will increase by an amount proportional to the 'degree' of the non-degeneracy introduced. Note that if only one accident (loosely speaking) is introduced, like parallelism, then this new 4-edge block will be the lowest possible odds model, yet the odds will have systematically gone up from the 3-edge interpretation. Similar arguments can be made for larger groups of lines, based on how constrained a block model is at a particular number of edges. Any block that has six or more edges is fully constrained. Adding another edge necessarily introduces more non-degeneracy (provided the edge is consistent with a block), and thus the lowest possible odds are increased. In practice, the 'tail-costs' mentioned in Section 3.3 make some of the lower bounds lower than one might actually think.

5. The distinct windows of odds further corroborate our earlier point that only hypotheses with the same number of instantiated edges can be fairly compared for

the purposes of search.

6. The window for the three edge interpretations is one of the narrowest, indicating that the different three edge-interpretations do not differ significantly in quality, according to QP. Since the most number of interpretations have three edges, and since many 'accidents' occur with three edges, this makes them even harder to sift through.

4.3 Verifying the Asymptotic Nature of QP

Recall that in Section 3.4, the spatial resolution parameter, ϵ, was introduced. We said that the position of planar model points was known to within a disk or radius ϵ. From this we were then able to calculate ϵ-order estimates of the posterior probability of various models, given the image data. In this section we set out to experimentally verify the asymptotic predictions made by QP.

The prior probability of a block model, B^ϵ, in a random line edge image, according to QP, is proportional to its epsilon odds (Equation 4.2). The reason this should be is that the odds are directly related to estimates of how often certain accidents are likely to occur, such as two lines coterminating, *etc.*. Thus, if we randomly generate lines, we would hope to observe these accidents with the prescribed probability, and hence block models with probability proportional to the epsilon odds.

$$p(B^\epsilon) \in \Theta(\epsilon^k) \tag{4.2}$$

How can we verify that this is so? Suppose we had a camera with different resolution settings. Then if we captured the same images at several different resolutions and did a full depth-first search through the models, we could see exactly how the number of models changes with changed resolution. Equivalently, we could use the same random line images used in Section 4.2.1, changing the resolution parameters in the search code rather than the resolution of the actual images. Such parameters would include the tolerance allowed for two line segments to be collinear, the angle tolerance for two lines to be parallel, the tolerance for the least-squares fit, and the tolerance for co-terminating lines. Each of these parameters is either a linear function of ϵ or can be approximated as such. Thus by changing ϵ, we change each of them by an amount proportional to ϵ. Since the images were randomly generated and have no inherent resolution (the random lines had real-valued endpoints), the results should be the same as if we had captured the images at different resolutions. This second experiment is the one we opted for because of its convenience, and will now be discussed.

4.3.1 Theoretical Predictions

As the resolution increases our intuition tells us that the number of models found should decrease. QP predicts exactly how this number should decrease, in the limit as $\epsilon \to 0$. Let the number of block models with epsilon odds of order ϵ^k be denoted by $N(\epsilon^k)$, then in the limit $\epsilon \to 0$,

$$N(\epsilon^k) \propto p(B^\epsilon) \in \Theta(\epsilon^k). \tag{4.3}$$

Therefore, so long as ϵ is sufficiently small, [3]

$$c\epsilon^k \leq N(\epsilon^k) \leq C\epsilon^k, \tag{4.4}$$

for some c, C, such that $0 < c \leq C$. In order to experimentally verify the asymptotic behaviour, it will be convenient to have the log of this expression:

$$\log_{10} c \leq \log_{10}(N(\epsilon^k)) - k \log_{10} \epsilon \leq \log_{10} C \tag{4.5}$$

[3]Equation 4.4 is obtained as follows. Since, for some m, $N(\epsilon^k) = mp(B^\epsilon)$, and for some d_0, d_1, where $0 < d_1 \leq d_2$, $d_0\epsilon^k \leq p(B^\epsilon) \leq d_1\epsilon^k$. Thus it follows that $md_0\epsilon^k \leq N(\epsilon^k) \leq md_1\epsilon^k$. Setting $c = md_0$, and $C = md_1$, we obtain Equation 4.4

Now suppose we change the resolution, ϵ, by a factor of ρ, so that $\epsilon' = \rho\epsilon$, and let $d = \log_{10}(c\epsilon^k)$ and $D = \log_{10}(C\epsilon^k)$, then

$$d \leq \log_{10}(N(\epsilon'^k)) - k\log_{10}\rho \leq D \tag{4.6}$$

Thus, if we conduct the experiment described above, and then plot for a given epsilon odds, $\log_{10}(N(\epsilon'^k))$ as a function of $log_{10}(\rho)$, we should obtain a result bounded above and below by straight lines with slope equal to k. Next we will discuss exactly how we conducted the experiment.

4.3.2 Analysis and Results

Figure 4.11 shows the results of the experiment described at the start of this section. The data for this figure was generated as follows: Ten line images were generated randomly as described in Section 4.2.1. A depth-first search was performed on each of these images, for values of $\rho = 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}$, in order to obtain all possible models (as described in Section 4.2). Then, for the results of each of these fourty searches, histograms were calculated, with each bin counting the number of models for a given epsilon odds, ignoring how many edges were present. We will call the bins counting the number of $odds = x$ models, the '$odds = x$-bins'. We will call searches that were performed with the resolution scaled by a factor $\rho = k$, the '$\rho = k$-searches'.

To calculate, for example, the left-most point from the red line in Figure 4.11, the $odds = 0$-bins from each of the ten histograms that resulted from $\rho = 1$-searches were averaged. Call this average value, avg. The data point then is $\log_{10}(avg)$. The error bars on each data point is calculated as follows (see [23] for details): First calculate the standard error of the mean of avg and call this value $stderr$. Then the error on the plotted data point is the fractional error of the average, that is, $\frac{|errorbar|}{2} = \frac{stderr}{avg}$. Each line drawn in Figure 4.11 is a weighted least-squares fit, using the inverse error as a weight ([23]). The slopes presented in the figure are the slopes of the fitted lines, and

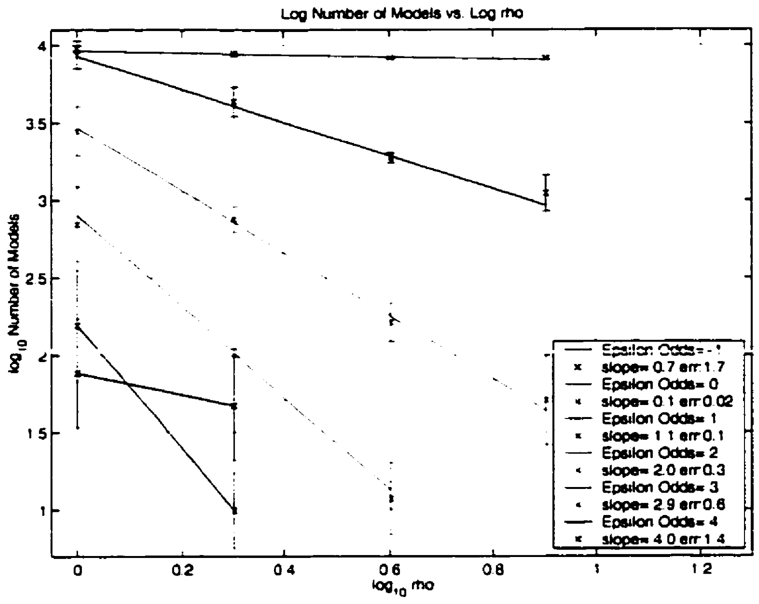


Figure 4.11: Results confirming the asymptotic behaviour of QP. Each line corresponds to the number of models found, at different resolutions, for a given epsilon odds. The slopes match the epsilon odds as predicted, except for epsilon odds of zero..

the errors on these slopes fall out of the weighted least-squares formulation ([23]).

Data points that have fewer than three models are not shown, nor are lines that have fewer than two data points. Though the odds do not go very high because random line images were used, we nevertheless have excellent agreement with theory. All lines except for epsilon odds of zero have slopes within error of their predicted values as dictated by Equation 4.6. For epsilon odds of both 4 and -1, there are not very many models, and the error bars are thus larger. Note that the odds -1 line is only just within error. Any model that has an odds of -1 necessarily has been penalized a tail cost (see Section 5.5). The part of the QP theory involving tail-costs is not as clean as the rest of the theory. Recall that the tail cost used was a bit of a guess – it was derived from guessing what the underlying explanation might be, on average, for the tail. Thus any theoretical predictions for models where the tail cost dominates should be taken with a grain of salt (just a small one). Only when the ratio of tails used to accidents present(*i.e.* parallelism, co-termination, collinearity) is small, can the tail cost dominate in this way. This happens in only a small percentage of cases and thus we needn't worry too much if our predictions are off in such circumstances. Though the error for the odds 4 line is also high, notice that the slope matches the theory without even taking the error into account. In other words, our experimental points with odds equal to 4 bounced around quite a bit, but they always bounced around the true theoretical value.

Thus we have convincingly shown that QP is asymptotic in practice as well as in theory, and that the theory and ensuing predictions form a sound basis upon which one may build. Moreover, the resolution used (*i.e.* for $\rho = 1$) appears to be within the asymptotic regime both in [20] and in subsequent sections of this thesis.

4.4 Resolution in Computational Vision

In Section 4.2 we showed that accidents happen - lots of them! We claimed that no matter how careful a measure of non-accidentalness can be found, that accidents will occur. Why then doesn't the human visual system have 'accidents'? Is it that we *do* have such accidents but that somehow we have evolved such a fantastically good measure for sifting through interpretations that we efficiently and effectively sift out these accidents? Perhaps. There is also another possibility, one not often discussed in the vision community.

The resolution in typical computational vision systems is far worse than human foveal resolution. If we used images with resolution as high as the human fovea, we could cut down tremendously on spurious hypotheses. However, this would be at a cost of increased computational time because of the increased number of edges that results from increased resolution. Presumably a human with full cognitive capabilities, in particular visual attention ([27, 28]), can avoid this trade-off by seeing in a more intelligent way, looking where it needs to when it needs to.

An interesting study of exactly how resolution affects false target rates can be found in [8]. By individually varying a number of 'basic parameters' related to resolution, such as focal length of the camera, separation of object points, *etc.*, Dickinson *et al* show how view degeneracy increases with decreased resolution. They also compare their table-top vision system view degeneracy rates with human view-degeneracy rates, showing that the two are order of magnitudes apart.

Chapter 5

Searching for Models

Chapter 4 briefly talked about using a depth-first search in order to generate all possible block models for a given image. Since search is a central part of the object recognition problem, we feel it is worthwhile to devote an entire chapter to properly describing how a simple depth-first search can be used to find block models from image line segments. Understanding this search is also crucial because it forms the basis for all other searches presented in this document.

In this chapter we will discuss how a depth-first search can be used to build up block models from image line segments, and exemplify this search in a simple, alphabet world domain.

5.1 Building Blocks

Recall from Section 4.1 that a block model consists, in part, of a labelling of some image edges describing which part of the block model the edges belong to, as for example in Figure 4.1. Keep in mind also that a block model may be only partially instantiated, that is, some edge labels between 0 and 8 may be missing because no image line segments have been assigned to these parts of the block model. Thus a block model may have anywhere between zero and nine block edges present, as a result of assigning any number of image edges to the nine block model edges.

One way of building up a block model from image line segments is to choose some image edge, assign it to some part of the empty block model (a model that has no edges assigned to it), and then to try adding, one after another, other image edges, in every possible way (*i.e.* every possible labelling), checking to make sure that the set of labelled edges is consistent with a block. At each step of this building process we have, by definition, a block model. When no further image edges may be added to a particular block model, (because of either a lack of consistency or a lack of further edges) the block model is said to be a *maximal block model.* If we start building from one particular edge, we may arrive at several different maximal block models, though it is unlikely that more than one of them corresponds to a true underlying block in the image. [1]

In the building up process of a block model, we said that all edges are added in every possible way to a particular model, so long as they are consistent. But during the control of the building process, how can we know which image edges have been tried and rejected, and which ones have simply not been tried yet? We do this by keeping a list of frozen edges with every block model that we build. A *frozen edge* is defined to be any edge that we have not yet tried to add to the model, and is defined in relation to a particular model, in the context of the building up process. Thus a particular model will have associated with it a *frozen list*, consisting of possibly many frozen edges.

The simplest (though not efficient) way to systematically build up block models from a given set of image line segments is to use a depth-first search to control the building process. In Section 5.2 we will write down an algorithm for depth first search for models, introducing new terms and concepts along the way.

[1]To get two maximal, correct block models, starting with the same initial edge segment, the two blocks would need to share an image edge, as would happen if the two blocks abutted one another, for example see Figure 3.3.

5.2 Depth-First Search

Algorithm 1 Depth-First Search with a Stack

```
stack.push(root)
while stack.notEmpty() {
   currentNode:=stack.pop()
   visit(currentNode)
   for every child, u, of currentNode {
       stack.push(u)
   }
}
```

A standard way of implementing a depth-first search and one convenient for describing the task at hand is to use a stack ([21]). Recall that a stack is a first-in-last-out data structure. To place an item on the stack, we say that we *push* the item on to the stack. To remove an item from the stack, we say that we *pop* the item from the stack. Any recursive algorithm can be written non-recursively using a stack; in particular, depth-first search can be written using a stack. For a tree (a directed, acyclic graph [6]), we conduct such a search starting at the root as shown in Algorithm 1. Later in this chapter we will show a picture of this algorithm in action.

When searching for block models from a set of image line segments, each *node* of the search tree consists of a block model and an associated frozen list. The first node that we push onto the stack is the node consisting of the empty model, and a frozen list consisting of all image line segments (since we have not yet tried to add any of them to this empty model). This starting node is the 'seed' of all other nodes, from which the depth-first search promises to deliver all possible models.

In our case, the function `visit`, in Algorithm 1, would add the model associated with the current node, to a *list of models already visited* during the search, so that we have a record of these for analysis. Also, during the course of the search, we want to keep a *list of maximal block models* that have so far been found. To do the second task we need the notion of a shadowed model.

Shadowed Model

A *shadowed* block model is a block model that is a sub-model of a model already encountered during the search. By *sub-model* we mean the following: Block model A is a sub-model of B, if, when all image edges that are in B and are not in A are removed from B to form B', then A is the same as B', as defined in Section 4.1. Put another way, A is a sub-model of B, if it does not contain any image edges that B does not, and if the labelling of its image edges is equivalent under symmetric relabelling to the labelling of those same image edges in B.

Instead of thinking about whether or not a given block model is shadowed by a known block model, it might be easier to understand the concept by thinking of all possible block models that a given block model can shadow. Suppose for example that we have a block model such as the one in Figure 5.1a. Then to find out which block models this block model has the potential to shadow, we systematically remove image edges. Thus a block model with k image edges shadows 2^k unique block models. These are shown in Figure 5.1b through (o) for the model in Figure 5.1a. Note also that blocks (b) through (o) can be called *sub-models* of the block model shown in (a). Of course, the same is true if block (a) is relabelled symmetrically.

With this definition in hand, we can now describe how to maintain the list of maximal block models.

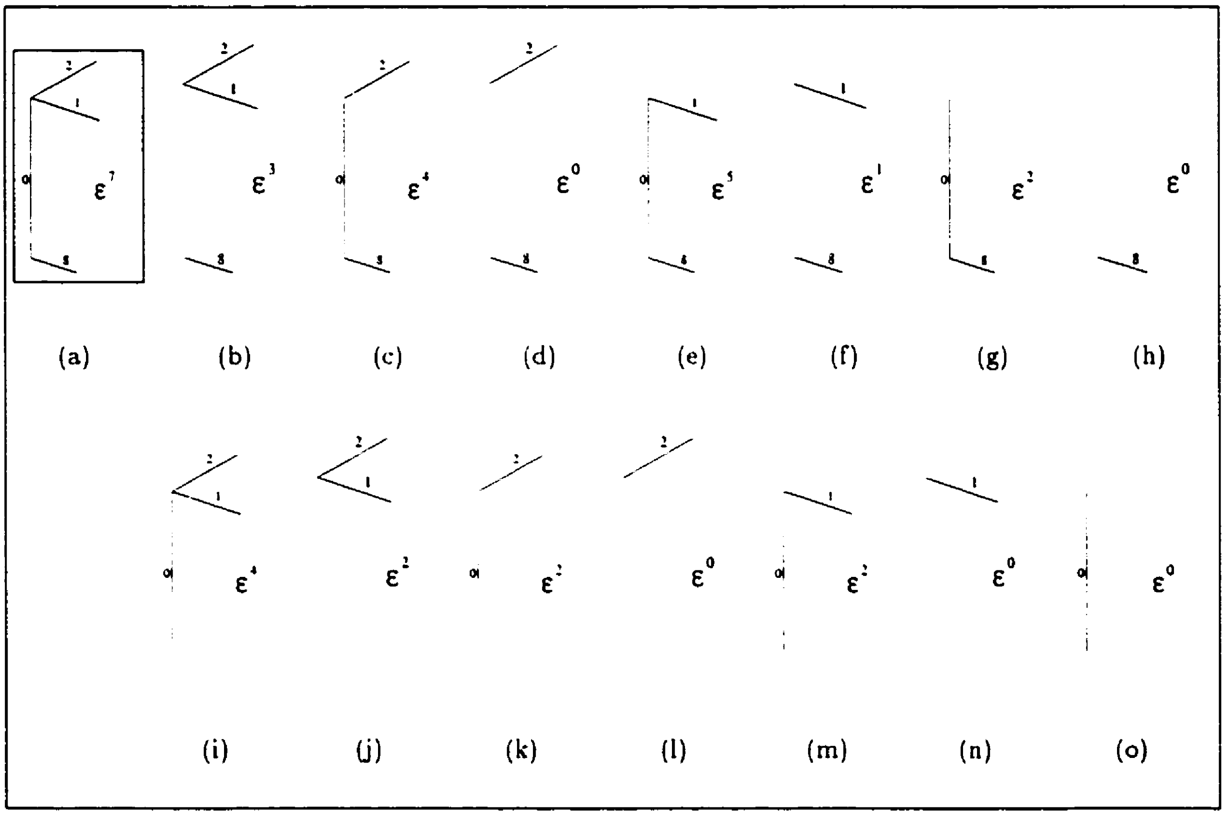


Figure 5.1: (a) shows a block model, while (b) through (o) show all block models that are shadowed by the model in (a). Additionally, we note that (b) shadows (d), that (i) shadows (m), *etc.* To the right of each figure is the asymptotic epsilon odds.

Updating the List of Maximal Models

To maintain an on-line list of maximal block models, we must update the list of maximal blocks when visiting each node during the search. We do this as follows: Upon visiting a node, check to see if the current model is either:

1. the same, is shadowed by any model in the list of maximal models that we have to date. If it is, then do not add it to the list. If it is not, then add it to the list of maximal models.

2. shadows any of the models in the list of maximal models (this can only happen if it was added to the list of maximal blocks). For every model that it shadows, remove that model from the list so that we may maintain the maximal property.

5.2.1 Finding the Children

An important detail of Algorithm 1 was glossed over: we did not specify how to get the children of `currentNode`. This is a key point as this is how we build new models from old ones.

Generating a child involves generating a new model, as well as an associated frozen list. The frozen list associated with a model contains the information telling us which line segments we should try adding to form a new model. To get a child model from a given node, we need to take an edge from the frozen list associated with this node, and add it in some way that is consistent with the model of the given node. For block models, there are up to 18 ways we can try (9 edges $\times$ 2 directions). To get the new frozen list associated with this model, we need to do one more thing - impose an ordering on the initial list of image segments. Then the frozen list associated with this new model is the frozen list associated with the original node, except without any edges that are smaller (according to the ordering we imposed) than the frozen edge we used to construct the child model. We use a function called `keepBigger(edge, edgeList)` to do this. If we were only to remove the edge we used, and no others, we would end up checking many more nodes than we need, as will be discussed in Section 5.4.

Pseudo-code for the depth-first search algorithm is shown in Algorithm 2. In the next section we will demonstrate how the search algorithm works in a simple alphabet world.

Algorithm 2 Function to get children for depth-first search

```
function childrenList = getChildren(aNode) {
   model:=aNode.model
   frozenList:=aNode.frozenList
   childrenList:=[]
   newChild:=null, newModel:=null, newFrozen:=null
   for each frozen item, fz, in frozenList {
      for each possible labelling, lb, of fz in model {
         newModel:=model+fz (according to labelling, lb)
         if isConsistent(newModel) {
            newChild.model:=newModel
            newFrozen:=frozenList-fz
            newFrozen:=keepBigger(fz, frozenList)
            newChild.frozenList:=newFrozen
            childrenList:=childrenList + newChild
         }
      }
   }
   return childrenList
}
```

5.3 Alphabet World

For illustrative purposes we will now introduce an alphabet world, analogous to our blocks world. The analogue to a set of input image edge lines will be the set of letters $\{A, B, C, D, E\}$. Instead of block models, we will have 'alphabet models', each consisting of a string of letters. Instead of searching for maximal block models, we will search for maximal alphabet models. Alphabet model consistency is defined by *fiat*: the set of maximal, (consistent) alphabet models is: $\{ACD, AE\}$. (Order is unimportant in the strings, thus $ACD = ADC = CDA$ *etc.*) We will say that if a set of letters is consistent, then so too are all subsets of this set. Thus the set of *all* consistent alphabet strings is $\{ACD, AC, AD, CD, A, C, D, AE, E\}$. [2] These are the only alphabet models in our world, since so far in this document we have called models only those sets that are consistent (it doesn't make sense to hypothesize a block model made up of edges that are inconsistent with a block).

Definitions for a frozen letter and a shadowed alphabet model are directly analogous to definitions for edges and block models described above. A frozen letter is one that we have not yet tried to add to the alphabet model in question. One alphabet model shadows another model if it contains all of the same letters (in any order) as the other model, and possibly more.

5.4 Permutations and Pruning

Note that as written, Algorithm 2 will search through only one *one permutation* of model edges/alphabet letters because of the way we create the new frozen list. Suppose we had only 3 letters, A, C, and D. Then the algorithm described above, would try only *one* of the following sequences (we assume that the three letters are consistent with one another):

1. $A \rightarrow AC \rightarrow ACD$
2. $A \rightarrow AD \rightarrow ADC$
3. $C \rightarrow CA \rightarrow CAD$
4. $C \rightarrow CD \rightarrow CDA$
5. $D \rightarrow DC \rightarrow DCA$
6. $D \rightarrow DA \rightarrow DAC$

For the tasks described so far this is perfect - we do not need to go through every one of these permutations, and to do so would be a waste of time. This is because if three letters are consistent with an alphabet model, then so too is any subset of those three

[2]The empty string is also considered to be consistent.

letters. Likewise, if any three edges are consistent with a block model, then any subset of these edges must also be consistent with a block model. Thus only one of the six sequences above need be attempted. A search that goes through *all* permutations would be $O(n!)$, when in fact, it need be only [3] $O(2^n)$, as it is in the pseudo-code presented. (where n is the number of input items: edges or letters).

Is there any reason that we might want to have a search that can go through all permutations? Yes. Suppose we wanted to make our search more efficient, and that we wanted to prune the search tree not only when a model that is inconsistent is found, but also when a model that is *implausible*, according to some criteria, is found. Depending on the criteria, we might need pursue all of the permutations, since many more will be pruned. This will be explained in more detail in Section 5.5, and we will come back to this point in Chapter 6. However, we now show the reader how the algorithm is changed so that it searches through all permutations.

We adjust our algorithm by changing our **getChildren** function ever so slightly - we no longer remove the smaller items from the frozen list in order to generate the new frozen list. This change is shown in Algorithm 3, where the only difference is that one line of code has been commented out. If the distinction between the two algorithms is not entirely clear yet, the example that follows will be helpful.

5.5 Example: Search for Alphabet Models

In this section we will trace through an example of a depth-first search to find all maximal alphabet models.

We remind the reader that consistency of an alphabet model has the same property as consistency of a block model. If a set of letters is consistent, then so too is every

[3]hah!

Algorithm 3 Modified Algorithm 2 so that all permutations are searched

```
function childrenList = getChildren2(aNode) {
   model:=aNode.model
   frozenList:=aNode.frozenList
   chidrenList:=[]
   newChild:=null, newModel:=null, newFrozen:=null
   for each frozen item, fz, in frozenList {
      for each possible labelling, lb, (of fz in model) {
         newModel:=model+fz (according to labelling, lb)
         if isConsistent(newModel) {
            newChild.model:=newModel
            newFrozen:=frozenList-fz
            //-------------------------------------
            // COMMENT OUT THIS LINE
            //newFrozen:=keepBigger(fz, frozenList)
            //-------------------------------------
            newChild.frozenList:=newFrozen
            childrenList:=childrenList + newChild
         }
      }
   }
   return List
}
```

subset. Thus we do not need to search all permutations. This is an important point that we will revisit in Section 6.2.1. We nevertheless will show both algorithms to reinforce in the reader's mind the difference between them. Figure 5.2 shows the search trees that result from each search. [4]

At the end of the search we are left with a list of all models visited, as described in Section 5.2. For the searches shown in Figure 5.2a, this would simply be the list of all the models shown in the diagram (*i.e.* models from nodes 1) through 17)). We are also left at the end of the search with a list of maximal models. For Figure 5.2, both (a) and (b), this list of maximal models is built in the following order:

[4]Note that each letter can be added to a given alphabet model in only one way, as opposed to a block model, where an edge can be added in up to eighteen different ways (nine model edges time two directions).

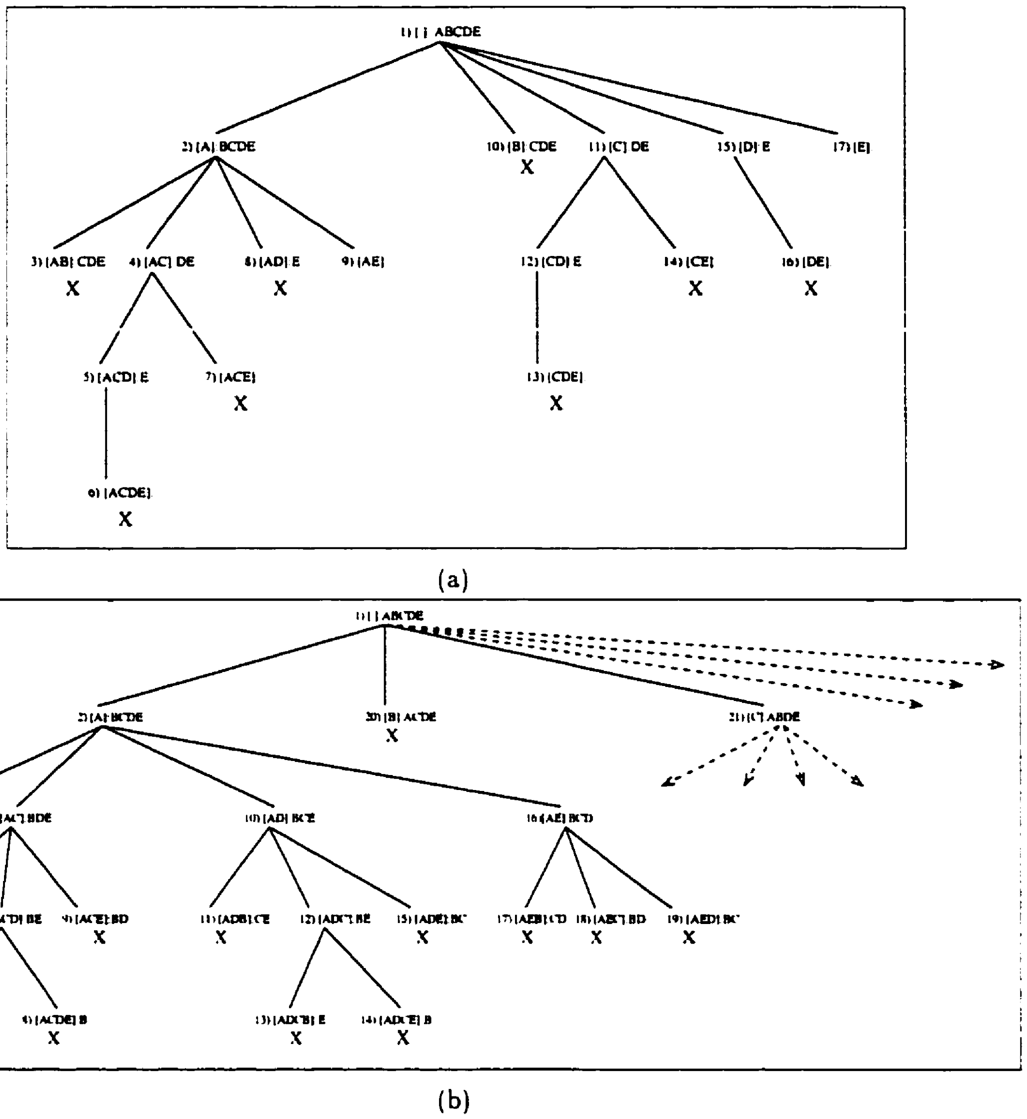


Figure 5.2: **Alphabet World Depth-First Search** for ACD and AE

a) not all permutations, using Algorithm 2 b) all permutations using Algorithm 3

Legend: Each node in the search tree is drawn as i) the number of the node, representing when the node was visited, ii) the proposed model, denoted by square brackets, iii) the associated frozen list, following the colon. For example, the twelfth node visited in (a) is model CD, with associated frozen list (E). A dashed line indicates that the subtree is not shown (due to lack of space). An 'X' below a proposed model denotes that the string of letters is not consistent. Note that if we were searching for blocks in an image, then we could never assume that a model was complete since at any stage it might be possible to add an extra image edge to a model edge, even if the model is fully constrained.

maximalList=[]
maximalList=[A]
maximalList=[AC]
maximalList=[ACD]
maximalList=[ACD, AE]

Of course use of Algorithm 2 will compile this list much more quickly, and with less redundancy. So why would we want to use Algorithm 3? Suppose that want to make the search more efficient by pruning certain hypotheses that we deem to be implausible. In particular, suppose that in the specified alphabet example, that only subsets $\{A, B, C, D, E, CD, AE, ACD\}$ are considered to be plausible (order is unimportant). If we *do not* use the all permutations search, we will cut out some intermediate nodes that we need to get to our final models. In particular, we would cut out the whole branch rooted at node 4) in Figure 5.2a, and thus end up missing ACD, a maximal alphabet model. In other words, some subsets of ACD are less plausible than others. Thus, using Algorithm 2, we might generate a non-plausible interpration first, pruning the branch at this interpretation, and never encountering the more promising one.

It is possible that the trade-off between increased computation due to searching all permutations and decreased computation due to implausibility pruning can have a net beneficial result. We revisit this point in Section 6.2.1.

Having seen how the depth-first search works in the alphabet world, we hope that the reader is now more comfortable with how a depth-first search can be used to look for block models from a set of image edges. In the next section, we will describe a more efficient search that uses QP, and is based on the depth-first search presented here. We will introduce yet another search, also dependent on the depth-first search, in Chapter 7.

Chapter 6

Comparison of QP to a Cover Measure

6.1 Can QP be used for Search ?

Chapters 3 and 4 suggested that QP might be useful in the search for block interpretations, but did not directly address the issue. It was shown how QP breaks down the set of all block models, but not how to traverse this set using QP. In Chapter 5, we described how a depth-first search can be used to find block models. In this chapter, we will show that using QP for search is feasible by using it in a simple search based upon the depth-first search. We will then go on to compare QP to another heuristic in the context of this simple algorithm.

6.2 Depth-First Search with Pruning

The simple search used in this chapter is a depth-first search (as described in Chapter 5, using some metric (either QP, or a cover measure which we will introduce) as a pruning measure. We will now describe pruning the search tree with QP.

Recall from Section 3.4.1 that the odds of a block model is a number that reflects

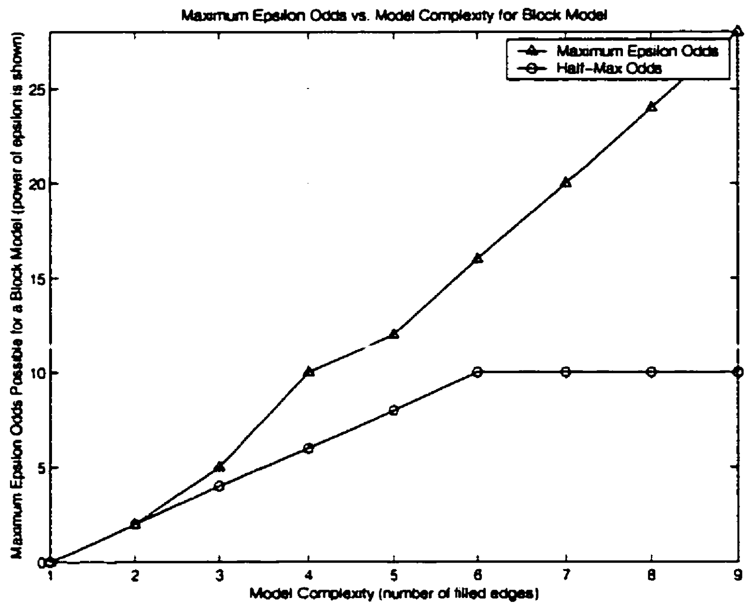


Figure 6.1: Plot of both the maximum QP odds possible for a block model, as well as the 'half-max' QP odds used as a search heuristic.

how much more likely the underlying set of image lines are of belonging to a block, B, as compared to independent sticks, S.

$$Odds(B) = \frac{p(B|I)}{p(S|I)} \in \Theta(\epsilon^n \delta^m) \tag{6.1}$$

The asymptotic odds are expressed as a power of epsilon times a power of delta. In this section we look only at the power of epsilon, but still refer to it as the odds. Roughly speaking, the higher the power of epsilon, the more promising a block model (recall that we dropped the negative sign on the power of epsilon in Chapter 3).

The pruning heuristic can be described as follows: during the search, whenever a node is reached that has a model with less than the minimum 'half-max' odds for a model with the same number of filled edges (shown in Figure 6.1), then the sub-tree rooted at this node is not explored further. [1] We will call this search, the *QP Half-Max Odds* search.

In terms of implementation, the change is very simple. Recall that a depth-first step

[1] Strictly speaking, it is not exactly half.

involves removing a node from the stack, adding an image edge to the corresponding model, altering the frozen list, and then placing this new model on the stack. With pruning, the only thing that changes is that before placing the new model on the stack, we first check to see if it satisfies our Half-Max criterion. Thus, we would add one line to Algorithm 1 as is shown in Algorithm 4, where `halfMaxOdds(aNode)` is a boolean function returning whether or not the model associated with `aNode` has at least half the maximum possible odds for a block model with the same number of edges.

6.2.1 Permutations, Subset-Independent Property and Shadow-Depth

In Section 5.4 we talked about how consistency of a block or alphabet model has a nice property that allows us to search more efficiently: we did not need to search all permutations. When we start pruning the search tree with QP, can the same thing be said? To answer this question we will now introduce the notion of a subset independent property.

A *subset independent property* is one, which, if the property holds for any model (block or otherwise), then it holds for all sub-models (models formed by removing for example image edges, such as in Figure 5.1).

Thus, as already mentioned, consistency of a block model or alphabet model is a subset independent property. The QP Half-Max odds property is not. If a block model has more than half the maximum possible odds for a given number of edges, this is *not* necessarily so for sub-models of this block model. Consider the block models in Figure 5.1, where (b) through (o) are all sub-models of the model shown in (a). Sibling models (those occurring at the same depth in the search tree) have different epsilon odds. Thus, if we follow only certain paths in the search tree, we are not guaranteed to end up with the same models as if we had followed all paths (*i.e.* all permutations). For example, according to the Half-Max function shown in Figure 6.1, models (b), (d) and (f) in Figure 5.1 will be pruned.

The implication this has on our search is that we must, if we want to use the QP Half Max property to prune, use the all permutations search, Algorithm 3. Is the loss larger than the gain? As it turns out, for the examples we consider, the pruning power of QP Half Max Odds is so effective, that it completely dominates the need to search all permutations. We can make it dominate even more if we add one little twist - keep track of the shadow-depth.

The *shadow-depth* of a model is a number, representing, how long, in terms of depth-first steps, this model has been shadowed by another known model (one that has already been generated during the course of the search). For example, in Figure 5.2b, node 10) has a shadow-depth of 1 (it is shadowed by node 6), while node 12) has a shadow-depth of 2 (it is shadowed by nodes 10) and 6)). In Figure 5.2a, there are also models with non-zero shadow-depths, for example nodes 11) and 12), but there are far fewer of them.

If a branch in the search tree terminates with a shadowed model (such as the branch 2-10-12 in Figure 5.2b, then nothing will have been gained by exploring that branch. In retrospect, we see that it was a waste of time. Is there some way to predict this ahead of time? There is no guaranteed way, but we can take a guess. The longer a model is shadowed, the less freedom it has to differentiate itself from the shadowing model, because it has fewer and fewer frozen edges associated with it. Thus we can make a guess as follows: if a model has a shadow-depth larger than some maximum shadow depth parameter, then we will not pursue that branch any further. In practice, by adjusting this parameter we can achieve a much more efficient search, while still obtaining the desired models. (This will be shown in the next section.)

What determines how large a shadow-depth is needed? Larger shadow-depths are needed when the same image edge plays a role in many plausible (as deemed by, say, QP Half-Max Odds) models. This in turn is related to the resolution of the image (see

Section 4.4), among other factors.

If one could find a property that is effective at filtering the good from the bad interpretations, *and* was subset independent, then one could cut down an even larger portion of the search space. Is such a subset independent property likely to exist ? Given that we want our algorithm to work on noisy and incomplete data, we can probably answer no. In the general case, imperfect data (*i.e.* missing edge segments, imperfect co-termination of line segments, line segments not exactly placed due to aliasing ...) can only be built up a small proportion of ways to form the correct model, if we maintain at each step of the building process a minimum criterion of 'goodness'. However, the better we are at guessing which branches will be useful, the less important it is to find such a property.

6.3 Results of the Simple QP Search

The QP Half-Max Odds search was run, using a maximum shadow-depth of 2, on each of the six images in [20], and the results aggregated. So that we may describe the results, we will now introduce a few more terms.

Recall from Section 5.2 that during the search, we keep a list of all of the models visited, as well as a list of the maximal models found. During the course of the search, some models are visited more than once. We call these ***duplicate models*** and make note of this in our list of models visited. For example, in Figure 5.2b, nodes 6) and 12) are duplicates. We will call the list of models visited containing no duplicates, ***unique models***, and the whole set, containing unique and duplicate models, simply as ***all models***. Note that the list of maximal models contains no duplicates.

6.3.1 Number of Nodes Visited

In Section 4.2.3, there were 592,952 unique models visited (over all six images). Using the QP Half-Max Odds search, the number of unique models visited was 7,346, only 1%

of the total number of unique models. Thus in terms of cutting down the search space, QP Half-Max Odds is doing an excellent job. The number of all models visited was 11,468, thus the search is not as efficient as one might like (because of the duplicates), but is still doing extremely well in terms of numbers.

Figure 6.2 (the blue lines) shows how how many nodes there are for each number of filled edges. Note that in contrast to Figure 4.8, the 3 and 4 edges interpretations no longer dominate. In fact, there is a visible dip in the graph for these numbers of filled edges. QP Half-Max Odds is managing to cut a swath through the densest portion of the search space. We will see shortly whether or not it is doing so with a compass. [2]

6.3.2 Quality of Nodes Visited

In order to discuss the quality of the results, we need the notion of a *true block*. True blocks for each image used are those block models deemed by the author to correspond to real-world blocks. This is in fact far less subjective than it might seem, as agreement was easily obtained from several people. There is a one-to-one correspondence between a given block and its true model.

Given this definition, there are four ways to assess the quality of the search results:

1. *False Positive Rate:* Of the unique models visited, the percentage that were not either sub-models of a true block model or equivalent to a true block model. This is calculated by examining the list of unique models visited, checking each one. (We compared this measure, using both the unique models visited, and all models visited, and found little difference - thus we use only the one.)

2. *True Positive Rate:* Each image has either three of four true blocks in it (see for example Figure 4.9). The True Positive Rate is the ratio of the number of true block

[2] *i.e.* is the algorithm wandering around in no particular direction, or is it following paths in the tree that lead to the desired solution?

models visited to the total number of true blocks. This is calculated by examining the *list of models visited*, and, anytime we encounter a block model that is the same as a true block model (not a sub-model of a true block), we count it. (Obviously if a true block is found more than once, it is only counted a single time.)

3. *True and Maximal Positive Rate:* In order for the search to be most useful, we would like at the end of the search to look at the *list of maximal blocks* found, not at *all* the nodes visited. The True Positive Rate does not take this into account. Thus the True and Maximal Positive Rate is the ratio of the number of true blocks found in the maximal list to the total number of true blocks. It is calculated by examining the list of maximal blocks, and, when we encounter a true block model (or equivalent), then it is counted.

4. *Position of True Blocks:* Not only is the presence (or lack thereof) of a true block an important issue, but so too is where in the final list of maximal blocks the true blocks lie. If QP is indeed a good metric, then if we order the list of maximal blocks according to their QP odds, the true blocks will lie near the top.

The first three of these values, for the QP Half-Max Odds search, appear in the first row of Table 6.2. [3] Clearly QP is finding the true blocks most of the time, and is also finding many other blocks, which is reasonable because of the enormous search space (of course we would like to do better). The positions of the true blocks can be seen in Table 6.3. The last column of the table shows how many maximal blocks are found by the exhaustive search in Section 4.2.3 (*i.e* search with no pruning). With the exception of "im4", all of the blocks lie in the top 25 blocks of the maximal blocks list. This is tremendous given the number of nodes in the search space, and the number of possible maximal models that can be found! As noted in [20], one of the blocks in "im4" is not

[3]All of the data presented in this chapter are aggregate data, that is, the data for the six images used were added together. For the rates described above, this means that the denominators and numerators were each aggregated separately, and then the division carried out.

Table 6.1: Number of Nodes Visited for Different Search Heuristics

Search Heuristic	Number of Unique Nodes Visited	Total Number Nodes Visited
QP Half Max Odds	7,346	11,468
Cover=70%	68,918	105,224
Cover=80%	51,647	64,539

fully constrained, and even a human observer is unable to fully describe the underlying true block. It is this block that causes the 115 rank in Table 6.3.

Having demonstrated that QP shows promise in helping the search for interpretations, we will compare QP to another metric in the next section.

6.4 Comparing QP to a Cover Measure

Images such as those used in [20] are simple. There are not too many extraneous lines. Blocks are simple objects. Sure a few of the lines are broken or missing, and some are not perfectly aligned, or coterminating, but take a look at Figure 4.9, and note that the blocks virtually pop out to the human observer. Do we really need something like QP to get a machine to see the same blocks that we do ? Maybe our brains aren't cavorting among fanciful priors and perverse normalizing constants ... maybe they are.

At present we may not know what our brains are computing, but we can demonstrate that seemingly easy problems are in fact quite hard. For the specific problem domain discussed in this thesis, block interpretations, one might argue that a much simpler measure than QP could be used. For example, one might argue that a simple cover measure will suffice, such as one motivated by Jacob's convex hull heuristic[18]. In this section, we describe such a cover measure, and then systematically compare it to QP.

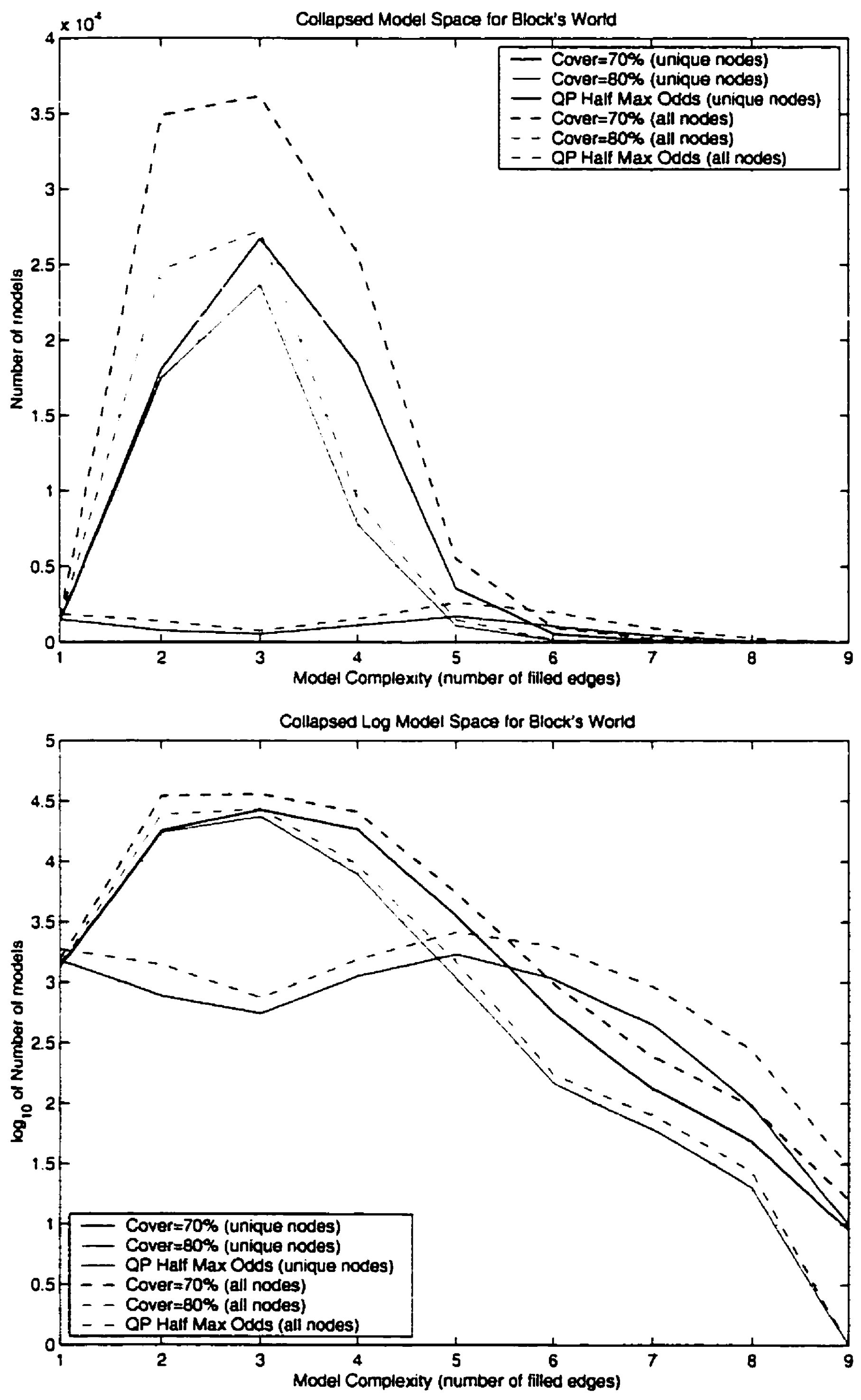


Figure 6.2: Number of models for each number of filled edges. Data generated from depth-first search using three different pruning heuristics. i) cover=80%, ii) cover=70%, iii) QP half-max odds. *Top*: normal *Bottom*: $\log_{10}$ scale

Table 6.2: True Positive and False Positive Rates for Different Search Heuristics

Search Heuristic	True Positive Rate	True and Maximal Positive Rate	False Positive Rate
QP Half Max Odds	0.95	0.90	0.78
Cover=70%	0.76	0.76	0.97
Cover=80%	0.33	0.33	0.97

6.4.1 Definition of Cover Measure for a Block

Recall that a block model, b_{l_n}, has associated with it a set of image line segments, l_n (see Section 4.2.3), as well as a least squares fit to an eight-parameter block model. There are some cases where the eight parameters of a block model are fully specified, while in others, some are free. For example, in Figure 4.9, the left-most image has a block model with one free parameter - downward length, while the right-most image has a block model that is fully constrained.

For the model in the right-most image, we know the endpoints of every model edge - we say that each model edge has no *free endpoints*, rather, each has two *fixed endpoints*. In contrast, for the model in the left-most image, where there is one free block model parameter, there are three model edges, 2, 4, and 6 for which we do not know an endpoint (in each case, the bottom-most endpoint). We will say that each of these three model edges has one free endpoint and one fixed endpoint. It is possible that some models have an edge with two free endpoints, such as a model consisting of only a single line segment assigned to some model edge. In such a case, we don't know where either of the two model edge endpoints might be. Also, we will say that if a model edge has no image edges assigned to it, then the model edge is not *instantiated*.

Now we can define cover fraction for a block model, which varies continuously in the range $[0, 1]$. An example is shown in Figure 6.3.

$$Cover\ Fraction = \frac{Amount\ acutally\ covered}{Maximum\ potential\ cover} \tag{6.2}$$

The *Amount actually covered* is calculated as follows:

Algorithm 4 Depth-First Search with QP Half-Max Odds Pruning

```
stack.push(root)
while stack.notEmpty() {
   currentNode:=stack.pop()
   visit(currentNode)
   for every child, u, of currentNode {
       if halfMaxOdds(u) {
           stack.push(u)
       }
   }
}
```

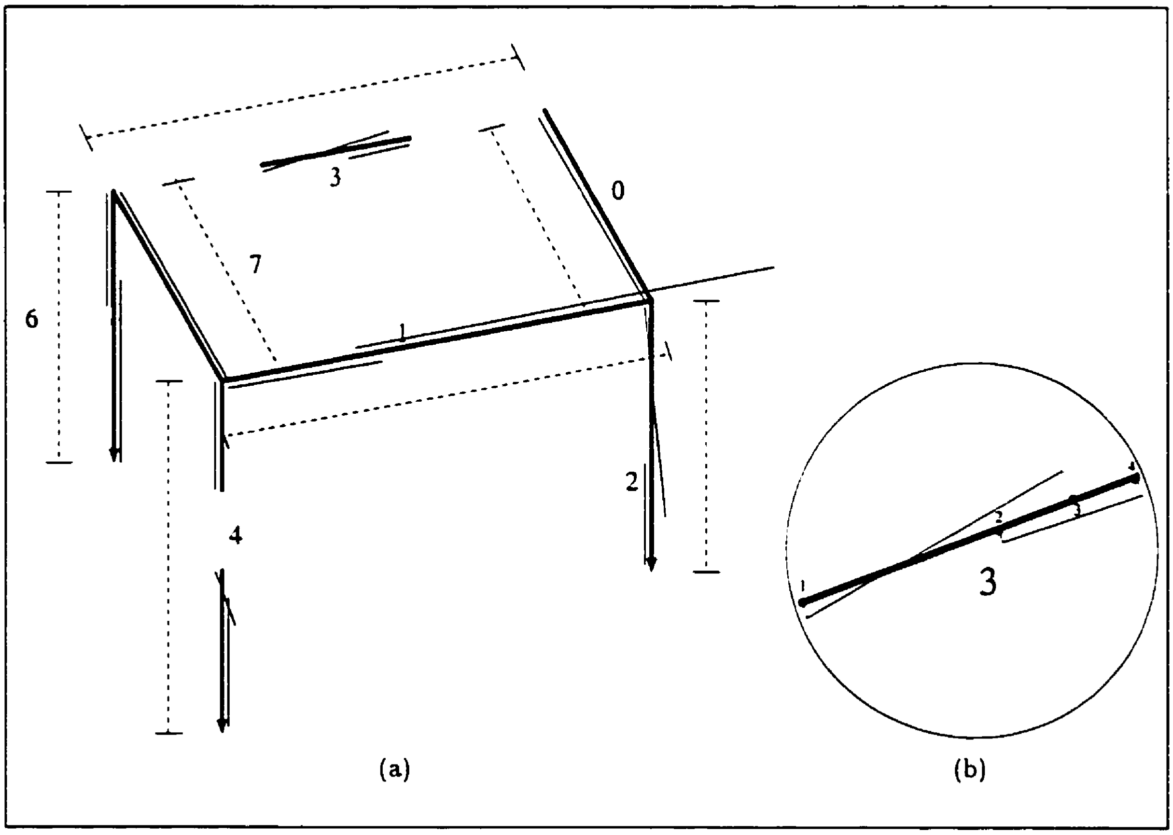


Figure 6.3: Calculating the Cover Fraction (a) *Legend*: *Solid bold line* – Covered segments of model edges *Dashed line* – Maximum potential cover, *Solid line* – Actual image edge *Arrow* – Indicates a free endpoint. (b) Blow-up of model edge 3, showing the projected endpoints and the covered segment of the model edge.

- Add up the lengths of all the covered segments of model edges.

The *Maximum potential cover* is calculated as follows:
For each model edge do the following, and then add up the values from all of the model edges:

- If the model edge is not instantiated, ignore it.
- If the model edge has no free endpoints, then add the length of the model edge. This would be the case for model edges 0, 1, 3, 7 in Figure 6.3.
- If the model edge has only one free endpoint, then add the following: the length of the model edge, from the fixed endpoint, to the furthest projected endpoint from this fixed endpoint. This would be the case for model edges 2, 4, 6 in Figure 6.3.
- If the model edge has two free endpoints then add the following: the length of the model edge between the two projection endpoints that are furthest away from one another.

6.4.2 Experimental Results

In order to compare how a cover measure compares to QP Half Max Odds, we ran the search described in Section 6.2, with maximum shadow depth of 2, except that each of three different pruning heuristics were used, on the same data set (the six images from [20]).

1. QP Half-Max Odds - call this *QPHM* for brevity
2. Pruning whenever the cover fraction is less than 0.7 - call this *C70* for brevity
3. Pruning whenever the cover fraction is less than 0.8 - call this *C80* for brevity

Pruning with the cover measure is directly analogous to pruning with QP - we do not pursue branches of the search tree when the 'pruning' measure is not satisfied for the root of that branch.

Table 6.3: Positions of True Blocks Found in QP Half Max Search

Image	Positions of True Blocks Found	Actual Number True Blocks	Number of Maximal Blocks from Exhaustive Search
im4	{2, 9, 115}	4	9,785
im5	{1, 3, 5}	3	5,649
im7	{1, 3, 12}	4	13,922
im9	{1, 2, 12}	3	15,728
imd	{1, 2, 3}	3	10,499
ime	{1, 2, 10, 22}	4	11,272

(These are the positions of true blocks found in the list of *maximal* blocks)

Table 6.4: Positions of True Blocks Found in Cover=80% Search

Image	Positions of True Blocks Found	Actual Number True Blocks	Number of Maximal Blocks from Exhaustive Search
im4	{1}	4	9,785
im5	{1}	3	5,649
im7	{1, 2, 6, 21}	4	13,922
im9	{}	3	15,728
imd	{}	3	10,499
ime	{2}	4	11,272

Table 6.5: Positions of True Blocks Found in Cover=70% Search

Image	Positions of True Blocks Found	Actual Number True Blocks	Number of Maximal Blocks from Exhaustive Search
im4	{4, 6}	4	9,785
im5	{1, 3, 4}	3	5,649
im7	{1, 2, 8}	4	13,922
im9	{1, 2, 10}	3	15,728
imd	{1, 2, 4}	3	10,499
ime	{1, 2}	4	11,272

Number of Nodes Visited

Table 6.1 shows how many nodes (with and without duplicates) were visited by each of the three searches. QPHM visits 10% the number of unique nodes that C70 does, and 14% that C80 does. For all nodes visited, the numbers are respectively 11% and 17%. Thus QPHM explores far less of the search space then either of C70 or C80.

We can get even more information if we show how many nodes occur at each number of filled edges, for each of the three searches. This is shown in Figure 6.2. A few points should be noted:

- Again, we note that the number of nodes visited by QPHM is far less than either of C70 or C80. C70 has the most nodes visited, and also has the largest, absolute duplication in nodes. However, the fraction of duplicate nodes visited is roughly the same for each of the searches, as can be seen on the log scale.
- The shapes of the curves for C70 and C80 strongly resemble those in Figure 4.8 - they have a large concentration of 3 and 4 edge models. In contrast, the QP curve is far flatter, and in fact has more 5 and 6 edge models than 3 and 4 edge models.
- That QPHM found more nodes than C70 and C80 for models with more than 6 edges can likely be explained by the fact that QPHM is missing far fewer of the true blocks than C70 and C80. We will see this in the next section.

Quality of Nodes Visited

Table 6.2 and Tables 6.3, 6.4, 6.5 show how the measures of quality set out in Section 6.3.2 compare for each of the three searches. Additionally, Figure 6.4 shows the breakdown of false positives for each number of filled edges. Some important points follow:

- From Table 6.2, we see that QPMH finds far more of the true blocks than either C70 or C80, and also finds significantly fewer false targets. In light of the fact that

QPMH explores an order of magnitude fewer nodes than do C70 and C80, these results are all the more striking. One point of interest here is that the True Positive Rate and the True and Maximal Positive Rate are the same for both C70 and C80, but not for QPMH. This means that QPMH is incorrectly 'growing' some of the true blocks, while C70 and C80 are not. In other words, once a true block model is found, there are still other image edges that can be added to the model, while still satisfying the QP Half Max Odds criterion, even thought they don't 'truly' belong to the model. However, we do need to keep in mind that C70 and C80 are finding far fewer of the true blocks to start with, and thus have fewer to 'grow'.

- Tables 6.3, 6.4, 6.5 show the positions of the true blocks in the list of maximal blocks. Overall, QPMH is finding more of the true blocks, and is doing no worse in ordering them. Note that for im7, using C80, all 4 blocks are found, while neither QP or C70 find the fourth block. How is it possible that C80 found blocks that C70 did not, given that C80 should filter out more models? The answer lies with the shadow-depth (which was set to 2 for all of C70, C80 and QPMH). Somewhere during the search, C70 must have abandoned a particular path because the shadow-depth of a model became too large. However, C80 could very well have arrived at this same model with a smaller shadow-depth, because C80 may have filtered out some of the models which shadow this model, which C70 did not filter out. Thus the shadow-depth would have been smaller, and we could have continued to explore this path. Though counter-intuitive at first, and probably a very rare event, this actually makes perfect sense.

- Figure 6.4 shows how the false positive rate changes for models with different numbers of filled edges. QPMH has a lower rate than do C70 and C80 for 3 and 4 edge interpretations. This is a crucial point since we showed that this is where the search space is most dense, and needs the most pruning. For 6 to 9 edge

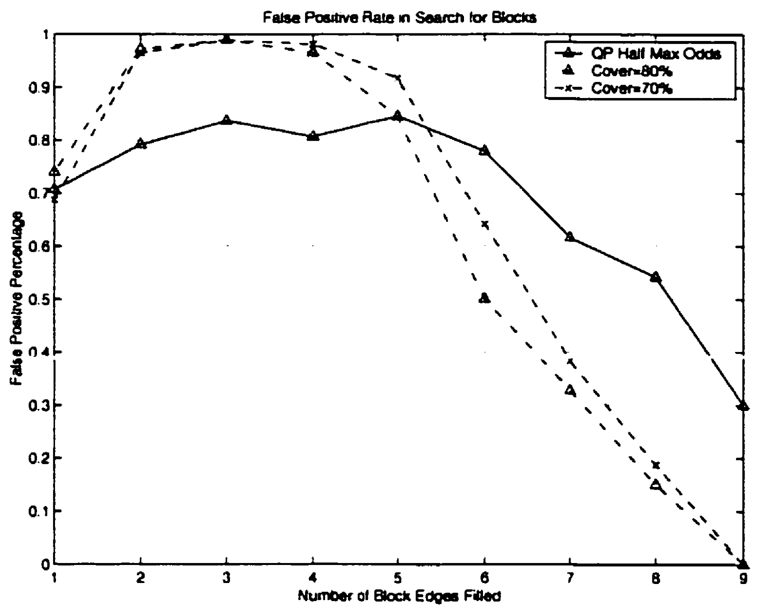


Figure 6.4: False Positive Rate for Each Number of Filled Edges

models, C70 and C80 have a lower false positive rate, but we must keep in mind that they are not finding many of the correct blocks - thus we are likely witnessing the trade-off between false positives and true positives.

6.5 Summary

In this chapter we demonstrated two important ideas. We showed that QP can be incorporated into a relatively naive search algorithm, and produce pleasing results, in terms of both the number of nodes visited, and the quality of the results. Secondly, we showed that QP as a pruning heuristic compares very favorably to the simpler cover measure. It visited roughly an order of magnitude fewer nodes than the cover measure, while at the same time producing higher quality results. This indicates that QP is on the right track, and that the simpler cover measure does not suffice. In the next chapter, we discuss how to use QP in a more sophisticated search algorithm.

Chapter 7

Cascade Search

In this chapter we will present a search algorithm that we call *Cascade Search*. First we will motivate its development, then we will explain the algorithm in detail, and lastly, we will present experimental results with discussion.

7.1 Motivation

Both the search used in [20], and the QP Half-Max Odds search presented in Chapter 6 use the Half-Max Odds function shown in Figure 6.1. Though both of these searches work extremely well, and prove that QP can be effectively used with search, they are not ideal, as we will now explain.

Recall that in Chapter 3 and Chapter 4 we showed how block models that have different numbers of edges filled cannot be compared fairly. Models with different numbers of filled edges create different 'windows' of odds (see Figure 4.7). The way we make use of this fact in the Half-Max Odds search in Chapter 6 is to use a pruning measure that is dependent on the number of edges filled for the model in question. Specifically, we said that if a given model had less than 'half' the maximum odds, then we would not explore the search branch rooted at this model. What's not ideal about this?

The fact that we have to derive/deduce the Half-Max Odds function is a little unpalatable. If we wanted to search for more complex models, or many different models, we would in each case need to re-derive this function. Furthermore, in Chapter 4 we did not use exactly half the maximum possible odds, rather, we tweaked it a little for more optimal results. It would be both more elegant and more practical if instead the algorithm were self-adapting. That is, we would like an algorithm that does not need to know ahead of time what kind of cut-off measures are needed for models at different stages (*i.e.* with different numbers of filled edges). We want an algorithm that can adapt on the fly, to different models, at different stages of development. The Cascade search presented in this chapter is a first attempt at achieving this goal.

7.2 Overview of Cascade Search

The cascade search operates by first building all 1-edge block models, using the depth-first search described in Chapter 5. [1] Once all 1-edge models have been found, it uses these models as 'seeds' [2] for another search, this time looking for all 2-edge models. This process continues, from one level to the next, until we have all 9-edge models. However, at each stage, after the search for k-model edges is complete, we filter some of these k-edge models, based on a QP criterion, before continuing onto the $k + 1$-edge model search. We call the algorithm that filters, *Grouping and Culling*. As in Chapter 6, we keep around a list of maximal models found, as well as lists of all models, and unique models visited during the course of the search.

In essence, we are cascading from window to window, only filtering out models from within one window at a time. Thus we call it Cascade Search. A pictorial representation of the search can be seen in Figure 7.1, though some of the elements have not yet been

[1]The basic depth-first search used in this chapter is searching for all permutations, using a shadow-depth of 0 (see Section 6.2.1). This is discussed at greater length in Section 7.4.1.

[2]We call them seeds because we build/grow other models from these models. These seed models are the *only* models upon which we build at the next stage. That is, we do not allow the depth-first search to build upon the empty model, except at the first stage, where the seed models consists of a single model, the empty model.

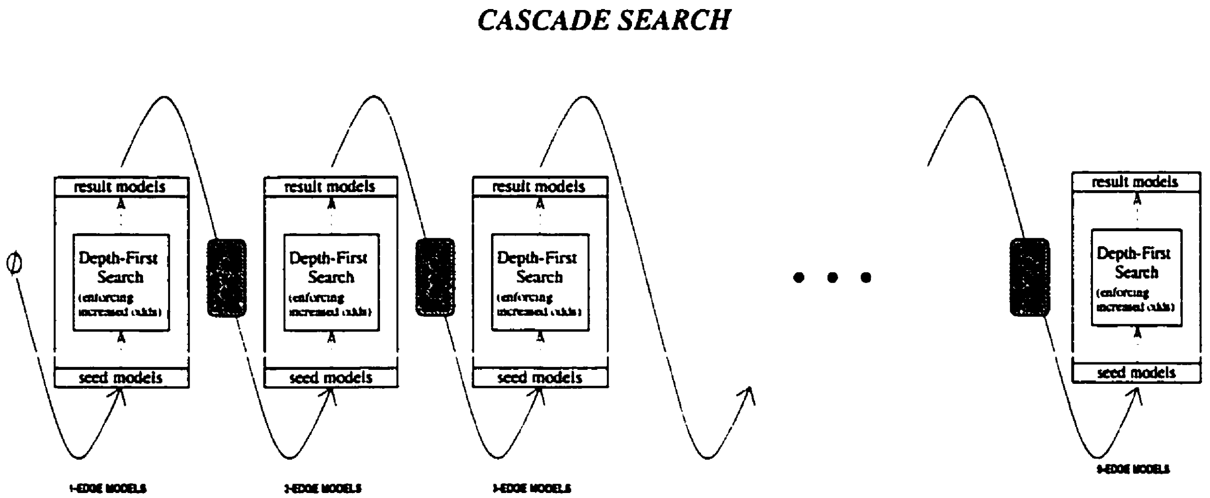


Figure 7.1: Schematic of Cascade Search – The search starts off looking for 1-edge models, starting from the empty model. It does so using no pruning criteria other than enforcing an increase in odds, as explained in Section 7.3. This produces a set of 1-edge result models. These result models are grouped and culled (G&C), as explained in Section 7.4, producing a smaller list of models which are then used as seeds/starting models for the next level of search. Note that at each stage, the depth-first search is looking for all k-edge models from all (k-1)-edge models. Several depth-first steps may be needed because collinear image segments can be added to existing model edges.

discussed.

In addition to the motivation set out at the start of this chapter for Cascade search, there are a few more minor ones, stemming from characteristics of QPHM that can be improved upon. We will now delve in to these, after which point Cascade search should be clear in its entirety.

7.3 Enforcing an Increase in the Odds

If we look at the bottom-most plot in Figure 6.2, showing the number of models for different numbers of filled edges resulting from QPMH, we note that the graph dips down at 3-edge models and then curves back up, peaking at 5-edge models before dipping down again. Why is the number of models found increasing from the 3-edge models through to the 5-edge models? Part of the explanation has to do with the combinatorics, as explained in Section 4.2.2. That is, with more image edges, there are in general more ways to build blocks. However, a more interesting part of the explanation has to do with the Half-Max Odds function we use for pruning, shown in Figure 6.1. Using this function to prune the search tree restricts us to models with odds that lie above the Half-Max Odds curve in Figure 6.1. However, having shown in Table 6.3 that true blocks typically lie in the top fraction of all block interpretations, we could potentially do even better.

Instead of restricting the search to an *entire area* above the curve, we might be able to restrict the search to a *band* above the curve, thereby cutting out even more of the search space. How can we do this intelligently? Can we simply shift the Half-Max Odds curve upward, or warp it upward? No, this will not do – we would likely prune block hypotheses that we need. Furthermore, this makes the whole Half-Max Odds function even more *ad hoc*, something we were trying to avoid in the first place. What other approach might we take?

Consider that during the course of the QPHM search, we may come across a 4-edge model that has close to the maximum possible odds of ϵ^{10}, say, ϵ^{8}. Then, proceeding along in the QPHM search, we might add some very 'bad' image edges to this model, while maintaining the necessary Half-Max criterion, since we had such high odds to start with. Thus block models that are actually quite bad are kept around and built upon, when in fact, we can actually deduce that they are bad. By simply stating that we will not accept a new block model unless it has higher odds than its parent model, we can help to eliminate this particular problem. In doing this, we will very likely reduce the larger area explored above the Half-Max Odds curve, to some sort of band, still bounded above by the maximum possible odds, and below by the Half-Max Odds function. This also helps us achieve our goal of a self-adapting algorithm.

If we enforce an increase in odds, as described, we would hope at best that the same number of true blocks would be found, with lower false positive rates for block models with more than 3 edges, while visiting fewer nodes during the course of the search. However, since we have also changed the underlying control of the search to a cascade, we may not get all that we hope for. In Section 7.5 we will see the outcome.

For the results presented in this chapter, we enforced an increase in epsilon odds of at least a power of 1. That is, given any particular block model, we only accept it (*i.e.* pursue the branch rooted at this model) if its epsilon odds are at least a power of one greater than its parent model in the search tree. This has the effect of allowing us to add only image edges that:

1. are collinear with one of the instantiated block model edges. [3] (For example see Figure 7.2a)

2. meet with the line defined by the base/cover of one of the instantiated block model edges, if the image line were to be extended. We call this a *type 1 endpoint on line.* (For example see Figure 7.2b)

3. have an endpoint that is on the extended line defined by the base/cover of one the instantiated block model edges. (For example see Figure 7.2c) We call this a *type 2 endpoint on line.*

Of course a perfect 'V' (between image edge and model edge) is either case 2 or 3. Also, we have a maximum gap allowed for the endpoint on lines (Figure 7.2b,c). Lastly, we also allow any image edge to be added to an empty model.

[3]By instantiated block model edge, we mean a model edge that either has at least one image segment assigned to it, or is fully constrained (*i.e.* has no free endpoints).

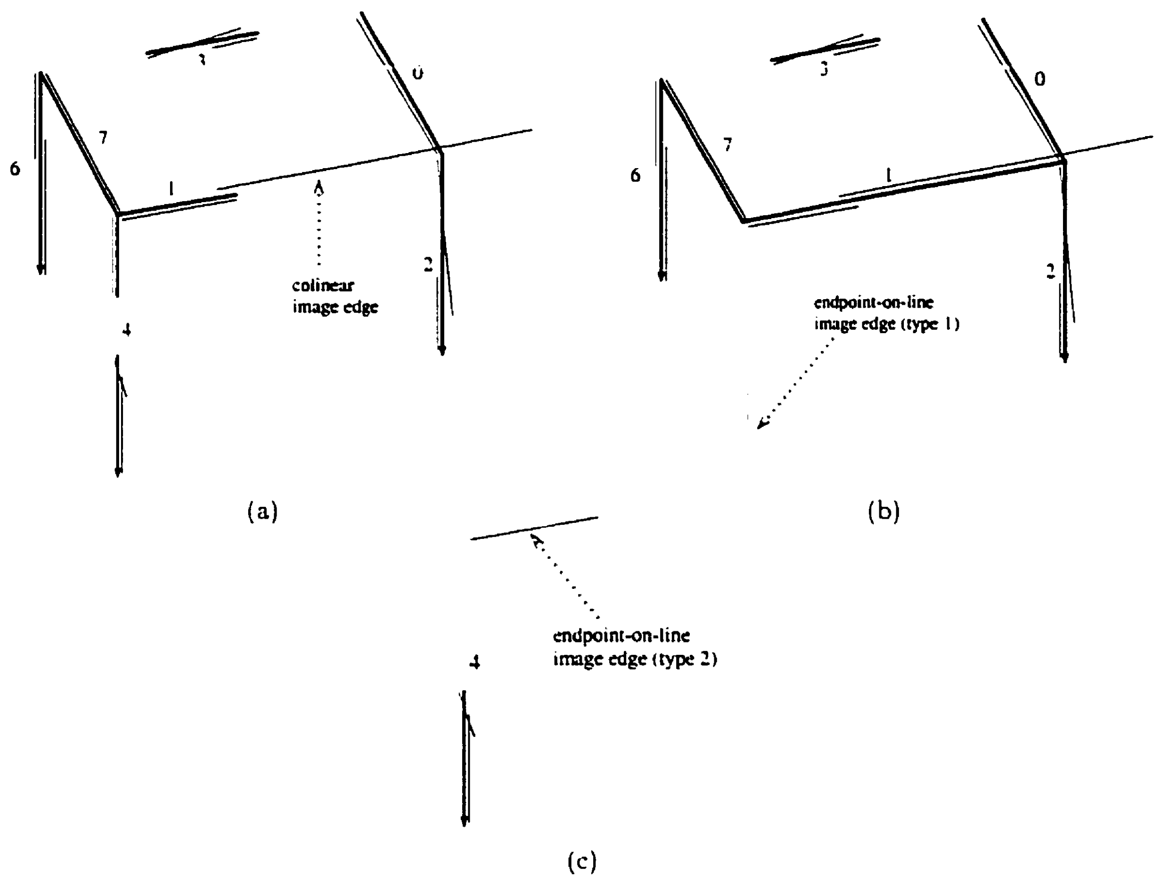

3
0
7
6
1
2
4
colinear image edge
(a)
3
0
7
6
1
2
endpoint-on-line image edge (type 1)
(b)
endpoint-on-line image edge (type 2)
4
(c)

Figure 7.2: Examples of the only three types of image edges that can be added to a current model such that the increase in odds is sufficient. (a) image edge is collinear with model edge 1 (b) model edge 7 (and, incidentally, model edge 1) has endpoint on image line (c) image edge has endpoint on line formed by model edge 4 (and, incidentally, a type 1 endpoint on line with model edges 0 and 2)

7.4 Grouping and Culling as a Branch Predictor

Recall that in Section 6.2.1 we talked about using the shadow-depth to estimate how fruitful a particular branch of the search tree might be. [4] We said that the larger the shadow-depth, the less useful a particular branch would likely turn out to be. Because the control process during the Cascade search is fundamentally different, we are able to do a similar thing in a quite different way, one which leaves more room for improvement.

7.4.1 Maximum Shadow-Depth of Zero

Note that during the cascade search we have all of the k-edge models that will be explored during the search, on hand at the same time (as opposed to QPHM, which is in essence depth-first, and has a list of models at any stage of development at any given time). This is the case because we are cascading from one level to the next, only moving on to the search for k-edge models when we have found all of the (k-1)-edge models. Because of this fact, we can simply set the maximum shadow-depth parameter to 0. It is best to think of this in an inductive way - not as a formal proof, but as a convincing argument:

Initially, we are looking for 1-edge models from the empty model. There is clearly no need to have a non-zero shadow depth since we do not need 1-edge models that subsume one another. Then, assume we are looking for k+1-edge models, from a sufficient set of k-edge models (*i.e.* this is our inductive assumption - we did not prune any vital models at the previous level). Again, it is clear that we do not need to have a non-zero shadow-depth, since we do not need shadowed models if we are only adding at most one edge.

In other words, QPHM needs to, conceivably, guess if some 3-edge model, which is shadowed by some 4-edge model, or 5-edge model, will lead to a better model than the 4-edge or 5-edge model. In contrast, Cascade search needs to guess if some 3-edge model

[4] Of course any time we prune the search tree in any way, we are attempting to make such a prediction.

will lead to a better model than *only* other 3-edge models. Rather than using shadow-depth to make this guess, it instead uses the Grouping and Culling algorithm that we will now explain.

Note that it is still possible for us to miss some needed models by setting the shadow-depth to zero. This is because we might need several depth-first steps in forming a(k+1)-edge model from a k-edge model, since collinear edges can be added to an existing model edge. However, with the images used, it is unlikely; in practice, it is not a significant problem.

7.4.2 Grouping and Culling

When we are finished searching for all k-edge models, having used the 'best' (k-1)-edge models as seeds, we are left with a group of maximal k-edge block models. Call these models the k-edge *result models* (see Figure 7.1). Keep in mind that we have *not* been using the Half-Max Odds criterion when building the models from one level to the next, though we have been enforcing an increase in odds.

By grouping together k-edge models that are similar (definition to follow), and then keeping only the best of these (*i.e.* culling the bad ones), according to their QP odds, we are predicting which k-edge models will be useful for building up the true blocks, without the need to play around with a shadow-depth parameter.

We will now present our similarity definition. The culling and grouping algorithm will follow.

Similarity Definition for Two Block Models

Firstly, the similarity measure takes a parameter, called the *similarity fraction*. Thus, we say that two block models are k-similar, for $k = 0.8$ if they are 80% similar according to our definition. The similarity is basically a measure of how many image line segments two block models have in common. Now we will formally define k-similarity.

Let $I_1 = \{l_{11}, l_{12}, \ldots, l_{1n_1}\}$ be the image segments that have been assigned to one block, B_1. Similarly, let $I_2 = \{l_{21}, l_{22}, \ldots, l_{2n_2}\}$ be the image segments that have been assigned to another block, B_2. Now let $S = |I_1 \cap I_2|$. Then, B_1 and B_2 are said to be *k-similar* iff either i) $\frac{S}{|I_1|} \geq k$ or ii) i) $\frac{S}{|I_2|} \geq k$. Clearly $0 \leq k \leq 1$. Now we are ready to describe how grouping and culling works.

The G&C Algorithm

The Grouping and Culling algorithm requires that the initial list of models be sorted by non-increasing epsilon odds. Then, starting with the first model in the list, it looks for all other models that are k-similar, grouping them together. Then, it keeps some predefined `topFraction` of these (according to their epsilon odds), ensuring that no fewer than `minGroupSize` are kept around, unless the group is not large enough (in practice, we set `k=0.6, minGroupSize=10 and topFraction=0.10`). We do this iteratively, until every model has been placed in a group. No model is assigned to more than one group, even if it is culled from this group. Pseudo-code is given in Algorithm 5.

The G&C algorithm is exploiting the fact that image line segments, in general, play a role in at most one, or very few real world objects. By using the best of the interpretations, as determined by QP, we can hope to eliminate the less useful search paths. We can only use G&C in Cascade search, and not in QPHM (which is depth-first), because it requires that we have all k-edge models at the same time.

7.5 Results and Discussion

7.5.1 List of Maximal Block Models

In order to compare our Cascade search results with our QPHM search, it will be useful to construct a list of maximal blocks from the cascade search.

Algorithm 5 Grouping and Culling

```
//precondition: modelList is sorted by non-increasing epsilon odds
//-----------------------------------------------------------------
//k is the similarity fraction to be used by the similarity function
//topFraction is the fraction of each 'group' to keep around
//minGroupSize is the minimum number of models to keep in a 'group'
//-----------------------------------------------------------------
function newModelList = GroupAndCull(modelList, k, topFraction, minGroupSize) {
   remainingModels:=modelList  //models not yet grouped
   modelGroup:= nil            //current model group
   oneModel:= nil              //representative model for a group
   while (remainingModels != nil) {
      oneModel:= removeFirst(remainingModels)
      modelGroup:=oneModel
      remainingModels:=allButFirst(remainingModels)
      for each model, md, in remainingModels {
         if kSimilar(oneModel, md, k)
            modelGroup:= modelGroup + md //append to list
            //note this keeps them in non-increasing odds order
      }
      remainingModels:=remainingModels-modelGroup //set difference
      if size(keepTopFraction(modelGroup, topFraction))>=minGroupSize
         modelGroup:=keepTopFraction(modelGroup, topFraction)
      else
         modelGroup:=keepMinGroupSize(modelGroup, minGroupSize)
      newModelList:=newModelList+modelGroup //append to list
   }
}
```

Suppose that a true block is poorly imaged, and that one or more lines are missing. Then the maximal model for this block will have fewer than 9 model edges. For the cascade search, we obtain the final list of maximal models by i) concatenating all the maximal, k-edge models that made it past G&C, for any $1 \leq k \leq 9$. Then, starting at level $k = 2$ edges, and going up to $k = 9$ edges, we remove all (k-1)-edge models that are shadowed by k-edge models. This leaves us with a final, maximal list of block models, analogous to the maximal list found in Chapter 5.

7.5.2 Experimental Results and Discussion

Again, the same six images used in [20], and shown in Figure 4.10 were used, and again, the results aggregated.

Results are presented in a similar way to those in Chapter 6.

Table 7.1 shows the positions of true blocks found, and compares these to the QPHM search. We see that Cascade search is doing very nearly as well as QPHM. In im4, the right-most block is missed by Cascade, but found by QPHM. This is because one of the bottom edges does not form an endpoint-on-line with existing model edges (specifically, the bottom, left-most edge, because the gap is too large). Additionally, Cascade is missing the bottom-most block in ime, though QPHM is finding it. The same explanation as for the missing block in im4 is appropriate.

Table 7.2 shows the number of nodes visited. The number visited by QPHM is comparable to those found by Cascade. Thus, two different algorithms, QPHM and Cascade, though controlling the search very differently, are operating at the same level of efficiency. This suggests that for this type of search (*i.e.*, not using fragments of blocks first, but building up blocks in one go), we are doing as well as we could hope.

Table 7.3 shows that the false positive rates are roughly the same for Cascade as for QPHM. However, looking at Figure 7.3b, we see that the breakdowns according to number of filled edges are quite different. For models with more than 3 and fewer than 9 edges, Cascade has a much lower false positive rate. This is directly related to our discussion earlier in the chapter about how QPHM allows us to add 'bad' image edges to models that have fairly high odds, thereby producing many 'bad' models. Cascade search, likely through its use of enforcing an increase in odds, has helped avoid this problem.

The aberration in this pattern, for models with 9 edges, can be traced back to the left-most block in im4, which does not have enough data to be properly constrained, even to the human viewer. Thus quite poor hypotheses are put forth, and built upon. In fact, because of the vertical structure in the image, as well as a few accidents, Cascade is able to form consistent 9 edge block models for this block, that are not very good, but do not get pruned because there are no better ones. Since there are not very many 9 edge models found overall, the false-positive rate appears to be very high. In fact, only 2 false 9-edge models are found over all six images.

Table 7.3 also shows the two different true positive rates. The True and Maximal Positive Rate is lower than the True Positive Rate for both Cascade and QPHM. As stated in Chapter 6, this is because both algorithms allow us to incorrectly grow blocks, past the state when they are true blocks. Though both of the positive rates are lower for Cascade than for QPHM, we should remember that this difference can be explained by missing only two blocks, over all 21 blocks.

Lastly, Figure 7.3a shows a breakdown of the number of models, by number of edges filled, visited by both Cascade and QPHM. The increase in models from 3-edge models to 5-edge models that we spoke about at the start of this Chapter has vanished, again, likely due to the fact that we forced the odds to increase at every depth-first step. Note that there are more 2 and 3 edge models found by Cascade than by QPHM. This could be because of the difference in the pruning-during-search that the two algorithms use. QPHM prunes during search using the Half-Max function, while Cascade demands that the odds increase. For models with fewer than 3 edges, it is relatively easy to have an increase in odds from a parent, while having less than the epsilon odds prescribed by the Half-Max function.

7.6 Summary

The control of the building process in Cascade search allows us to improve upon some of the less desirable properties of QPHM.

Table 7.1: Positions of True Blocks Found using Cascade Search

Image	Positions of True Blocks	Actual Number True Blocks	Positions of True Blocks Using QPHM Search
im4	{2, 16}	4	{2, 9, 115}
im5	{1, 3, 4}	3	{1, 3, 5}
im7	{1, 2, 5}	4	{1, 3, 12}
im9	{1, 2, 5}	3	{1, 2, 12}
imd	{1, 2, 3}	3	{1, 2, 3}
ime	{1, 2, 12}	4	{1, 2, 10, 22}

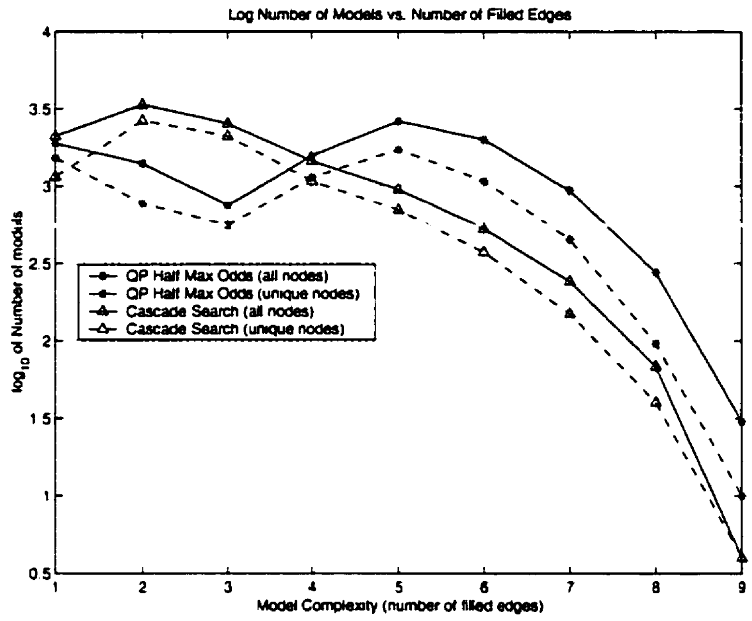
(These are the positions of true blocks found in the list of *maximal* blocks)

Table 7.2: Number of Nodes Visited

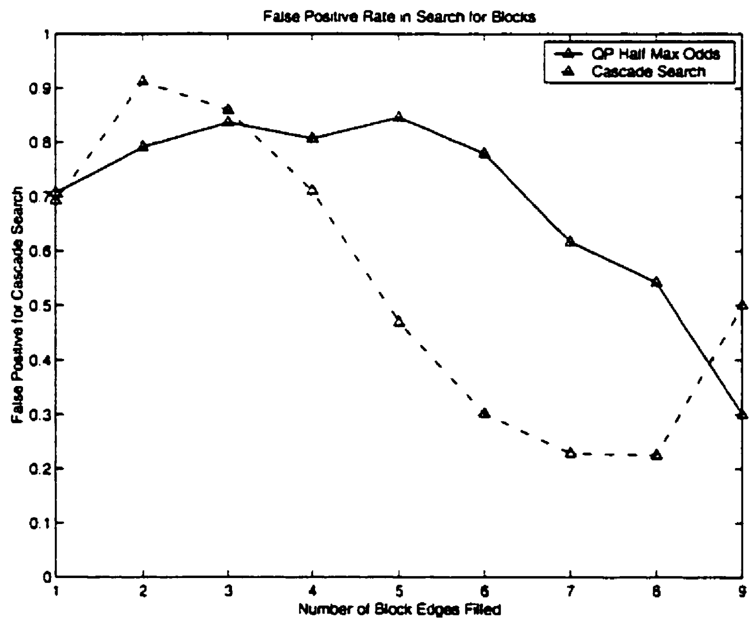
Search	Number of Unique Nodes Visited	Total Number Nodes Visited
Cascade	8,244	11,255
QP Half Max Odds	7,346	11,468

Table 7.3: True Positive and False Positive Rates for Different Search Heuristics

Search Heuristic	True Positive Rate	True and Maximal Positive Rate	False Positive Rate
Cascade	0.86	0.81	0.76
QP Half Max Odds	0.95	0.90	0.78



(a)



(b)

Figure 7.3: Cascade Search Results (a) Number of models for different numbers of filled edges (b) False positive rate for different number of filled edges.

- We no longer need to calculate the Half-Max Odds function since Cascade search is self-adapting. This is more practical because there is less of a need to play with parameters, and the algorithm has the potential to scale up better with more models. It is also more elegant.

- Due to the fact that with Cascade search we have all k-edge models on hand at the same time, we can automatically set the shadow-depth to zero, eliminating the need to fiddle with yet another parameter. (Of course, we did introduce a few parameters of our own for Cascade, namely, the similarity fraction, and the group size).

Additionally, independent from the control process of Cascade search, we tried enforcing an increase in odds during the depth-first portion of the search. We found that this was useful and produced nice results.

The end results of Cascade Search are almost identical to those of QPHM. Though the results are quite good, there is still much room for improvement in terms of getting closer to a general purpose search algorithm. However, Cascade search is an important step toward the goal of achieving an efficient, effective, and self-adapting search algorithm for line-based object recognition using Qualitative Probabilities.

Chapter 8

Conclusion

8.1 Contributions

We summarize the contributions of this thesis as follows:

1. We explored the search space induced by Qualitative Probabilities in a block world domain. It was found that the odds calculated using QP were 'windowed', with the size and position of the window dependent on how many model edges were accounted for by image edges. Furthermore, we gained a better understanding of how the combinatorics of forming geometrically consistent models from image edges interplays with the QP odds of the corresponding models.

2. We showed that 'accidents' do happen, and showed that the number of different types (*i.e.* degree of genericity) of accidents is correctly modeled by QP.

3. We incorporated QP into a simple search to find block models. The search was found to be effective and efficient. Additionally, we compared the use of QP in this search, to a more standard measure of goodness, a 'cover' measure. We found that QP is a much stronger predictor of good interpretations than the cover measure. That is, search using QP was far more efficient, and produced better block models

than the cover measure.

4. Based on observations in 1, we designed a search algorithm tailored to use of QP. This search algorithm does not require calculation of a 'Half-Max Odds' function, as the search in 3 required. This makes for a more elegant and practical search algorithm.

8.2 Discussion and Future Directions

The work presented in this thesis is only the beginning of how QP can be used to achieve object recognition. Many additions and extensions can be made, both to the QP framework, as well as to the search framework. We will now mention a few, ranging from issues quite specific to work presented in this thesis, to those more loosely associated with it.

1. The Grouping and Culling function presented in Chapter 7 was useful as a first attempt, but could potentially be much improved upon. The grouping part of the algorithm is dependent upon the order of the list of models given to it. We work around this by insisting that the list is in order of non-increasing odds. However, it would better if the grouping function was not dependent on order. Rather, it is desirable for the function to partition the list of models into equivalence classes. This requires the similarity definition to be reflexive, symmetric and transitive, the first two of which are currently satisfied. If this was achieved, the grouping would correspond better to our intuition of grouping as a precursor to culling. Additionally, the notion of similarity presented is rather rudimentary, and could no doubt be improved upon in other ways as well.

2. The Grouping and Culling algorithm is an attempt to predict which paths in the search tree will be useful. It looks only at the current set of models with their

associated QP odds. As is, the search algorithm does not use the object model to predict where image edges should be found. We could probably make better predictions by also taking into account this information. For instance we could do a 'look-ahead' of what different hypotheses are likely to lead to. That is, given a hypothesis, and given some general knowledge of the domain we are working in, for example, the density of image edges, their average length, *etc.*, as well as which parts of the hypothesized model have not yet had image edges assigned to them, we might be able to further distinguish the good hypotheses from the bad.

3. Other cues such as colour/grayscale could to be integrated into the search framework in cooperation with the Qualitative Probabilities. It will be interesting to see whether a trade-off between added computation due to more calculations, and less computation due to better pruning power exists, and if so, to what degree. Moreover, how do we intelligently incorporate the cues? Simply ordering their importance is likely too naive.

4. QP itself will need to be extended to handle more complex objects made up of parameterized curves, as well as straight lines. The number of resolved parameters for a particular curve model should provide the order of the prior probability.

5. The motivation behind the work in this thesis is to develop a method that can be applied to the extraction of primitives which are sufficiently complex, that with a large enough, finite group of them we might be able to represent most objects, and achieve generic object recognition. This extraction method must be robust to many types of noise, including occlusions, shadows and missing data. A logical next step would be to find a suitable set of primitives for generic object recognition, and try to search for them using an extended form of Qualitative Probabilities.

6. In the work presented in this thesis, we only search for one object at a time. How can we intelligently look for more than one object at a time? Clearly the human

brain is not running serial computations, nor even parallel ones, as information about one object reinforces information about other potential objects. We need to find an computationally efficient way of searching for multiple objects.

Bibliography

[1] J. S. Beis and D. G. Lowe. Learning indexing functions for 3-D model-based object recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 275–280, Los Alamitos, CA, USA, June 1994. IEEE Computer Society Press.

[2] J. S. Beis and D. G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1000–1006, Puerto Rico, 1997. IEEE Computer Society Press.

[3] R. Bergevin and M. Levine. Generic object recognition: Building and matching coarse descriptions from line drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, to appear, 1993.

[4] I. Biederman. Recognition by components. *Psychological Review*, 94:115–147, 1987.

[5] M. J. Black and A. D. Jepson. Eigen tracking: robust matching and tracking of articulated objects using a view-based representation. *Lecture Notes in Computer Science*, 1064:329, 1996.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press and McGraw-Hill Book Company, 6th edition, 1992.

[7] S. J. Dickinson. A. P. Pentland. and A. Rosenfeld. 3-D shape recovery using distributed aspect matching. *IEEE Transactions on Pattern Analysis and machine Intelligence*, 14(2):174–198, February 1992.

[8] Sven Dickinson, David Wilkes. and John Tsotsos. A computational model of view degeneracy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. August 1999.

[9] Sven J. Dickinson. Robert Bergevin. Irving Biederman, Jan-Olof Eklundh, Roger Munck-Fairwood. Anil K. Jain. and Alex Pentland. Panel report: The potential of geons for generic 3-d object recognition. Technical Report TR-29, 1994.

[10] Sven Josef Dickinson. The recovery and recognition of three-dimensional objects using part-based aspect matching. Technical Report CAR-TR-572, Center for Automation Research. University of Maryland. College Park. MD, August 1991.

[11] J. Dolan and R. Weiss. Perceptual grouping of curved lines. In *Image Understanding Workshop (Palo Alto. CA. May 23-26. 1989)*. pages 1135–1145. San Mateo. CA. 1989. Defense Advanced Research Projects Agency. Morgan Kaufmann.

[12] Jacob Feldman. Regularity-based perceptual grouping. *Computational Intelligence*, 13(4):582–623. 1997.

[13] Jacob Feldman. The role of objects in perceptual grouping. *Acta Psychologica*. 102:137–163. 1999.

[14] E. Grimson. *Object Recognition by Computer*. MIT Press, May 1990.

[15] W. E. L. Grimson and D. P. Huttenlocher. On the sensitivity of the hough transform for object recognition. In *Second International Conference on Computer Vision (Tampa., FL. December 5–8. 1988)*, pages 700–706, Washington, DC,, 1988. Computer Society Press.

[16] W. E. L. Grimson and T. Lozano-Perez. Localizating Overlapping Parts by Searching the Interpretation Tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(4):469–482. 1987.

[17] David W. Jacobs. Grouping for recognition. Technical Report AIM-1177. Massachusetts Institute of Technology. November 1989.

[18] David W. Jacobs. Robust and Efficient Detection of Convex Groups. *IEEE Trans. on Pattern Analysis and Machine Intelligence*. 1995.

[19] A. Jepson. W. Richards. and D. Knill. *Perception as Bayesian Inference*. chapter 2: Modal structure and reliable inference. pages 53–92. Cambridge Univ. Press. 1966.

[20] Allan Jepson and Richard Mann. Qualitative probabilities for image interpretation. In *Proc. IEEE Inter. Conference on Computer Vision*. Greece. 1999.

[21] Robert L. Kruse. *Data Structures and Program Design. Third Edition.* Prentice Hall. New Jersey. 1994.

[22] D. Lowe. *Perceptual Organization and Visual Recognition.* Kluwer Academic Publishers. Norwell. MA. 1985.

[23] Louis Lyons. *A Practical Guide to Data Analysis for Physical Science Students.* Cambridge University Press. 1994.

[24] Lawrence G. Roberts. Machine perception of three-dimensional solids. TR 315, Lincoln Lab. MIT. Lexington. MA. May 1963.

[25] Sudeep Sarkar and Kim L. Boyer. *Computing Perceptual Organization in Computer Vision.* World Scientific. 1994.

[26] Eric Saund. Perceptual organization of occluding contours of opaque surfaces. *Computer Vision and Image Understanding: CVIU*, 76(1):70–82, October 1999.

[27] John K. Tsotsos. Analysing vision at the complexity level. *Behavioral and Brain Sciences*, 13(3):423–496, 1990.

[28] John K. Tsotsos, Sean M. Culhane, Winky Yan Kei Wai, Yuzhong Lai, Neal Davis, and Fernando J. Nuflo. Modeling visual attention via selective tuning. *Artificial Intelligence*, 78(1-2):507–545, 1995.

[29] Lance Williams and Karvel Thornber. A comparison of measures for detecting natural shapes in cluttered backgrounds. In *European Conference on Computer Vision*, LNCS. Springer-Verlag, 1998.

[30] Wolfson and Rigoutsos. Geometric hashing: An overview. *CSAE: Computational Science & Engineering, IEEE Computer Society*, 4, 1997.