Weighted Geometric Grammars for Object Detection in Context

by

Margaret Aycinena Lippow

B.S., Stanford University (2003)
S.M., Massachusetts Institute of Technology (2005)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2010

Author . . . .
Department of Electrical Engineering and Computer Science
May 21, 2010

Certified by . . . .
Leslie Pack Kaelbling
Professor
Thesis Supervisor

Certified by . . . .
Tomás Lozano-Pérez
Professor
Thesis Supervisor

Accepted by . . . .
Professor Terry P. Orlando
Chair, Department Committee on Graduate Students

Weighted Geometric Grammars for Object Detection in Context

by

Margaret Aycinena Lippow

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2010, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

Abstract

This thesis addresses the problem of detecting objects in images of complex scenes. Strong patterns exist in the types and spatial arrangements of objects that occur in scenes, and we seek to exploit these patterns to improve detection performance. We introduce a novel formalism—weighted geometric grammars (WGGs)—for flexibly representing and recognizing combinations of objects and their spatial relationships in scenes. We adapt the structured perceptron algorithm to parameter learning in WGG models, and develop a set of original clustering-based algorithms for structure learning. We then demonstrate empirically that WGG models, with parameters and structure learned automatically from data, can outperform a standard object detector. This thesis also contributes three new fully-labeled datasets, in two domains, to the scene understanding community.

Thesis Supervisor: Leslie Pack Kaelbling
Title: Professor

Thesis Supervisor: Tomás Lozano-Pérez
Title: Professor

Acknowledgments

I am so grateful to many people for their support while working on this thesis.

My advisors, Leslie Pack Kaelbling and Tomás Lozano-Pérez, have provided incredibly valuable guidance and insight on many aspects of research, but also on teaching, writing, giving good talks, and communicating ideas in general. Most importantly, their kind encouragement and perpetual sense of humor have made a sometimes very difficult road much smoother. I always look forward to our meetings, and I feel very fortunate to have advisors whose company I genuinely enjoy, and whose opinion I so respect.

My third committee member, Antonio Torralba, has been a wonderful and approachable resource for all things computer vision. I have learned so much from his expertise in scene understanding and object detection in context, as well as from his knowledge of the history of computer vision as a field. His advice on collecting and labeling data, and on using LabelMe, has also been extremely valuable.

I have been lucky to have amazing officemates during my time at MIT. I am indebted to all past and present occupants of G585 for creating such a supportive, (mostly) productive, and social work environment. I am also thankful for the varied perspectives and thoughtful feedback provided by the many present and former members of the LIS research group. I have learned a huge amount from the exposure to problems and ideas from many areas of artificial intelligence and machine learning, not just computer vision. I am particularly grateful to Luke Zettlemoyer, for many useful discussions about research, insight into natural language processing techniques and the structured perceptron, and excellent timing for afternoon coffee breaks (based on the sunlight on his screen).

Among the many former G585 and LIS people I consider friends, I am especially thankful for the friendship of Emma Brunskill and Sarah Finney. Emma's suggestion in 2006 that we form a "research buddy group", meeting weekly to discuss the ups and downs of our research lives, was a crucial turning point for me. I am so grateful for Sarah and Emma's constant empathy and encouragement, even continuing to talk regularly when they had already graduated and moved away.

When I developed carpal tunnel syndrome and related RSI problems this past January, I still had an entire dataset to label and a thesis to write. My husband Shaun and my parents Alex and Peggy could not write this thesis for me, but they could and did label almost the entire house dataset, a task which took them many days and unending patience. I will be forever grateful for their help when I needed it most.

I owe everything to the love and understanding of my family and friends. My parents Peggy and Alex, my sister Diana, and my brother J. Alex continually help me walk the line between aiming high and working hard, and remembering what is really important in life. I know that no matter what I do, they will support me, and I cannot express how much that has helped me.

Finally, I am so grateful to my husband Shaun for his endless encouragement, patience, cooking skills, humor, and love.

Contents

1 Introduction **9**
1.1 The Problem of Object Detection 9
1.2 The Promise of Context 10
1.3 Our Approach 11
1.4 Related Work 13
1.4.1 Object Detection using Local Image Features 13
1.4.2 Scene Classification 16
1.4.3 Object Detection in Context 17
1.4.4 Grammatical Approaches to Vision 19
1.4.5 Structure Learning and Grammar Induction 20
1.5 What's To Come 22

2 Weighted Geometric Grammars **23**
2.1 Linear Models 23
2.2 Weighted Geometric Grammars 24
2.2.1 Structural Variability 26
2.2.2 Representing Geometry in Scene Trees 27
2.3 Features and Weights in WGGs 27
2.3.1 Features and Weights on Object Detector Output 27
2.3.2 Features and Weights on Geometry 29
2.3.3 Features and Weights on Tree Structure 30
2.3.4 WGGs as Linear Models 31
2.4 Parsing 32
2.4.1 Scoring a Scene Tree 32
2.4.2 Finding the Max-Scoring Tree 33
2.4.3 Practical Considerations for Parsing 36

3 Parameter Learning in WGGs **41**
3.1 The Structured Perceptron Algorithm 41
3.1.1 Theoretical Justification 43
3.1.2 The Structured Perceptron for WGGs 44
3.2 Experiments with a Hand-built Structure 47
3.2.1 The Small Placesetting Dataset 47
3.2.2 Evaluation Metrics 50
3.2.3 Baseline Model 53
3.2.4 Results on the Small Placesetting Dataset 54

4 Two-Level Structure Learning in WGGs **63**
4.1 Learning Class Templates 64
4.1.1 Tree Node Geometry Vectors 65
4.1.2 Normalized Variance in Geometry 68
4.1.3 Matching Templates To Minimize Geometry Variance 69
4.1.4 A Clustering-Based Algorithm for Learning Class Templates 70
4.2 Experiments with Two-Level Learned Structures 71
4.2.1 The Big Placesetting Dataset 71
4.2.2 The House Dataset 72
4.2.3 Results on the Small Placesetting Dataset 72
4.2.4 Results on the Big Placesetting Dataset 78
4.2.5 Results on the House Dataset 82

5 Three-Level Structure Learning in WGGs **91**
5.1 Learning Composite Classes 92
5.1.1 Phase 1: Learning Composite Classes 94
5.1.2 Phase 2: Learning Scene Class Rules 101
5.1.3 Assigning Single Objects to Composite Classes 101
5.2 Experiments with Three-Level Learned Structures 105
5.2.1 Results on the Small Placesetting Dataset 105
5.2.2 Results on the Big Placesetting Dataset 110
5.2.3 Results on the House Dataset 112

6 Conclusions and Future Work **117**
6.1 Future Work 117
6.2 Conclusions and Contributions 119

A Datasets **121**

B Results **129**

Bibliography **155**

Chapter 1

Introduction

In this thesis, we address the problem of detecting and localizing objects in images of complex scenes. We seek to exploit the fact that scenes do not consist of random assortments of unrelated objects. Rather, there are rich patterns in the types of objects that occur together, and their spatial arrangements with respect to one another. To capture these patterns, we introduce weighted geometric grammars to flexibly represent and recognize combinations of objects in scenes.

1.1 The Problem of Object Detection

Object detection is an extraordinarily challenging problem. Despite over fifty years of research, current object detection systems do not remotely approach the level of human ability.

A number of factors contribute to making the problem so difficult. Objects appear from a huge number of viewpoints and rotations. They display wide changes in scale, both intrinsic to the object itself and due to the object's distance from the viewer or camera. Some objects are articulated, such that their parts move with respect to one another, while other objects are not even rigid at all. Some object classes have members that vary widely in shape, appearance, and material properties; this can be especially true when the object class is defined only by functional constraints and custom (see Figure 1-1). Objects are often occluded, both by their own parts and by other objects. And they rarely appear alone; they are usually surrounded by background clutter and distractions. Finally, illumination and lighting can change an object's appearance drastically under different conditions.

Researchers in computer vision work on a variety of tasks related to object recognition, so it is important to clarify our problem setting. Given a static image of a scene, we are interested in identifying the class labels and locations of the objects in the image. Imagine a robot putting dishes away after a meal. We might hope to provide useful sensory information to the robot such that it can determine the set of objects laid on the table in front of it, and know where to reach in order to grasp each one.

We focus on class-level, rather than instance-level, identification. Put another way, we are asking "Is this a coffee mug?" as opposed to "Is this my favorite coffee mug?" Object classes provide a useful level of abstraction in scene understanding, because they can serve as a proxy for object function. The robot likely does not need to know *which* coffee cup it is considering, only that it is a cup at all, in order to put it away in the correct location. In general, knowledge of an object's class label can help predict its role in the world.

We assume complex and cluttered images, with many objects. Thus, we are not classifying the entire image ("Is this a picture of a motorbike or a teacup?"). Many researchers focus on such problems, which are highly relevant when performing image search on the internet or organizing

Figure 1-1: Chimneys, like many other object classes, vary widely in shape and appearance.

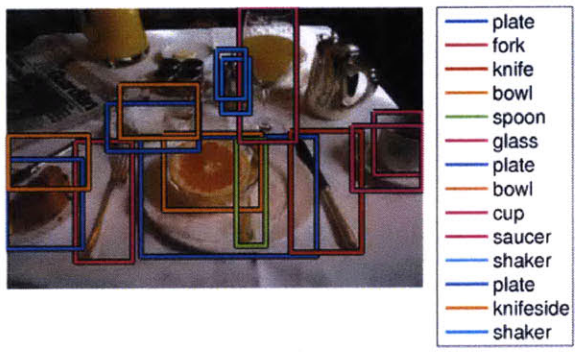


Figure 1-2: An example of the desired output of our system. We are not seeking to explain every pixel of an image with a class label, but rather identify what objects are present, and their locations.

huge image databases. However, for a robot looking to understand the visual world, classifying images of isolated objects in uncluttered images is not as helpful.

On the other hand, we are not seeking a segmentation of the image—to explain each and every pixel with a class label. Going further, we are not developing a single unified model to capture every element of an image, from its scene category and objects to curves, shadows, and edges. Again considering our robot, it does not need to know the class identity of each pixel; it simply needs to know what objects are present, and their locations. Figure 1-2 shows an example of the ideal result of our desired system.

We aim to improve the ability of existing local object detectors (that consider only the image pixels belonging to the object itself) to find salient objects. In fact, a goal of this work is to build on state-of-the-art object detectors, such that we could substitute in new detectors as the field improves. We view our work as complementary with research on local-image-based object detectors, in which much recent progress has been made.

1.2 The Promise of Context

Despite the factors that make object detection so difficult, opportunities exist for a detection system that considers more than just the local pixels where an object might appear. In particular, images from the same scene category (e.g., bedroom, kitchen, tabletop, street) have large amounts of structure. Patterns exist in the number and types of objects present in the scene. And they exist in the absolute size and locations of objects in the image, and in their relative sizes and locations with respect to one another. The main goal of this thesis is to exploit these contextual relationships among objects in a scene, in order to improve detection results.

Significant research in cognitive psychology has attested to the crucial role of context in human recognition abilities. For example, people are faster to recognize objects placed in the correct

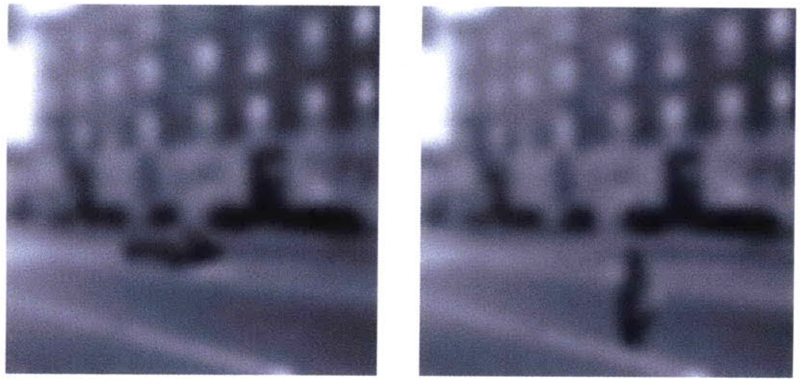


Figure 1-3: These images from Torralba (2003) demonstrate the extent to which object recognition is affected by context when the image quality is degraded. Humans would usually interpret the dark blurred object in the left image as a car, while they would interpret the blurred object in the right image as a pedestrian. However, both objects consist of identical pixels, simply rotated 90 degrees.

context (i.e., a semantically correct scene) than objects appearing in inconsistent or unexpected settings (Palmer, 1975; Friedman, 1979). Contextual cues arise from more than just the general scene context. As we have discussed, the geometric properties of the objects, including their absolute locations and sizes, and the relationships among them, provides a major source of information. The importance of these cues has been shown in human recognition; Biederman, Mezzanotte, and Rabinowitz (1982) showed humans recognize objects that violate expected relationships (e.g., support relationships, position, and size) less accurately than those that do not. See Figure 1-3 for another example of the effect of an object's context on its interpretation.

In general, context informs and guides the interpretation of objects, helping to rule out unlikely labels or functions. However, it is not clear how to effectively incorporate contextual cues into an object detection system. Two images from the same scene category may contain completely different numbers and types of objects. Furthermore, the geometric relationships among objects are flexible and multimodal. Thus, the pattern of objects in a scene cannot be easily modeled with rigid graph structures and tight spatial relations, like those used for part-based models of objects. And the properties of objects on opposite sides of the scene may be correlated. These long-range dependencies mean that simple object adjacency is not always sufficient to determine which objects should affect one another.

There may also be different levels of context in a scene. When we are looking to detect an object, we may find some subsets of the other objects in the scene more informative than others. As a simple example, the problem of finding a cup in an image becomes much easier if we have already found a saucer. Cups and saucers tend to co-occur, and they also tend to "move together"—they have lower variance spatial relationships with respect to one another than to rest of the scene. Thus, we hypothesize that a model that can capture such a hierarchical notion of context, through the use of composite classes, may further improve detection.

1.3 Our Approach

As we have argued, we need a flexible way to model many possible numbers and combinations of objects in a scene, with complex, hierarchical spatial relationships.

A natural choice is to model each image and its objects as a tree, in which the root represents the

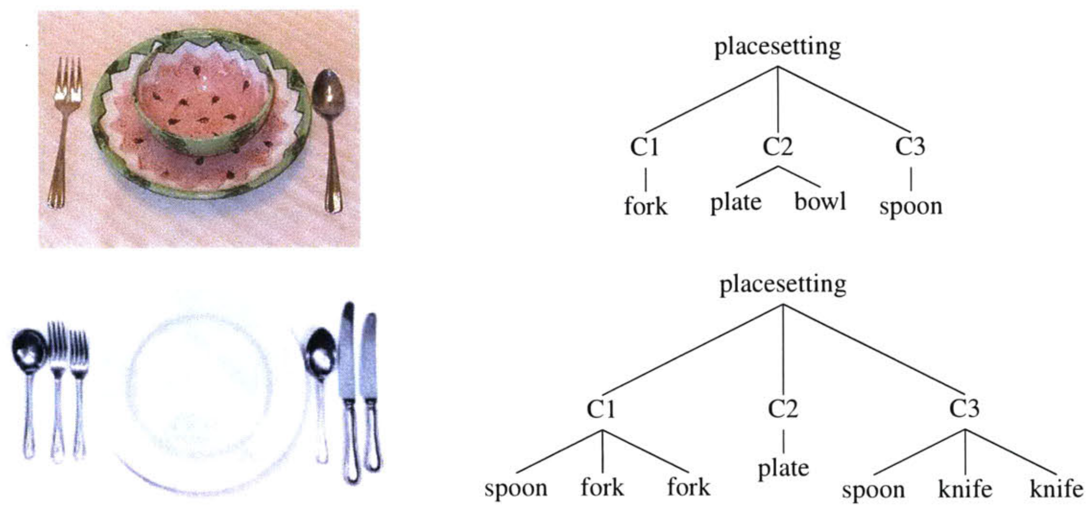


Figure 1-4: Examples of possible scene tree structures for two images.

scene class of the image, the leaves represent the objects, and the internal nodes, if any, represent groups of objects. We will refer to this representation as a *scene tree*. Tree-structured models are appealing because they capture relationships among objects in a way that enables efficient inference.

The scene tree for each image has a different structure, reflecting the different objects present in the image. We can also associate geometry and image-based information with the nodes of a scene tree, thus representing the spatial arrangement and appearance of the objects in the image. Two examples of scene tree structures are shown in Figure 1-4.

Grammars provide a natural way to model different tree structures. In particular, grammatical models offer a compact representation of the many different scene trees associated with images, and thus many combinations of objects and their spatial relationships.

Context-free grammars (CFGs), and their probabilistic brethren PCFGs, have been used extensively in natural language processing (Allen, 1995; Jurafsky & Martin, 2000; Manning & Schütze, 2002). In PCFGs, the probabilities on rules represent a distribution over tree structures, thus assigning likelihoods to different sets of symbols that may occur in the data. This matches our need to model co-occurrence patterns among sets of objects. However, as we discuss in detail in Chapter 2, traditional PCFGs are not a perfect fit for our needs, because the number of possible combinations of objects that may appear in a scene is too large. We also need to incorporate spatial and scale relationships among different components of the model, which PCFGs do not innately address.

Furthermore, a PCFG is a generative model over tree structures. (We discuss generative models again in the next section.) Generative models can have limitations, particularly in situations in which there are multiple, possibly conflicting, sources of information from which to make a decision. This is somewhat problematic, since one of the goals of this work is to incorporate arbitrary image features into our model, including the output of existing object detectors as well as more global or local image features in the future. Discriminative models offer this type of flexibility, and also allows us to leverage existing powerful methods for discriminative learning.

To address all of these issues, this thesis introduces a new class of grammatical models, *weighted geometric grammars* (WGGs), which extend models developed recently for structured prediction in natural language processing (Collins, 2002, 2004). Because WGGs are linear models, we can incorporate arbitrary features of scene trees. In this thesis, we use features representing object detector output, geometric properties and relationships, and tree structure; it would be possible to use additional features as well. We can then learn weights on the features using the structured perceptron algorithm. Thus, weight learning corresponds to determining how best to balance object detector scores, geometry, and co-occurrence information, in a principled way. Finally, we develop algorithms for learning effective WGG structures from data.

1.4 Related Work

This thesis draws on several lines of related work in computer vision, machine learning, and natural language processing. In this section, we describe these bodies of research, and place our approach in their context.

As we mentioned above, our weighted grammatical models are inspired by recent work done on structured prediction in the natural language processing community (Collins, 2002, 2004). Because we build directly on this work, we discuss it in detail in Chapters 2 and 3, rather than in this section.

1.4.1 Object Detection using Local Image Features

This thesis focuses on the relationships among objects, rather than their intrinsic image features, to improve detection. However, we build on existing techniques for object detection using local image features. Furthermore, we have been inspired by many of the historical and recent advances in object class models in designing our own scene models.

Research on object recognition goes back almost as far as the entire field of artificial intelligence, so we cannot hope to survey all of the literature. We focus here on trends in local object detection that have influenced our own work.

Template Matching and Bag-of-Features Models

Perhaps the simplest class of approaches to object detection is that of template matching. These methods use a separate template (perhaps in a transformed feature space) for each object category and view. Detection then takes place using a scanning-window approach, in which the template is scanned over every location and scale in the image, computing some measure of the similarity between the template and each image window. Windows that match well enough are deemed object detections. These methods have been successful for face and pedestrian detection (Turk & Pentland, 1991; Papageorgiou & Poggio, 2000; Viola & Jones, 2001). The most prominent recent example of a template-based detector is that of Dalal and Triggs (2005), who introduced histograms of oriented gradients (HOG) features, and achieved state-of-the-art results at the time.

Some work on object class models has focused on the problem of classification, rather than detection. That is, the models are designed to recognize (and perhaps segment) objects in isolated settings, where the object occupies most of the image. The task is then to classify the entire image into a single object class.

The class of "bag-of-features" models fall into this category. In these models, the object is represented as collection of parts, each with some model of appearance or image features, but there are no spatial relationships among them.[1] These models can be very flexible, but cannot provide much or any localization information about the object, only an estimate of the class of the image as a whole. Representative examples of this class of models include the work of Csurka, Dance, Fan, Willamowski, and Bray (2004), Sivic, Russell, Efros, Zisserman, and Freeman (2005), and Grauman and Darrell (2005, 2006).

[1] Although these models are technically 'parts-based', the literature has tended to reserve that term for models that incorporate spatial relationships among the parts; we follow the same convention.

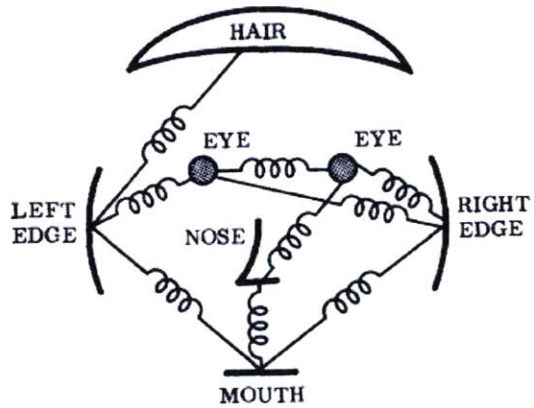


Figure 1-5: Fischler and Elschlager introduced one of the first parts-based models with spatial relations among parts. This famous figure from their (1973) paper illustrates their "spring" analogy.

Parts-Based Models

In parts-based models, an object class is represented as collection of object parts, each with an appearance model, and a set of flexible relative spatial relationships among the parts. Detection usually occurs through some version of weighted voting; each part, at each possible location, votes for where it thinks the object's center should be, and the part's vote is weighted by the quality of the image at that location, according to its appearance model.

Parts-based models have existing in the literature for decades. Binford (1971) introduced generalized cylinders as a way to perform parts-based modeling in 3D. Along similar lines, Biederman (1985, 1987) introduced *geons* as generic 3D volumetric primitives with which to describe object parts. This work was often motivated by cognitive psychology rather than computational image-based detection systems, and placed more emphasis on defining the set of primitives, rather than representing the relationships among the parts.

Fischler and Elschlager (1973) were one of the first to address the issue of spatial relationships among the parts. Their pictorial structures model, in which they formulated the parts as having "springs" between them (see Figure 1-5), has strongly influenced more recent work.

One line of such research developed "constellation" or "star" models of object classes. Early work modeled the spatial relations among all pairs of parts, leading to inefficient inference (Weber, Welling, & Perona, 2000; Fergus, Perona, & Zisserman, 2003); more recent work used a single center "landmark" part, such that the model forms a tree, allowing efficient recognition and more parts (Fergus, Perona, & Zisserman, 2005). This research tended to focus on image classification, rather than efficient object localization.

Meanwhile, a related vein of work by Felzenszwalb and Huttenlocher (2000, 2005), Crandall, Felzenszwalb, and Huttenlocher (2005), Crandall and Huttenlocher (2006) extended Fischler and Elschlager's (1973) pictorial structures framework to a statistical interpretation. The use of "k-fans" (tree-structured models), dynamic programming, and distance transforms allowed efficient localization and detection. One interesting result of this research was the demonstration that modeling relations among a greater number of pairs of parts gives only slightly better recognition accuracy, but with much greater computational cost (Crandall et al., 2005). This finding justifies simpler tree-structured models.

Quattoni, Wang, Morency, Collins, and Darrell (2004, 2007) were one of the first to apply conditional random fields (CRFs) to parts-based object recognition. This allowed the relaxation of the traditional conditional independence assumptions among parts required by generative models, and also provides an example of discriminative learning applied to parts-based models.

Finally, the discriminatively trained parts-based model developed by Felzenszwalb, Girshick, McAllester, and Ramanan (2008, 2010) is the one we use as a component in our system. This model extends the HOG models described above (Dalal & Triggs, 2005) to multiple parts and resolutions for robust object detection.

Part Sharing Across Models

In the simplest version of parts-based models, each object class has its own unique set of parts. However, researchers quickly observed that sharing parts across object class models can be very powerful. Torralba, Murphy, and Freeman (2004, 2007) were one of the first to advocate this approach. They showed that sharing can produce more robust detectors, because choosing parts that generalize to other classes often results in parts that generalize to new members of the same class as well. Furthermore, part sharing can produce faster detectors, since the scores for each part need only be computed once, even when detecting several classes.

Other prominent lines of research that make heavy use of part sharing include the work of Sudderth, Torralba, Freeman, and Willsky (2005a, 2005b, 2006, 2008), who develop hierarchical Dirichlet processes for learning and sharing parts among different object classes, and of Ommer and Buhmann (2005, 2006, 2007), who learn compositional object class models that share primitive parts among object models.

Hierarchical Parts-Based Models

In the same way that an object consists of parts in the models we have described so far, an object part can itself consist of subparts. In fact, an object model can have arbitrarily many levels of hierarchy.[2] Many hierarchical parts-based models also exhibit part sharing, in that parts share subparts or features (or that scenes share objects), in the same way that objects share parts or features.

Research on hierarchical models has been highly popular in recent years. The line of research by Sudderth et al. (2005a, 2005b, 2006, 2008), which we already mentioned, falls into this category. These models use hierarchical Dirichlet processes for detecting and recognizing objects with variable numbers of parts, and scenes with variable numbers of objects. The compositional object class models of Ommer and Buhmann (2005, 2006, 2007) are also prominent examples. Other representative work includes research by Epshtein and Ullman (2005a, 2005b, 2006), Fidler and Leonardis (2007), Ranzato, Huang, Boureau, and LeCun (2007), and Felzenszwalb and Schwartz (2007).

A noteworthy category of hierarchical models focus on capturing structural variation within an object class. Wang and Mori (2007) used a boosted forest of trees, each with different dependencies, for articulated human pose recognition. Hess, Fern, and Mortensen (2007) used mixture-of-parts pictorial structures to model object classes whose parts can vary in location, number, and type. And our own previous work developed generative parts-based grammatical models for representing structural variability within object classes (Lippow, Kaelbling, & Lozano-Pérez, 2008).

Generative versus Discriminative Approaches

The field of machine learning often draws a distinction between generative and discriminative approaches to modeling. Most of the classes of models we have discussed so far (parts-based, part-

[2] Note that these are *compositional* hierarchies, where the child-parent relationship means "is a component of", rather than *taxonomic* hierarchies, where the child-parent relationship means "is an instance of".

sharing, hierarchical) can be generative or discriminative; this is an orthogonal distinction.

Let X be some data (e.g., the image pixels) and Y be the desired class labels (e.g., object classes and their locations). In generative models, we learn a joint distribution $P(X,Y)$ on both the image data and their class labels. This distribution represents the probabilistic process by which a class model *generates* new instances of its class, perhaps all the way down to the pixels themselves. Generative approaches offer the potential advantage of more understandable, interpretable models.

Many of the models discussed so far fall into this category, including constellation or star models (Weber et al., 2000; Fergus et al., 2003, 2005), pictorial structures or k-fans (Felzenszwalb & Huttenlocher, 2005; Crandall et al., 2005; Crandall & Huttenlocher, 2006), and the hierarchical Dirichlet process models for objects and scenes (Sudderth et al., 2005a, 2005b, 2006, 2008).

In discriminative models, we instead learn a conditional distribution $P(Y|X)$ on the class labels *given* the image data. In fact, the model need not be probabilistic at all; any function mapping X to Y is sufficient. In these approaches, we only learn what aspects of members of Y make them different from other classes; other properties are ignored. Scanning-window approaches to object detection lend themselves easily to discriminative models, because each candidate window can be seen as an independent data vector which the detector classifies to decide whether an object is present.

In general, discriminative approaches have been shown to have better test performance than generative models, due to their ability to incorporate arbitrary features of the data, X, without having to build a consistent probabilistic model that only explains each part of X once (so as to not overcount likelihoods). Conditional random fields (CRFs) are a class of discriminative models that leverage this flexibility and have been employed in object detection (Quattoni et al., 2004; Ramanan & Sminchisescu, 2006; Quattoni et al., 2007).

Furthermore, a wide class of powerful discriminative learning methods have been developed in recent years. One such method is boosting, which has been used for face detection (Viola & Jones, 2001) and feature selection for object detection (Torralba et al., 2004, 2007). Another is support vector machines (SVMs) and related kernel methods, which appear in some of the most successful detection methods in recent years (Dalal & Triggs, 2005; Felzenszwalb et al., 2008, 2010).

For these dual reasons—flexibility of features, and powerful machine learning tools—we adopt exclusively discriminative approaches in this thesis.

1.4.2 Scene Classification

Turning from object recognition temporarily, we consider the task of scene classification—determining the scene category of an image (e.g., outdoor, city, street, living room).

Research into human scene understanding has shown that people recognize the scene category (or some sense of its semantic meaning), and its global layout and structure, *faster* than they recognize individual objects within the scene (Potter, 1975; Navon, 1977; Oliva & Schyns, 2000). This somewhat surprising result suggests that object recognition leverages scene information, rather than vice versa. The term "gist" was first used in the psychological literature to refer to this visual information about scene category or meaning that a viewer captures at a glance (Friedman, 1979).

From a computational perspective, these findings have motivated an approach to scene understanding which bypasses the objects and captures only global features. Oliva and Torralba (2001, 2006) argued that the primary perceptual units of scene understanding at the global level seem to be low resolution spatial configurations. They demonstrated that the spatial envelope representation (also called gist), in which an image is modeled as a vector of values for a small set of global properties (e.g., openness, naturalness, expansion, roughness), allows accurate scene classification without recognizing individual objects or regions.

Most researchers in this area have primarily focused on representations for capturing these low-dimensional global image features, then using off-the-shelf machine learning tools such as SVMs to perform classification; although there are exceptions (e.g., Kivinen, Sudderth, & Jordan, 2007).

Bag-of-features representations appear in this setting as well. Fei-Fei and Perona (2005) modeled a scene as a set of local regions, learning "themes" as hidden variables to group sets of regions, but with no spatial relationships among the regions. Rasiwasia and Vasconcelos (2008) adopted a similar approach, but showed that weak supervision of these themes produces results that better correlate with human understanding.

However, incorporating some coarse spatial information seems important. Lazebnik, Schmid, and Ponce (2006) classified images of both scenes and isolated objects by partitioning the image into increasingly fine regions, and then computing histograms of features for each region. Kivinen et al. (2007) developed nonparametric Bayesian models that capture multiscale spatial relationships among features, and showed that these models classified images into scene categories better than methods with no spatial knowledge.

1.4.3 Object Detection in Context

We return now to object detection. With ideal image conditions, an object can be recognized based only on the image features belonging to that object. However, as we have discussed, a variety of factors make this difficult and sometimes impossible in practice. Thus, many researchers have turned to the question of how to incorporate image-based information and background knowledge outside the object itself to improve recognition. We can think of this external information as the *context* of the object.

Research on object detection in context can be loosely divided into two categories: approaches that model relationships between the scene as a whole and individual objects ("scene-centered" approaches), and approaches that model relationships among objects ("object-centered" approaches).

Scene-Centered Approaches

Scene-centered approaches model relationships between global scene features and individual objects. Thus, there is no direct modeling of the spatial relationships among objects, only between the objects and the global context.

One of the first, and perhaps the most influential, pieces of work in this area is that of Torralba (2003). He modeled the expected location and scale of an object given the gist representation of the scene, producing a computational model for object priming. The system learns, for example, that pedestrians are often found in horizontal bands in the image, and the y-position of the band is constrained by the scene context, but the x-location can vary widely. Because the approach used the global features of the scene, rather than the recognition of other objects, to provide contextual cues, it remains a canonical example of a scene-centered context-based approach to object detection.

Another example of "scene-centered" approaches is that of Murphy, Torralba, and Freeman (2003), who used a conditional random field that incorporated global gist features of the scene, but not relationships among objects, to help improve object detection.

Torralba, Murphy, Freeman, and Rubin (2003) focused on specific place recognition as well as scene category, and using temporal cues rather than static images. They detected specific locations, and then used that knowledge to provide contextual priors for object recognition.

More recently, Russell, Torralba, Liu, Fergus, and Freeman (2007) used gist features to find a "retrieval set" of images in LabelMe that match the global structure of a test image. They then built a probabilistic model to transfer object labels from the retrieval set to produce candidate object detections in the query image. Spatial relationships among objects were modeled indirectly, also allowing multimodal distributions in geometry.

Object-Centered Approaches

The notion of modeling a scene as a collection of objects has a long tradition in both biological and computational vision, dating back at least as far as Minsky's (1974) frame systems. As does this thesis, most early work attempted to perform object detection jointly among different object classes, taking into account their expected relationships as a representation of the context of the scene. However, most of this research used expert systems with hand-written rules to encode background knowledge of expected scene arrangements. Examples include the VISIONS system (Hanson & Riseman, 1978), the "Schema" system (Draper, Collins, Brolio, Hanson, & Riseman, 1989), and the CONDOR system (Strat & Fischler, 1991), as well as work by Bobick and Pinhanez (1995) and Moore, Essa, and Hayes (1999).

More recently, research has leveraged modern machine learning tools to model the relationships among objects more robustly. Fink and Perona (2003) developed "mutual boosting" (extending the work of Viola and Jones (2001)) to build object detectors that incorporated features from large windows surrounding the object, as well as the output of boosting from other objects.

Often, graph structures are used to model objects and their relations. Some work has used a nearest-neighbor topology, in which adjacent objects are connected in the graph. Carbonetto, de Freitas, and Barnard (2004) framed multiclass object recognition as a segmentation task. They used a Markov random field to label pixels, and captured spatial relationships among object by enforcing consistency between neighboring labels and between labels and pixels.

However, approaches that only model relationships among adjacent objects have difficulty capturing long-distance patterns. Thus, researchers have turned to alternative graph structures. He, Zemel, and Carreira-Perpiñán (2004) performed pixel-wise labeling to segment an image with multiple object classes, using a conditional random field (CRF) with latent variables to capture long-distance correlations between labels. And Torralba, Murphy, and Freeman (2005) learned a graph structure among informative objects using boosting to select from a dictionary of connectivity templates. This allowed reliable objects (e.g., monitors) to be detected first, then providing strong contextual cues for more challenging objects (e.g., keyboards and mice).

One class of graph structures model the pairwise spatial relations among objects. Several recent papers have taken this approach. Galleguillos, Rabinovich, and Belongie (2008) frame object recognition as a per-pixel labeling task, as others have done. They incorporate co-occurrence and relative locations among objects using a CRF with discretized spatial relationships (above, below, inside, around) learned from vector quantizing pairwise relations between objects in training data. However, because of the pairwise relationships, they cannot perform exact inference, using Monte Carlo importance sampling instead. Gould, Rodgers, Cohen, Elidan, and Koller (2008) also tackle multiclass per-pixel segmentation, and incorporate inter-class spatial relationships among pairs of objects. But they encode these global features as local features by learning nonparametric relative location maps, thus allowing efficient inference.

The work of Desai, Ramanan, and Fowlkes (2009) shares many traits with our own. They also use the output of existing state-of-the-art object detectors, and take a fully discriminative approach to learning. However, as in the two papers we just described, they explicitly represent spatial relations between pairs of objects with discrete canonical relations (above, below, next-to). Thus, they use simple greedy forward search, because inference cannot be done exactly. They also do not explicitly model co-occurrence among object classes, only relative location.

In contrast to these approaches, tree-structured models allow exact and efficient inference over spatial locations and scale, while still capturing important geometric relationships among objects. However, as we discussed in the introduction, rigid tree structures are usually not flexible enough to model the widely varying numbers and types of objects that appear in scenes. Grammatical models, which can produce trees with variable structures, provide one way to model these patterns. This is the approach we take, so we devote Section 1.4.4 to discussing related work in that area.

But other types of work have leveraged tree-structured models for efficient inference. Crandall and Huttenlocher (2007) extended their generative parts-based k-fan models (Crandall et al., 2005; Crandall & Huttenlocher, 2006) to use scene context. They created a two-level hierarchical model that represented the spatial configuration of regions in the scene at a coarse level and the multi-part object models at a finer scale. They showed that using scene context in this way improved their localization results.

One way to address the variability in numbers and types of objects is with nonparametric hierarchical Bayesian models, which are based on generative topic models developed for language tasks. The most prominent example of this approach is the work of Sudderth et al. (2005a, 2005b, 2006, 2008), as we have mentioned. The framework is hierarchical: scenes are composed of objects, which are composed of parts, which are themselves composed of features. Hierarchical transformed Dirichlet processes are then used to model objects with variable numbers of parts and scenes with variable numbers of objects, with mixture components representing object positions in the scene. Because the models have infinite components, sophisticated sampling techniques are needed for both inference and learning.

Both of these lines of work represent holistic approaches to scene and object modeling, such that the same framework governs both the way that scenes are composed of objects and the way that objects are composed of parts. As we have argued, we propose that it may be more flexible and robust to avoid this assumption, allowing any approach to local object detection to coexist with a higher scene model.

Another very recent line of work adopting this philosophy is that of Choi, Lim, Torralba, and Willsky (2010), who share many common themes with our own approach. They use a tree-structured graphical model to learn dependencies in co-occurrence and spatial relationships among object classes. The probabilistic model incorporates both global gist features and local object detector output.[3] One difference with our approach is that in their model, each object class corresponds to a single node in the tree; a hierarchical tree structure is learned in which all nodes correspond to object classes. Thus, there is no notion of composite classes. Another difference is the probabilistic interpretation, versus our purely discriminative linear model.

1.4.4 Grammatical Approaches to Vision

Since we take a grammatical approach to the problem of object detection in context, we devote a section to reviewing previous work on using grammars in vision. The use of (deterministic) grammars and syntactic models was quite popular in early computer vision and pattern recognition research (Rosenfeld, 1973). But until relatively recently, grammars had largely disappeared from modern object recognition.

The past decade, however, has seen a resurgence of research in grammatical approaches to vision. Many of these lines of work focus on creating rich models that capture much of the imaging process, from scene structure and object models to segmentation and curve detection. In contrast, one of the goals of our research is to easily incorporate existing work on local object detection.

One example is the work of Pollak, Siskind, Harper, and Bouman (2003) and Siskind, Sherman,

[3]Thus, this work can be thought of as somewhat "scene-centered" as well.

Jr., Pollak, Harper, and Bouman (2007). They developed a generative model to directly extend probabilistic context-free grammars (PCFGs) to two dimensions, in order to segment and explain every pixel of images.

In a different body of work, Tu, Chen, Yuille, and Zhu (2003, 2005) attempted to unify segmentation and recognition while parsing images into "constituent visual patterns". Han and Zhu (2005) used graph grammars to decompose an image into its constituent components, rectangles and straight lines. Related work by Tu and Zhu (2006) focused on parsing images into "middle level vision representations," including regions and curves. And Zhu and Mumford (2006) developed stochastic context-sensitive image grammars, with hierarchical decompositions from scenes to objects, parts, primitives, and pixels. Because of horizontal links in the model, MCMC sampling with specially tuned proposal probabilities was necessary for efficient inference. In general, this body of work tends to emphasize rich modeling of scene structure and long-range correlations, rather than efficient recognition or learning. In the last paper cited, for example, learning occurs from a small set of fully-labeled parse trees, and then the model is sampled to produce more synthetic data for further training.

Another line of research has focused on tree-structured hierarchies or AND-OR graphs for deformable object matching and segmentation, with greater emphasis on efficient inference and learning (Zhu & Yuille, 2005; Zhu, Chen, & Yuille, 2006; Zhu, Chen, Ye, & Yuille, 2008b; Zhu, Chen, Lin, Lin, & Yuille, 2008a). This research shares many motivations and attributes with our own, using compositional models to represent geometry and appearance features hierarchically. The last two papers (Zhu et al., 2008b, 2008a) even use the structured perceptron algorithm (Collins, 2002) to learn the model parameters, as we do. However, the focus tends to be on segmentation and matching, particularly for articulated objects that occupy most of the image, rather than object detection in large and complex cluttered scenes. And again, the emphasis is on a holistic model for the entire image, from regions to local image patches and curves.

Finally, as we mentioned, our own previous work used grammars to model structural variability within object classes in synthetic 3D data (Aycinena, 2005) and images (Lippow et al., 2008).

1.4.5 Structure Learning and Grammar Induction

A huge body of literature addresses the related problems of structure learning and grammar induction, which we only briefly address here. In particular, we focus on mechanisms for learning structured models for cases in which the training data has not been fully labeled with internal structure. The majority of work on learning grammatical models for natural language processing has not focused on the unsupervised case, because the problem is so challenging, and because of the availability of large corpora of labeled tree banks of parsed sentences.

In structure learning, the goal is to find a model structure which both fits the training data and generalizes to test data well. Many researchers have adopted a search-and-score approach to this problem. In general, these algorithms explore the space of possible models using search operators to modify a current structure into a slightly different one. They evaluate each structure using a score that trades off the goodness-of-fit of the model to the training data and some estimate of the model's ability to generalize to unseen data (e.g., compactness). Searching continues until a structure with a locally-maximal score is found. Heckerman (1999) gives an overview of how algorithms of this type have been used to learn the structure of Bayesian networks from data.

Classical approaches to unsupervised grammar induction for natural language processing often take this form. For example, Chen (1995) describes an induction algorithm for probabilistic context-free grammars (PCFGs) that uses greedy heuristic search in a Bayesian framework. Along similar lines, de Marcken (1996) fits a stochastic, generative model of language to the evidence, using concepts such as concatenation and substitution/composition, estimation-maximization (EM) to estimate parameters, and structural refinement operators to improve the grammar and reduce its description length.

Our previous work on learning grammatical structures for object recognition also took a search-and-score approach (Lippow et al., 2008). We explored the space of candidate grammars using search operators to create rules capturing repeated patterns of object parts, as well as OR-structures—cases in which different patterns of parts appear in the same role or context. Candidate grammars were evaluated using a score that combined the log likelihood of the training data under the model with a penalty on the model's complexity.

However, search-and-score approaches can have serious drawbacks. First, the search problem presents inherent computational challenges, often requiring that the best parameters for each candidate model be learned from the data as an inner loop within structure search. When parameter learning is expensive, this is not a practical approach. Furthermore, the search space itself is usually huge and highly non-convex, so greedily walking through the space of candidate structures until reaching a local optimum is not always effective.

Second, constructing a good structure score is itself challenging. In particular, it is difficult to effectively balance goodness-of-fit and generalization ability when scoring a candidate model. A measure of the goodness of fit of a model is usually straightforward to obtain; the log likelihood of the data under the model is a common choice. Quantifying the generalization ability, and in a way that can be combined with the model's goodness-of-fit, is more challenging. A Bayesian approach would use a structural prior to bias the search towards compact structures that are likely to generalize well. However, the choice of structural prior is itself somewhat arbitrary.

In a related vein, researchers in the minimum description length (MDL) community have argued that the complexity of the model should be a measure of how many data samples would be fit by it (Grünwald, 2005). For example, in "refined MDL", the complexity of a model M is measured as the log of the sum, over all possible datasets of a fixed size n, of the likelihood of that dataset according to the maximum-likelihood parameters for M learned from the data. This directly and elegantly measures the expressive power of the model in terms that can be combined with the log likelihood of the data under the model. However, Grünwald shows that it is intractable to compute for all but the simplest model classes, and is actually undefined in many cases.

For these reasons, this thesis adopts an approach to structure learning that departs from the search-and-score methodology. The work of Nevill-Manning and Witten (1997) is an example of (older) research on grammar induction in natural language that also does not explicitly use a search-and-score technique. Rather, the authors infer hierarchical structure from a sequence of symbols by recursively applying operators that replace repeated phrases with grammatical rules. In some ways, this research is similar to our clustering-based approach to learning composite classes (Chapter 5), in which we look for commonly occurring low-variance object groups.

Recent work on unsupervised grammar induction in natural language tends to focus on learning classes of grammars that are less related to our model, such as dependency grammars or constituent-context models (e.g., Klein & Manning, 2001, 2002, 2004, and others). Notable exceptions include Mohri and Roark (2006), who use statistical tests to decide if a nonterminal combination is unobserved due to sparse data or due to hard syntactic constraints. Haghighi and Klein (2006) take a semi-supervised approach, in which they label a few canonical examples of each phrase type, and then propagate the prototype information to other examples in a PCFG model.

1.5 What's To Come

In the next chapter, we formally introduce the weighted geometric grammar (WGG) formalism. We define the structure of the model, and the features that capture object detector output, geometric properties and relationships among objects, and scene tree structure. We then describe a parsing algorithm for efficiently finding the best-scoring scene tree for a test image, given a WGG model. The leaves of this tree correspond to detected objects in the image.

Chapter 3 addresses the problem of parameter learning in WGG models. We review the structured perceptron algorithm (Collins, 2002, 2004), and adapt it to our framework. We then introduce the first of three new datasets contributed by this thesis, and show experimental results with a hand-built WGG structure on the dataset. These results demonstrate that WGG models do improve detection results over the object detector's performance.

In Chapter 4, we turn to the problem of structure learning in WGGs, constraining our learned models to two levels (one for the scene, and one for all of the objects). In this setting, we can view structure learning as a matching problem across training images. We propose a clustering-based algorithm for finding these correspondences, and also for inferring the latent parent geometry on the root scene nodes. In the second half of the chapter, we introduce the other two new datasets in this thesis. We present full results on all three datasets, comparing the performance of the learned two-level structures to the hand-built structure in Chapter 3, and to the performance of the object detector. Again, the results show an advantage over the object detector alone.

Chapter 5 extends the structure learning algorithms in Chapter 4 to finding hierarchical grammars with three levels. We develop a set of algorithms for finding composite classes that group objects with high co-occurrence and low internal geometry variance. We then present experiments comparing the performance of the learned hierarchical models to the two-level structures from the previous chapter.

Finally, Chapter 6 discusses the conclusions and contributions of this thesis, and proposes avenues for future work.

Chapter 2

Weighted Geometric Grammars

In Chapter 1, we argued for a tree-structured grammatical approach to modeling sets of objects and their relationships in scenes. We described how a scene can be represented as *scene tree*, with its leaves corresponding to the objects in the scene.

We can draw an analogy between our scene trees, and parse trees used to model the syntactic and semantic structure of sentences in the field of natural language processing (NLP). Recently, several researchers in NLP have developed successful linear models and learning algorithms for parsing sentences (Collins, 2002, 2004; Zettlemoyer & Collins, 2005, 2007). Our approach will extend this work to modeling and detecting arrangements of objects in scenes.

2.1 Linear Models

The class of models introduced in this thesis, weighted grammatical models (WGGs), are instances of weighted linear models, as described by Collins (2004). Basing WGGs on weighted linear models gives us the flexibility to incorporate arbitrary features of an image and scene tree into the framework.

Before we formally define the elements of a WGG in the next section, we apply the general definition of linear models from Collins (2004) to our context. We want to learn a function $F : \mathbb{I} \rightarrow \mathbb{T}$, where $\mathbb{I}$ is a set of input images, and $\mathbb{T}$ is a set of scene trees with associated class, geometry, and image information in the nodes. We assume:

- A set of training examples $\langle I_i, T_i \rangle$ for $i = 1, ..., m$, where $I_i \in \mathbb{I}, T_i \in \mathbb{T}$. (For most of this thesis, we will not actually assume fully-labeled scene trees for the training images. But for clarity, we start by making this assumption.)
- A function **GEN** which enumerates a set of candidate scene trees $\mathbf{GEN}(I)$ for an image I.
- A feature representation Φ mapping each $\langle I, T \rangle \in \mathbb{I} \times \mathbb{T}$ to a feature vector $\Phi(I, T) \in \mathbb{R}^n$.
- A parameter vector $\Theta \in \mathbb{R}^n$.

The components **GEN**, Φ, and Θ allow us to map an input image I to its scene tree $T = F(I)$ with this equation:

$$F(I) = \underset{T \in \mathbf{GEN}(I)}{\operatorname{argmax}} \; \Phi(I, T) \cdot \Theta \tag{2.1}$$

In words, the function F maps the image I to the scene tree T which maximizes the inner product between the feature vector $\Phi(I, T)$ computed from the image and the tree, and the global parameter vector Θ. We will define Φ and **GEN** for our context in the next few sections.

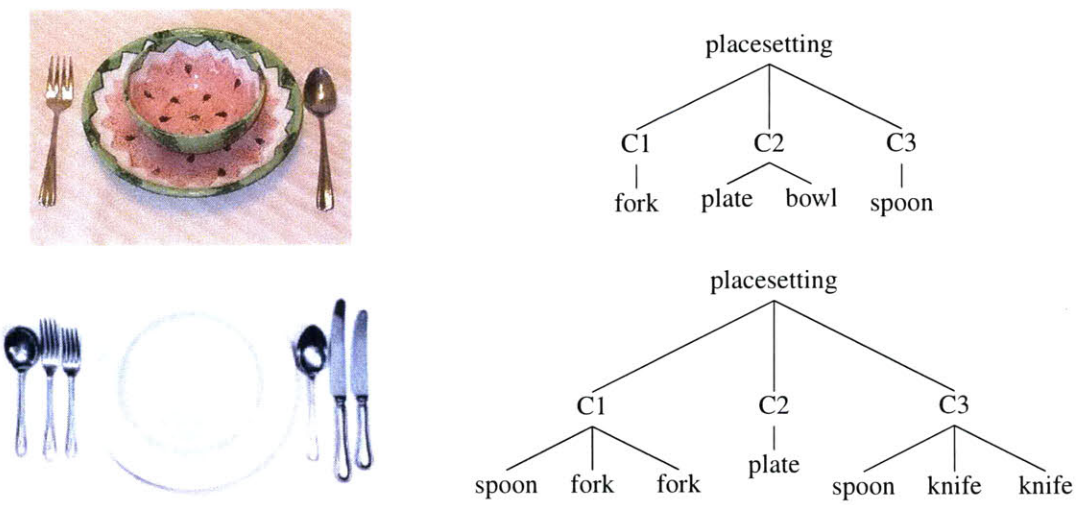


Figure 2-1: Examples of possible scene tree structures for two images.

2.2 Weighted Geometric Grammars

Continuing the analogy to language approaches, we define a type of grammatical model for scene classes. Recall that a traditional context-free grammar (CFG) defines a set of legal parse trees, each of which maps to a string of symbols. This set of strings is the language of the CFG. In the same way, our grammatical model represents a set of legal scene trees, each of which maps to a set of object instances in an image. The model lets us represent the possible combinations of objects we expect to see in an image, as well as their expected spatial arrangement and sizes.

Formally, a *weighted geometric grammar* (WGG) is a tuple $G = \langle B, C, R, S, \Phi, \Theta \rangle$, where:

- B is a set of primitive classes;
- C is a set of composite classes;
- R is a set of rules of the form

$$X \rightarrow Y_1, Y_2, \ldots, Y_n$$

  for $n \geq 0$, $X \in C$, and $Y_i \in (B \cup C)$;
- $S \subseteq C$ is the set of root scene classes;
- Φ is a feature representation mapping each image-tree pair $\langle I, T \rangle$ to a feature vector $\Phi(I, T)$; and
- $\Theta \in \mathbb{R}^n$ is a parameter vector.

We will refer to the first four of these elements, $\langle B, C, R, S \rangle$, as the *structure* of the model.[1] In the rest of this section, we focus on these first elements; we will describe the last two elements Φ and Θ in the next section.

The WGG structure $\langle B, C, R, S \rangle$ closely corresponds to the elements of a CFG. A primitive class is like a terminal symbol in a CFG. In our context, each primitive class corresponds to an object class.

[1]Sometimes we also use "structure" to refer to all of the components of a WGG except the weight vector; i.e., $\langle B, C, R, S, \Phi \rangle$.

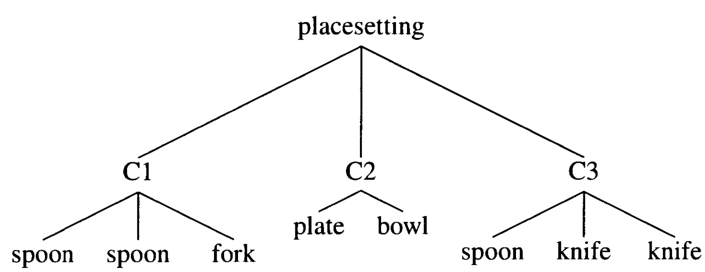
A composite class is like a nonterminal in a CFG; it represents a (possibly hierarchical) group of objects. In a CFG for language, a nonterminal corresponds to a part of speech, such a noun or verb phrase. In a WGG, a composite class (that is not a scene class) represents a group of objects that we expect to co-occur, and to maintain a relatively low-variance spatial arrangement with respect to one another when they do occur. Put another way, we expect these objects to move together in the scene—to be easier to find relative to one another than relative to other objects in the scene. We will formalize this notion in greater detail in Chapter 5 when we discuss hierarchical structure learning.

Here is an example of a simple WGG structure for placesettings (Φ and Θ are not shown):

$$
\begin{aligned}
B &= \{\text{bowl}, \text{fork}, \text{knife}, \text{plate}, \text{spoon}\} \\
C &= \{\text{placesetting}, \text{C1}, \text{C2}, \text{C3}\} \\
R &= \left\{ \begin{array}{r} \text{placesetting} \rightarrow \text{C1 C2 C3} \\ \text{C1} \rightarrow \text{spoon fork fork} \\ \text{C2} \rightarrow \text{plate bowl} \\ \text{C3} \rightarrow \text{spoon knife knife} \end{array} \right\} \\
S &= \{\text{placesetting}\}
\end{aligned}
$$

In the same way that a rule expands a nonterminal in a CFG, a rule in a WGG means that an instance of the class on the left of the arrow *consists of* instances of the class on the right. We will refer to the elements on the right side of each rule as *rule parts*. (The order in which the parts are listed in the rule does not matter, as we will discuss below.)

We can draw the structure of a scene tree that might be produced by the above WGG like this:

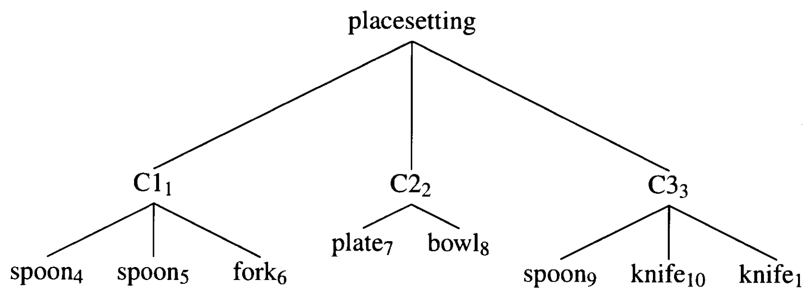


Notice that more than one part on the right side of a rule may have the same class label. These parts have different geometry parameters—we may expect the first fork in the rule for C2 to be in the middle of the group of utensils to the left of the plate, while the second fork in the rule is expected to appear on the right, closer to the plate.

To keep track of these differences, we identify each rule part in the grammar using an arbitrary but unique rule part identifier (RPID). For a rule part with RPID k, we denote its class label as $\text{class}(k)$. Then, we would write the rules R in the above grammar as follows, with each rule part written as its class label followed by its RPID as a subscript:

$$
R = \left\{ \begin{array}{r} \text{placesetting} \rightarrow \text{C1}_1\ \text{C2}_2\ \text{C3}_3 \\ \text{C1} \rightarrow \text{spoon}_4\ \text{fork}_5\ \text{fork}_6 \\ \text{C2} \rightarrow \text{plate}_7\ \text{bowl}_8 \\ \text{C3} \rightarrow \text{spoon}_9\ \text{knife}_{10}\ \text{knife}_{11} \end{array} \right\}
$$

Similarly, each node t in a scene tree T has both a class label $\text{class}(t)$ and an RPID $\text{rpid}(t)$, so it can be mapped to a unique rule part in the grammar:



The root of a scene tree has no RPID, since it does not map to a rule part in the grammar, so we define $\text{rpid}(\text{root}(T))$ to return some null symbol.

The use of RPIDs means that the order in which the parts of a rule or the children of a tree node are listed does not matter; all that matters is the RPID associated with each of these components.

2.2.1 Structural Variability

Of course, not all placesettings have exactly the set of objects listed in the scene tree above—we might see any subset of them in an actual scene, as in the examples in Figure 2-1. The number of possible combinations of objects we might see in a scene is enormous. In placesettings, the group of utensils to the right of a plate—the composite class C3 in the grammar above—might include a spoon, a spoon and a knife, a spoon and two knives, a knife, and so on. We'll refer to this phenomenon as *structural variability*.

How would we express structural variability using traditional CFGs? There are two options. Recall that in a CFG, a nonterminal can be expanded in different ways by having multiple rules with the same symbol on the left side. So one approach would be to write a separate rule for each possible combination (using the original RPIDs from above, to show the correspondences):

$$R = \left\{ \begin{array}{l} \ldots \\ \text{C3} \rightarrow \\ \text{C3} \rightarrow \text{spoon}_9 \\ \text{C3} \rightarrow \text{knife}_{10} \\ \text{C3} \rightarrow \text{knife}_{11} \\ \text{C3} \rightarrow \text{spoon}_9\ \text{knife}_{10} \\ \text{C3} \rightarrow \text{spoon}_9\ \text{knife}_{11} \\ \text{C3} \rightarrow \text{knife}_{10}\ \text{knife}_{11} \\ \text{C3} \rightarrow \text{spoon}_9\ \text{knife}_{10}\ \text{knife}_{11} \end{array} \right\}$$

We could then associate a weight with each rule to represent the relative likeliness of seeing that particular subset of objects.

However, this approach is less than ideal. First, it requires enumerating an exponential number of rules. Furthermore, we wrote the rule parts in each rule to reuse the same set of three RPIDs for clarity only; in reality, each rule would have an unrelated set of rule parts. This is problematic because we expect these objects to maintain a similar spatial arrangement with respect to one another, regardless of which subset of them occur. Having separate rules prevents the sharing of geometry parameters among these related variations.

A second approach would be to introduce intermediate classes, one for each part on the right of each rule. Each intermediate class then represents that its rule part is optional.

$$R = \left\{ \begin{array}{l} \quad\quad \ldots \\ \quad\quad \mathrm{C3} \rightarrow \mathrm{spoon9}_9\ \mathrm{knife10}_{10}\ \mathrm{knife11}_{11} \\ \mathrm{spoon9} \rightarrow \mathrm{spoon}_{12} \\ \mathrm{spoon9} \rightarrow \\ \mathrm{knife10} \rightarrow \mathrm{knife}_{13} \\ \mathrm{knife10} \rightarrow \\ \mathrm{knife11} \rightarrow \mathrm{knife}_{14} \\ \mathrm{knife11} \rightarrow \end{array} \right\}$$

By associating weights with each rule in the intermediate classes, we can represent how likely each object is to occur in its specific role. This approach seems to have the expressive power we need, although it is not very compact.

In WGGs, we adopt a structural model which is closest to the second approach, but more compact. Specifically, each part on the right side of a rule in a WGG is *always* optional. We can then associate weights with the parts to model the relative likelihood of different subsets of the parts occurring together; we will describe these weights in greater detail in Section 2.3.3 below.

In this thesis, we assume that each composite class c in a WGG is expanded by a single rule, denoted $\mathrm{rule}(c)$. All of the structural variability within each class is therefore modeled by the optionality of the parts in its one rule. However, this is by no means a necessary assumption, and expanding the framework to include multiple rules for each composite class (each with optional parts), would be straightforward.

2.2.2 Representing Geometry in Scene Trees

So far, our scene tree representation includes class labels and RPIDs to capture the set of objects and composite classes present in an image. But in order to model the location of these objects in the image, we also need geometry information in the tree.

Therefore, each node t in a scene tree has an associated geometry vector $\mathrm{geom}(t) = \langle x, y, s \rangle$. If we think of a node's geometry as a bounding box, then $\langle x, y \rangle$ is the box's centroid, while s is the scale factor at which the box's area equals that of a class-specific canonical size $\langle W_{c,\mathrm{width}}, W_{c,\mathrm{height}} \rangle$ for class c. For a leaf node corresponding to a single object, the bounding box interpretation is very natural—it is simply the bounding box of the labeled object silhouette. For a non-leaf node, we can instead think of the geometry vector representing a coordinate frame, with origin $\langle x, y \rangle$ and scale factor s. Since $\mathrm{geom}(t)$ is the location and scale of a node t in absolute image coordinates, we refer to it as the *absolute geometry vector* for t.

2.3 Features and Weights in WGGs

We are now ready to define the feature representation $\Phi(I, T)$ for an image I and scene tree T, and the corresponding weights Θ, of a WGG.

2.3.1 Features and Weights on Object Detector Output

Each primitive class in a WGG corresponds to an object class. We assume that we have a trained object detector that can provide a score $D(c, \mathbf{v}, I)$ for each object class c at each location and scale

$\mathbf{v} = \langle x, y, s \rangle$ in a (possibly subsampled) image I. In principle, any object detector can be used. In this thesis, we use the discriminatively-trained parts-based object detector developed by Felzenszwalb et al. (2008, 2010). The first type of feature we define is based on this object detector output.

In order to make learning more effective,[2] it is important to keep feature values from taking on arbitrarily high or low values with respect to the other features in the model. We also want to make it likely that detector scores at correct locations and scales are positive, so that a single linear weight can be effective. Therefore, we add a class-specific offset τ_c to the detector output, and then scale and cap the result so that it lies in $[-1, 1]$:

$$f(c, \mathbf{v}, I) = \begin{cases} \min\left(1, \max\left(-1, \frac{D(c,\mathbf{v},I)+\tau_c}{\kappa_{\text{obj}}}\right)\right) & \text{if } c \in B \\ 0 & \text{otherwise} \end{cases}$$

where κ_{obj} is a constant chosen such that most (offset) detector scores will already fall into $[-1, 1]$ before capping. The Felzenszwalb detector learns a conservative per-class threshold, intended to be used to produce high-recall (and low-precision) results. We use the negative of these thresholds for the values of τ_c in this thesis—thus, ideally, all but very low-confidence object detector scores are positive. We then use $\kappa_{\text{obj}} = 1$.

Then, for each class $c \in (B \cup C)$ in the WGG, we define a feature:

$$\phi_c^{\text{f},1}(I, T) = \sum_{\substack{t \in T \text{ s.t.} \\ \text{class}(t) = c}} f(c, \text{geom}(t), I)$$

There is also corresponding scalar weight $\theta_c^{\text{f},1}$, which will be set during learning. The feature and its corresponding weight for class c are always zero if c is not primitive, but it will be convenient later to make the features well-defined for all classes in the WGG.

Notice that in the features we have just defined, the same weight is used for each node with primitive class c in a tree, regardless of which rule part is associated with the node. Thus, we can interpret the weight for each class as a measure of confidence in the quality of the object detector for that class.

However, it would be more expressive to allow each rule part its own weight. Then, the model could represent that we may trust the object detector to a greater or less degree depending on the context of the object in the scene. This can be achieved easily. For each rule part k, we define a feature:

$$\phi_k^{\text{f},2}(I, T) = \sum_{\substack{t \in T \text{ s.t.} \\ \text{rpid}(t) = k}} f(\text{class}(t), \text{geom}(t), I)$$

and a corresponding scalar weight $\theta_k^{\text{f},2}$. Again, this feature and its weight are always 0 if rule part k does not have a primitive class label.

The disadvantage of using rule-part-based object detector features is that it introduces many more parameters, and thus makes the learning problem harder. We will evaluate both types of features in the experiments in Chapters 3 and 4.

[2] We will describe the reasons for this when we discuss the structured perceptron algorithm in Section 3.1.

2.3.2 Features and Weights on Geometry

The first type of geometry features we define are *relative*; they capture the relative location and scale of each non-root node with respect to the geometry of its parent.[3] Relative geometry features allow the model to represent expected relationships among objects while being insensitive to the absolute position or size of objects in the image, which we expect to vary widely across different images.

Given a child node with RPID k and absolute geometry vector $\mathbf{v}' = \langle x', y', s' \rangle$, and a parent node with absolute geometry vector $\mathbf{v} = \langle x, y, s \rangle$, we compute the relative geometry feature vector of the child with respect to the parent as:

$$g(k, \mathbf{v}', \mathbf{v}) = \langle \bar{x}^2, \bar{y}^2, \bar{s}^2, \bar{x}, \bar{y}, \bar{s} \rangle$$

where

$$\bar{x} = \min\left(1, \max\left(-1, \frac{s(x'-x) - \alpha_{k,\mathrm{x}}}{\kappa_{\mathrm{geom,x}}}\right)\right)$$
$$\bar{y} = \min\left(1, \max\left(-1, \frac{s(y'-y) - \alpha_{k,\mathrm{y}}}{\kappa_{\mathrm{geom,y}}}\right)\right)$$
$$\bar{s} = \min\left(1, \max\left(-1, \frac{\log s' - \log s - \alpha_{k,\mathrm{s}}}{\kappa_{\mathrm{geom,s}}}\right)\right)$$

The values $s(x'-x)$ and $s(y'-y)$ in the first two expressions represent the position of the child node relative to the parent position, at an image scale s such that the parent has the canonical size $\langle W_{c,\mathrm{width}}, W_{c,\mathrm{height}} \rangle$ for its class c. Similarly, $\log s' - \log s$ in the third expression is the relative difference in scale between the child and parent nodes, computed in log space since s and s' are multiplicative factors.[4] Then, $\langle \alpha_{k,\mathrm{x}}, \alpha_{k,\mathrm{y}}, \alpha_{k,\mathrm{s}} \rangle$ are rule-part-specific offsets that represent the expected position and scale of the child relative to the parent. Finally, $\langle \kappa_{\mathrm{geom,x}}, \kappa_{\mathrm{geom,y}}, \kappa_{\mathrm{geom,s}} \rangle$ are constants to encourage the values of $\bar{x}$, $\bar{y}$, and $\bar{s}$ to lie in $[-1, 1]$, and then they are capped to ensure this is true. For the experiments in this thesis, we use:

$$\kappa_{\mathrm{geom,x}} = \frac{W_{c,\mathrm{width}}}{2} \qquad \kappa_{\mathrm{geom,y}} = \frac{W_{c,\mathrm{height}}}{2} \qquad \kappa_{\mathrm{geom,s}} = 2$$

where we choose the last value because the log of the smallest scale factor the object detector considers is roughly 2.

Now, for each rule part k in the WGG, and a given scene tree T, we define a vector of features by summing over the relative geometry feature vectors for all nodes in the tree with RPID k:

$$\phi_k^{\mathrm{g}}(I, T) = \sum_{\substack{t \in T \text{ s.t.} \\ \mathrm{rpid}(t) = k}} g(k, \mathrm{geom}(t), \mathrm{geom}(\mathrm{parent}(t)))$$

And there is, of course, a corresponding vector of weights θ_k^{g}.

By taking the inner product of the relative geometry features $g(k, \mathbf{v}', \mathbf{v}) = \langle \bar{x}^2, \bar{y}^2, \bar{s}^2, \bar{x}, \bar{y}, \bar{s} \rangle$ and the vector of weights, we can express a quadratic function on the relative position and scale of a child given its parent. Put another way, the representation is a parabola in 4-dimensional space in which the maximum output score is obtained at the expected values of the input variables $\langle \bar{x}, \bar{y}, \bar{s} \rangle$. The weights determine both the values of $\langle \bar{x}, \bar{y}, \bar{s} \rangle$ at which the function is maximized, as well as

[3]This explains why we can interpret the geometry vector of a non-leaf node as a coordinate frame in which we model the geometry of its children.

[4]All logs in this thesis are the natural log.

the width or steepness of the parabola—how quickly the score drops off as the input variables get further from their expected values.

It may seem counterintuitive that the values $\langle \alpha_{k,\mathrm{x}}, \alpha_{k,\mathrm{y}}, \alpha_{k,\mathrm{s}} \rangle$ are not necessarily the final expected relative location and scale—in fact, they only represent initial guesses. This fact arises from the basic equation for a parabola. The quadratic function $ax^2 + bx + c = 0$ is maximized when $x = -b/2a$. This means that by changing the weights (the 3-dimensional equivalents of a and b), the parabola can be centered on any arbitrary values of $\langle \overline{x}, \overline{y}, \overline{s} \rangle$.

The second type of geometry features are *absolute*; they capture the absolute location and scale of the root node of a scene tree. The absolute geometry features represent the expected location and size of the entire group of objects in the image. For this reason, we only compute them for the root scene classes in the grammar.

Given a node with class label c and absolute geometry vector $\mathbf{v} = \langle x, y, s \rangle$, and an image I with dimensions $\langle \mathrm{width}(I), \mathrm{height}(I) \rangle$, we compute the absolute geometry feature vector of the node as:

$$h(c, \mathbf{v}, I) = \langle \widehat{x}^2, \widehat{y}^2, \widehat{s}^2, \widehat{x}, \widehat{y}, \widehat{s} \rangle$$

where

$$\widehat{x} = \frac{2x - \mathrm{width}(I)}{\mathrm{width}(I)} \qquad \widehat{y} = \frac{2y - \mathrm{height}(I)}{\mathrm{height}(I)} \qquad \widehat{s} = \min\left(1, \max\left(-1, \frac{\log s}{\kappa_{\mathrm{geom,s}}}\right)\right)$$

Again, the first two expressions simply shift and scale x and y according to the image dimensions, while the third expression scales and caps the log scale factor, so that $\widehat{x}$, $\widehat{y}$, and $\widehat{s}$ all lie in $[-1, 1]$.

Now, for each class c in the WGG, we define a vector of features:

$$\phi_c^{\mathrm{h}}(I, T) = \sum_{\substack{t \in T \text{ s.t.} \\ \mathrm{class}(t) = c}} h(c, \mathrm{geom}(t), I)$$

And again there is a corresponding vector of weights θ_c^{h}, which we assume to be zero if $c \notin S$. Thus, in practice, the feature will only be computed on the root node of the tree, for mathematical convenience we write it as a sum over all the nodes in the tree.

Consider the different roles played by the relative versus absolute geometry features. In a place-setting, for example, the relative geometry features might represent that we expect to see a fork to the left of the plate, and a knife to the right, while remaining independent to the absolute location of the plate, knife, and fork in the image. The absolute geometry features, on the other hand, might represent that we expect to see the entire group of plate, knife, and fork in the center of the image and occupying most of it. The latter expectation may be less strong than the former, in which case the weights on the absolute geometry features can be lower than those on the relative features.

2.3.3 Features and Weights on Tree Structure

The final category of features and weights are based on the structure of a scene tree. Recall that in a WGG, each part on the right side of a rule is always optional. The structural features, and the weights on them, let the model capture the relative likelihood of different subsets of the parts of a rule occurring together.

First, for each rule part in k in the WGG, and a given scene tree T, we define a feature:

$$\phi_k^{\mathrm{rp}}(I, T) = \sum_{\substack{t \in T \text{ s.t.} \\ \mathrm{rpid}(t) = k}} 1$$

and a corresponding scalar weight θ_k^{rp}. These features simply count the number of times each rule part appears in the tree, independently of any other parts. Thus, they provide only a weak model for the expected appearance of each part, and cannot model preferences for the co-occurrence of multiple parts.

For this reason, we define an additional set of *pairwise* structural features. For each pair of rule parts $\langle k_1, k_2 \rangle$ in the same rule in the WGG, and given a scene tree T, we define a feature:

$$\phi^{\mathrm{pw}}_{k_1,k_2}(I,T) = \sum_{\substack{t_1,t_2 \in T \text{ s.t.} \\ \mathrm{parent}(t_1)=\mathrm{parent}(t_2), \\ \mathrm{rpid}(t_1)=k_1 \text{ and } \mathrm{rpid}(t_2)=k_2}} 1$$

and a corresponding scalar weight $\theta^{\mathrm{pw}}_{k_1,k_2}$. These features count the number of times each pair of rule parts occurs (as a pair of sibling nodes) in the tree.

In Section 2.2.1, we discussed ways to express the notion of optional rule parts within the traditional CFG framework. We said that the idea of having every rule part always be optional was equivalent to simply introducing an intermediate composite class for each rule part. But in fact, the pairwise structural features we just introduced would not be possible if we were using intermediate classes, because the rules of the grammar representing that the parts are optional would be separate from the rule grouping the parts into a composite class. This observation makes clear that the WGG formalism is actually *not* context-free, but rather context-sensitive in a very mild way. The decision of whether to include each part in a rule depends on what other parts are also being included. This is similar to the first approach we proposed for adapting CFGs—we could have multiple rules for each composite class that are variations on a theme, and have weights on each rule. But recall that in that approach, we could not easily share geometry or object detector weights across the rules. The framework of optional parts with pairwise weights in WGGs lets us borrow some of the best of both of these approaches to capturing structural variability.

2.3.4 WGGs as Linear Models

We can now define the feature representation $\Phi(I,T)$ in a WGG as the concatenated output of the feature functions we defined in the previous three sections. Similarly, the parameter vector Θ is the concatenation of the weight vectors:[5]

$$\Phi(I,T) = \begin{bmatrix} \forall_{c \in B \cup C}\, \phi^{\mathrm{f},1}_c(I,T) \\ \forall_{r \in R}\, \forall_{k \in r}\, \phi^{\mathrm{f},2}_k(I,T) \\ \forall_{r \in R}\, \forall_{k \in r}\, \phi^{\mathrm{g}}_k(I,T) \\ \forall_{c \in B \cup C}\, \phi^{\mathrm{h}}_c(I,T) \\ \forall_{r \in R}\, \forall_{k \in r}\, \phi^{\mathrm{rp}}_k(I,T) \\ \forall_{r \in R}\, \forall_{\langle k_1,k_2 \rangle \in r} \phi^{\mathrm{pw}}_{k_1,k_2}(I,T) \end{bmatrix} \qquad \Theta = \begin{bmatrix} \forall_{c \in B \cup C}\, \theta^{\mathrm{f},1}_c \\ \forall_{r \in R}\, \forall_{k \in r}\, \theta^{\mathrm{f},2}_k \\ \forall_{r \in R}\, \forall_{k \in r}\, \theta^{\mathrm{g}}_k \\ \forall_{c \in B \cup C}\, \theta^{\mathrm{h}}_c \\ \forall_{r \in R}\, \forall_{k \in r}\, \theta^{\mathrm{rp}}_k \\ \forall_{r \in R}\, \forall_{\langle k_1,k_2 \rangle \in r} \theta^{\mathrm{pw}}_{k_1,k_2} \end{bmatrix} \tag{2.2}$$

where we use the subscripts $k \in r$ and $\langle k_1, k_2 \rangle \in r$ to denote enumerating over each rule part k or pair of rule parts $\langle k_1, k_2 \rangle$, respectively, on the right side of rule r.

Finally, we must define the function **GEN** which enumerates the candidate scene trees $\mathbf{GEN}(I)$ for an image I. We take $\mathbf{GEN}(I)$ to be the set of scene trees that have valid structure according to G, and whose geometry vectors contain only valid locations and scales given the dimensions and feasible resolutions of the (possibly subsampled) image I.

[5] In practice, we only use *either* the class-based object detector features $\phi^{\mathrm{f},1}_c(I,T)$ or the rule-part-based version $\phi^{\mathrm{f},2}_k(I,T)$. But we include both in what follows; whichever one is unused is simply removed from the parsing expressions.

2.4 Parsing

Imagine we are given a WGG $G = \langle B, C, R, S, \Phi, \Theta \rangle$. The parsing task is to compute Equation (2.1); i.e., to find the best scene tree T given an image I:

$$F(I) = \underset{T \in \mathbf{GEN}(I)}{\operatorname{argmax}} \; \Phi(I,T) \cdot \Theta \tag{2.1}$$

The challenge is that the number of candidate trees in $\mathbf{GEN}(I)$ is exponential in the size of the grammar and the image. A naive approach, in which we enumerate each tree $T \in \mathbf{GEN}(I)$, compute the tree's score $\Phi(I,T) \cdot \Theta$, and choose the tree with the highest score, would be completely intractable.

However, we intentionally constructed the feature functions in $\Phi(I,T)$ to be simple counts or sums of component functions, each of which operates on a local piece of a scene tree—a single node, a node and its parent, or two sibling nodes. Thus, we can derive a CYK-style parsing algorithm (Jurafsky & Martin, 2000) that uses dynamic programming to find the best scene tree in polynomial time in the size of the grammar (number of classes and rule parts) and the size of the image.

2.4.1 Scoring a Scene Tree

First, let's consider how we would score a given tree. Let $Q(I,T) = \Phi(I,T) \cdot \Theta$ be the score of a fixed scene tree T, given an image I. Expand $\Phi(I,T)$ and Θ using Equation (2.2):

$$\begin{aligned}
Q(I,T) &= \Phi(I,T) \cdot \Theta \\
&= \left(\sum_{c \in B \cup C} \theta_c^{\mathrm{f},1} \phi_c^{\mathrm{f},1}(I,T) \right) + \left(\sum_{r \in R} \sum_{k \in r} \theta_k^{\mathrm{f},2} \phi_k^{\mathrm{f},2}(I,T) \right) + \left(\sum_{r \in R} \sum_{k \in r} \theta_k^{\mathrm{g}} \cdot \phi_k^{\mathrm{g}}(I,T) \right) \\
&\quad + \left(\sum_{c \in B \cup C} \theta_c^{\mathrm{h}} \cdot \phi_c^{\mathrm{h}}(I,T) \right) + \left(\sum_{r \in R} \sum_{k \in r} \theta_k^{\mathrm{rp}} \phi_k^{\mathrm{rp}}(I,T) \right) + \left(\sum_{r \in R} \sum_{\langle k_1, k_2 \rangle \in r} \theta_{k_1,k_2}^{\mathrm{pw}} \phi_{k_1,k_2}^{\mathrm{pw}}(I,T) \right)
\end{aligned} \tag{2.3}$$

Consider the first term in this expression. Replace $\phi_c^{\mathrm{f},1}(I,T)$ with its definition, and then rearrange to get an expression in terms of nodes $t \in T$:

$$\begin{aligned}
\sum_{c \in B \cup C} \theta_c^{\mathrm{f},1} \phi_c^{\mathrm{f},1}(I,T) &= \sum_{c \in B \cup C} \theta_c^{\mathrm{f},1} \sum_{\substack{t \in T \text{ s.t.} \\ \mathrm{class}(t) = c}} f(c, \mathrm{geom}(t), I) \\
&= \sum_{t \in T} \theta_{\mathrm{class}(t)}^{\mathrm{f},1} f(\mathrm{class}(t), \mathrm{geom}(t), I)
\end{aligned}$$

We can do something similar with the second term:

$$\begin{aligned}
\sum_{r \in R} \sum_{k \in r} \theta_k^{\mathrm{f},2} \phi_k^{\mathrm{f},2}(I,T) &= \sum_{r \in R} \sum_{k \in r} \theta_k^{\mathrm{f},2} \sum_{\substack{t \in T \text{ s.t.} \\ \mathrm{rpid}(t) = k}} f(\mathrm{class}(t), \mathrm{geom}(t), I) \\
&= \sum_{\substack{t \in T \text{ s.t.} \\ t \neq \mathrm{root}(T)}} \theta_{\mathrm{rpid}(t)}^{\mathrm{f},2} f(\mathrm{class}(t), \mathrm{geom}(t), I)
\end{aligned}$$

and the last four terms:

$$\sum_{r \in R} \sum_{k \in r} \theta_k^{\mathrm{g}} \cdot \phi_k^{\mathrm{g}}(I,T) = \sum_{\substack{t \in T \text{ s.t.} \\ t \neq \mathrm{root}(T)}} \theta_{\mathrm{rpid}(t)}^{\mathrm{g}} \cdot g(\mathrm{rpid}(t), \mathrm{geom}(t), \mathrm{geom}(\mathrm{parent}(t)))$$

$$\sum_{c\in B\cup C} \theta^{\mathrm{h}}_c \cdot \phi^{\mathrm{h}}_c(I,T) = \sum_{t\in T} \theta^{\mathrm{h}}_{\mathrm{class}(t)} \cdot h(\mathrm{class}(t), \mathrm{geom}(t), I)$$

$$\sum_{r\in R}\sum_{k\in r} \theta^{\mathrm{rp}}_k \phi^{\mathrm{rp}}_k(I,T) = \sum_{\substack{t\in T \text{ s.t.}\\ t\neq \mathrm{root}(T)}} \theta^{\mathrm{rp}}_{\mathrm{rpid}(t)}$$

$$\sum_{r\in R}\sum_{\langle k_1,k_2\rangle\in r} \phi^{\mathrm{pw}}_{k_1,k_2}(I,T)\theta^{\mathrm{pw}}_{k_1,k_2} = \sum_{\substack{\langle t_1,t_2\rangle\in T \text{ s.t.}\\ \mathrm{parent}(t_1)=\mathrm{parent}(t_2)}} \theta^{\mathrm{pw}}_{\mathrm{rpid}(t_1),\mathrm{rpid}(t_2)}$$

Now, replace the terms in Equation (2.3) with those we just found, and group them according to the domain of each sum:

$$\begin{aligned}
Q(I,T) = &\sum_{t\in T}\left(\theta^{\mathrm{f},1}_{\mathrm{class}(t)} f(\mathrm{class}(t), \mathrm{geom}(t), I) + \theta^{\mathrm{h}}_{\mathrm{class}(t)} \cdot h(\mathrm{class}(t), \mathrm{geom}(t), I)\right)\\
&+ \sum_{\substack{t\in T \text{ s.t.}\\ t\neq \mathrm{root}(T)}} \Big(\theta^{\mathrm{rp}}_{\mathrm{rpid}(t)} + \theta^{\mathrm{f},2}_{\mathrm{rpid}(t)} f(\mathrm{class}(t), \mathrm{geom}(t), I)\\
&\qquad + \theta^{\mathrm{g}}_{\mathrm{rpid}(t)} \cdot g(\mathrm{rpid}(t), \mathrm{geom}(t), \mathrm{geom}(\mathrm{parent}(t)))\Big)\\
&+ \sum_{\substack{\langle t_1,t_2\rangle\in T \text{ s.t.}\\ \mathrm{parent}(t_1)=\mathrm{parent}(t_2)}} \theta^{\mathrm{pw}}_{\mathrm{rpid}(t_1),\mathrm{rpid}(t_2)}
\end{aligned}$$

Let $\mathrm{children}(t)$ be the child nodes of node t, and $\mathrm{subtree}(t)$ be the subtree of T rooted at node t. Then we can rewrite $Q(I,T)$ recursively:

$$\begin{aligned}
&= \theta^{\mathrm{f},1}_{\mathrm{class}(\mathrm{root}(T))} f(\mathrm{class}(\mathrm{root}(T)), \mathrm{geom}(\mathrm{root}(T)), I)\\
&\quad + \theta^{\mathrm{h}}_{\mathrm{class}(\mathrm{root}(T))} \cdot h(\mathrm{class}(\mathrm{root}(T)), \mathrm{geom}(\mathrm{root}(T)), I)\\
&\quad + \sum_{t\in \mathrm{children}(\mathrm{root}(T))} \Big(\theta^{\mathrm{rp}}_{\mathrm{rpid}(t)} + \theta^{\mathrm{f},2}_{\mathrm{rpid}(t)} f(\mathrm{class}(t), \mathrm{geom}(t), I)\\
&\qquad + \theta^{\mathrm{g}}_{\mathrm{rpid}(t)} \cdot g(\mathrm{rpid}(t), \mathrm{geom}(t), \mathrm{geom}(\mathrm{root}(T))) + Q(I, \mathrm{subtree}(t))\Big)\\
&\quad + \sum_{\langle t_1,t_2\rangle\in \mathrm{children}(\mathrm{root}(T))} \theta^{\mathrm{pw}}_{\mathrm{rpid}(t_1),\mathrm{rpid}(t_2)}
\end{aligned}$$

2.4.2 Finding the Max-Scoring Tree

For the parsing task, we want to find the scene tree T that maximizes the score $Q(I,T)$. Let $Z(c,\mathbf{v},I)$ be the score of best-scoring subtree rooted at class c and location/scale $\mathbf{v}$:

$$\begin{aligned}
Z(c,\mathbf{v},I) = &\max_{\substack{T\in\mathbf{GEN}(I) \text{ s.t.}\\ \mathrm{class}(\mathrm{root}(T))=c\ \wedge\\ \mathrm{geom}(\mathrm{root}(T))=\mathbf{v}}} Q(I,T)\\
= &\ \theta^{\mathrm{f},1}_c f(c,\mathbf{v},I) + \theta^{\mathrm{h}}_c \cdot h(c,\mathbf{v},I)\\
&+ \max_{\substack{T\in\mathbf{GEN}(I) \text{ s.t.}\\ \mathrm{class}(\mathrm{root}(T))=c\ \wedge\\ \mathrm{geom}(\mathrm{root}(T))=\mathbf{v}}} \sum_{t\in \mathrm{children}(\mathrm{root}(T))} \Big(\theta^{\mathrm{rp}}_{\mathrm{rpid}(t)} + \theta^{\mathrm{f},2}_{\mathrm{rpid}(t)} f(\mathrm{class}(t), \mathrm{geom}(t), I)\\
&\qquad + \theta^{\mathrm{g}}_{\mathrm{rpid}(t)} \cdot g(\mathrm{rpid}(t), \mathrm{geom}(t), \mathbf{v}) + Q(I, \mathrm{subtree}(t))\Big)\\
&\qquad + \sum_{\langle t_1,t_2\rangle\in \mathrm{children}(\mathrm{root}(T))} \theta^{\mathrm{pw}}_{\mathrm{rpid}(t_1),\mathrm{rpid}(t_2)}
\end{aligned}$$

Without Pairwise Rule Part Features

First, consider the case in which we ignore the pairwise rule part features:

$$Z^{\text{nopair}}(c,\mathbf{v},I) = \theta_c^{\text{f},1} f(c,\mathbf{v},I) + \theta_c^{\text{h}} \cdot h(c,\mathbf{v},I)$$
$$+ \max_{\substack{T\in\mathbf{GEN}(I)\text{ s.t.}\\ \text{class}(\text{root}(T))=c\ \wedge\\ \text{geom}(\text{root}(T))=\mathbf{v}}} \sum_{t\in\text{children}(\text{root}(T))} \Big(\theta^{\text{rp}}_{\text{rpid}(t)} + \theta^{\text{f},2}_{\text{rpid}(t)} f(\text{class}(t),\text{geom}(t),I)$$
$$+\theta^{\text{g}}_{\text{rpid}(t)}\cdot g(\text{rpid}(t),\text{geom}(t),\mathbf{v}) + Q(I,\text{subtree}(t))\Big)$$

We can flip the max and the sum over child nodes, because the search for the max subtree for each rule part k can happen independently. Also, because each rule part is optional, we take the max of 0 and the score for the best subtree T' rooted at part k. Only subtrees with positive scores can contribute to the best global tree, so if the best subtree T' has a negative score, it does not get added:

$$= \theta_c^{\text{f},1} f(c,\mathbf{v},I) + \theta_c^{\text{h}} \cdot h(c,\mathbf{v},I)$$
$$+ \sum_{k\in\text{rule}(c)} \max_{\substack{T'\in\mathbf{GEN}(I)\text{ s.t.}\\ \text{rpid}(\text{root}(T'))=k}} \max\Big(0, \theta_k^{\text{rp}} + \theta_k^{\text{f},2} f(\text{class}(k),\text{geom}(\text{root}(T')),I)$$
$$+\theta_k^{\text{g}}\cdot g(k,\text{geom}(\text{root}(T')),\mathbf{v}) + Q(I,T')\Big)$$

Now, we can separate the max over subtrees T' into a max over root location/scales $\mathbf{v}'$ for the subtree, and a max over subtrees T' with root location/scale $\mathbf{v}'$:

$$= \theta_c^{\text{f},1} f(c,\mathbf{v},I) + \theta_c^{\text{h}} \cdot h(c,\mathbf{v},I)$$
$$+ \sum_{k\in\text{rule}(c)} \max\Big(0, \theta_k^{\text{rp}} + \max_{\mathbf{v}'}\Big(\theta_k^{\text{f},2} f(\text{class}(k),\mathbf{v}',I) + \theta_k^{\text{g}}\cdot g(k,\mathbf{v}',\mathbf{v})$$
$$+ \max_{\substack{T'\in\mathbf{GEN}(I)\text{ s.t.}\\ \text{class}(\text{root}(T'))=\text{class}(k)\ \wedge\\ \text{geom}(\text{root}(T'))=\mathbf{v}'}} Q(I,T')\Big)\Big)$$

Finally, we recognize the final max over subtrees as the recursive step:

$$= \theta_c^{\text{f},1} f(c,\mathbf{v},I) + \theta_c^{\text{h}} \cdot h(c,\mathbf{v},I)$$
$$+ \sum_{k\in\text{rule}(c)} \max\Big(0, \theta_k^{\text{rp}} + \max_{\mathbf{v}'}\Big(\theta_k^{\text{f},2} f(\text{class}(k),\mathbf{v}',I) + \theta_k^{\text{g}}\cdot g(k,\mathbf{v}',\mathbf{v}) + Z^{\text{nopair}}(\text{class}(k),\mathbf{v}',I)\Big)\Big)$$

Rewriting slightly, we have:

$$Z^{\text{nopair}}(c,\mathbf{v},I) = \underbrace{\theta_c^{\text{f},1} f(c,\mathbf{v},I)}_{\textbf{object detector output}} + \underbrace{\theta_c^{\text{h}} \cdot h(c,\mathbf{v},I)}_{\textbf{absolute geometry}} + \sum_{k\in\text{rule}(c)} \underbrace{\max(0,\ z^{\text{nopair}}(c,k,\mathbf{v},I))}_{\textbf{whether to add rule part}} \tag{2.4}$$

where

$$z^{\text{nopair}}(c,k,\mathbf{v},I) = \underbrace{\theta_k^{\text{rp}}}_{\textbf{part weight}} + \underbrace{\max_{\mathbf{v}'}}_{\textbf{child locations}} \Big(\underbrace{\theta_k^{\text{f},2} f(\text{class}(k),\mathbf{v}',I)}_{\textbf{object detector output}} + \underbrace{\theta_k^{\text{g}}\cdot g(k,\mathbf{v}',\mathbf{v})}_{\textbf{relative geometry}} + \underbrace{Z^{\text{nopair}}(\text{class}(k),\mathbf{v}',I)}_{\textbf{recursion}}\Big) \tag{2.5}$$

is the score of the best subtree rooted at rule part k, given a parent with class c and geometry $\mathbf{v}$.

Writing the score in this manner shows that, in the absence of pairwise rule part features, each individual rule part weight θ_k^{rp} can be interpreted as (the negative of) a threshold on the score of the subtree rooted at rule part k. If the score of the subtree rooted at the best child location $\mathbf{v}'$ is less than $-\theta_k^{\text{rp}}$, the subtree score $z^{\text{nopair}}(c,k,\mathbf{v},I)$ will be non-positive, and the subtree will not be included in the global best tree.

Thus, in the absence of pairwise rule part features, Equations 2.4 and 2.5 lead directly to an efficient and exact dynamic programming parsing algorithm, in which a table is maintained that stores the score $Z^{\text{nopair}}(c,\mathbf{v},I)$ for each class c at each location and scale $\mathbf{v}$ in the image I. Once the table is fully constructed, the score of the best overall scene tree for the image I is given by:

$$\max_{T\in\mathbf{GEN}(I)} Q^{\text{nopair}}(I,T) = \operatorname*{argmax}_{c\in S,\mathbf{v}} Z^{\text{nopair}}(c,\mathbf{v},I)$$

where $Q^{\text{nopair}}(I,T)$ is as in Equation 2.3 but without the pairwise term, and S is the set of root scene classes in the WGG. By maintaining an additional chart that keeps track of the best subtree (which rule parts are added, and their geometry values $\mathbf{v}'$) associated with each parent $(c,\mathbf{v})$ in the table, the best-scoring scene tree T itself can be reconstructed.

With Pairwise Rule Part Features

To reintroduce pairwise rule part features, we can still find the best subtree for each rule part k independently, but we cannot decide whether to add each subtree without making a joint decision. To compute this exactly, we would need to consider all subsets of rule parts in the rule:

$$Z(c,\mathbf{v},I) = \underbrace{\theta_c^{\text{f},1} f(c,\mathbf{v},I)}_{\textbf{object detector output}} + \underbrace{\theta_c^{\text{h}}\cdot h(c,\mathbf{v},I)}_{\textbf{absolute geometry}} + \underbrace{\max_{K\in\mathcal{P}(\text{rule}(c))}}_{\textbf{all subsets of rule parts}} \left(\underbrace{\sum_{\langle k_1,k_2\rangle\in K} \theta_{k_1,k_2}^{\text{pw}}}_{\textbf{pairwise weights}} + \underbrace{\sum_{k\in K} z(c,k,\mathbf{v},I)}_{\textbf{subtree scores}} \right) \tag{2.6}$$

where $\mathcal{P}(\text{rule}(c))$ is the set of all subsets of the rule parts in the rule for class c, and

$$z(c,k,\mathbf{v},I) = \underbrace{\theta_k^{\text{rp}}}_{\textbf{part weight}} + \underbrace{\max_{\mathbf{v}'}}_{\textbf{child locations}} \left(\underbrace{\theta_k^{\text{f},2} f(\text{class}(k),\mathbf{v}',I)}_{\textbf{object detector output}} + \underbrace{\theta_k^{\text{g}}\cdot g(k,\mathbf{v}',\mathbf{v})}_{\textbf{relative geometry}} + \underbrace{Z(\text{class}(k),\mathbf{v}',I)}_{\textbf{recursion}} \right) \tag{2.7}$$

as before.

In general, it will not be tractable to enumerate all subsets of rule parts for each rule in the grammar. Therefore, we adopt a greedy approach to this one aspect of an otherwise optimal algorithm. Our approach is inspired by Torralba et al. (2007), who proposed a similar algorithm to search for the best set of object classes to share a feature on each round of boosting.

For a parent class c and location/scale $\mathbf{v}$, we must decide which of the N rule parts in rule(c) should have their subtrees included in the best tree rooted at $(c,\mathbf{v})$. For each rule part k, we precompute the score $z(c,k,\mathbf{v},I)$ of its best subtree. First, we select the rule part k_1 with the best score:

$$k_1 = \operatorname*{argmax}_{k\in\text{rule}(c)} z(c,k,\mathbf{v},I)$$

Then, we select the next rule part k_2 with the best score, jointly with the previously chosen part k_1:

$$k_2 = \operatorname*{argmax}_{k\in(\text{rule}(c)-\{k_1\})} z(c,k,\mathbf{v},I) + \theta_{k_1,k}^{\text{pw}}$$

This continues, so that the ith selected part is chosen as:

$$k_i = \operatorname*{argmax}_{k \in (\text{rule}(c) - \cup_{j=1}^{i-1}\{k_j\})} z(c,k,\mathbf{v},I) + \sum_{j=1}^{i-1} \theta^{\text{pw}}_{k_j,k}$$

until all parts k in rule(c) have been selected.

Each k_i for $i = 1 \ldots N$ corresponds to a subset of i of the N parts—specifically k_1 through k_i. And the ith subset has an associated joint score:

$$\text{score}(i) = \sum_{j=1}^{i-1} \sum_{\ell=j+1}^{i} \theta^{\text{pw}}_{k_j,k_\ell} + \sum_{j=1}^{i} z(c,k_j,\mathbf{v},I)$$

We then pick the subset i with the highest score, unless no subsets have positive scores, in which case we add no rule parts. Thus the final set of rule parts can have any size between 0 and N.

Because this algorithm is greedy, it is incapable of looking past rule parts with very good individual scores $z(c,k,\mathbf{v},I)$—it must either include the top-scoring part or none at all. However, in practice it tends to be quite robust.

This approach replaces the exhaustive search over all rule part subsets in Equation 2.6. Otherwise, we can implement a parsing algorithm based on Equations 2.6 and 2.7 in the same way we described for the no-pairwise case.

2.4.3 Practical Considerations for Parsing

The parsing algorithm must be implemented with an awareness of the domain of $\mathbf{v}$. The variable $\mathbf{v}$ represents the location and scale $\langle x,y,s\rangle$ of an object or composite class instance, and in practice it ranges over the $\langle x,y\rangle$ coordinates of each pixel at each level of an image pyramid—the (possibly subsampled) image I, resized using a discrete set of scale factors s.

This has several implications. The first is computational. To perform the max over child locations $\mathbf{v}'$ in Equation 2.7, a naive approach would consider every possible child location $\langle x',y'\rangle$ and scale s' for each fixed parent location $\langle x,y\rangle$ and scale s, which would result in a parsing algorithm with complexity that is squared in the total number of pixels in the image pyramid. To perform this search more efficiently, we adapt the generalized distance transform techniques developed by Felzenszwalb and Huttenlocher (2004, 2005).

Using Distance Transform Techniques

Felzenszwalb and Huttenlocher (2004) define the distance transform as follows. If $\mathcal{G}$ is a regular grid and $f : \mathcal{G} \to \mathbb{R}$ is a function on the grid, the distance transform of f is:

$$\mathcal{D}_f(p) = \min_{q \in \mathcal{G}} (d(p,q) + f(q))$$

where $d(p,q)$ is a measure of the distance between p and q. In the case when the distance function is the squared Euclidean norm $d(p,q) = (p-q)^2$, they show that performing the one-dimensional distance transform is equivalent to computing the lower envelope of the set of parabolas rooted at $(q, f(q))$, as shown in Figure 2-2, and give an algorithm for computing this envelope in linear time.

For more complicated distance functions, linear minimization is often still possible. Felzen-

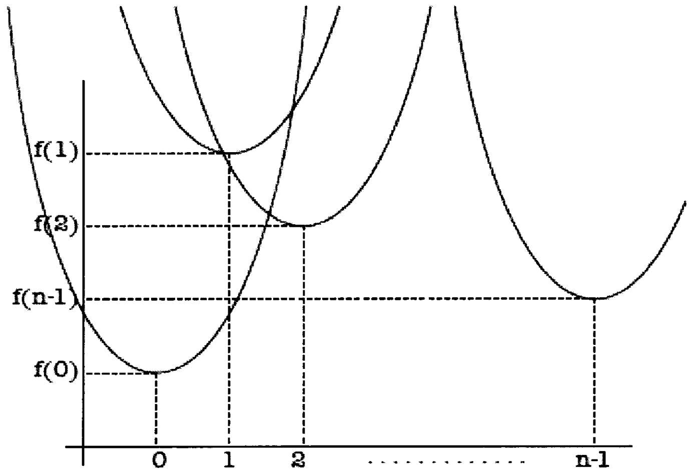


Figure 2-2: The one dimensional distance transform for squared Euclidean distance, as the lower envelope of n parabolas. From Felzenszwalb & Huttenlocher, 2004.

szwalb et al. (2008) use this distance function in their object recognition system:

$$d(p,q) = a(p-q)^2 + b(p-q)$$

They use a recursive algorithm to compute the distance transform under this function in linear time. Although the recursive algorithm does not seem to be documented in their papers, the released code has an implementation.[6]

In our context, we want to write the maximization over child locations in Equation 2.7:

$$\max_{\mathbf{v}'} \left(\theta_k^{\mathrm{f},2} f(\mathrm{class}(k), \mathbf{v}', I) + \theta_k^{\mathrm{g}} \cdot g(k, \mathbf{v}', \mathbf{v}) + Z(\mathrm{class}(k), \mathbf{v}', I) \right) \tag{2.8}$$

in terms of a distance transform over child locations $\langle x', y' \rangle$. First, define

$$\hat{f}(x', y', s') = \theta_k^{\mathrm{f},2} f(\mathrm{class}(k), \langle x', y', s' \rangle, I) + Z(\mathrm{class}(k), \langle x', y', s' \rangle, I) \tag{2.9}$$

to be everything inside the max except the relative geometry term. (We use $\hat{f}$ to distinguish this function from f, our features on object detector output.) Then, recall the definition of $g(k, \mathbf{v}', \mathbf{v})$, and that we can write θ_k^{g} as a vector:

$$g(k, \mathbf{v}', \mathbf{v}) = \langle \bar{x}^2 \bar{y}^2 \bar{s}^2, \bar{x}, \bar{y}, \bar{s} \rangle \qquad \theta_k^{\mathrm{g}} = \langle \theta_{k,\mathrm{x2}}^{\mathrm{g}} \theta_{k,\mathrm{y2}}^{\mathrm{g}} \theta_{k,\mathrm{s2}}^{\mathrm{g}} \theta_{k,\mathrm{x}}^{\mathrm{g}} \theta_{k,\mathrm{y}}^{\mathrm{g}} \theta_{k,\mathrm{s}}^{\mathrm{g}} \rangle$$

Now we can rewrite Equation 2.8 as:

$$\begin{aligned}
&\max_{\mathbf{v}'} \left(\theta_k^{\mathrm{g}} \cdot g(k, \mathbf{v}', \mathbf{v}) + \hat{f}(x', y', s') \right) \\
&= \max_{x', y', s'} \left(\theta_{k,\mathrm{x2}}^{\mathrm{g}} \bar{x}^2 + \theta_{k,\mathrm{y2}}^{\mathrm{g}} \bar{y}^2 + \theta_{k,\mathrm{s2}}^{\mathrm{g}} \bar{s}^2 + \theta_{k,\mathrm{x}}^{\mathrm{g}} \bar{x} + \theta_{k,\mathrm{y}}^{\mathrm{g}} \bar{y} + \theta_{k,\mathrm{s}}^{\mathrm{g}} \bar{s} + \hat{f}(x', y', s') \right) \\
&= \max_{s'} \left(\theta_{k,\mathrm{s2}}^{\mathrm{g}} \bar{s}^2 + \theta_{k,\mathrm{s}}^{\mathrm{g}} \bar{s} + D(x, y, s, s') \right)
\end{aligned}$$

[6] Available at `http://people.cs.uchicago.edu/~pff/latent/`.

where

$$D(x,y,s,s') = +\max_{x',y'} \left(\theta^{g}_{k,x2}\bar{x}^2 + \theta^{g}_{k,y2}\bar{y}^2 + \theta^{g}_{k,x}\bar{x} + \theta^{g}_{k,y}\bar{y} + \hat{f}(x',y',s') \right)$$

We can write $D(x,y,s,s')$ as a distance transform as follows. First, we assume without loss of generality that the parent and child scales s and s' are fixed and equal (we will discuss this again in the next section). Then, we can consider x and y locations in an image already scaled by s. This means that in the definition of $\bar{x}$:

$$\bar{x} = \min\left(1, \max\left(-1, \frac{s(x'-x) - \alpha_{k,\mathrm{x}}}{\kappa_{\mathrm{geom,x}}}\right)\right)$$

we can assume that $s = 1$. Next, since the x and y dimensions are separable, and their models have the same form, we will consider just the x dimension. Finally, we always can turn a max into a min by negating the arguments. This gives us:

$$D(x) = \min_{x'} \left(d(x,x') - \hat{f}(x')\right)$$

where

$$d(x,x') = a(d'(x,x'))^2 + bd'(x,x')$$

and

$$d'(x,x') = \min\left(1, \max\left(-1, \frac{x'-x-m}{w}\right)\right) \tag{2.10}$$

with

$$a = -\theta^{g}_{k,x2} \qquad b = -\theta^{g}_{k,x} \qquad m = \alpha_{k,\mathrm{x}} \qquad w = \kappa_{\mathrm{geom,x}} \tag{2.11}$$

So by substituting our definition of $d(x,x')$ into the recursive algorithm of Felzenszwalb et al. (2008), we can perform the maximization over child locations for *all* parent locations (at a fixed scale s) in time that is linear, rather than squared, in the size of the scaled image.

Different Parent and Child Scales

To derive the distance transform, we assumed that the parent and child scale factors were fixed and equal. However, we need to be able to consider the relationship between a child location and a parent location at two *different* scales in the image pyramid, despite the fact that we can only apply a distance transform in one scale.

Note that the locations $\langle x',y' \rangle$ over which $\hat{f}$ is defined in Equation 2.9 are pixels in the image pyramid at the child scale s'. In fact, $\hat{f}(x',y',s')$ is actually a matrix of scores of the same size as the image scaled by s'. We have two options. We could either rescale the child score matrix to the parent scale s, and then perform the distance transforms in x and y at the parent scale. Or, we could perform the distance transforms at the child scale, and then rescale the resulting matrix of scores to the parent scale afterwards.

The first option is conceptually simpler, since the location features and weights for a rule part k are already defined at the parent's scale. But in practice, it turns out to be very important to take the second approach. It is often the case that the image pyramid level corresponding to the child scale s' is much higher resolution than the parent scale s, because child instances tend to be (much) smaller than their parents. For example, in a two-level grammar, a child might represent a tiny fork, while the parent represents the group of all the objects in the scene. The scale at which the fork is at its canonical size may be close to the full image resolution, while the scale at which the parent is at its canonical size would be greatly subsampled.[7] By scaling the child scores *before* applying the geometry models, we would lose a great deal of precision in possible locations for the fork with respect to the parent.

To apply the distance transforms at the child scale, however, we must scale the geometry model parameters accordingly, since they are defined at the parent scale s. (To see this, note that in the definition of $\bar{x}$, we scale both the image coordinates x and x' by s before doing anything else.) So now we want to have Equation 2.10 continue to be correct, but when x and x' are locations at the child scale s', rather than the parent.

We can transform a location x at the child scale to the parent scale by dividing by $\sigma = s'/s$. So we replace x and x' in Equation 2.10 by x/σ and x'/σ, and simplify:

$$\begin{aligned} d'(x,x') &= \min\left(1, \max\left(-1, \frac{x'/\sigma - x/\sigma - m}{w}\right)\right) \\ &= \min\left(1, \max\left(-1, \left(\frac{x' - x - \sigma m}{\sigma w}\right)/\sigma\right)\right) \\ &= \min\left(1, \max\left(-1, \frac{x' - x - \sigma m}{\sigma^2 w}\right)\right) \end{aligned}$$

Thus, by redefining Equation 2.11 as:

$$a = -\theta^{g}_{k,x2} \qquad b = -\theta^{g}_{k,x} \qquad m = \sigma\alpha_{k,x} \qquad w = \sigma^2\kappa_{\text{geom},x} \tag{2.12}$$

we can use the original Equation 2.10 on locations at the child scale and have the result be correct.

Finally, the question remains of how to rescale a matrix of scores between two levels of the image pyramid. If we need to upsample the matrix (because the target pyramid level is higher resolution than the source one), we can simply replicate scores to neighboring pixels. However, if we need to downsample the matrix (because the target level is lower resolution), some amount of information must be lost. We use the fact that the sampling process is happening in the context of a maximization algorithm, and set the value of each single pixel at the target scale to be the local max score of its corresponding region of pixels at the source scale.

Mapping from Scene Trees to Object Detections

The last practical issue about parsing that we must address is how a predicted scene tree T maps to a set of object detections for an image I. As we have said, the leaves of T correspond to the set of detections, where the predicted class for each leaf node t is simply the object class corresponding to $\text{class}(t)$. So we must only specify how the geometry vector $\text{geom}(t) = \langle x, y, s\rangle$ for the node maps to a predicted bounding box in image coordinates. This mapping is motivated by the bounding box interpretation for the geometry vectors of leaf nodes in Section 2.2.2. The location values $\langle x, y\rangle$ specify the centroid of the bounding box, while the scale factor s determines the dimensions of the box as:

$$\frac{\langle W_{c,\text{width}}, W_{c,\text{height}}\rangle}{s}$$

where $\langle W_{c,\text{width}}, W_{c,\text{height}}\rangle$ is the class-specific canonical size for $c = \text{class}(t)$.

The mapping above produces a set of bounding boxes $\{b_j\}$ with class labels $\{c_j\}$, one for each leaf in the predicted tree T. However, some of these bounding boxes may overlap heavily with

[7] We use the same canonical sizes chosen by the Felzenzswalb detector. For each object class, a box is selected with median aspect ratio such that most of the training objects have larger area.

one another. Two or more bounding boxes may even be exactly equal, which can happen when the object detector score at a location and scale is very strong, thus tempting several rule parts to choose that pixel as their centroid. Even if a ground-truth object actually exists at that location and scale, only one of the predicted bounding boxes would be considered correct, while the others would be labeled false positives.

Therefore, we perform a very conservative form of non-maximal suppression (NMS) on the leaves of a predicted scene tree. We must adapt traditional NMS techniques, since we do not have a single independent score for each predicted object. Instead, we must consider how removing each overlapping object would affect the score of the entire scene tree.

The overlap between two bounding boxes b_1 and b_2 is defined as the area of their intersection divided by the area of their union:

$$\text{overlap}(b_1, b_2) = \frac{\text{area}(b_1 \cap b_2)}{\text{area}(b_1 \cup b_2)}$$

We choose a maximum threshold ρ on the allowable overlap between detected bounding boxes of the same class. Then, we take the following best-first approach to removing detections that overlap too much:

1. Find the set of leaf nodes $\{t_j\}$ in T such that each leaf's bounding box overlaps with at least one other leaf with the same class label by more than ρ.

2. For each overlapping leaf t_j, let T_j be the tree formed by removing t_j from T (and all ancestor nodes of t_j that no longer have any object descendents).

3. Update T to be the tree T_j with the highest score:

$$T = \underset{T_j}{\text{argmax}}\, \Phi(I, T_j) \cdot \Theta$$

We iterate this process until no leaf nodes overlap by more than ρ in Step 1.

For the experiments in this thesis, we apply NMS to detected scene trees at test time, with a conservative overlap threshold of $\rho = 0.95$. However, we do not perform any NMS during parameter learning; we will discuss the reasons for this in the next chapter.

Chapter 3

Parameter Learning in WGGs

In the last chapter, we introduced weighted geometric grammars (WGGs), and showed how to perform parsing efficiently with a given model. In this chapter, we discuss the problem of learning a good set of parameters for a WGG with fixed structure. Formally, we can state the parameter learning problem in WGGs as follows. We are given:

- A WGG structure $\langle B, C, R, S \rangle$ and feature representation Φ.
- A set of training images and scene trees $\langle I_i, T_i \rangle$ for $i = 1, \ldots, m$.

The goal is to find a good weight vector Θ.

At the beginning of the last chapter, we said that we would not assume fully labeled scene trees T_i for the training images. However, in the context of parameter learning, we do assume that we have labeled trees, with class and rule part ID tags corresponding to the WGG structure $\langle B, C, R, S \rangle$, and with geometry information in all nodes. We can make this assumption because the structure learning algorithms in the next two chapters output tree structure and geometry for the training images, in addition to a learned grammar structure. Furthermore, the experiments in this chapter use a hand-built grammar structure, for which we hand-annotated corresponding scene trees.

3.1 The Structured Perceptron Algorithm

To learn the weights for a WGG model, we adapt the structured perceptron algorithm, introduced by Collins (2002, 2004) for a variety of natural language processing tasks. The simplicity and efficiency of this algorithm are immediately appealing, but it also has some nice theoretical properties. The algorithm extends the classical perceptron algorithm for classification (Rosenblatt, 1958, 1962), and the voted or averaged versions of the perceptron (Freund & Schapire, 1999), to cases in which the output is much richer than a simple binary class.

The original structured perceptron algorithm, written in our notation, is given in Algorithm 3.1. The algorithm proceeds as follows. We choose an initial setting of the weights Θ_0 (e.g., all zero). Then, we make J passes over the training images. For the ith image on the jth pass, we perform parsing on the image I_i using the current weights Θ_ℓ. If the predicted scene tree T^* is incorrect, we update the current weight vector by the difference between the feature vector $\Phi(I_i, T_i)$ for the correct tree, and the feature vector $\Phi(I_i, T^*)$ for the predicted tree. If the predicted tree is correct, however, the weights are unchanged. Finally, after all passes have completed, we set the final weight vector Θ to the average of the weight vectors after each image on each pass.

This last step may seem a bit surprising, but it approximates the voted perceptron of Freund and Schapire (1999). In the voted perceptron, a single pass is made over the training examples (so

Input:

- Training examples $\langle I_i, T_i \rangle$ for $i = 1, \ldots, m$
- WGG components $\langle B, C, R, S, \Phi \rangle$
- Initial weight vector $\Theta_0 \in \mathbb{R}^n$
- Number of iterations J

Output: Final weight vector Θ

1.	$\ell = 0$	**weight vector index**
2.	for $j = 1, \ldots, J$	
3.	for $i = 1, \ldots, m$	
4.	$T^* = \operatorname{argmax}_{T \in \mathbf{GEN}(I_i)} \Phi(I_i, T) \cdot \Theta_\ell$	**find best tree for I_i**
5.	if $T^* \neq T_i$	**check tree correctness**
6.	$\Theta_{\ell+1} = \Theta_\ell + \Phi(I_i, T_i) - \Phi(I_i, T^*)$	**update weights**
7.	else	
8.	$\Theta_{\ell+1} = \Theta_\ell$	**do not change weights**
9.	$\ell = \ell + 1$	
10.	$\Theta = \frac{1}{mJ} \sum_{\ell=1}^{mJ} \Theta_\ell$	**approximates voted perceptron**

Algorithm 3.1: The structured perceptron algorithm (Collins, 2002).

$J = 1$). Let Θ_i be the weight vector after the ith training example. The voted perceptron outputs all m of these weight vectors, rather than a single vector. Then, for a new test image I_{test}, parsing is performed m separate times, once with each weight vector Θ_i:

$$T_{\text{test},i} = \operatorname*{argmax}_{T \in \mathbf{GEN}(I_{\text{test}})} \Phi(I_{\text{test}}, T) \cdot \Theta_i$$

These m trees are "votes" for the output, and the most frequently occurring tree is chosen.

Thus, the averaging performed on Line 13 in Algorithm 3.1 can be seen as an approximation to the voted perceptron. But averaging has the crucial advantage that we perform parsing on each test image only once using the averaged weights, rather than parsing each test image m times with m different weight vectors (Collins, 2002). Collins also shows that using the averaged version of the algorithm can improve performance significantly compared to the unaveraged version.

Although this algorithm is beautifully simple, it can be quite powerful. In particular, the iterative parsing steps serve to automatically find hard negative examples with which to update the model weights. If a training image has an especially attractive, but incorrect, interpretation under the current settings of the weights, the predicted scene tree will reflect that, and the weights will be updated accordingly. This discriminative feature of the algorithm makes it a good fit for the task of object detection, for which the mining of hard negative examples has been shown to be crucial for good learning (e.g., Torralba et al., 2007; Felzenszwalb et al., 2008, 2010).

We will see that the structured perceptron is a good fit for the task of parameter learning in WGGs, but we could have used other learning algorithms instead. For example, we could apply the latent SVM formalism developed by Felzenszwalb et al. (2008, 2010), which adapts multiple-instance SVMs to the problem of finding good parameters in a structured model. We could also apply general optimization techniques, such as conjugate gradient descent, although gradient-based approaches would need to be approximate, since scoring a training image using a candidate weight vector involves taking a max over trees.

3.1.1 Theoretical Justification

Collins (2002) presents several theorems on the convergence and generalization of the structured perceptron algorithm, which we paraphrase briefly here.

The first result is on the convergence of the algorithm when the training set is separable. A training set $\langle I_i, T_i \rangle$ is **separable with margin** δ if there exists some vector $\mathbf{U}$ with $||\mathbf{U}|| = 1$ such that

$$\forall i, \forall T \in \overline{\mathbf{GEN}}(I_i), \quad \mathbf{U} \cdot \Phi(I_i, T_i) - \mathbf{U} \cdot \Phi(I_i, T) \geq \delta$$

where $\overline{\mathbf{GEN}}(I_i) = \mathbf{GEN}(I_i) - \{T_i\}$ is the set of *incorrect* candidate trees for image I_i, and $||\mathbf{U}||$ is the L^2 norm of $\mathbf{U}$. Collins proves that for a training set that is separable with margin δ, the training error of the algorithm is bounded by:

$$\text{number of mistakes} \leq \frac{R^2}{\delta^2}$$

where R is a constant such that

$$\forall i, \forall T \in \overline{\mathbf{GEN}}(I_i), \quad ||\Phi(I_i, T_i) - \Phi(I_i, T)|| \leq R$$

This implies that the algorithm is guaranteed to converge to a weight vector with zero training error in a finite number of iterations, if such a vector exists.

Two observations are worth making. First, as Collins points out, the number of mistakes is not dependent on the size of $\mathbf{GEN}(I_i)$, only on the margin of separability δ. This is important since in practice the size of $\mathbf{GEN}(I_i)$ might be huge.

Second, notice that R depends on the max Euclidean distance between feature vectors of correct and incorrect trees for an image. This is why it is so important to cap the features we defined in Section 2.3 in a bounded range (e.g., $[-1, 1]$). Theoretically, even with feature vectors with unbounded values, the algorithm could still converge, if the training data is separable. But in practice, convergence will be much faster if the features cannot range too far. Intuitively, if the feature vectors can take on unbounded values, then a particularly bad prediction in Line 4 of Algorithm 3.1 could result in an update vector $\Phi(I_i, T_i) - \Phi(I_i, T^*)$ with arbitrarily large magnitude. This update would push the weight vector to a far extreme of the feature space, such that many steps might be required to get the vector back into a good part of the space. Using feature vectors with bounded magnitude means that no single image will be able to change the weight vector by a more than a fixed amount, avoiding the scenario we just described.[1]

The next result presented by Collins (2002) concerns the training error when the training set is *not* separable, which is virtually always the case in the context of this thesis. We will not state the theorem here, but summarize its implications. If the training set is "close" to being separable with margin δ (for a formal definition of "close"), then theorem implies that the algorithm will eventually make a small number of mistakes. This result demonstrates the robustness of the algorithm to training sets in which some examples will never be parsed correctly.

Finally, Collins describes generalization results for the voted perceptron algorithm that were originally presented by Freund and Schapire (1999). The results suggest that if the voted perceptron makes relatively few mistakes on the training set, it is likely to generalize well to test examples.

[1]This is likely to be even more important in cases when the training set is not actually separable.

Input:
- Training examples $\langle I_i, T_i \rangle$ for $i = 1, \ldots, m$
- WGG components $\langle B, C, R, S, \Phi \rangle$
- Initial weight vector $\Theta_0 \in \mathbb{R}^n$
- Number of iterations J
- Learning rate vector $\Psi \in \mathbb{R}^n$

Output: Final weight vector Θ

1.	$\ell = 0$	**weight vector index**
2.	for $j = 1, \ldots, J$	
3.	for $i = 1, \ldots, m$	
4.	$T^* = \text{argmax}_{T \in \mathbf{GEN}(I_i)} \Phi(I_i, T) \cdot \Theta_\ell$	**find best tree for I_i**
5.	if $\neg\text{correct}(\text{detections}(T^*), \text{labels}(I_i))$	**check detection correctness**
6.	$\Theta_{\ell+1} = \Theta_\ell + \Psi \circ (\Phi(I_i, T_i) - \Phi(I_i, T^*))$	**update weights (Equation 3.1)**
7.	else	
8.	$\Theta_{\ell+1} = \Theta_\ell$	**do not change weights**
9.	$\ell = \ell + 1$	
10.	$\Theta = \frac{1}{mJ} \sum_{\ell=1}^{mJ} \Theta_\ell$	**approximates voted perceptron**

Algorithm 3.2: The structured perceptron algorithm, applied to parameter learning in WGGs. It is identical to Algorithm 3.1, except for modifications in Lines 5 and 6.

3.1.2 The Structured Perceptron for WGGs

We can apply Algorithm 3.1 to learn the parameters of a WGG virtually as written, with only a small number of tweaks, which we describe in this section. Algorithm 3.2 shows the modified version.

Evaluating Predicted Trees

Consider the test for tree correctness in Line 5 of Algorithm 3.1. Our scene trees have continuous geometry values in their nodes, so performing a naive test for perfect equality between the predicted tree T^* and the labeled tree T_i seems inappropriate. Furthermore, we only want to penalize trees which would not produce correct object detections at test time; otherwise, there should be no need to update the weight vector.

In Section 3.2.2, we will formally define the precision/recall/f-measure metrics we use for the experiments in this thesis. Briefly, the cumulative f-measure of a set of predicted bounding boxes and a set of ground truth bounding boxes, each with associated object classes, is a number between 0 and 1. An f-measure of 1 means that all the predicted boxes were correct, and that all the ground truth boxes were predicted.

Thus, we take the following approach to evaluating whether a predicted tree T^* for a training image I_i is correct in Line 5 of Algorithm 3.2. If the set of bounding boxes produced by the leaves of the predicted tree T^* (as described in Section 2.4.3) has a cumulative f-measure of 1 when compared with the set of human-annotated bounding boxes for image I_i, then the tree is considered correct. Otherwise, it is not, and the weight vector is updated in Line 6.

However, it is important that no non-maximal suppression (NMS) be applied to the leaves of the predicted tree T^*, despite the fact that we do apply conservative NMS at test time. This is because NMS will produce a tree $T^{*\prime}$ which scores lower according to the model than the original detected tree T^* (even though the detections resulting from NMS may score higher against ground truth labels). So using $T^{*\prime}$ instead of T^* to update the weights in Line 6 would be misleading to the perceptron, because it gives inaccurate feedback about how the current setting of the weights affect the features of the highest-scoring detected tree.

A Learning Rate for Features on Tree Structure

We discussed in Section 2.4.2 how the weight θ_k^{rp} on each rule part tree structure feature can be interpreted as (the negative of) a threshold on the score of the subtree rooted at rule part k. Similarly, the pairwise weights $\theta_{k_1,k_2}^{\mathrm{pw}}$ have a thresholding role in determining how high the sum of scores for sets of subtrees need to be in order to be included in the predicted tree.

However, these tree structure features themselves are binary—they are 0 or 1. Thus, performing the simple additive update in Line 6 of Algorithm 3.1 would result in tree structure weights that take only integer values (at least until the averaging step in Line 10). This seems too coarse, since these weights are acting as thresholds on subtree scores which are functions of object detector and geometry features that are capped to lie in $[-1,1]$.

In the limit of many iterations, the perceptron could always increase the object detector and geometry weights so that the integer-valued tree structure weights can effectively threshold the subtree scores. But in order to speed up learning, we introduce a learning rate ψ on *only* the tree structure features (both single rule part and pairwise). Let $\Psi \in \mathbb{R}^n$ be a vector with the same length as the feature representation Φ. All elements of Ψ are 1 except for those corresponding to tree structure features in Φ; those elements have value ψ. Then, in Line 6 of Algorithm 3.2, we take the entrywise product of the delta vector $\Phi(I_i,T_i) - \Phi(I_i,T^*)$ with the learning rate vector Ψ, before updating the weight vector:

$$\Theta_{\ell+1} = \Theta_\ell + \Psi \circ (\Phi(I_i,T_i) - \Phi(I_i,T^*)) \tag{3.1}$$

where $\circ$ denotes the entrywise product between vectors.

For all the experiments in this thesis, we use $\psi = 0.01$, chosen based on preliminary results on a development set. Again, this modification is only to achieve faster learning. In practice, we found that after enough iterations, the perceptron could set the weights effectively even without a learning rate (equivalently, with $\psi = 1$), but that lower values of ψ produced faster convergence.

Initialization

For most of the experiments in this thesis, we initialize the weight vector Θ_0 in Algorithm 3.2 to zero. However, another reasonable initialization scheme is as follows:

- Initialize all object detector output weights $\theta_c^{\mathrm{f},1}$ and $\theta_k^{\mathrm{f},2}$ to 1. This expresses our expectation that the object detector output correlates positive with object presence.

- Initialize all absolute and relative geometry weight vectors θ_k^{g} and θ_c^{h} to $\langle -1,-1,-1,0,0,0\rangle$. This initializes the geometry models to simple parabolas with maximum score achieved when the input variables $\langle \bar{x},\bar{y},\bar{s}\rangle$ and $\langle \hat{x},\hat{y},\hat{s}\rangle$ equal the initial relative offsets $\langle \alpha_{k,\mathrm{x}},\alpha_{k,\mathrm{y}},\alpha_{k,\mathrm{s}}\rangle$ and the center and native scale of the image, respectively.

- Initialize all other weights (on tree structure) to 0.

We will use this initialization scheme in the next section.

Geometry in Ground Truth Trees

As we have said, we assume that we have fully specified structure for the "ground truth" trees T_i. In this section, we discuss how we set the geometry values in the ground truth trees.

The leaves of the ground truth trees correspond to human-annotated objects in the training images. The obvious approach would be to use each labeled bounding box b_{gt} to set the geometry vector in the corresponding tree leaf directly. Instead, we use the object detector output on the training image to choose a location and scale $\langle x, y, s \rangle$ for the leaf node that corresponds to a locally-maximal object detector score, while still producing a bounding box that overlaps sufficiently with b_{gt} to be considered correct. This improves the learning, because it provides training examples with correct leaf geometry that are more likely to be separable in object detector feature space.

To set the geometry of the internal nodes of the ground truth trees, we have two options. As we mentioned at the beginning of this chapter, the structure learning algorithms and hand-built dataset provide scene trees with both structure and geometry as input to parameter learning. So the first option is simply to use the given geometry directly for the internal nodes. We will refer to this method as "fixed ground truth internal geometry."

The second option is to perform an additional parsing at each step of the perceptron, in which we find a tree that has the same structure and leaf geometry as the ground truth tree, while freely setting the geometry on the internal and root nodes according to the current setting of the weights:

$$\hat{T}_i = \operatorname*{argmax}_{\substack{T \in \mathbf{GEN}(I_i) \text{ s.t.} \\ \text{structure}(T) = \text{structure}(T_i) \,\wedge \\ \text{geom}(\text{leaves}(T)) = \text{geom}(\text{leaves}(T_i))}} \Phi(I_i, T) \cdot \Theta_\ell$$

We would insert this computation before Line 4 in Algorithm 3.2, and use $\hat{T}_i$ in place of T_i in Lines 5 and 6. We will refer to this method as "free ground truth internal geometry."

When using free ground truth geometry, it turns out to be important to use the alternative initialization scheme we described last section. This is because we are more sensitive to the initial weight values when using them to choose the ground truth geometry as well as the predicted geometry. Again, the algorithm can still converge given enough time under either initialization scheme, but it will probably take longer. Therefore, whenever we use free internal geometry, we will initialize the weights as described in the previous section; otherwise, we will initialize the weights to all zeros.

We will investigate both methods for setting the internal geometry of ground truth trees in the experiments in the next section and in Chapter 4.

Setting the Rule Part Offsets

Finally, we cannot apply the perceptron to a WGG structure $\langle B, C, R, S, \Phi \rangle$ before setting the offsets $\langle \alpha_{k,\text{x}}, \alpha_{k,\text{y}}, \alpha_{k,\text{s}} \rangle$ for each rule part k—these offsets are used to compute the relative geometry features (Section 2.3.2). Since we assume scene trees T_i with labeled rule part IDs and geometry, we simply compute the offsets for each rule part k as the sample means over all nodes with label k:

$$\alpha_{k,\text{x}} = \frac{1}{N_k} \sum_i \sum_{\substack{t \in T_i \text{ s.t.} \\ \text{rpid}(t) = k}} \text{geom}_\text{x}(t) \qquad \alpha_{k,\text{y}} = \frac{1}{N_k} \sum_i \sum_{\substack{t \in T_i \text{ s.t.} \\ \text{rpid}(t) = k}} \text{geom}_\text{y}(t)$$

$$\alpha_{k,\text{s}} = \frac{1}{N_k} \sum_i \sum_{\substack{t \in T_i \text{ s.t.} \\ \text{rpid}(t) = k}} \log(\text{geom}_\text{s}(t)) \qquad N_k = \sum_i \sum_{\substack{t \in T_i \text{ s.t.} \\ \text{rpid}(t) = k}} 1$$

Again, we put the scale values $\text{geom}_\text{s}(t)$ into log space, since they are multiplicative factors.

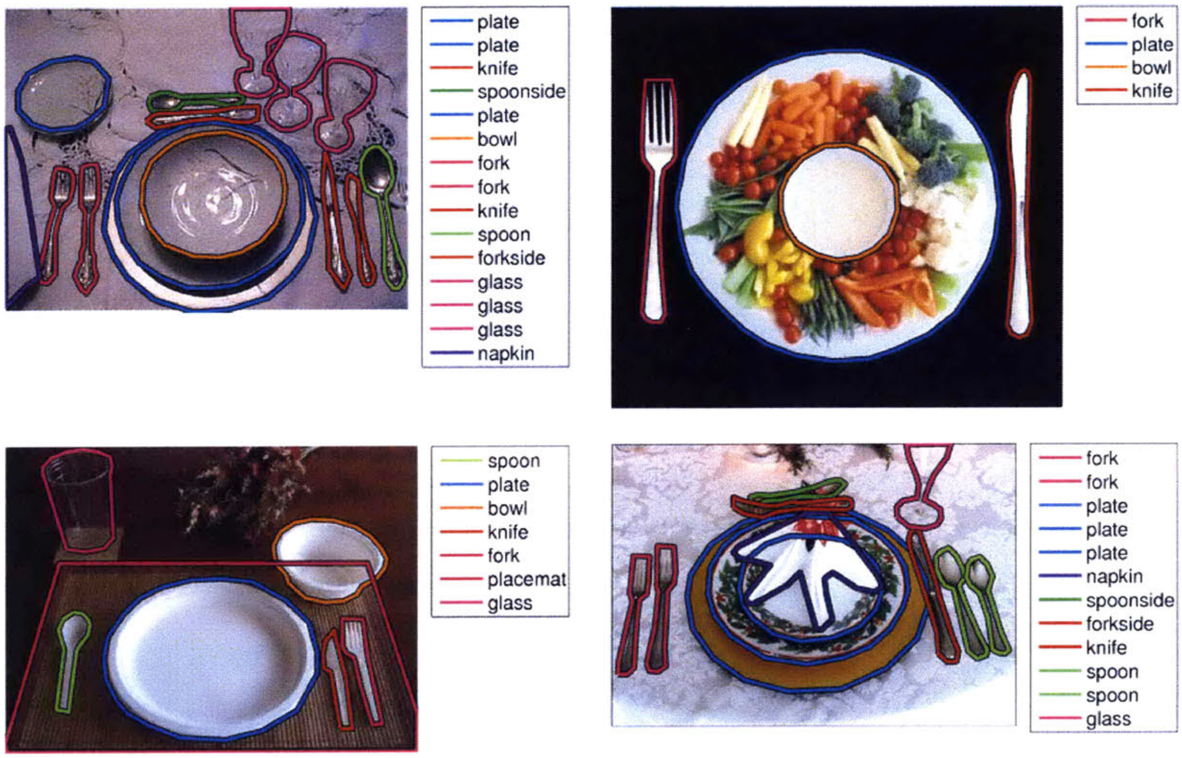


Figure 3-1: Examples of images and their labels from the small placesetting dataset.

3.2 Experiments with a Hand-built Structure

In this section, we introduce the first of the three datasets we use in this thesis, and show a WGG structure we hand-built for the dataset. We explain the evaluation metrics we use for the experiments, and present the baseline object detector against which we compare. Finally, we present experiments evaluating the hand-built WGG structure on the first dataset, and exploring the variations on WGG models and perceptron learning that we discussed in the first of half of the chapter.

3.2.1 The Small Placesetting Dataset

This thesis contributes three new datasets for the task of object detection in context. The first, the "small placesetting dataset," consists of 167 images of placesettings. The images have been labeled with silhouettes for 1499 objects from 15 object classes.[2] Figure 3-1 shows example images and labels from the dataset. Additional images are shown in Appendix A (Figure A-1). Table 3.1 lists the object classes and their frequencies, while Figure 3-2 shows examples of each class.

The dataset is quite challenging. The number of objects per image is relatively high (an average of 9), and also has high variance (a standard deviation of 5.7). Furthermore, the objects appear at a wide range of scales and rotations, with occlusion and poor image quality not uncommon.

The small placesetting dataset was labeled using the LabelMe web-based image annotation tool (Russell, Torralba, Murphy, & Freeman, 2008), and is freely available through the same sys-

[2]Our system does not actually use the labeled silhouettes, only the bounding boxes.

object class	# instances	% objects
bowl	53	3.5
candle	32	2.1
cup	44	2.9
fork	251	16.7
forkside	37	2.5
glass	201	13.4
knife	187	12.5
knifeside	25	1.7
napkin	93	6.2
placemat	24	1.6
plate	316	21.1
saucer	27	1.8
shaker	33	2.2
spoon	120	8.0
spoonside	56	3.7

Table 3.1: The small placesetting dataset, with 167 images and 1499 labeled objects.

(a) bowl (b) candle (c) cup

(d) fork (e) forkside (f) glass (g) knife (h) knifeside

(i) napkin (j) placemat

(k) plate (l) saucer

(m) shaker (n) spoon (o) spoonside

Figure 3-2: Examples of each object class in the small placesetting dataset. Notice the difference between the fork and forkside, knife and knifeside, and spoon and spoonside classes.

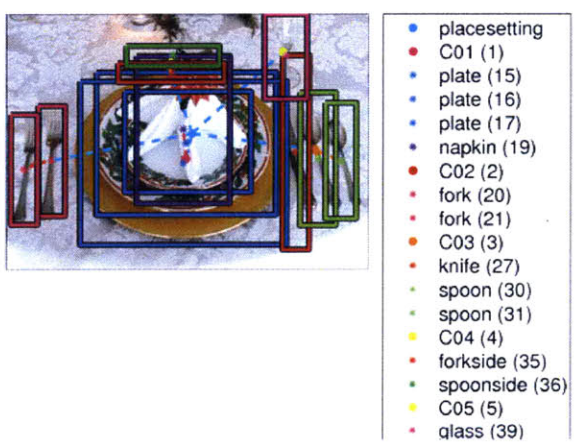
$B = \{\text{bowl}, \text{candle}, \text{cup}, \text{fork}, \text{forkside}, \text{glass}, \text{knife}, \text{knifeside}, \text{napkin}, \text{placemat}, \text{plate}, \text{saucer}, \text{shaker}, \text{spoon}, \text{spoonside}\}$

$C = \{\text{placesetting}, \text{C01}, \text{C02}, \text{C03}, \text{C04}, \text{C05}, \text{C06}, \text{C07}, \text{C08}, \text{C09}, \text{C10}, \text{C11}, \text{C12}, \text{C13}\}$

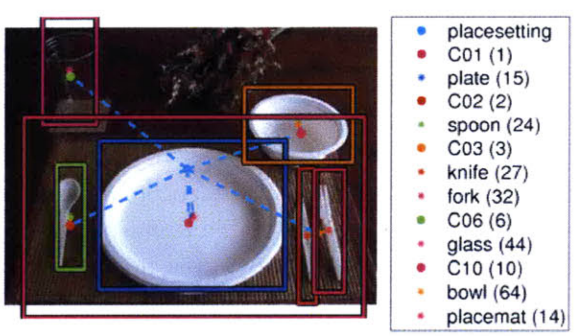
$$R = \left\{ \begin{array}{ll}
\text{placesetting} \to \text{C01}_1\ \text{C02}_2\ \text{C03}_3\ \text{C04}_4\ \text{C05}_5\ \text{C06}_6\ \text{C07}_7\ \text{C08}_8\ \text{C09}_9\ \text{C10}_{10}\ \text{C11}_{11}\ \text{C12}_{12}\ \text{C13}_{13}\ \text{placemat}_{14} & \\
\quad \text{C01} \to \text{plate}_{15}\ \text{plate}_{16}\ \text{plate}_{17}\ \text{bowl}_{18}\ \text{napkin}_{19} & \textbf{dinner, salad, charger} \\
\quad \text{C02} \to \text{fork}_{20}\ \text{fork}_{21}\ \text{fork}_{22}\ \text{knife}_{23}\ \text{spoon}_{24}\ \text{spoon}_{25}\ \text{napkin}_{26} & \textbf{left of C01, right to left} \\
\quad \text{C03} \to \text{knife}_{27}\ \text{knife}_{28}\ \text{knife}_{29}\ \text{spoon}_{30}\ \text{spoon}_{31}\ \text{fork}_{32}\ \text{fork}_{33}\ \text{napkin}_{34} & \textbf{right of C01, left to right} \\
\quad \text{C04} \to \text{forkside}_{35}\ \text{spoonside}_{36}\ \text{knifeside}_{37}\ \text{napkin}_{38} & \textbf{above C01} \\
\quad \text{C05} \to \text{glass}_{39}\ \text{glass}_{40}\ \text{glass}_{41}\ \text{glass}_{42}\ \text{napkin}_{43} & \textbf{glasses, upper right} \\
\quad \text{C06} \to \text{glass}_{44}\ \text{glass}_{45}\ \text{glass}_{46}\ \text{glass}_{47}\ \text{napkin}_{48} & \textbf{glasses, upper left} \\
\quad \text{C07} \to \text{plate}_{49}\ \text{plate}_{50}\ \text{knifeside}_{51}\ \text{knife}_{52}\ \text{napkin}_{53} & \textbf{bread plate, upper left} \\
\quad \text{C08} \to \text{cup}_{54}\ \text{saucer}_{55}\ \text{plate}_{56}\ \text{spoonside}_{57}\ \text{spoon}_{58} & \textbf{cup/saucer, upper right} \\
\quad \text{C09} \to \text{cup}_{59}\ \text{saucer}_{60}\ \text{plate}_{61}\ \text{spoonside}_{62}\ \text{spoon}_{63} & \textbf{cup/saucer, upper left} \\
\quad \text{C10} \to \text{bowl}_{64}\ \text{bowl}_{65}\ \text{bowl}_{66}\ \text{plate}_{67} & \textbf{bowls/plate, upper right} \\
\quad \text{C11} \to \text{bowl}_{68}\ \text{bowl}_{69}\ \text{bowl}_{70}\ \text{plate}_{71} & \textbf{bowls/plate, upper left} \\
\quad \text{C12} \to \text{shaker}_{72}\ \text{shaker}_{73} & \textbf{above, left to right} \\
\quad \text{C13} \to \text{candle}_{74}\ \text{candle}_{75}\ \text{candle}_{76} & \textbf{above, left to right}
\end{array} \right\}$$

$S = \{\text{placesetting}\}$

Figure 3-3: A hand-built WGG structure for the small placesetting dataset. Comments show guidelines for how each composite class was assigned in annotating scene trees for the training images.




```
+-+ placesetting
  +-+ C01 (1)
  | +-+ plate (15)
  | +-+ plate (16)
  | +-+ plate (17)
  | +-+ napkin (19)
  +-+ C02 (2)
  | +-+ fork (20)
  | +-+ fork (21)
  +-+ C03 (3)
  | +-+ knife (27)
  | +-+ spoon (30)
  | +-+ spoon (31)
  +-+ C04 (4)
  | +-+ forkside (35)
  | +-+ spoonside (36)
  +-+ C05 (5)
    +-+ glass (39)
```




```
+-+ placesetting
  +-+ C01 (1)
  | +-+ plate (15)
  +-+ C02 (2)
  | +-+ spoon (24)
  +-+ C03 (3)
  | +-+ knife (27)
  | +-+ fork (32)
  +-+ C06 (6)
  | +-+ glass (44)
  +-+ C10 (10)
  | +-+ bowl (64)
  +-+ placemat (14)
```

Figure 3-4: Examples of images and their labeled scene trees from the small placesetting dataset. Notice that the bounding box on each leaf node is not exactly the bounding box of the labeled silhouette; rather, it is centered on a locally-maximal object detector score.

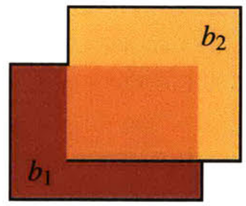


Figure 3-5: The overlap score between b_1 and b_2 is the area of their intersection (orange) divided by the area of their union (red plus orange plus yellow).

tem.[3] The dataset is a subset of the `static_silverware` and `static_web_submitted_meg_placesettings` LabelMe directories; Appendix A lists the 167 filenames from these directories.

This dataset is smaller than the others we consider. However, in addition to labeling object silhouettes, we also hand-built a WGG structure for the dataset, and annotated corresponding scene tree structures for the images in the dataset. Figure 3-3 shows the hand-built structure, while Figure 3-4 show examples of labeled scene trees. This level of annotation would be very difficult for a larger dataset. Therefore, the small placesetting dataset provides a useful testbed for exploring the WGG framework and parameter learning, while controlling for the problem of structure learning.

3.2.2 Evaluation Metrics

Both the WGG parsing algorithm and the object detector against which we compare output bounding boxes with class labels. How do we evaluate these detections to determine their correctness with respect to the labeled annotations?

We use the overlap metric to determine whether a predicted bounding box is correct. This metric has become the standard in object recognition, used among other contexts in the PASCAL visual object classes (VOC) challenges (Everingham, Van Gool, Williams, Winn, & Zisserman, 2008). As we discussed in Section 2.4.3, the overlap between two bounding boxes b_1 and b_2 is defined as the area of their intersection divided by the area of their union:

$$\text{overlap}(b_1, b_2) = \frac{\text{area}(b_1 \cap b_2)}{\text{area}(b_1 \cup b_2)}$$

We consider a predicted bounding box b_{pred} with object class c to be "correct" if it overlaps by more than 0.5 with a ground truth bounding box b_{gt} with class c. Overlap is an intuitive metric for object detection, because it provides a measure in $[0,1]$—it is one when the regions are identical, and zero when they do not overlap at all.

We then evaluate a set of predicted bounding boxes for a class c using precision-recall. Let $\{b_i\}$ be the set of detected bounding boxes with class c for an image, and $\{b_j\}$ be the set of ground truth "target" boxes with class c. First, we compute the overlap score $o_{i,j}$ between each detected bounding box b_i and each target box b_j.

Then, we sort the detected bounding boxes by decreasing confidence. For the object detector, the confidence of a detection is simply its score. For a box produced from a scene tree, we use the object detector feature times its weight, plus the dot product of the local relative geometry features and their weights—this is a local estimate of the contribution of the object to the overall tree score.

We walk through the detected boxes in order. For the current box b_i, find the target b_j with highest overlap score $o_{i,j}$. If $o_{i,j} > 0.5$, we mark b_i as correctly detecting the target object b_j. We

[3] `http://labelme.csail.mit.edu/browseLabelMe/static_silverware.html`
`http://labelme.csail.mit.edu/browseLabelMe/static_web_submitted_meg_placesettings.html`

also set the overlap score between b_j and all other detected objects to $-\infty$, so that once a target has been assigned to a detection, it cannot be used again. However, if no target b_j overlaps with b_i by more than 0.5, we mark b_i as a false positive, and move on.

In the end, each detected object has either been marked correct (assigned a target object), or marked as a false positive. Then we can sum the number of correct detections for class c across all test images, and compute the *recall* and *precision* of the detections as:

$$\text{recall} = \frac{\text{\# correct detections}}{\text{\# target objects}} \qquad \text{precision} = \frac{\text{\# correct detections}}{\text{\# detected objects}}$$

We can also compute the cumulative scores across all object classes, by summing the number of correct detections, target objects, and detected objects across all images and all classes, rather than within each class separately.

Recall captures the percentage of target objects were correctly detected, while precision captures the percentage of detected objects that were correct. Detection systems often display a trade-off, explicit or not, between higher recall and higher precision. In the simplest case, in which a detector outputs a score for each candidate detection, thresholding the score at a low value will result in a large number of detections. The target objects are more likely to be correctly detected, making the recall higher, but many of the detections are also likely to be wrong, making the precision lower. On the other hand, thresholding at a high value reverses this trend, often producing relatively low recall but high precision.

To allow easier comparison among different detection systems, particularly because of this trade-off, it is useful to have a single score summarizing both recall and precision. The *f-measure* of a set of detections is defined as the harmonic mean of the recall and and precision:

$$\text{f-measure} = \frac{2 \times \text{recall} \times \text{precision}}{\text{recall} + \text{precision}}$$

The harmonic mean is the appropriate method for averaging rates, and is lower than the arithmetic mean. In practice, it tends to magnify the effect of the lower of the two contributing values; this means that it is difficult for a detection system to achieve high f-measure without having *both* relatively high recall and relatively high precision. Thus, f-measure is a challenging and effective measure of comparison across multiple systems, even for systems which may fall on different sides of the implicit trade-off between high recall and high precision.

Using Precision-Recall Rather than ROC

Precision-recall and f-measure are traditional evaluation metrics in the field of information retrieval. However, the task of object detection is inherently similar to that of information retrieval. In particular, in both tasks, the number of *possible* detections is enormous—many orders of magnitude greater than the number of targets. For object detection, even when an image contains many target objects, the number of objects is tiny compared with the number of pixels at each scale of the image pyramid!

For this reason, precision-recall is far more appropriate as a metric for object detection tasks than ROC, which many authors have used. The problem with ROC is that it can present an overly optimistic view of a system's performance in cases when there is great imbalance between the number of positive and negative examples; i.e., when the number of possible detections is much greater than the number of targets.

In a detection task, we can use a confusion matrix to summarize an algorithm's performance:

	actual positive	actual negative
predicted positive	TP	FP
predicted negative	FN	TN

TP is the number of true positives (target objects that were correctly detected) while FN is the number of false negatives (target objects that were not detected). FP is the number of false positives (detected objects that were *not* correct), while TN is the number of true negatives (candidate detections that were *correctly* not detected, because they do not map to a true target object).

Rewriting recall and precision in terms of the confusion matrix results in:

$$\text{recall} = \frac{TP}{TP+FN} \qquad\qquad \text{precision} = \frac{TP}{TP+FP}$$

ROC, however, evaluates detection systems by computing the true-positive rate and the false-positive rate:

$$\text{true-positive rate} = \frac{TP}{TP+FN} \qquad\qquad \text{false-positive rate} = \frac{FP}{FP+TN}$$

Notice that the true-positive rate of a system is identical to its recall. However, its false-positive rate is very different from its precision. While a very large number of incorrect detections will result in very low precision, it may not result in such a low false-positive rate. Although the numerator FP would be large, the denominator $FP+TN$ would also be large, because there are so many *candidate* detections in an image. Therefore, the false-positive rate can mask the fact that a system may make a huge number of incorrect detections. Precision, on the other hand, simply computes the percentage of detected objects that were correct.

Furthermore, it is not always clear how to even count the number of "actual negatives" in an object detection task. The number of different candidate detections a system might consider is heavily dependent on the details of a system (e.g., the number of scales in the image pyramid, the subsampling rate of the image, etc.). So even if a number could be computed, it would not be obvious how to compare fairly across different systems.

Precision-recall does not have these problems, because it never uses the number of "actual negatives." This sidesteps the issue of how to count this number, but more importantly, it produces metrics which correctly penalize a system for making many incorrect detections. Davis and Goadrich (2006) make many of these same arguments, and even prove that a ROC curve for one system dominates the ROC curve for another system if and only if its precision-recall curve dominates the precision-recall curve for the second system; i.e., that precision-recall is a strictly more informative metric on an algorithm's performance than ROC.

Producing Detection Sets Rather than Precision-Recall Curves

Parsing with a WGG model produces a single highest-scoring scene tree for a test image, which then results in a single set of detected objects. This process differs from most current object detection systems (Dalal & Triggs, 2005; Torralba et al., 2007; Felzenszwalb et al., 2010), which assign an independent score to each candidate detection in a test image, but do not directly address the question of how to threshold these scores to produce a final set of detections. Indeed, the vast majority of recent object recognition research has focused on how best to score candidate detections, rather than on how to threshold those scores.

Instead, most systems control for the thresholding question by producing precision-recall or ROC curves—sweeping over a large range of thresholds and plotting the scores associated with each one. Different systems are then compared by computing the area under their curves or by overlaying their curves on the same axes.

Object detection systems which maintain an agnostic stance towards threshold-setting do have some advantages. A user with a particular task in mind may have a greater need for high recall, at the cost of low precision, or vice versa; decision theorists would refer to this as the utility of various outcomes. By outputting what is essentially a set of score matrices rather than a set of detections, these systems theoretically allow the user to choose a threshold (i.e., a point on the precision-recall curve) that best meets their needs.

However, this line of reasoning has allowed the field to largely ignore the challenging question of how to best set these thresholds in a general, not-task-specific manner. A user who simply wants an effective object detection system, but does not want to be deeply involved in tweaking the internals, has few options.

In this context, we view the fact that WGG parsing produces a single set of detections as a possible advantage, rather than a disadvantage, of the system. We can view parameter learning as jointly setting both the weights on object detector and geometry features, and also the context-specific thresholds across all components of the model. Thus, it attempts to address both the scoring and threshold problems, rather than just the first. Nonetheless, one area of future work would be to consider ways to incorporate a measure of utility, or the relative importance of recall versus precision, directly into the learning process.

3.2.3 Baseline Model

The object detector features in the WGG model (Section 2.3.1) are based on the discriminatively-trained parts-based object detector (DTPBM) developed by Felzenszwalb, Girshick, McAllester, and Ramanan (2008, 2010). This detector is one of the best in the field, achieving state-of-the-art results on the PASCAL VOC benchmarks in 2007 and 2008 (Everingham et al., 2008). For both of these reasons, it is the most natural system against which to compare the WGG framework.

We used code available from `http://people.cs.uchicago.edu/~pff/latent/`, training a single-component model with one root and six deformable parts for each object class, as is done in Felzenszwalb et al.'s paper. For each object class, the code outputs a score at each location and scale in each image; these scores provide the basis of the WGG object detector features. It also learns a canonical height and width for the "root" part of each object class c, which we used as our canonical size $\langle W_{c,\text{width}}, W_{c,\text{height}} \rangle$ for that class.

The DTPBM code also learns a conservative threshold for each object class, chosen to produce high recall but very low precision. Then simple non-maximal suppression is used to produce a large number of candidate detections as follows. The first detection is created from the root box at the best-scoring location and scale. The next detection is created from the root box at the next best-scoring location and scale, but constrained to not overlap with any previous detection by more than 0.5. This continues until no more valid detections score higher than the conservative threshold. This very large set of detections is used by Felzenszwalb et al. to produce precision-recall curves.

There is also a variant in which, instead of simply using the root box associated with the best-scoring location, a bounding box prediction is made based on both the root box and the predicted locations of the parts in the deformable model. Then, the same non-maximal suppression scheme as described above is applied to produce a slightly different large set of detections. Felzenszwalb et al. (2010) found that this predicted-box version performed slightly better than the simple root-box version described above.

In our experiments, we produced detection sets using both versions. We found that on the small placesetting dataset, the root-box method outperformed the predicted-box version by a slight margin, while the reverse was true on our other two datasets. Although these differences are slight, we report baseline results based on whichever version performed better on the dataset in question.

Producing a Good Detection Set

Regardless of which bounding box prediction method is used, the DTPBM system produces a huge set of detections for a test set, intentionally achieving high recall and low precision. So we must select a good subset of these detections against which to fairly compare our system. For example, we could choose a less conservative threshold for each object class, perhaps to maximize f-measure on the training set or a development set.

Instead, however, we use a simple heuristic based on the number of objects of each class in an image. Let n_c be the maximum number of objects of class c ever seen in a training image. Then, for each object class c in each test image, we choose the n_c highest-scoring detections from the set produced by the DTPBM, discarding the rest. These detections already score above the conservative threshold and do not overlap with one another by much, so no additional processing is needed.

Comparing against DTPBM detections selected this way allows us to control somewhat for any advantage the WGG gains from simply capturing the maximum number of objects of a class ever seen in a training image (which is an upper bound on the number of objects a WGG will predict).

To further investigate the power of knowing the number of objects of each class in an image, we also compare against DTPBM detections chosen as follows. Let $n_{c,i}$ be the actual number of ground truth target objects with class c in the ith test image. We then choose the $n_{c,i}$ highest-scoring detections from the set produced by the DTPBM, discarding the rest. This is cheating—no actual system would have access to this information. However, the performance of detections chosen with this knowledge lends insight into both the quality of the object detector, and the value of good estimates for the expected number of target objects in an image.

3.2.4 Results on the Small Placesetting Dataset

We are now ready to present experimental results using the hand-built WGG structure on the small placesetting dataset. First, we compare the performance of the simplest version of the WGG model to the DTPBM object detector. Then, we explore different variations of the WGG model and the perceptron algorithm. In particular, we consider the effects of:

- associating the WGG object detector features with an entire object class versus each rule part;
- using fixed versus free internal geometry in the ground truth trees during the perceptron; and
- removing the pairwise tree structure weights, using the individual weights on each rule part.

For each set of experiments, we randomly split the dataset into a training set with 87 images and a test set with 80 images. We trained and tested each model on exactly the same images, computing per-class and cumulative precision, recall, and f-measure on the test set. Then, to control for variance in the training and test sets, we performed the same experiments for 5 different splits of the dataset. The results we present here are averaged over these 5 splits, with the minimum and maximum scores across runs measuring the variability in performance.

In the following sections, we will compare numerical results for the different models cumulatively (across all object classes), only showing per-class results graphically. However, the complete numerical results, cumulative and per-class, for all models are listed in Appendix B.

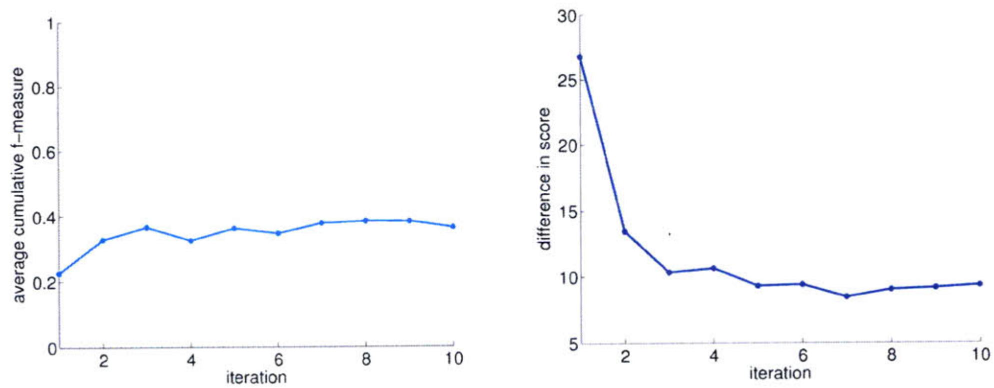
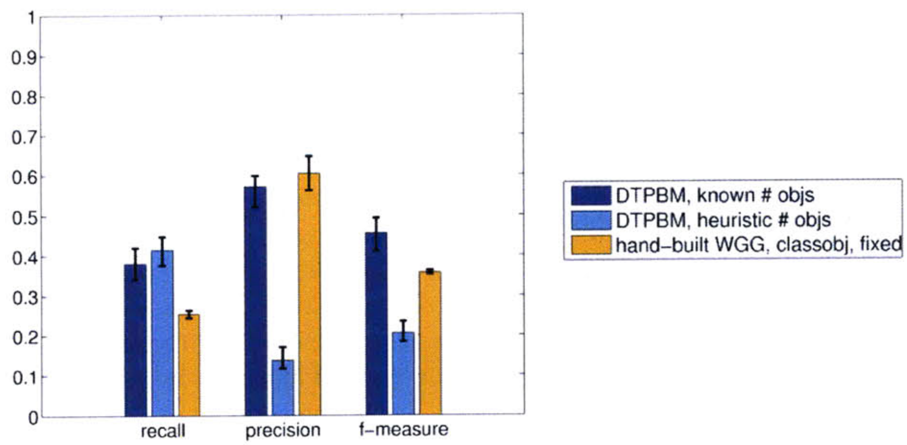


Figure 3-6: The performance of a WGG model over the 10 iterations of the perceptron. The left plot shows average cumulative f-measure across each iteration. The right plot shows the average difference in score between the detected tree and the ground truth tree.



 # targets (all models): 741.0

	recall	**precision**	**f-measure**	**# detections**	**# correct**
DTPBM, known # objs	0.380 [0.341 0.420]	0.571 [0.521 0.598]	0.456 [0.412 0.493]	493.0	280.8
DTPBM, heuristic # objs	0.415 [0.376 0.447]	0.138 [0.117 0.171]	0.205 [0.184 0.235]	2263.6	306.8
WGG, classobj, fixed	0.254 [0.244 0.264]	0.604 [0.561 0.647]	0.357 [0.353 0.362]	312.0	188.0

Figure 3-7: Cumulative results for the DTPBM object detector (dark and light blue) and the hand-built WGG structure (orange). Mean, minimum, and maximum scores across runs are shown. 'classobj' means class-based object detector features were used, while 'fixed' refers to fixed ground-truth internal geometry.

The WGG versus the DTPBM Object Detector

The most straightforward version of the WGG uses class-based object detector features and pairwise weights, with fixed ground-truth internal geometry during the perceptron. All WGG experiments in this chapter used 10 iterations of the perceptron, chosen based on preliminary experiments. Figure 3-6 illustrates one model's performance on the training data over the 10 iterations of the perceptron.

Figure 3-7 summarizes the cumulative average performance of the hand-built WGG model, and compares it to both versions of the DTPBM object detector. (We break the results out by object

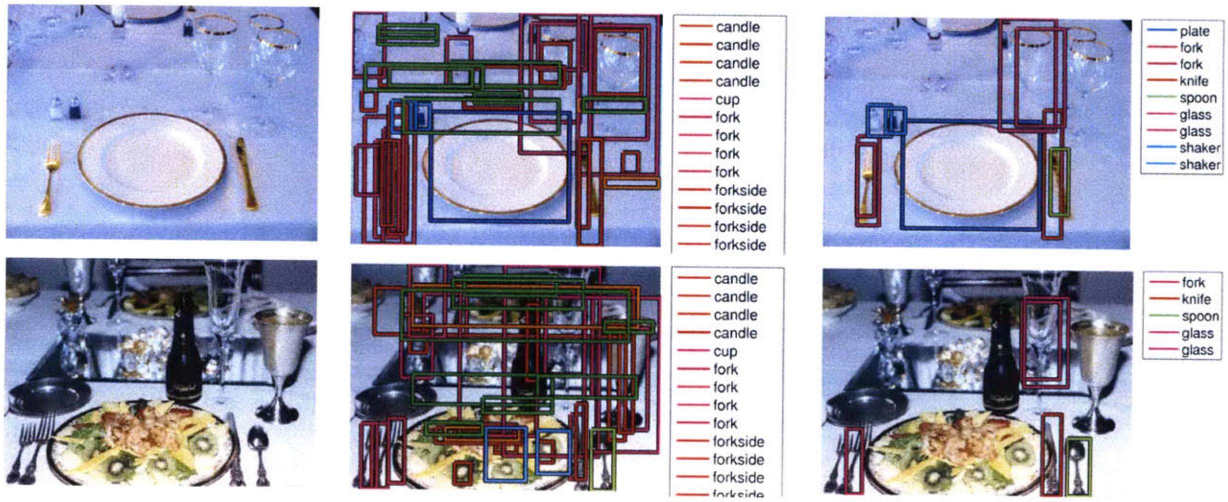


Figure 3-8: Representative detections from the heuristic DTPBM object detector (second column) and a WGG model (third column).

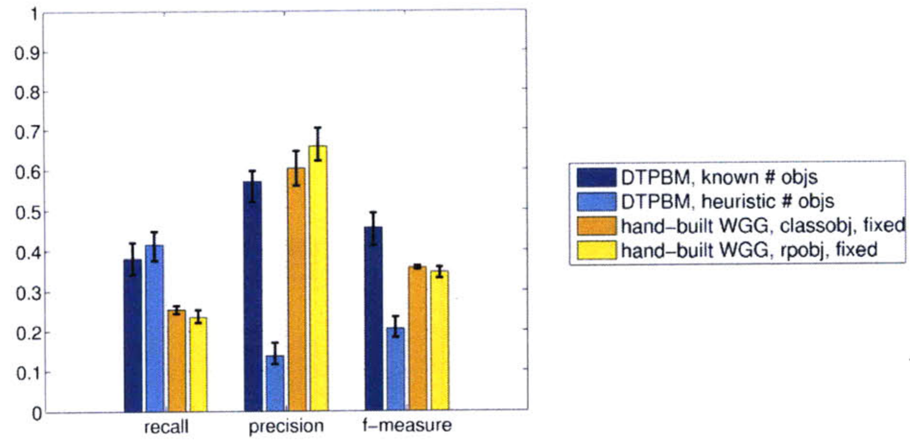
class later in the section.)

The first observation we can make is that the WGG (orange) performs significantly better according to f-measure than the heuristic DTPBM (light blue). Although the heuristic DTPBM has much higher recall than the WGG, the WGG's vastly higher precision results in an higher f-measure score overall. To further understand these numbers, consider the last two columns in the table in Figure 3-7. While the WGG detects an average of 312 objects and gets 188 correct, the DTPBM detects over 2200, getting 307 correct on average. The DTPBM's high recall is due to the fact that it detects such a large number of objects in each image that it is more likely to find the correct objects. The WGG, on the other hand, detects relatively few objects, but its detections are very likely to be correct. Figure 3-8 shows example detections from both systems, illustrating the qualitative difference in the results. Additional WGG detections are shown in Figure 3-14 at the end of the chapter.

A second observation is that the WGG performs quite a bit worse than the DTPBM when it knows the ground-truth number of objects of each class for each test image (dark blue). Remember that this version of the DTPBM is cheating, so it is not a completely fair comparison. However, the fact that the object detector can perform so well when it has access to this information is interesting. It suggests that the detector is doing a good job of ranking candidate detections, but that it has trouble knowing where to threshold them, even using a simple heuristic on the number of objects per image. A WGG model can thus be seen as one way to capture how many objects are expected from each class, while taking into account both the detector scores and geometry information.

The relatively low recall and high precision of the WGG model here will turn out to be a consistent feature of its performance. It makes few detections, but those detections are often correct. (And even when they are not correct, they are often close misses; see the examples in Figure 3-14.) It seems that the perceptron tends to set the weights on tree structure such that both the object detector and geometry features need to be satisfied in order for a detection to be made. This is likely due to the fact that the object detector's output is so noisy, while the inherent variance in spatial arrangements means that the geometry models cannot be informative enough to overcome the noise. In Chapter 6, we will discuss several paths of future work that could address this issue.

Finally, notice that *all* the models perform poorly, by absolute standards, on this dataset. Even



 # targets (all models): 741.0

	recall	precision	f-measure	# detections	# correct
DTPBM, known # objs	0.380 [0.341 0.420]	0.571 [0.521 0.598]	0.456 [0.412 0.493]	493.0	280.8
DTPBM, heuristic # objs	0.415 [0.376 0.447]	0.138 [0.117 0.171]	0.205 [0.184 0.235]	2263.6	306.8
WGG, classobj, fixed	0.254 [0.244 0.264]	0.604 [0.561 0.647]	0.357 [0.353 0.362]	312.0	188.0
WGG, rpobj, fixed	0.235 [0.221 0.253]	0.659 [0.623 0.705]	0.346 [0.332 0.359]	264.6	174.0

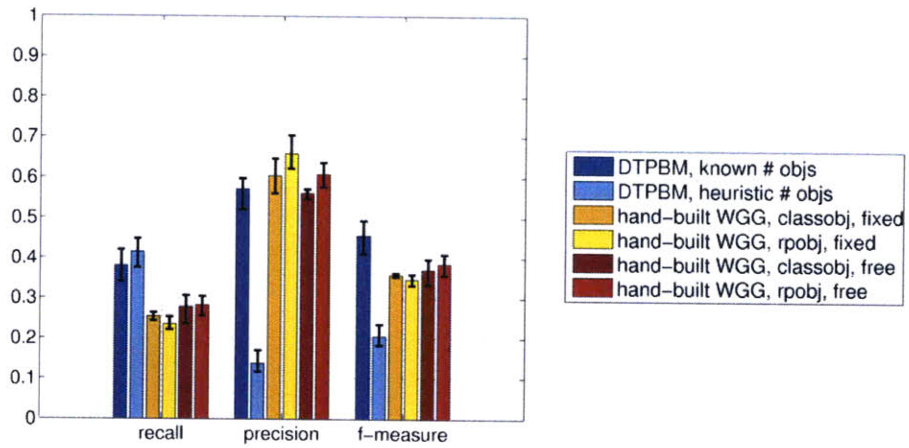
Figure 3-9: Cumulative results for the DTPBM object detector (dark and light blue) and the hand-built WGG structure with class-based (orange) versus rule-part-based (yellow) object detector features. 'classobj' means class-based object detector features were used, while 'rpobj' means rule-part-based features were used.

the DTPBM, a state-of-the-art object detector, when given ground truth knowledge about the number of each object class in each test image, produces a cumulative f-measure of significantly less than 0.5. This highlights the difficulty of the dataset. There are actually a fair number of objects which are strictly impossible to detect, because they are too large or small to be captured by the range of scales the detector considers, or because they are rotated such that their aspect ratio is too different from the detector's canonical object size. However, the poor performance also points to the inherent difficulty of the object recognition task in general, and emphasizes the extent to which the current abilities of the field are lacking.

Class-Based Versus Rule-Part-Based Object Detector Features

In Section 2.3.1, we described how the features and weights on the object detector output could either be associated with an entire object class c ($\phi_c^{f,1}(I,T)$), or with each individual rule part k in each rule ($\phi_k^{f,2}(I,T)$). We discussed how the rule-part-based version offers greater expressive power, since it allows the weights to capture that the object detector may be more or less reliable depending on the context of the object in the scene. However, the class-based version has the advantage of fewer parameters and greater sharing of training data across different parts of the model.

Figure 3-9 summarizes the cumulative performance of each approach, and compares them to both versions of the DTPBM object detector. We can see that the choice of class-based object detector features (orange) versus rule-part-based features (yellow) does not make much of a difference, at least on this dataset and with the hand-built WGG structure. The rule-part-based features perform better in precision but slightly worse in recall than the class-based features. But because the overall recall scores are lower, the class-based features' slight advantage in recall is more important than



 # targets (all models): 741.0

	recall	**precision**	**f-measure**	**# detections**	**# correct**
DTPBM, known # objs	0.380 [0.341 0.420]	0.571 [0.521 0.598]	0.456 [0.412 0.493]	493.0	280.8
DTPBM, heuristic # objs	0.415 [0.376 0.447]	0.138 [0.117 0.171]	0.205 [0.184 0.235]	2263.6	306.8
WGG, classobj, fixed	0.254 [0.244 0.264]	0.604 [0.561 0.647]	0.357 [0.353 0.362]	312.0	188.0
WGG, rpobj, fixed	0.235 [0.221 0.253]	0.659 [0.623 0.705]	0.346 [0.332 0.359]	264.6	174.0
WGG, classobj, free	0.278 [0.237 0.306]	0.561 [0.547 0.571]	0.371 [0.334 0.397]	365.4	205.0
WGG, rpobj, free	0.282 [0.256 0.304]	0.608 [0.576 0.637]	0.385 [0.357 0.409]	342.8	208.4

Figure 3-10: Cumulative results for the DTPBM object detector and four variations of the hand-built WGG structure. 'fixed' refers to fixed ground-truth internal geometry, while 'free' refers to free ground-truth internal geometry.

the rule-part-based features' advantage in precision. However, in general, the difference between the two versions is only barely significant in this context. We will revisit the question of class-based versus rule-part-based object detector features again next chapter.

Fixed Versus Free Internal Geometry in Ground-Truth Trees

We discussed in Section 3.1.2 how we have two options for setting the geometry of the internal nodes of the ground truth tree for each training image during the perceptron. First, we can simply use the provided tree geometry. For the hand-build WGG structure, we have labeled ground truth tree structure, and labeled bounding boxes on the leaf nodes (corresponding to labeled object silhouettes). We then compute the geometry vector for each internal node from the bounding box of its descendent leaves' bounding boxes. We call this "fixed ground truth internal geometry." All the WGG experiments we have presented so far have used this method.

However, this approach is sometimes less than optimal. The geometry vectors on the internal nodes represent coordinate frames in which the expected locations and scales of their children are represented. The bounding boxes of for two sets of corresponding leaf nodes, one of which is missing an object, are likely to be very different. This would result in very different relative geometry vectors for corresponding objects, in turn misleading the geometry models. (We will discuss this issue in much more detail in the context of structure learning next chapter.)

In these cases, a better approach is to run parsing to determine the best geometry vectors for the internal nodes, while constraining the tree structure to match the labeled structure, and the leaves to have the correct locations and scales. We refer to this as "free ground truth internal geometry".

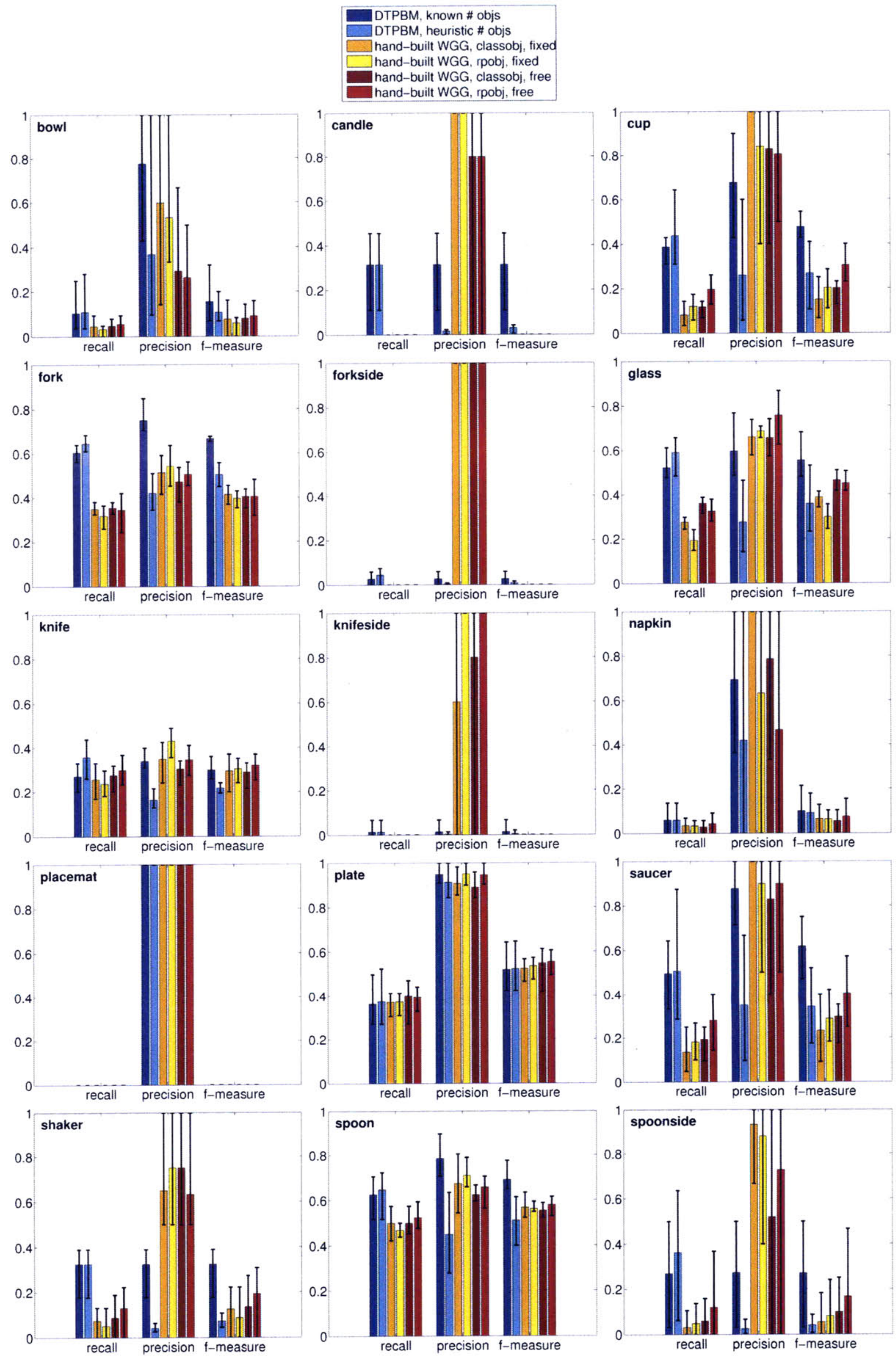
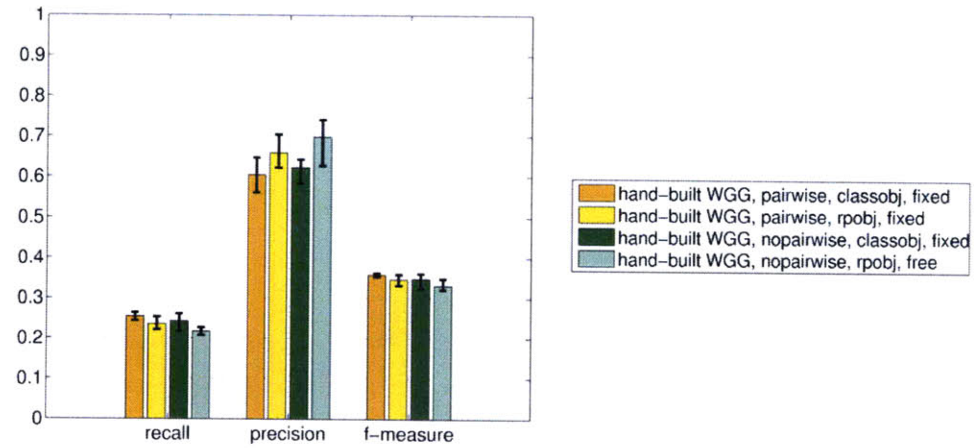


Figure 3-11: Per-class results for the DTPBM object detector and the WGG hand-built structure. For numeric per-class results, see Appendix B (Tables B.4, B.5, B.6, B.7, B.8, and B.9).



 # targets (all models): 741.0

	recall	precision	f-measure	# detections	# correct
pairwise, classobj, fixed	0.254 [0.244 0.264]	0.604 [0.561 0.647]	0.357 [0.353 0.362]	312.0	188.0
pairwise, rpobj, fixed	0.235 [0.221 0.253]	0.659 [0.623 0.705]	0.346 [0.332 0.359]	264.6	174.0
nopairwise, classobj, fixed	0.242 [0.218 0.260]	0.623 [0.584 0.643]	0.348 [0.325 0.361]	287.6	178.8
nopairwise, rpobj, fixed	0.217 [0.208 0.227]	0.698 [0.627 0.741]	0.331 [0.321 0.348]	231.2	161.0

Figure 3-12: Cumulative results for the hand-built WGG structure with and without pairwise features.

In Figure 3-10, we see the performance of the WGG model using both methods. All four combinations of class-based versus rule-part-based object detector features and fixed versus free internal geometry are presented, and the results are again compared to the two version of the DTPBM object detector. We also break out the results for all six models by object class in Figure 3-11.

As we would hope, using free internal geometry does give a slight improvement over fixed geometry, with the hand-built structure. The improvement is more marked with the rule-part-based features, and for certain object classes, such as cup, saucer, glass, and shaker. These objects almost always appear in pairs or groups, such that misleading parent coordinate frames resulting from fixed internal geometry would be more harmful.

Pairwise Tree Structure Features Versus None

The last set of experiments in this chapter explore the importance of the pairwise features on tree structure in the WGG model. Our intuition is that the pairwise features play a critical role in capturing co-occurrence information among sets of objects, because each rule part in a rule is optional. To test this hypothesis, we remove the pairwise features from the model completely, relying only on the individual rule part weights to act as thresholds on subtree scores.

Figure 3-12 compares the performance of the WGG model with and without pairwise features (each with class-based and rule-part-based object detector features), while Figure 3-13 shows the per-class results. We see that the pairwise weights do help by a small amount, but it is not significant. They would likely help more, but the hand-built structure already builds in a fair amount of co-occurrence information in the grouping of objects into composite classes.

We shall see in the next few chapters that pairwise features become more important in larger, more complex datasets, when the grammar structure is not predefined by a human. Indeed, the next two chapters focus on how to learn WGG models when there is no labeled structure.

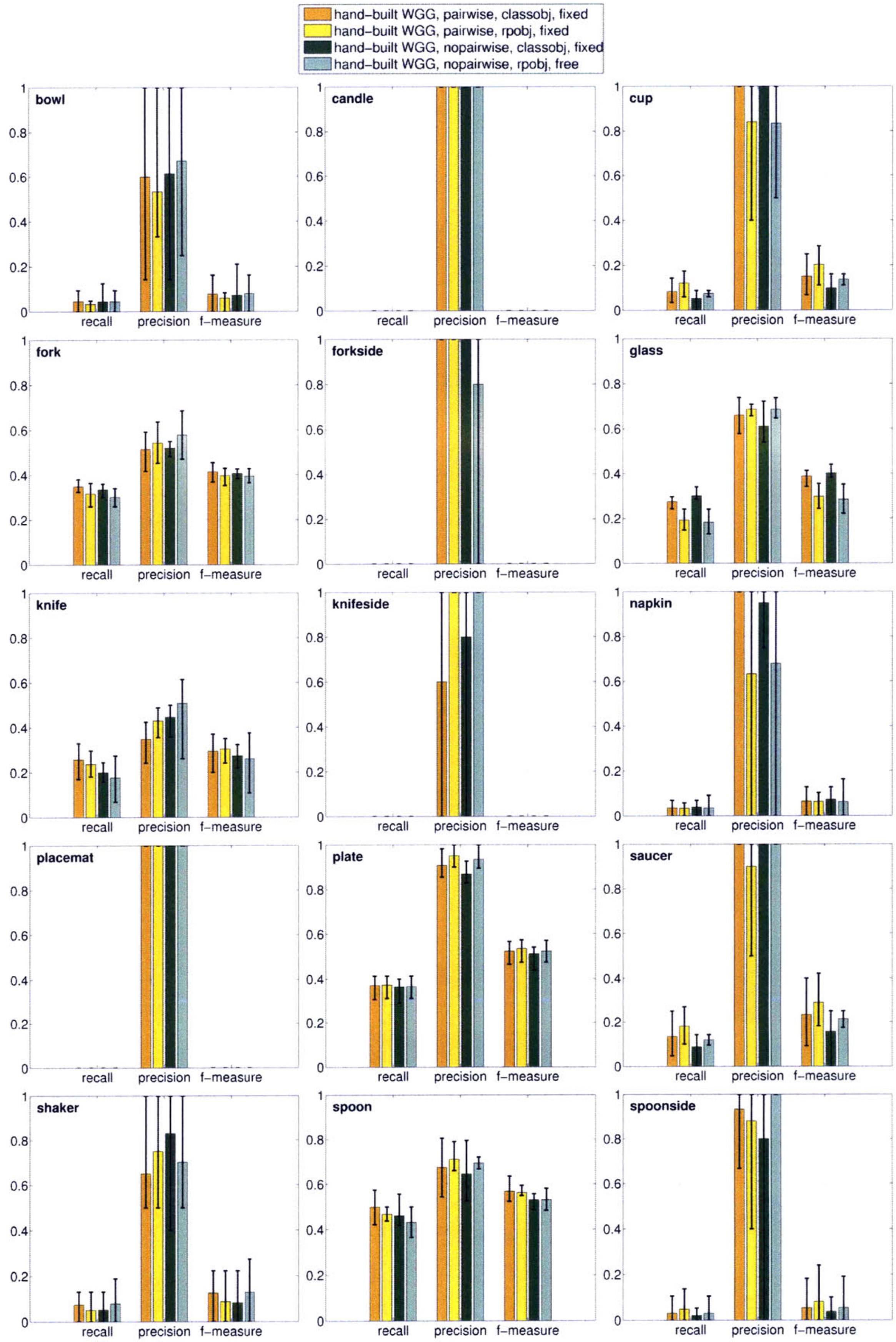


Figure 3-13: Per-class results for the WGG hand-built structure with and without pairwise features. For numeric per-class results, see Appendix B (Tables B.6, B.7, B.10, and B.11).

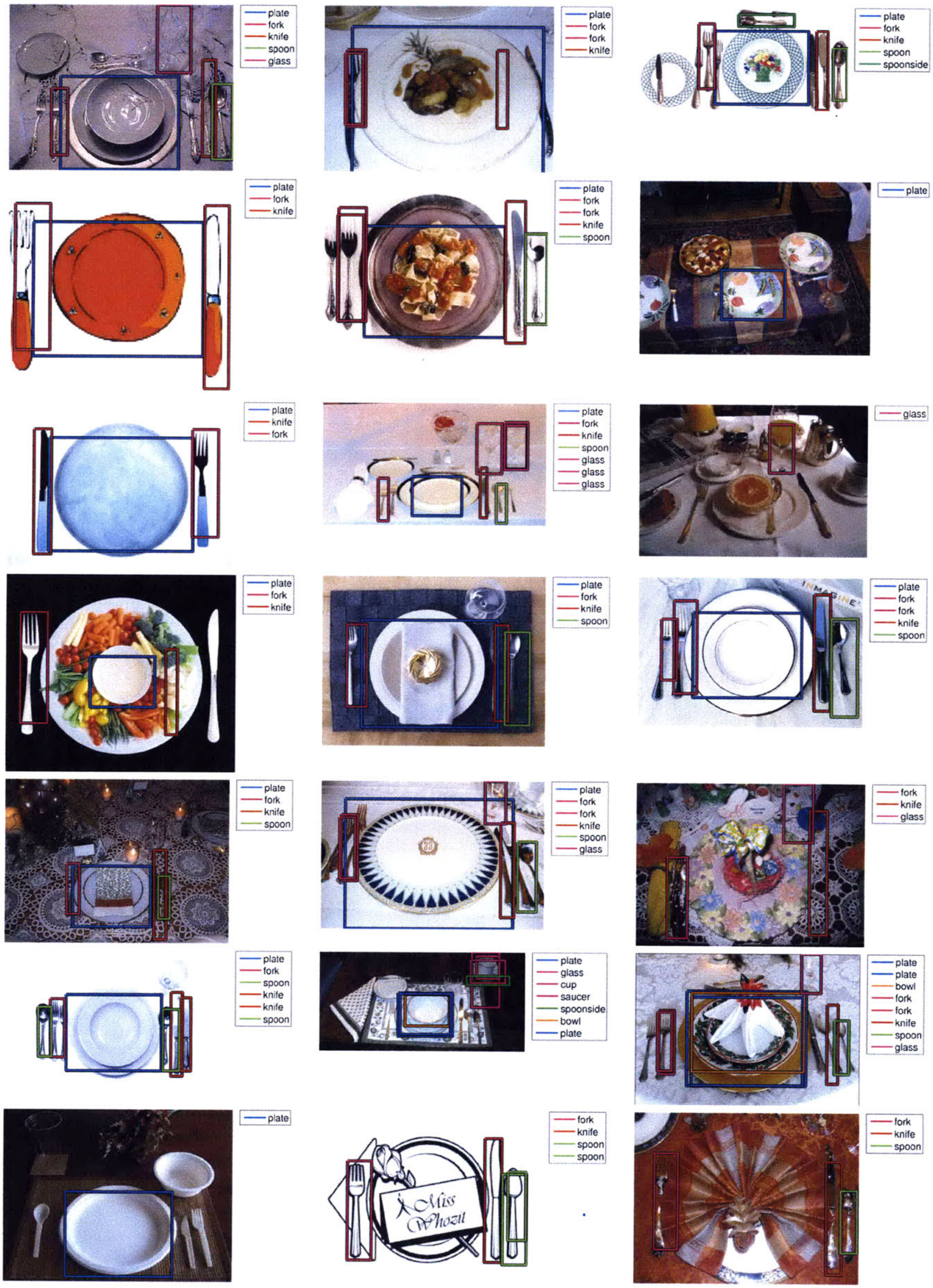


Figure 3-14: Representative detections from the hand-built WGG (pairwise, classobj, fixed).

Chapter 4

Two-Level Structure Learning in WGGs

In the last chapter, we presented the structured perceptron algorithm for learning the parameters of a WGG with fixed structure. In this chapter and the next, we will discuss several algorithms for learning the structure of the WGG itself.

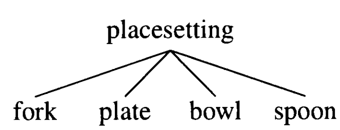
The formal structure learning problem in WGGs is as follows. We are given:

- A set of object classes B and scene classes S.
- A set of training images I_i, for $i = 1, \ldots, m$, in which the ith image is annotated with:
  - a scene class $c_i \in S$, and
  - a set of m_i objects, in which the jth object has object class $c_{i,j} \in B$ and bounding box $b_{i,j}$, for $j = 1, \ldots, m_i$.

The goal is to find:

- A small set of good composite classes C.
- A set of rules R—one for each learned composite class, and one for each scene class in S.
- A set of scene trees T_i for the training images, with class and rule part labels corresponding to C and R, and geometry vectors in the nodes.

In this chapter, we will simplify the problem by learning only two-level WGG structures. In other words, we will not learn any new composite classes; we will only learn a single rule for each scene class $c \in S$, in which the rule parts correspond to primitive object classes directly. So all of the scene trees for the WGG structures learned in this chapter will have two levels, like this:



In the next chapter, we will relax this assumption, and learn three-level hierarchical WGG structures. However, we shall see that in fact two-level WGGs are often powerful enough to capture the co-occurrence and relative geometry information we need. Furthermore, the techniques we develop to learn two-level structures will serve as building blocks for the algorithms in the next chapter.

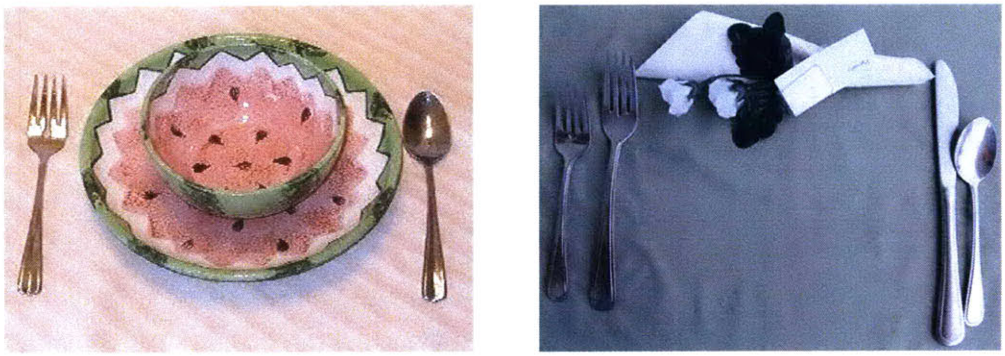


Figure 4-1: Two images of simple placesettings.

4.1 Learning Class Templates

In the two-level learning case, the labels on the training images make it easy to initialize reasonable scene trees. Each image I_i is labeled with a scene class c_i and a set of m_i objects with class and bounding box information $\langle c_{i,j}, b_{i,j} \rangle$. So we can write down this tree T_i for image I_i:

$$c_i \rightarrow c_{i,1} \quad \ldots \quad c_{i,j} \quad \ldots \quad c_{i,m_i}$$

where the geometry vector for in the jth leaf node is produced from the labeled bounding box $b_{i,j}$, and the geometry vector in the root is produced from the bounding box of the leaves.

Among images with the same scene class c, each tree will have the class label of c on the root node, but a different set of leaf nodes. For the two placesetting images in Figure 4-1, we might have the following tree structures (remember that the ordering of leaves is arbitrary):

placesetting
fork plate bowl spoon

placesetting
fork fork napkin knife spoon

In this simple example, we might want to learn this rule:

$$\text{placesetting} \rightarrow \text{fork}_1\ \text{fork}_2\ \text{plate}_3\ \text{bowl}_4\ \text{napkin}_5\ \text{knife}_6\ \text{spoon}_7$$

with this set of rule part labels (RPIDs) on the tree leaves:

placesetting
fork_2 plate_3 bowl_4 spoon_7

placesetting
fork_1 fork_2 napkin_5 knife_6 spoon_7

The choice of which RPID to assign to each node is also a choice of the correspondence between leaves in different trees, and it is important because assigning two leaves the same RPID means they will share the same model parameters. The choice might be straightforward when there is at most one instance of an object class in each image, and it appears at a fairly consistent position and size with respect to the other objects. However, it becomes more challenging when there is more than one object of the same class in some images (e.g., the two forks in the example above), and in cases of high variability in the presence, locations, and scales of objects.

In this light, we can view the task of learning a rule for a scene class as a constrained matching problem. The goal is to find an assignment of RPIDs to the leaves of the trees associated with the training images (i.e., a correspondance among the leaves across trees) such that the object class labels are respected, and that the variance in the geometry of the nodes assigned to the same RPID (relative to their parents' geometry) is as small as possible. The rule we learn for the scene class is is essentially a template specifying the set of RPIDs, and their associated object classes, that appear in the tree leaf labels. For this reason, we will refer to this matching problem as "learning a class template."

We will use agglomerative clustering on trees as the basis for our approach to learning class templates. A *cluster* is a tuple $\langle K, \mathcal{T} \rangle$, where:

- the template K is a set of RPIDs k and their class labels $\text{class}(k)$; and
- $\mathcal{T}$ is a set of two-level trees, where for each tree $T \in \mathcal{T}$, each child node $t \in T$ has RPID label $\text{rpid}(t) \in K$. (We will use $t \in T$ as shorthand for $t \in \text{children}(\text{root}(T))$ in this chapter.)

Again, the sets of child nodes in the trees of a cluster do not, in general, have the same number of members or class labels. The template and the RPID labels on the nodes specify the correspondence among these arbitrarily different sets.

Then, we can learn a template for a set of trees $\mathcal{T}$ (with the same root scene class) using the following strategy. First, we create an initial cluster for each training image I_i, with a single tree T_i and a template simply listing the initial (arbitrary but unique) RPIDs of its leaf nodes. Then, we perform agglomerative clustering on these clusters, merging the pair of clusters on each step that minimizes the geometry variance across the trees in the merged cluster, and searching over matchings between templates to resolve ambiguous RPID assignments. In this way, we incrementally build up clusters with templates that fit their component trees well. We continue until a single cluster contains all groups; the template for this final cluster is the learned rule.

Every approach to structure learning or clustering must somehow address the question of "How many clusters?" In the approach we have just outlined, there is an implicit assumption that the number of rule parts for each object class in the learned rule is the maximum number of instances of that class ever seen in a single training image. There are some cases in which this is not an appropriate assumption—for example, when an object class only occurs at most once in each image, but appears at several different locations with respect to the other objects. However, this type of situation is relatively rare in practice. In general, we expect that the advantage of having larger clusters, and therefore more training examples for the parameters in each rule part, will tend to outweigh the expressive power of having more clusters.

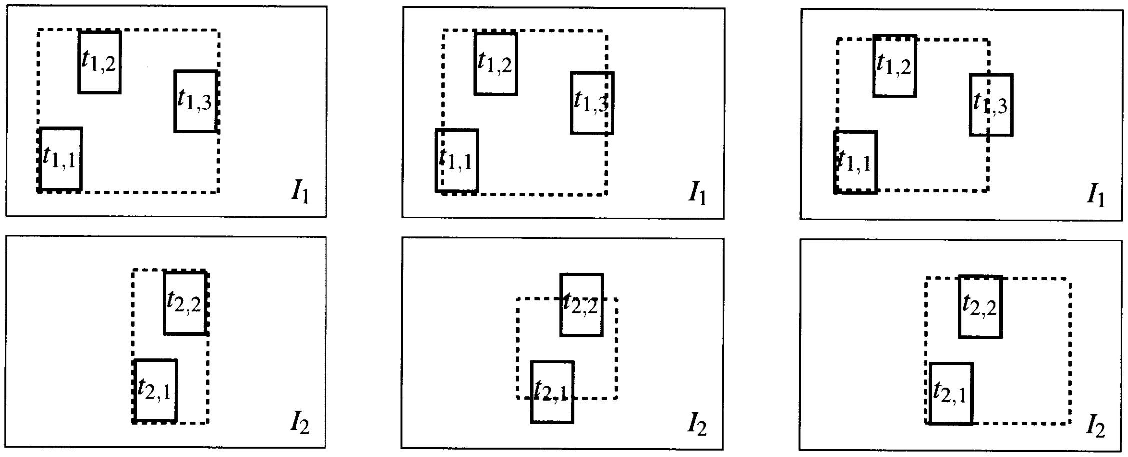
The next three sections describe the components we need to formalize this algorithm, which we present in detail in Section 4.1.4.

4.1.1 Tree Node Geometry Vectors

In Section 2.2.2, we introduced the absolute geometry vector $\text{geom}(t) = \langle x, y, s \rangle$ associated with a tree node t. In this chapter and the next, we will use the slightly modified version

$$\text{geom}_{\text{abs}}(t) = \langle x, y, \log s \rangle$$

which will be convenient mathematically.



(a) Initial parent bounding boxes (dashed).

(b) Initial parent geometry vectors (visualized as squares).

(c) Updated parent geometry, for a match of $t_{1,1}$ to $t_{2,1}$ and $t_{1,2}$ to $t_{2,2}$.

Figure 4-2: Two cartoon training images I_1 and I_2, with object bounding boxes and parent geometry. All objects have the same class.

We will also define a *relative geometry vector* $\text{geom}_{\text{rel}}(t)$ for each non-root node, which represents the geometry of a node t, transformed to lie in the coordinate frame defined by the geometry its parent node. If we have child and parent absolute geometry vectors:

$$\text{geom}_{\text{abs}}(t) = \langle x_c, y_c, \log s_c \rangle \qquad\qquad \text{geom}_{\text{abs}}(\text{parent}(t)) = \langle x_p, y_p, \log s_p \rangle$$

then the relative geometry vector of t is given by:

$$\text{geom}_{\text{rel}}(t) = \left\langle \frac{s_p(x_c - x_p)}{\kappa_{\text{geom,x}}}, \frac{s_p(y_c - y_p)}{\kappa_{\text{geom,y}}}, \frac{\log s_c - \log s_p}{\kappa_{\text{geom,s}}} \right\rangle \tag{4.1}$$

where $\langle \kappa_{\text{geom,x}}, \kappa_{\text{geom,y}}, \kappa_{\text{geom,s}} \rangle$ are the same constants used to scale the relative geometry features in Section 2.3.2.[1]

We will use the relative geometry vectors of the leaf nodes to perform the clustering and matching to minimize variance. Therefore, since the relative geometry vectors are computed with respect to the absolute geometry of the root, the value of the absolute root geometry vectors will play an important role of the quality of the matching we learn. To see this, consider Figure 4-2.

In Figure 4-2(a), the root bounding box for each image is computed from the labeled object bounding boxes. In Figure 4-2(b), each root bounding box from (a) is used to compute an absolute geometry vector for the root, which is visualized as a square.[2] The relative geometry vectors for the child nodes in each image are then computed with respect to these coordinate frames.

We want to find a matching such that $t_{1,1}$ corresponds to $t_{2,1}$ and $t_{1,2}$ corresponds to $t_{2,2}$, since these pairs of objects have similar spatial relationships. But using relative geometry computed from the parents in Figure 4-2(b) will not capture this pattern as clearly as we would like. For example, $t_{1,2}$'s x position relative to the centroid of its parent in (b) will be negative, while $t_{1,2}$'s relative x position will be positive.

[1] The relative geometry vector is related, but not identical, to the relative geometry feature vector defined in Section 2.3.2. There are no offset terms, and the values are not capped to lie in $[-1, 1]$.

[2] The centroid of the bounding box defines an origin, while the scale factor can be seen as defining the unit length in a coordinate frame. The process is the same as used in "Mapping from Scene Trees to Object Detections," in Section 2.4.3.

In fact, we would like to hypothesize parent coordinate frames like those shown in Figure 4-2(c), such that the relative geometry vectors of corresponding objects are numerically similar. We can think of the absolute geometry vectors for the root nodes as hidden variables in the learning problem. To address this issue, we incrementally update the parent absolute geometry vectors during clustering. We jointly find an assignment of template RPIDs to leaf nodes *and* a set of parent geometry values such that the variance of the relative geometry of child nodes, under the assignment and with respect to the parent geometry, is as low as possible.

Updating Parent Geometry

Here, we describe an update step that improves the absolute geometry vectors for the parent nodes of a set of trees, given a current set of parent absolute geometry vectors, and an assignment of template RPIDs to the child nodes. We will incorporate this step into the clustering algorithm in Section 4.1.4.

Given a cluster $\langle K, \mathcal{T} \rangle$, we update the parent geometry vectors of the trees as follows:

1. For each child node t in each tree $T \in \mathcal{T}$, compute $\mathrm{geom}_{\mathrm{rel}}(t)$ using Equation 4.1, with respect to the current setting of $\mathrm{geom}_{\mathrm{abs}}(\mathrm{root}(T))$.

2. For each rule part $k \in K$, compute the mean relative geometry vector μ_k across all child nodes that are currently assigned RPID k:

$$\mu_k = \frac{1}{n_k} \sum_{T \in \mathcal{T}} \sum_{\substack{t \in T \text{ s.t.} \\ \mathrm{rpid}(t)=k}} \mathrm{geom}_{\mathrm{rel}}(t) \tag{4.2}$$

   where n_k is the number of child nodes across all trees with RPID label k.

3. For each tree $T \in \mathcal{T}$,

   (a) For each child node $t_j \in T$, with absolute geometry $\mathrm{geom}_{\mathrm{abs}}(t_j) = \langle x, y, \log s \rangle$ and RPID $\mathrm{rpid}(t_j) = k$, compute the child's vote $\mathbf{v}_j = \langle x_j, y_j, \log s_j \rangle$ for its ideal absolute parent geometry vector, using the relative mean for k:

$$x_j = x - \frac{\kappa_{\mathrm{geom,x}} \mu_{k,\mathrm{x}}}{s_j} \qquad y_j = y - \frac{\kappa_{\mathrm{geom,y}} \mu_{k,\mathrm{y}}}{s_j} \qquad \log s_j = \log s - \kappa_{\mathrm{geom,s}} \mu_{k,\mathrm{s}}$$

   Notice that these expressions invert the relative geometry vector computation in Equation 4.1, so that if the relative geometry of t_j was recomputed with respect to its ideal parent geometry $\mathbf{v}_j$, it would exactly equal the mean vector μ_k.

   (b) Update the absolute parent geometry $\mathrm{geom}_{\mathrm{abs}}(\mathrm{root}(T))$ to the mean of the votes of its children:

$$\mathrm{geom}_{\mathrm{abs}}(\mathrm{root}(T)) = \frac{1}{m} \sum_{j=1}^{m} \mathbf{v}_j$$

   where $m = |\mathrm{children}(\mathrm{root}(T))|$.

In Figure 4-2, performing this update on the parent geometry in (b), with the correspondence of $t_{1,1}$ to $t_{2,1}$ and $t_{2,1}$ to $t_{2,2}$, produces exactly the updated parents shown in (c).

4.1.2 Normalized Variance in Geometry

The agglomerative clustering algorithm iteratively merges the pair of clusters resulting in the lowest variance in relative geometry. However, the trees in a cluster have different numbers of children, with different class and RPID labels; we cannot simply concatenate their geometry vectors in some canonical order. We need a measure of the variance of a cluster that can handle the fact that some cluster members may be missing some elements of the template. Furthermore, we need to be able to compare these measures across different choices of cluster pairs to merge, despite the fact that the merged clusters might have different size templates.

Recall that the L^p norm of a vector $\mathbf{x}$ is calculated as:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |\mathbf{x}_i|^p \right)^{1/p}$$

where n is the number of dimensions of $\mathbf{x}$ and $|x|$ is the absolute value of x. This norm is inherently sensitive to the number of dimensions n in $\mathbf{x}$. So we define the normalized L^p norm of a vector $\mathbf{x}$ as:

$$\|\mathbf{x}\|_p^* = \left(\frac{1}{n} \sum_{i=1}^{n} |\mathbf{x}_i|^p \right)^{1/p} \tag{4.3}$$

This norm is dimensionless, in the sense that it can be used to compare vectors with different numbers of dimensions.

Using the normalized L^2 norm, we define the *normalized geometry variance* of a set of trees $\mathcal{T}$ and a template K, as follows:[3]

1. Compute the mean relative geometry vector μ_k for each RPID $k \in K$, as in Equation 4.2.
2. For each tree $T_i \in \mathcal{T}$,
   (a) Let J_i be the set of indices of child nodes of T_i whose RPID labels appear in K.
   (b) Let $\mathbf{v}_i$ be the vector produced by concatenating the relative geometry vectors for the nodes in J_i:
   $$\mathbf{v}_i = \left\langle \mathrm{geom}_{\mathrm{rel}}(t_{i,j}) \right\rangle_{j \in J_i}$$
   (c) Let $\overline{\mu}_i$ be the vector produced by concatenating the mean vectors for the RPIDs on nodes in J_i:
   $$\overline{\mu}_i = \left\langle \mu_{\mathrm{rpid}(t_{i,j})} \right\rangle_{j \in J_i}$$
3. The normalized geometry variance is the average squared normalized L^2 distance to the mean, over all trees:
   $$\mathrm{var}_{\mathrm{geom}}(\mathcal{T}, K) = \frac{1}{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} \left(\|\mathbf{v}_i - \overline{\mu}_i\|_2^* \right)^2 \tag{4.4}$$
   If a tree T_i has no child nodes with RPIDs in K (so $J_i = \emptyset$), it is not included in the average.

The normalized geometry variance is a scalar value that captures the distance between relative geometry vectors of corresponding objects in different images, while handling the fact that not all RPIDs in the template are present in all trees.

[3]This definition intentionally allows the template K to be any subset of the full set of RPIDs on the child nodes of the trees in $\mathcal{T}$. This will be useful in the next section.

Input:

- Template $K_{1,c}$ and set of trees $\mathcal{T}_1$.
- Template $K_{2,c}$ and set of trees $\mathcal{T}_2$.
- Assume $n_1 = |K_{1,c}| \leq n_2 = |K_{2,c}|$ and $K_{1,c} \cap K_{2,c} = \emptyset$.

Output: Merged template K_c and updated RPID labels in $\mathcal{T}_2$.

Assign any fixed ordering $\mathbf{k}_1$ to elements of $K_{1,c}$.
Let $d^* = \infty$.
For each subset $K_2' \subseteq K_{2,c}$ such that $|K_2'| = n_1$,
    For each permutation $\mathbf{k}_2$ of K_2',
        For $\ell = 1, \ldots, n_1$,
            For each child node t in each tree $T \in \mathcal{T}_2$ such that $\text{rpid}(t) = \mathbf{k}_{2,\ell}$,
                Let $\text{rpid}(t) = \mathbf{k}_{1,\ell}$.
        Let $d = \text{var}_{\text{geom}}(\mathcal{T}_1 \cup \mathcal{T}_2, K_{1,c})$. **Equation 4.4**
        If $d < d^*$, let $d^* = d$, and $\mathbf{k}_2^* = \mathbf{k}_2$.
Let $K_c = K_{1,c} \cup (K_{2,c} - \{\mathbf{k}_2^*\})$.
For $\ell = 1, \ldots, n_1$,
    For each child node t in each tree $T \in \mathcal{T}_2$ such that $\text{rpid}(t) = \mathbf{k}_{2,\ell}^*$,
        Let $\text{rpid}(t) = \mathbf{k}_{1,\ell}^*$.

Algorithm 4.1: An exact algorithm for matching cluster templates (for a single class c).

4.1.3 Matching Templates To Minimize Geometry Variance

There is one more component problem we must solve before presenting the full learning algorithm. To merge two clusters, we need to search over correspondences to match their two templates, choosing the matching that minimizes the geometry variance we just defined. The problem is constrained, since only RPIDs with the same class label may be matched. So first, we consider the case of matching the portion of the templates with the same class.

Let $\langle K_1, \mathcal{T}_1 \rangle$ and $\langle K_2, \mathcal{T}_2 \rangle$ be two clusters, and let c be a class that appears in both their templates. Let $K_{1,c}$ be the set of n_1 RPIDs in K_1 with class c, and $K_{2,c}$ be the same for K_2, with size n_2. Without loss of generality, we can assume that $n_1 \leq n_2$, and that $K_{1,c} \cap K_{2,c} = \emptyset$. Then we can search exactly for the best matching of RPIDs in $K_{2,c}$ to RPIDs in $K_{1,c}$ by enumerating subsets and permutations of the elements in $K_{2,c}$ (assuming a fixed ordering of elements of $K_{1,c}$), and choosing the matching that minimizes the geometry variance of the trees under that assignment. Pseudocode is shown in Algorithm 4.1.

When n_1 and n_2 are large, we cannot afford to find the best matching exactly. In that case, we adopt an approximate strategy, in which we greedily assign elements of $K_{2,c}$ to elements of $K_{1,c}$ until all members of $K_{1,c}$ are matched. This algorithm is shown in detail in Algorithm 4.2.

In practice, we use the exact algorithm whenever the number of loops it would take is small enough:

$$\binom{n_2}{n_1} \times n_2! \leq M$$

where M is some constant (we use $M = 1000$), and the approximate algorithm otherwise.

We can match full templates by performing the matching independently for each class that appears in both templates. Any class that appears in only one of the templates can be included in the final template with no changes. The full matching algorithm is shown in Algorithm 4.3.

Input:

- Template $K_{1,c}$ and set of trees $\mathcal{T}_1$.
- Template $K_{2,c}$ and set of trees $\mathcal{T}_2$.
- Assume $n_1 = |K_{1,c}| \leq n_2 = |K_{2,c}|$ and $K_{1,c} \cap K_{2,c} = \emptyset$.

Output: Merged template K_c and updated RPID labels in $\mathcal{T}_2$.

1. Assign any fixed orderings $\mathbf{k}_1$ and $\mathbf{k}_2$ to the elements of $K_{1,c}$ and $K_{2,c}$, respectively.

2. Let $\mathbf{D}$ be a matrix of size $n_1 \times n_2$.

3. For $\ell_1 = 1, \ldots, n_1$,

4.   For $\ell_2 = 1, \ldots, n_2$,

5.   For each child node t in each tree $T \in \mathcal{T}_2$ such that $\text{rpid}(t) = \mathbf{k}_{2,\ell_2}$,

6.   Let $\text{rpid}(t) = \mathbf{k}_{1,\ell_1}$.

7.   Let $\mathbf{D}_{\ell_1,\ell_2} = \text{var}_{\text{geom}}(\mathcal{T}_1 \cup \mathcal{T}_2, \{\mathbf{k}_{1,\ell_1}\})$. **Equation 4.4**

8. Let $\mathbf{k}_2^* = \mathbf{0}$ with length n_1.

9. Loop:

10.   Let $\langle \ell_1^*, \ell_2^* \rangle = \text{argmin}_{\ell_1,\ell_2} \mathbf{D}_{\ell_1,\ell_2}$

11.   If $\mathbf{D}_{\ell_1^*,\ell_2^*} = \infty$, exit loop.

11.   Let $\mathbf{k}_{2,\ell_1^*}^* = \mathbf{k}_{2,\ell_2^*}$.

12.   For $\ell_1 = 1, \ldots, n_1$, let $\mathbf{D}_{\ell_1,\ell_2^*} = \infty$.

13.   For $\ell_2 = 1, \ldots, n_2$, let $\mathbf{D}_{\ell_1^*,\ell_2} = \infty$.

15. Let $K_c = K_{1,c} \cup (K_{2,c} - \{\mathbf{k}_2^*\})$.

16. For $\ell = 1, \ldots, n_1$,

17.   For each child node t in each tree $T \in \mathcal{T}_2$ such that $\text{rpid}(t) = \mathbf{k}_{2,\ell}^*$,

18.   Let $\text{rpid}(t) = \mathbf{k}_{1,\ell}^*$.

Algorithm 4.2: An approximate algorithm for matching cluster templates (for a single class c).

Input:

- Template K_1 and set of trees $\mathcal{T}_1$.
- Template K_2 and set of trees $\mathcal{T}_2$.

Output: Merged template K and updated RPID labels in $\mathcal{T}_1$ and $\mathcal{T}_2$.

1. Let $K = \emptyset$.

2. For each class c in K_1 but not K_2, let $K = K \cup K_{1,c}$.

3. For each class c in K_2 but not K_1, let $K = K \cup K_{2,c}$.

4. For each class c in both K_1 and K_2,

5.   If $|K_{1,c}| \leq |K_{2,c}|$, match $K_{2,c}$ to $K_{1,c}$ to get K_c (and update $\mathcal{T}_2$). **Algorithms 4.1, 4.2**

6.   Otherwise, match $K_{1,c}$ to $K_{2,c}$ to get K_c (and update $\mathcal{T}_1$). **Algorithms 4.1, 4.2**

7.   Let $K = K \cup K_c$.

Algorithm 4.3: An algorithm for matching cluster templates, across all classes.

4.1.4 A Clustering-Based Algorithm for Learning Class Templates

Now we are ready to define our algorithm for learning a template for a set of trees $\mathcal{T}$. The algorithm is given in Algorithm 4.4, and proceeds as follows. We begin by creating an initial cluster for each tree in $\mathcal{T}$ (Lines 1-5). Then, we perform agglomerative clustering on these clusters (Lines 6-15). On each iteration, we try merging each pair of existing clusters, finding the best-matching template K (Line 9), updating the parent geometry in the trees $\mathcal{T}$ (Line 11), and computing the relative geometry

Input: Set of trees $\mathcal{T}$, in which all trees have the same root class.

Output: Template K, and updated RPID labels on the child nodes in $\mathcal{T}$.

1. Let $\Gamma = \emptyset$.
2. For each tree $T \in \mathcal{T}$,
3.   Assign each child node $t \in T$ a globally unique RPID.
4.   Let K be the set of RPID (and class) labels on the child nodes in T.
5.   Let $\Gamma = \Gamma \cup \{\langle K, \{T\}\rangle\}$.
6. While $|\Gamma| > 1$,
7.   Let $d^* = \infty$.
8.   For each pair of clusters $\langle\langle K_i, \mathcal{T}_i\rangle, \langle K_j, \mathcal{T}_j\rangle\rangle \in \Gamma$,
9.   Match templates K_i and K_j to get K (and update $\mathcal{T}_i$ and $\mathcal{T}_j$). **Algorithm 4.3**
10.   Let $\mathcal{T} = \mathcal{T}_i \cup \mathcal{T}_j$.
11.   Update the parent geometry for $\mathcal{T}$ under template K. **Section 4.1.1**
12.   Let $d = \mathrm{var}_{\mathrm{geom}}(\mathcal{T}, K)$. **Equation 4.4**
13.   If $d < d^*$, let $d^* = d$, $i^* = i$, $j^* = j$, $K^* = K$, and $\mathcal{T}^* = \mathcal{T}$.
14.   Let $\Gamma = \Gamma - \{\langle K_{i^*}, \mathcal{T}_{i^*}\rangle, \langle K_{j^*}, \mathcal{T}_{j^*}\rangle\}$.
15.   Let $\Gamma = \Gamma \cup \{\langle K^*, \mathcal{T}^*\rangle\}$.
16. Let $\langle K, \mathcal{T}\rangle$ be the one remaining cluster in Γ.

Algorithm 4.4: An agglomerative clustering algorithm for learning a template for a set of trees.

variance for the resulting cluster (Line 12). We choose to merge the pair of clusters that produces the lowest variance (Lines 13-15). When only one cluster remains, its template K is the final result.

Using Algorithm 4.4, we learn a template K for a scene class c, using trees initialized for the training images labeled with scene class c, as described at the beginning of this chapter. From the learned template, it is then trivial to write down the rule for class c:

$$c \to \ldots, \mathrm{class}(k)_k \ldots \quad \forall_{k \in K}$$

We remove any rule part k that is assigned to less than two child nodes across all trees (and we remove the single tree node as well—it will not be used in training).

We can independently learn a rule for each scene class that appears in the training data. These rules form the learned WGG structure R, while the labeled trees T_i for the training images are exactly those produced by the template learning.

4.2 Experiments with Two-Level Learned Structures

In this section, we introduce the other two datasets we use in this thesis. We then present experiments evaluating two-level WGG models learned using the algorithms we just described. We compare these results to the DTPBM object detector and hand-built WGG structure presented last chapter.

4.2.1 The Big Placesetting Dataset

The second dataset used in this thesis, the "big placesetting dataset," has the same domain as the small placesetting dataset, but is much larger. It consists of 1052 images of placesettings, labeled with silhouettes for 10,177 objects from the same 15 object classes as the small placesetting dataset.

object class	# instances	% objects
bowl	509	5.0
candle	271	2.7
cup	461	4.5
fork	1185	11.6
forkside	243	2.4
glass	1815	17.8
knife	900	8.8
knifeside	200	2.0
napkin	842	8.3
placemat	345	3.4
plate	2064	20.3
saucer	285	2.8
shaker	195	1.9
spoon	574	5.6
spoonside	288	2.8

Table 4.1: The big placesetting dataset, with 1052 images and 10,177 labeled objects.

Figure 4-3 shows example images and labels from the dataset. Additional images are shown in Appendix A (Figure A-5). Table 4.1 lists the object classes and their frequencies. (Refer back to Figure 3-2 for examples of each object class).

The big placesetting dataset is even more challenging than its small version, with an average and standard deviation number of objects per image of 9.7 and 6.2, respectively. Again, the objects appear at a wide range of scales and rotations, with a wider range of object poses and occlusions than in the small dataset.

As before, the big placesetting dataset was labeled using the LabelMe web-based image annotation tool (Russell et al., 2008), and is freely available through the same system.[4] It is exactly equivalent to the `static_web_submitted_meg_placesettings2` LabelMe directory.

4.2.2 The House Dataset

Our final dataset, the "house dataset" consists of 848 images of houses, labeled with silhouettes for 13,801 objects from 10 object classes. Figure 4-4 shows example images and labels from the dataset. Additional images are shown in Appendix A (Figure A-6). Table 4.2 lists the object classes and their frequencies, and Figure 4-5 shows examples of each object class.

The house dataset is the most challenging of the three we consider. There are an average of 16.3 objects per image, with a standard deviation of 9, so there is enormous variability in the number and types of objects present. There is also more variability in the scales and aspect ratios of the objects, and their arrangement with respect to one another.

Again, the house dataset was labeled using LabelMe, and is freely available there, in the `static_web_submitted_meg_placesettings2` directory.[5]

4.2.3 Results on the Small Placesetting Dataset

In the next three sections, we evaluate the performance of learned two-level WGG models on each of the three datasets. We explore the same model and perceptron variations as in Chapter 3, and

[4] `http://labelme.csail.mit.edu/browseLabelMe/static_web_submitted_meg_placesettings2.html`

[5] `http://labelme.csail.mit.edu/browseLabelMe/static_web_submitted_meg_houses.html`

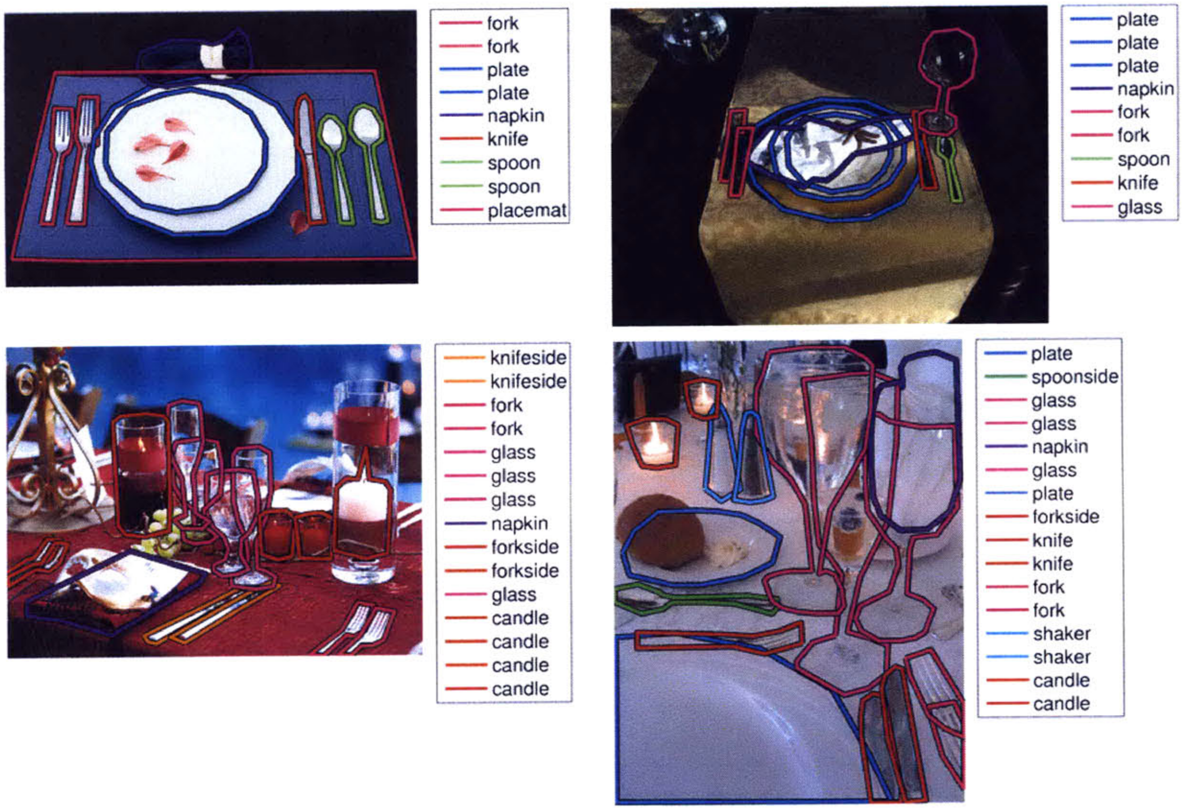


Figure 4-3: Examples of images and their labels from the big placesetting dataset.

Figure 4-4: Examples of images and their labels from the house dataset.

object class	# instances	% objects
car	224	1.6
chimney	709	5.1
door	1076	7.8
driveway	312	2.3
garagedoor	306	2.2
path	435	3.2
roof	1873	13.6
shutter	1743	12.6
steps	591	4.3
window	6532	47.3

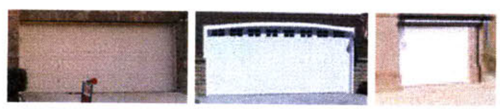
Table 4.2: The house dataset, with 848 images and 13,801 labeled objects.

(a) car

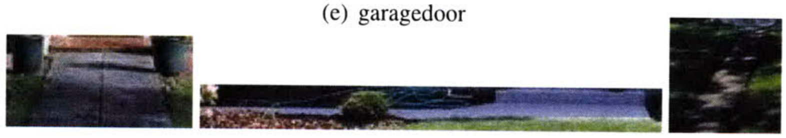
(b) chimney

(c) door

(d) driveway

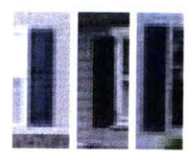


(e) garagedoor



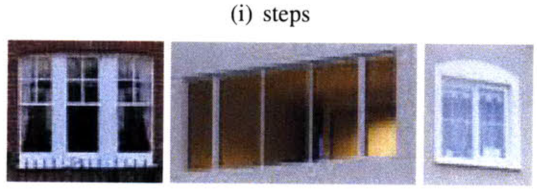
(f) path

(g) roof



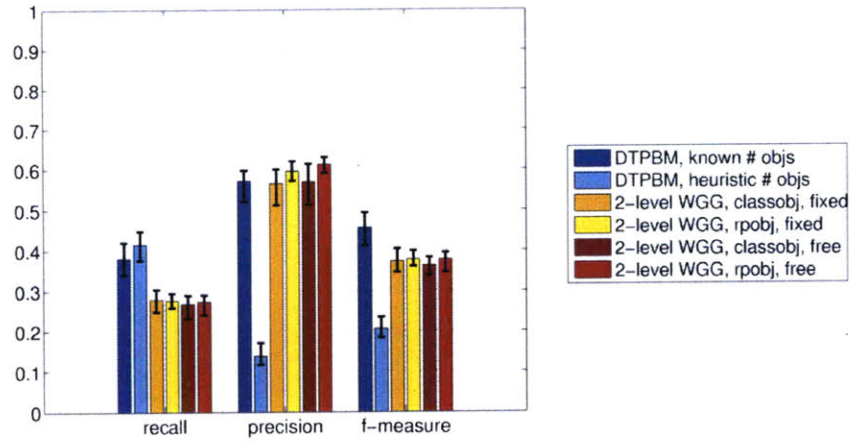
(h) shutter

(i) steps



(j) window

Figure 4-5: Examples of each object class in the house dataset.



 # targets (all models): 741.0

	recall	precision	f-measure	# detections	# correct
DTPBM, known # objs	0.380 [0.341 0.420]	0.571 [0.521 0.598]	0.456 [0.412 0.493]	493.0	280.8
DTPBM, heuristic # objs	0.415 [0.376 0.447]	0.138 [0.117 0.171]	0.205 [0.184 0.235]	2263.6	306.8
hand-built WGG, classobj, fixed	0.254 [0.244 0.264]	0.604 [0.561 0.647]	0.357 [0.353 0.362]	312.0	188.0
hand-built WGG, rpobj, fixed	0.235 [0.221 0.253]	0.659 [0.623 0.705]	0.346 [0.332 0.359]	264.6	174.0
hand-built WGG, classobj, free	0.278 [0.237 0.306]	0.561 [0.547 0.571]	0.371 [0.334 0.397]	365.4	205.0
hand-built WGG, rpobj, free	0.282 [0.256 0.304]	0.608 [0.576 0.637]	0.385 [0.357 0.409]	342.8	208.4
2-level WGG, classobj, fixed	0.278 [0.248 0.304]	0.565 [0.512 0.601]	0.372 [0.346 0.404]	363.8	205.2
2-level WGG, rpobj, fixed	0.276 [0.258 0.294]	0.596 [0.571 0.621]	0.377 [0.360 0.399]	342.6	204.2
2-level WGG, classobj, free	0.267 [0.233 0.289]	0.569 [0.513 0.614]	0.362 [0.338 0.383]	349.0	197.6
2-level WGG, rpobj, free	0.273 [0.241 0.290]	0.612 [0.592 0.631]	0.377 [0.346 0.395]	330.0	201.8

Figure 4-6: Cumulative results for the DTPBM object detector, and four variations of hand-built and learned two-level WGG models, on the small placesetting dataset. (Note that the table includes the DTPBM and both the hand-built and learned WGGs, while the bar graph shows only the DTPBM and learned WGGs.) 'classobj' means class-based object detector features were used, while 'rpobj' means rule-part-based features were used. 'fixed' refers to fixed ground-truth internal geometry, while 'free' refers to free ground-truth internal geometry.

compare them to the DTPBM object detector. For the small placesetting dataset, we also compare learned two-level WGGs to the hand-built structure from last chapter.

We used the same setup for our experiments with learned two-level WGGs on the small placesetting dataset as with the hand-built structure (see Section 3.2.4). In particular, we used exactly the same 5 splits of the dataset into training and test sets (each with 87 images and 80 images, respectively), to allow comparison between the hand-built and learned structures. Again, the perceptron was trained with 10 iterations.

Figure 4-6 shows the performance of the DTPBM object detector, the hand-built WGG structure, and learned two-level WGG models. For both categories of WGG structures, all four combinations of class-based versus rule-part-based object detector features and fixed versus free internal geometry are presented. As before, the results are compared to the two version of the DTPBM object detector. Figure 4-7 breaks out the results for the DTPBM and the learned WGGs by object class.

First, we see that the learned two-level WGG models perform roughly the same as the hand-built structure. This means our structure learning algorithms are finding reasonable object correspondences across images. On the other hand, it is interesting that even a hand-built hierarchical

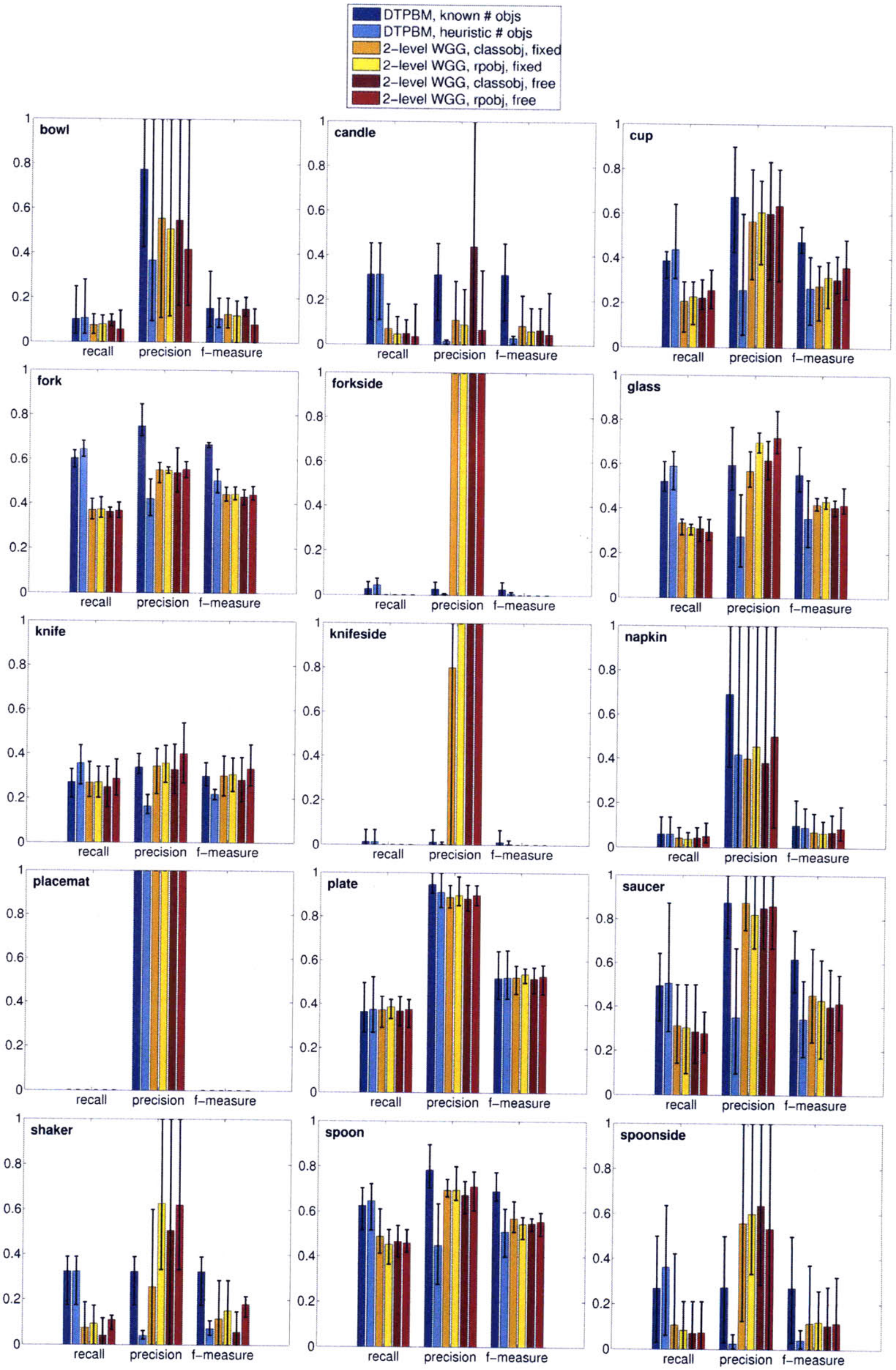
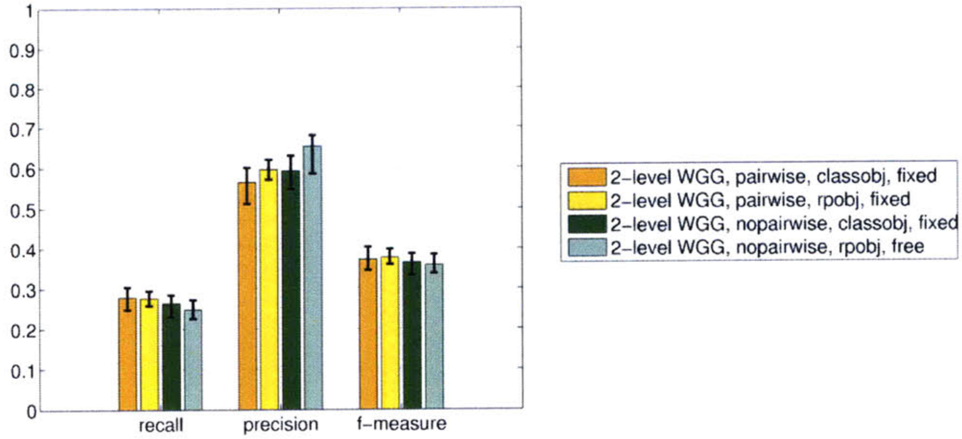


Figure 4-7: Per-class results for the DTPBM object detector and learned two-level WGG models on the small placesetting dataset. For numeric per-class results, see Appendix B (Tables B.4, B.5, B.12, B.13, B.14, and B.15).



 # targets (all models): 741.0

	recall	**precision**	**f-measure**	**# detections**	**# correct**
hand-built, pairwise, classobj, fixed	0.254 [0.244 0.264]	0.604 [0.561 0.647]	0.357 [0.353 0.362]	312.0	188.0
hand-built, pairwise, rpobj, fixed	0.235 [0.221 0.253]	0.659 [0.623 0.705]	0.346 [0.332 0.359]	264.6	174.0
hand-built, nopairwise, classobj, fixed	0.242 [0.218 0.260]	0.623 [0.584 0.643]	0.348 [0.325 0.361]	287.6	178.8
hand-built, nopairwise, rpobj, fixed	0.217 [0.208 0.227]	0.698 [0.627 0.741]	0.331 [0.321 0.348]	231.2	161.0
2-level, pairwise, classobj, fixed	0.278 [0.248 0.304]	0.565 [0.512 0.601]	0.372 [0.346 0.404]	363.8	205.2
2-level, pairwise, rpobj, fixed	0.276 [0.258 0.294]	0.596 [0.571 0.621]	0.377 [0.360 0.399]	342.6	204.2
2-level, nopairwise, classobj, fixed	0.264 [0.231 0.284]	0.593 [0.549 0.631]	0.365 [0.333 0.387]	329.8	195.0
2-level, nopairwise, rpobj, fixed	0.248 [0.226 0.272]	0.654 [0.587 0.681]	0.359 [0.338 0.385]	281.4	183.6

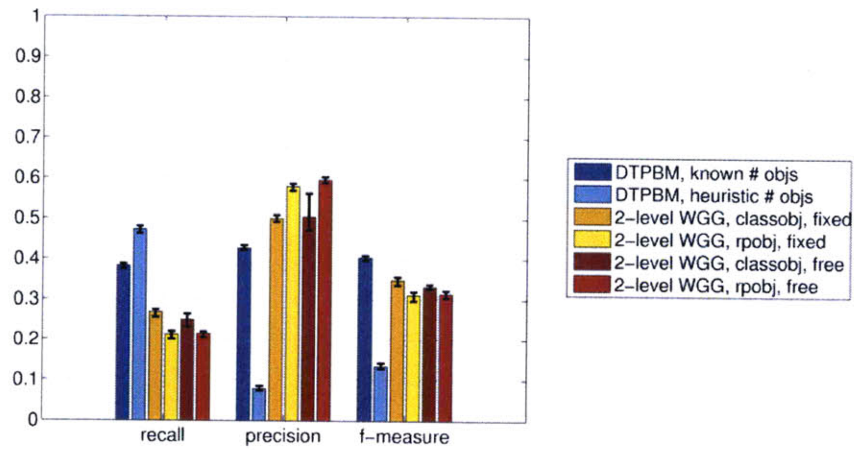
Figure 4-8: Cumulative results for the hand-built and learned two-level WGG models, with and without pairwise features, on the small placesetting dataset. (Note that the table lists both the hand-built and learned WGGs, while the bar graph shows only the learned models.)

structure does not outperform learned two-level models to a significant degree. We will investigate this issue in greater depth next chapter, but these results are already offering a taste of those to come.

Next, notice that free ground-truth internal geometry no longer seems to offer the small advantage over fixed geometry in the learned models that it had in the hand-built structure. This matches what we would expect. During the structure learning algorithm, we intentionally find parent geometry vectors for the internal trees nodes for the training images that represent good coordinate frames. Therefore, using free geometry to find better internal geometry during the perceptron provides no additional benefit. Furthermore, by finding good parent geometry jointly with learning the grammar structure, we make better choices about which objects should be matched than we might by using fixed bounding boxes for internal nodes.

Now consider the per-class results in Figure 4-7. The WGG models have an advantage over the heuristic DTPBM in some of the classes. However, for some other classes, there is no advantage, or the heuristic DTPBM even outperforms the WGGs in f-measure. These include classes with a lot of training data (plate, fork) for which the DTPBM is already quite strong; perhaps the WGG cannot hope to improve on the object detector by much. The WGGs also struggle in some classes with relatively little data that are more challenging for the DTPBM (forkside, knifeside, napkin, placemat). In these cases, the object detector is giving a noisy signal, while at the same time there is very high variance in the spatial arrangements and very little data with which to learn these patterns.

We can also explore the role played by the pairwise tree structure features in the learned structures. Figure 4-8 compares the performance of learned two-level WGG models with and without



 # targets (all models): 3899.0

	recall	precision	f-measure	# detections	# correct
DTPBM, known # objs	0.387 [0.381 0.390]	0.432 [0.428 0.437]	0.408 [0.403 0.412]	3494.7	1509.3
DTPBM, heuristic # objs	0.475 [0.465 0.482]	0.079 [0.075 0.085]	0.135 [0.130 0.144]	23666.3	1850.3
2-level WGG, classobj, fixed	0.267 [0.255 0.274]	0.499 [0.492 0.508]	0.348 [0.336 0.356]	2086.7	1040.7
2-level WGG, rpobj, fixed	0.212 [0.201 0.220]	0.579 [0.569 0.587]	0.311 [0.297 0.320]	1429.0	828.0
2-level WGG, classobj, free	0.249 [0.231 0.264]	0.504 [0.471 0.562]	0.332 [0.327 0.338]	1946.0	970.3
2-level WGG, rpobj, free	0.214 [0.205 0.219]	0.595 [0.588 0.603]	0.314 [0.305 0.322]	1400.3	832.7

Figure 4-9: Cumulative results for the DTPBM object detector and four variations of learned two-level WGG models, on the big placesetting dataset. 'classobj' means class-based object detector features were used, while 'rpobj' means rule-part-based features were used. 'fixed' refers to fixed ground-truth internal geometry, while 'free' refers to free ground-truth internal geometry.

pairwise features (each with class-based and rule-part-based object detector features), and also includes the same variations of the hand-built WGG structure.

Our intuition is that the pairwise features should be more important in two-level structures that do not have composite classes, since they serve to capture co-occurrences among objects. But the results on the small placesetting dataset only weakly confirm this hypothesis. However, it is important to remember that the small placesetting dataset is, as its name suggests, small. Rather than read too much into these results, we will evaluate this question again with the big placesetting dataset.

4.2.4 Results on the Big Placesetting Dataset

We turn now to the large datasets introduced in this chapter. For each set of experiments with the big placesetting dataset, we randomly selected a training set of 400 images and a test set of 400 images. Again, we trained and tested each model on exactly the same images, computing per-class and cumulative precision, recall, and f-measure on the test set. Because the training sets for this dataset are so much larger than those for its small counterpart, the perceptron needs fewer iterations over the training data. All experiments here use 5 perceptron iterations, rather than 10.

Similarly, the larger test sets for this dataset result in less variance in performance across different training and test sets. Thus, we performed the same experiments for 3 different random choices of training and test sets, rather than 5. All results we present here are averaged over these 3 runs, with the minimum and maximum scores across runs measuring the variability in performance.

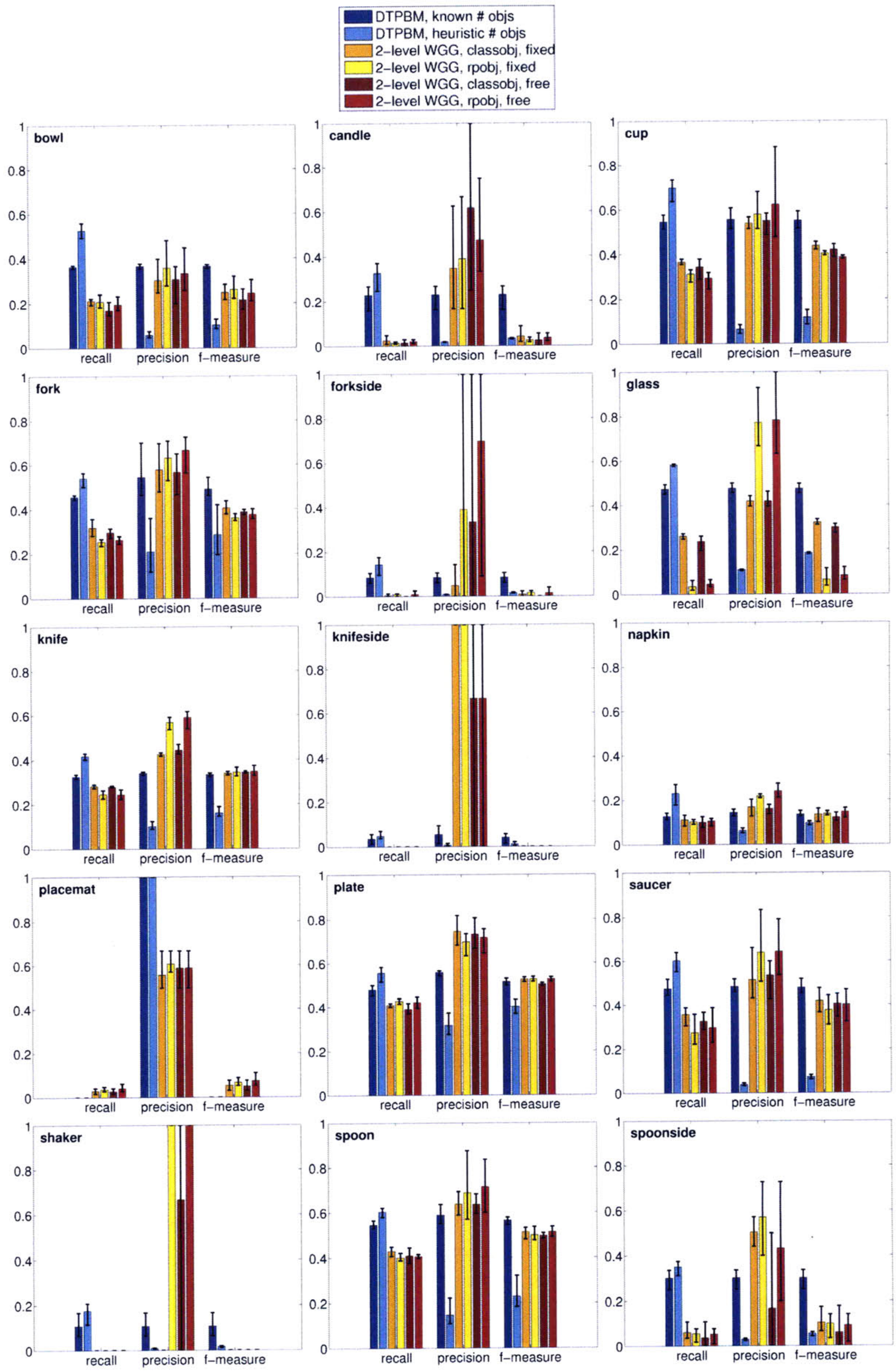
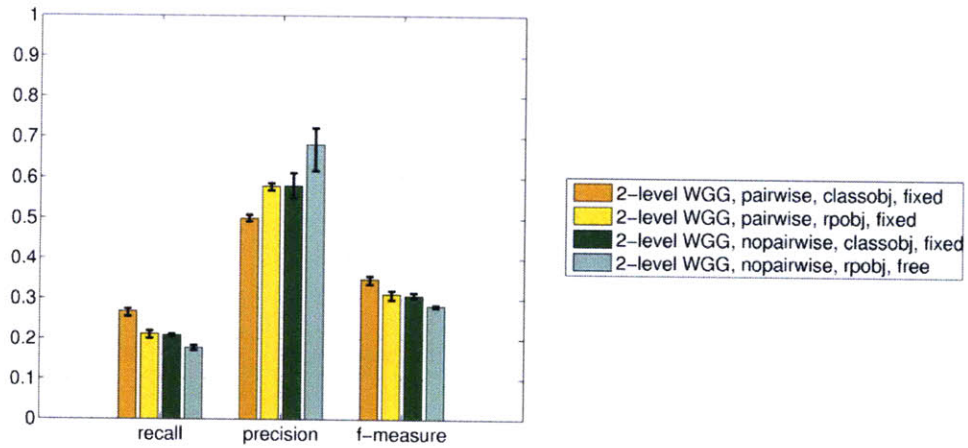


Figure 4-10: Per-class results for the DTPBM object detector and learned two-level WGG models on the big placesetting dataset. For numeric per-class results, see Appendix B (Tables B.30, B.31, B.32, B.33, B.34, and B.35).



 # targets (all models): 3899.0

	recall	**precision**	**f-measure**	**# detections**	**# correct**
pairwise, classobj, fixed	0.267 [0.255 0.274]	0.499 [0.492 0.508]	0.348 [0.336 0.356]	2086.7	1040.7
pairwise, rpobj, fixed	0.212 [0.201 0.220]	0.579 [0.569 0.587]	0.311 [0.297 0.320]	1429.0	828.0
nopairwise, classobj, fixed	0.209 [0.206 0.212]	0.580 [0.550 0.612]	0.307 [0.303 0.315]	1405.7	814.0
nopairwise, rpobj, fixed	0.178 [0.172 0.184]	0.682 [0.618 0.722]	0.282 [0.278 0.284]	1023.3	693.7

Figure 4-11: Cumulative results for learned two-level WGG models, with and without pairwise features, on the big placesetting dataset.

Figure 4-9 shows the performance of the DTPBM object detector and learned two-level WGG models on the big placesetting dataset. As before, we present all four combinations of class-based versus rule-part-based object detector features and fixed versus free internal geometry, and compare the results to the two versions of the DTPBM. Figure 4-10 shows the results by object class.

The general profile of the results has not changed substantially. The two versions of the object detector continue to differ dramatically, while the WGG models outperform the heuristic DTPBM in f-measure by a wide margin. Again, this advantage is due to the much higher precision of the WGG models. Notice that the heuristic DTPBM makes over ten times as many detections as the WGG models, but only correctly detects less than twice as many target objects. The difference in average f-measure between the DTPBM with ground-truth object numbers and the WGG models seems to have shrunk slightly, which is good. In general, the scores for all the models are lower than in the small dataset, despite the large increase in training data, confirming that this dataset is more challenging than its small counterpart. (See Figure 4-17 at the end of the chapter for representative detections from a WGG model on this dataset.)

In the larger dataset, the advantage of the class-based object detector features over their rule-part-based versions is more apparent. This is likely because the scenes in the big dataset are more complex than in the small dataset, so each object class appears in a greater number of rule parts. (Put another way, the maximum number of objects of each class ever seen in a training image is higher.) This means that the training data for each object class is more fragmented across the rule parts, so the parameter learning problem becomes more challenging. As the data becomes more complex, it becomes more important to tie the object detector features across all rule parts with the same object class, to avoid overfitting.

As with the learned WGGs in the small placesettings, the difference between fixed and free internal ground-truth geometry is relatively low; if anything, it seems to hurt slightly. Again, this suggests that the structure learning does a better job of finding geometry vectors for the internal tree

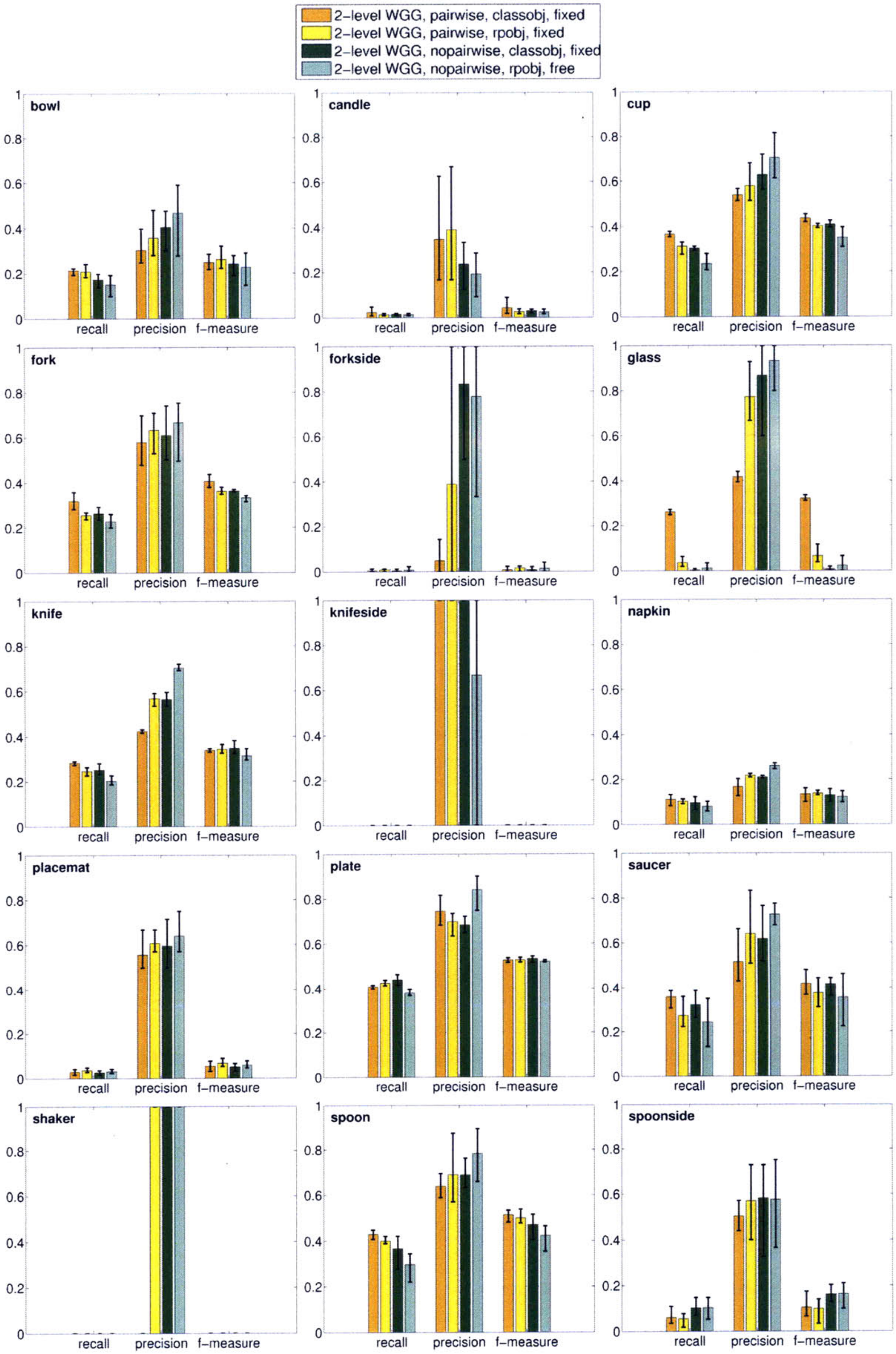
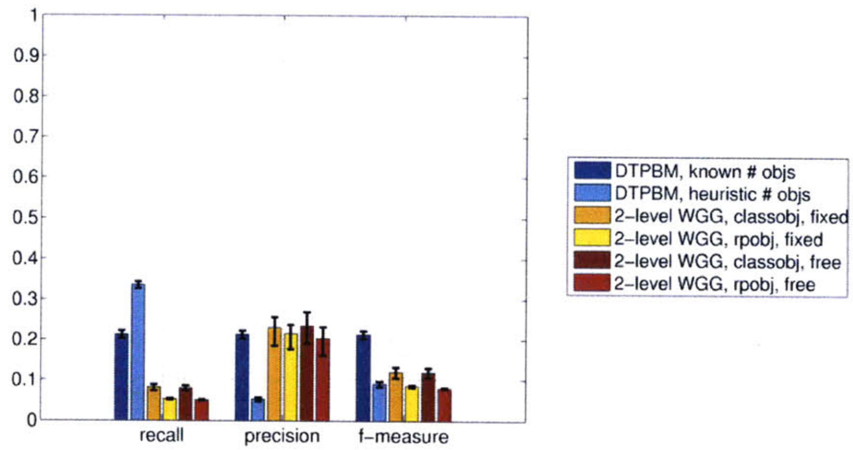


Figure 4-12: Per-class results for learned two-level WGG models with and without pairwise features, on the big placesetting dataset. For numeric per-class results, see Appendix B (Tables B.32, B.33, B.36, and B.37).



 # targets (all models): 6558.0

	recall	precision	f-measure	# detections	# correct
DTPBM, known # objs	0.214 [0.206 0.223]	0.216 [0.206 0.224]	0.215 [0.206 0.223]	6521.3	1406.0
DTPBM, heuristic # objs	0.337 [0.329 0.346]	0.053 [0.049 0.058]	0.092 [0.085 0.098]	41846.3	2210.0
2-level WGG, classobj, fixed	0.082 [0.073 0.089]	0.230 [0.185 0.256]	0.120 [0.105 0.132]	2347.3	535.3
2-level WGG, rpobj, fixed	0.053 [0.051 0.054]	0.215 [0.177 0.237]	0.085 [0.082 0.088]	1639.3	347.0
2-level WGG, classobj, free	0.080 [0.074 0.086]	0.234 [0.191 0.269]	0.119 [0.107 0.130]	2269.3	526.0
2-level WGG, rpobj, free	0.050 [0.049 0.052]	0.203 [0.161 0.231]	0.080 [0.078 0.082]	1665.3	330.0

Figure 4-13: Cumulative results for the DTPBM object detector and four variations of learned two-level WGG models, on the house dataset. 'classobj' means class-based object detector features were used, while 'rpobj' means rule-part-based features were used. 'fixed' refers to fixed ground-truth internal geometry, while 'free' refers to free ground-truth internal geometry.

nodes for the training images than parsing the "ground truth" trees during the perceptron.

Turning to the per-class results in Figure 4-10, we see that the WGG models' advantage over the heuristic DTPBM is more consistent across object classes than in the small placesetting dataset. The exceptions occur in classes where the object detector output is so poor that there is very little signal to build upon (candle, forkside, knifeside, shaker).

Finally, we again remove pairwise tree structure features to test their importance. Figure 4-11 compares the performance of the learned WGG models with and without pairwise features on the big placesetting dataset, while Figure 4-12 shows the per-class results.

In the larger dataset, we see a more substantial improvement from using pairwise features. It is possible that the small dataset did not offer enough training data to sufficiently learn the pairwise features, again leading to overfitting. Looking at the per-class results, the classes for which the pairwise features help the most are those which tend to appear in groups (cup, fork, glass, spoon). The glass class gets by far the most dramatic benefit from the pairwise features. On the other hand, it is somewhat surprising that the saucer class does not get a greater advantage, especially since the cup class does show a small improvement, and they tend to co-occur.

4.2.5 Results on the House Dataset

For the house dataset, we used the same experimental setup as with the big placesetting dataset, with randomly chosen training and test set of 400 images each, 5 perceptron iterations, and results averaged over 3 different choices of training and test sets.

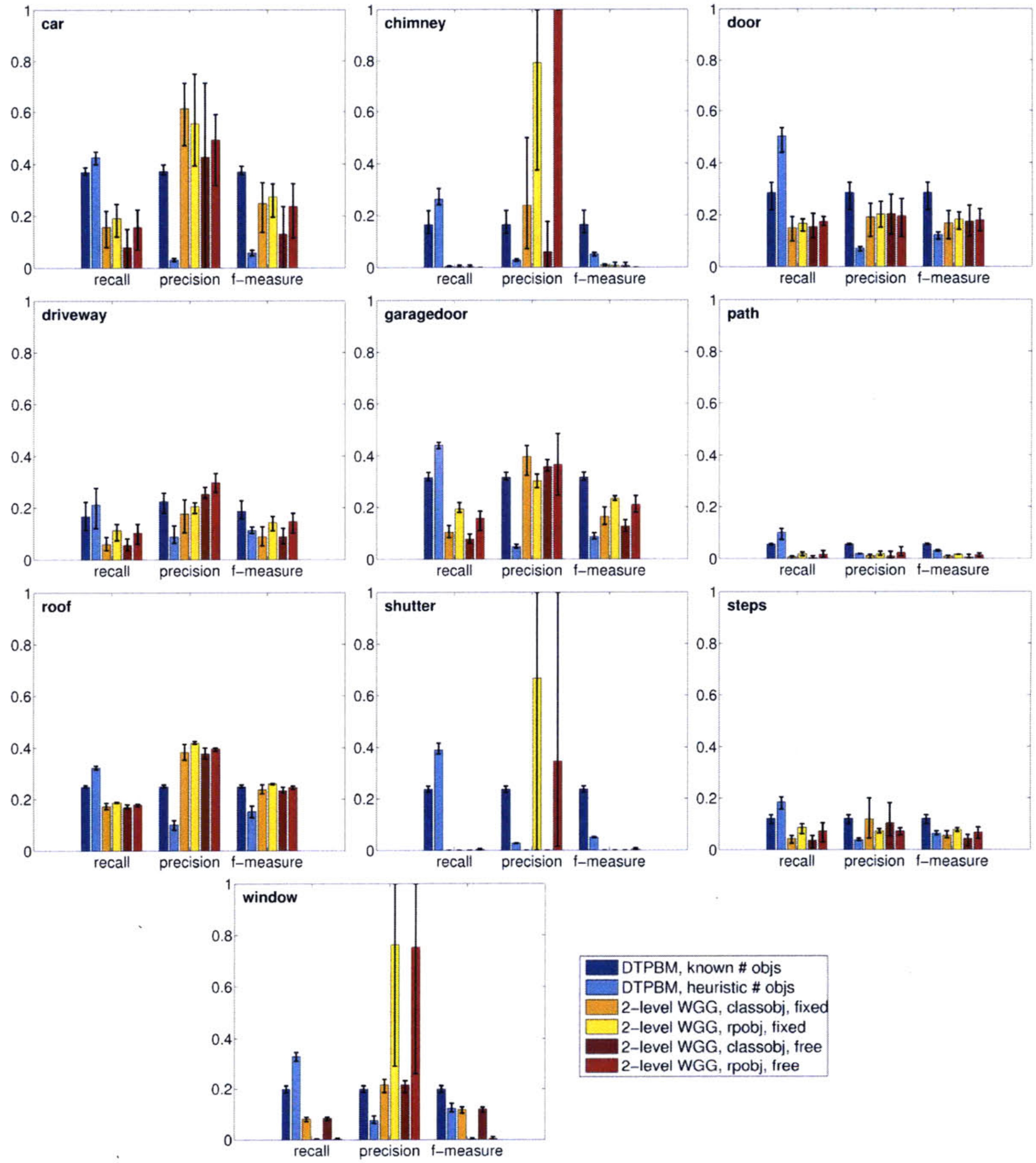


Figure 4-14: Per-class results for the DTPBM object detector and learned two-level WGG models on the house dataset. For numeric per-class results, see Appendix B (Tables B.50, B.51, B.52, B.53, B.54, and B.55).

Figure 4-13 shows the performance of the DTPBM object detector and learned two-level WGG models on the big placesetting dataset. As before, we present all four combinations of class-based versus rule-part-based object detector features and fixed versus free internal geometry, and compare the results to the two version of the DTPBM. Figure 4-14 breaks out the results by object class.

Right away, we notice that the scores for the all models on this dataset are significantly lower than they were on the placesetting datasets. Even the ground-truth-numbers DTPBM only achieves an average cumulative f-measure of 0.215, versus an average of 0.408 on the large placesetting dataset. As we have said, the house dataset is by far the most challenging of the three we consider.

To understand why, look back at Table 4.2, where we listed the object classes and their frequencies. Notice that almost half of the 13,801 objects in the dataset are windows, and another 13% are shutters. These objects are often tiny; in the subsampled images used by the object detector (with a subsampling rate of 8), some of these objects are only a few pixels wide. Thus, if the detected object is not centered on almost exactly the right *pixel*, the overlap score is not high enough to be considered correct. At least shutters tend to be very uniform in appearance, whereas windows vary widely in shape and aspect ratio.

Also, several of the larger object classes, such as path, driveway, and steps, have widely varying aspect ratios, so they are not as easily detected with a sliding-window approach. The driveway, roof, and path classes do not have as many local distinguishing features; they tend to be somewhat textureless. And the chimney class varies widely in appearance—chimneys can be tall and thin or short and squat or tiny and tubular, and made of brick or plaster or metal.

These difficulties manifest themselves in the per-class results in Figure 4-14. For many of the classes, not only does the heuristic DTPBM (light blue) perform poorly, but the ground-truth-numbers DTPBM (dark blue) does not do that much better. But there are several notable exceptions. One is the shutter class; this is because houses tend to either have many shutters, or none. So when the DTPBM is given the ground-truth number of shutters in a test image, this is hugely informative. The detector then does not produce false alarms in the many images which have no shutters at all. The same argument can be made for the car and garagedoor classes. Most houses have at most one of each of these classes, but many houses do not have them at all; thus being told whether the current test image does or does not have a car or garagedoor is very useful.

Overall WGG Performance

Now we consider the performance of the WGG models. They make even fewer detections, relative to the number of target objects, than in the placesetting datasets. This results in low recall, although the precision remains relatively high (at least compared to the heuristic DTPBM). In general, the versions of the WGGs with class-based object detector features show a slight but significant improvement in cumulative f-measure over the heuristic DTPBM, while the rule-part-based versions are slightly worse. (See Figure 4-18 at the end of the chapter for representative detections from a WGG model on this dataset.)

Looking at the per-class results, we see that actually the WGG models' advantage varies quite a bit for different object classes. For some classes (car, door, driveway, garagedoor, roof), some or all of the WGG models outperform the heuristic DTPBM in f-measure by fair margin. These are cases in which the object detector performs well enough to provide sufficient image-based information, and the co-occurrence and geometry models are informative enough to improve on the detector.

Then, in some classes for which the DTPBM performs very poorly (path, steps), the WGG has too little signal upon which to build, and performs equally poorly. As we have seen before, when the detector essentially provides noise, no amount of background knowledge of expected scene arrangements can overcome the lack of image-based information for the current scene.

Finally, the chimney, shutter, and window classes are cases in which the detector does only somewhat poorly, but the WGGs perform the same or even worse than the detector. In these cases, it seems that the WGG models cannot sufficiently model the spatial arrangements. While chimneys do occur on roofs, they can appear almost anywhere in that region, and their sizes are usually tiny compared to the variance in expected location. Thus, the variance in geometry is too high to overcome the noise in the object detector, so the model has trouble predicting whether a chimney is present in the image at all. Since chimneys are relatively uncommon, the perceptron is penalized early and often for predicting nonexistent chimneys. So eventually it adjusts the threshold weights to detect them very rarely or not at all, rather than continue to get them wrong.

For windows, there is actually relatively little spatial information to capture; windows tend to appear uniformly scattered across the front of a house. All we can learn is their expected scale relative to the entire house, and perhaps a broad region of expected location. In this situation, the WGG makes detection decisions based primarily on the object detector output, combined with tree structure weights (acting as thresholds) and the expected number of objects per class encoded in the grammar structure itself. And as expected, the f-measure performance for the WGG models (with class-based object detector scores) is roughly equivalent to the heuristic DTPBM.

The same lack of spatial information is true for shutters, except that we might hope to capture the fact that they occur right next to windows (and usually in pairs). However, because we are only learning a single flat rule for the entire house, the location of a shutter is conditionally independent from that of a window—we cannot model this relationship as tightly as we want. When we learn composite classes, we would hope to capture this pattern better.

Although the WGG models perform reasonably well on roughly half the object classes, this is reflected only somewhat in the cumulative scores. This is because two of the classes on which it performs less well—windows and shutters—account for 60% of the total objects.

Class-Based Versus Rule-Part-Based Object Detector Features

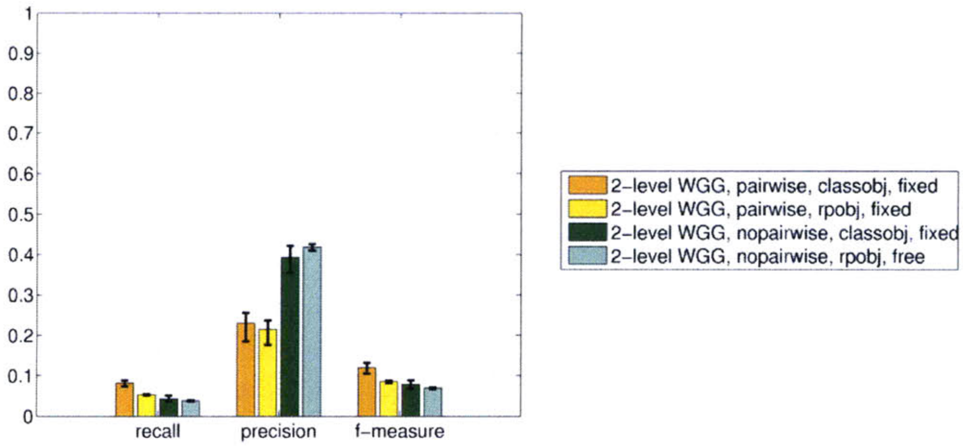
In Figure 4-13, the difference in cumulative f-measure between the class-based object detector features (orange and dark red) and the rule-part-based versions (yellow and bright red) is more marked, with the class-based features appearing to dominate. But again we have to look at the per-class results in Figure 4-14 to see the full story. For the window class, the rule-part-based features indeed get no traction at all, while the class-based features perform the same as the heuristic DTPBM (as we discussed). However, for almost every other class, the rule-part-based features outperform the class-based features, or at least do no worse. In some cases, the rule-part-based features do significantly better.

Remember the qualitative difference in expressive power between the two types of object detector features. The class-based features force the model to put the same amount of weight on the object detector output, regardless of the object's role in the scene. And the value of these weights affect the balance between the object detector scores and the geometry features in the overall tree score. The balance can still be different for different rule parts, because the geometry weights are always rule-part-specific. But there is less freedom.

The rule-part-based features, in contrast, allow different weights on the object detector scores for different rule parts. This means the perceptron will find a different balance between the object detector scores and the geometry features for different rule parts. However, the increased expressive power comes with an increased risk of overfitting.

Thus, for object classes which appear at most only a once or twice per image (if at all), the rule-part-based features provide additional expressive power without too much cost in parameter learning difficulty, because the training data is not fragmented by much across rule parts. This is true of the car, driveway, garagedoor, and steps classes, which usually occur either once or not at all in an image. These are also the classes where the rule-part-based features have a significant advantage over the class-based features.

For the window class, however, the reverse is true. Most images have many windows, so the data is very fragmented. Furthermore, as we discussed above, there is very little advantage to the additional expressive power, because the geometry models are not helping much. With the class-



 # targets (all models): 6558.0

	recall	**precision**	**f-measure**	**# detections**	**# correct**
pairwise, classobj, fixed	0.082 [0.073 0.089]	0.230 [0.185 0.256]	0.120 [0.105 0.132]	2347.3	535.3
pairwise, rpobj, fixed	0.053 [0.051 0.054]	0.215 [0.177 0.237]	0.085 [0.082 0.088]	1639.3	347.0
nopairwise, classobj, fixed	0.043 [0.038 0.051]	0.392 [0.355 0.421]	0.078 [0.069 0.089]	730.7	283.0
nopairwise, rpobj, fixed	0.038 [0.037 0.039]	0.418 [0.410 0.426]	0.069 [0.067 0.071]	593.7	247.7

Figure 4-15: Cumulative results for learned two-level WGG models, with and without pairwise features, on the house dataset.

based features, the perceptron learns to trust the object detector to a greater degree, by increasing the single weight on the window object detector score. But with the rule-part-based features, the perceptron cannot learn this rule sufficiently. It might detect a window correctly, but using the wrong rule part, according to the "ground truth" tree found during structure learning. So the object detector weight on the wrong rule part is decremented by the object detector score, while the weight on the right rule part is incremented. As the perceptron continues, these weights fluctuate, sometimes even becoming negative temporarily (meaning the model thinks the object detector score is *inversely* correlated to object presence). So eventually the perceptron stops trusting the window rule parts, raising the thresholds to the point it detects windows very rarely or not at all. Compare the row for the window class in Table B.52 versus Table B.53. In the former, the WGG with class-based features makes an average of 1177 window detections on the test set, getting 253 correct on average. In the latter, the WGG with rule-part-based features makes only 12.7 detections on average, over a test set of 400 images.

Thus, it seems that some object classes need the expressive power of rule-part-based object detector features, while others need the safeguard against overfitting that the class-based features provide. To address this issue, one area of future work might be to include both types of object detector features (in addition to other types of image-based features), as we will discuss in Chapter 6.

Pairwise Tree Structure Features Versus None

Figure 4-15 compares the performance of learned two-level WGG models with and without pairwise features, each with class-based and rule-part-based object detector features. Figure 4-16 shows the per-class results.

In this dataset, we again see a greater improvement from pairwise features than in the small dataset. And again, the classes in which they help the most seem to be those in which tend to occur

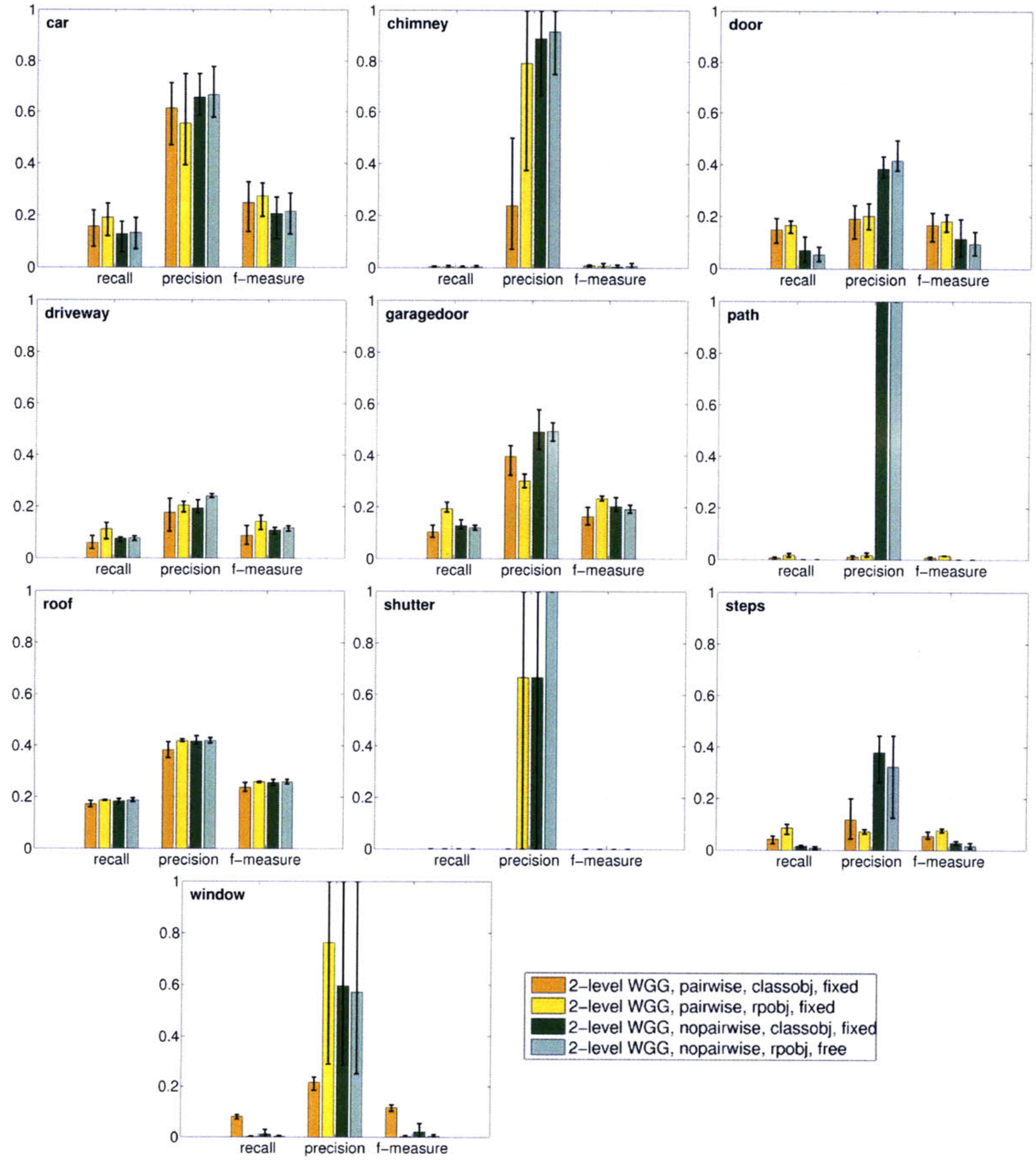


Figure 4-16: Per-class results for learned two-level WGG models with and without pairwise features, on the house dataset. For numeric per-class results, see Appendix B (Tables B.52, B.53, B.56, and B.57).

in pairs or groups. For example, the car, driveway, and garagedoor classes tend to occur together, while roofs occur often, regardless of what other objects are present. For the window class, the pairwise features seem to be crucial; and since windows are so common, this explains much of the overall advantage in cumulative scores.

Pairwise tree structure features are only one approach to modeling co-occurrence patterns. In the next chapter, we will explore the problem of learning composite classes, in which rules of the grammar structure explicitly capture these patterns.

Figure 4-17: Representative detections from a learned two-level WGG (pairwise, classobj, fixed) on the big placesetting dataset.

Figure 4-18: Representative detections from a learned two-level WGG (pairwise, classobj, fixed) on the house dataset.

Chapter 5

Three-Level Structure Learning in WGGs

At the beginning of the last chapter, we presented the formal structure learning problem in WGGs. We are given:

- A set of object classes B and scene classes S.
- A set of training images I_i, for $i = 1, \ldots, m$, in which the ith image is annotated with:
  - a scene class $c_i \in S$, and
  - a set of m_i objects, in which the jth object has object class $c_{i,j} \in B$ and bounding box $b_{i,j}$, for $j = 1, \ldots, m_i$.

The goal is to find:

- A small set of good composite classes C.
- A set of rules R—one for each learned composite class, and one for each scene class in S.
- A set of scene trees T_i for the training images, with class and rule part labels corresponding to C and R, and geometry vectors in the nodes.

In the last chapter, we restricted the problem to learning only two-level WGG structures. In this chapter, we extend our algorithms to learn one layer of composite classes C and their rules, in addition to the top-level rules for the scene classes. Thus, the scene trees for the learned WGG structures can have up to three levels.

As an example, for the simple placesettings shown in Figure 5-1, we might learn this set of composite classes and rules:

$$C = \{\text{placesetting}, \text{C1}, \text{C2}, \text{C3}\}$$
$$R = \left\{ \begin{array}{c} \text{placesetting} \rightarrow \text{C1}_1\ \text{C2}_2\ \text{C3}_3\ \text{napkin}_4 \\ \text{C1} \rightarrow \text{fork}_5\ \text{fork}_6 \\ \text{C2} \rightarrow \text{plate}_7\ \text{bowl}_8 \\ \text{C3} \rightarrow \text{knife}_9\ \text{spoon}_{10} \end{array} \right\}$$

with these scene trees:

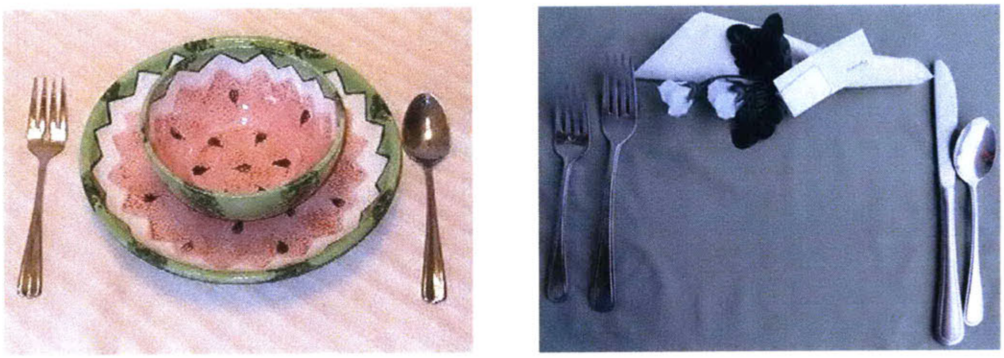
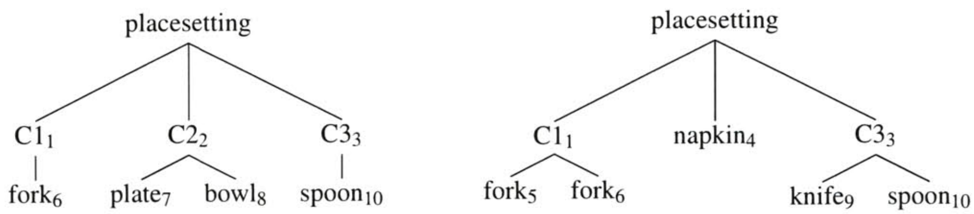


Figure 5-1: Two images of simple placesettings.



We also present experiments in this chapter to explore the effect of adding this additional layer of hierarchy.

5.1 Learning Composite Classes

Before addressing the problem of learning composite classes, it is worth discussing again why they might be desirable. Our notion of a composite class is a group of objects that:

1. have low structural variance (they often co-occur), and

2. have low geometry variance with respect to one another.

We hope that learning composite classes with these properties will improve the model's ability to capture patterns in the data. This could happen for several reasons.

First, composite classes can capture stronger patterns in object co-occurrence than using pairwise weights among rule parts. For example, if a saucer never appears without a cup, learning a composite class for the cup-saucer combination allows the weight learning to focus on modeling how often the *pair* occurs (and with what other objects), rather than on realizing that the presence of the cup and saucer themselves are correlated.

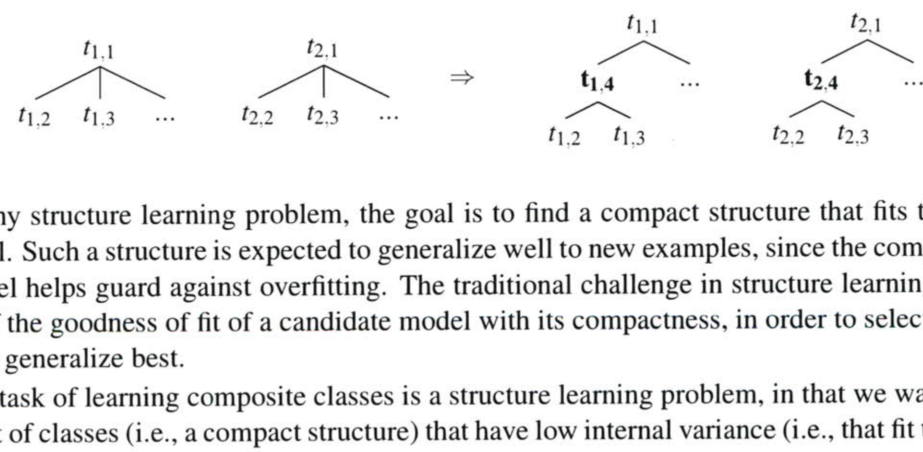
Second, composite classes might help lessen the effects of the model's conditional independence assumption in geometry—the assumption that, for a fixed parent location and scale, the best location and scale for each child subtree is found independently of all other children. By introducing a composite class for a subset of the objects in a scene class, we define a local parent coordinate frame that captures the fact that these objects vary less in geometry with respect to one another than they do to the rest of the objects in the scene. This may make the model better able to detect new objects, in a similar arrangement, at test time. In the cup and saucer example, the location of the saucer is actually heavily constrained, once the cup's location is known (see Figure 5-2). This relationship might be better captured using a local coordinate frame.

Figure 5-2: Cups and saucers often co-occur, and vary less in location and scale with respect to one another than with respect to other objects in the scene.

This interpretation of a composite class as a low-variance local coordinate frame leads to a more specific definition of what it means for a composite class to have low variance in geometry. In particular, we can state the following principle:

> **The variance in geometry for child nodes should be significantly lower with respect to the geometry of a composite parent than with respect to the root geometry.**

For example, in the two trees drawn below, introducing the composite parents as shown on the right should lead to a reduction in variance in the relative geometry vectors across corresponding child nodes in the two trees.



In any structure learning problem, the goal is to find a compact structure that fits the training data well. Such a structure is expected to generalize well to new examples, since the compactness of the model helps guard against overfitting. The traditional challenge in structure learning is how to trade off the goodness of fit of a candidate model with its compactness, in order to select the model that will generalize best.

The task of learning composite classes is a structure learning problem, in that we want to find a small set of classes (i.e., a compact structure) that have low internal variance (i.e., that fit the training data well). In our context, however, the principles we have described suggest a natural approach to trading off the quality of a set of classes with their compactness. Specifically, we can define thresholds on the structural and relative geometry variance of a class—a definition of what makes a class a "good enough" fit. Then, we use a clustering-based algorithm to find the most compact set of classes that are "good enough", where a compact set of classes is one in which each class covers a relative large number of object instances.

The algorithm has two phases. In Phase 1, described in Section 5.1.1, we use iterative agglomerative clustering to find low-variance composite classes. In Phase 2, described in Section 5.1.2, we learn the top-level scene class rules, with rule parts corresponding to both primitive object classes and learned composite classes. Finally, Section 5.1.3 describes an optional step, in which single objects may be assigned to existing composite classes.

5.1.1 Phase 1: Learning Composite Classes

The goal of Phase 1 is to find a small set of good composite classes, and the subtrees for the groups of objects covered by those classes. As in the last chapter, our approach is based on clustering. A cluster is a tuple $\langle K, \mathcal{T} \rangle$, where:

- the template K is a set of RPIDs k and their class labels $\text{class}(k)$, as in the last chapter; and
- $\mathcal{T}$ is a set of two-level *subtrees*, where for each tree $T \in \mathcal{T}$, each child node $t \in T$ has RPID label $\text{rpid}(t) \in K$. (Again, we will use $t \in T$ as shorthand for $t \in \text{children}(\text{root}(T))$.)

In the context of learning composite classes, the members of each cluster are subtrees of full scene trees. There will be multiple subtrees for each training image, and the leaf nodes of a subtree may correspond to only a subset of the objects in a training image.

During the course of Phase 1, we maintain a set of clusters Γ^*, which correspond to the composite classes that have been learned so far. Note that the sets of subtrees for the clusters in Γ^* must be mutually exclusive and non-overlapping. Put another way, each object instance in the training images must appear in at most one of the subtrees in the clusters in Γ^*. Otherwise, we will not be able to produce a set of consistent scene trees by merging the subtrees at the end of the learning process, because some leaf nodes will have more than one parent.

In addition to the clusters Γ^*, we also maintain a set of subtrees $\mathcal{T}^*$ that includes the subtrees for all clusters in Γ^*, as well as a single-node subtree for each object in each training image that has not been included in a cluster. At the beginning of Phase 1, $\mathcal{T}^*$ consists of a single-node subtree for each object in each training image, and Γ^* is empty.

We then iterate three steps. On Step 1, we use clustering to propose candidate low-variance, but possibly incompatible, composite classes. On Step 2, we choose the best set of mutually-consistent classes from those proposed in Step 1, learning a rule template for each. On Step 3, we update the subtrees for the training images using the clusters chosen in Step 2. We then return to Step 1. This continues until no new classes can be created.

In this section, we will discuss each of these steps in detail, and then present the full algorithm. But first, we describe how we compute the structural and geometry variance for composite classes, and how we apply thresholds to obtain low-variance classes.

Structural and Geometry Variance for Composite Classes

In Section 4.1.2, we described a way to compute the normalized variance in relative geometry of a cluster, while allowing the trees within the cluster to have different structure. We will use the same method to compute the relative geometry variance of a composite class cluster in this chapter.

However, we want to find clusters that are low in both relative geometry variance and structural variance. We would like to encourage a cluster to have subtrees with similar structure. Put another way, each subtree in the cluster should contain as many of the RPIDs in the cluster template as possible. Thus, we use the following simple definition of the structural variance of a set of subtrees $\mathcal{T}$, under a template K:

$$\text{var}_{\text{struct}}(\mathcal{T}, K) = \frac{1}{|K|} \sum_{k \in K} \left(\frac{n_k}{|\mathcal{T}|} - 1 \right)^2 \tag{5.1}$$

where n_k is the number of subtrees in $\mathcal{T}$ with a child node with RPID label k. (Each subtree can have at most one child labeled with k.) The quantity $n_k / |\mathcal{T}|$ is the proportion of trees in which k appears. So we can think of this as a measure of the variance across RPIDs in the template, in which the "mean" is one, because that represents the state in which all RPIDs are present in all subtrees.

To perform clustering, we need a single scalar variance value for each cluster. So we take a simple linear combination of the relative geometry variance and the structural variance:

$$\mathrm{var}(\mathcal{T},K) = \beta\,\mathrm{var}_{\mathrm{geom}}(\mathcal{T},K) + (1-\beta)\,\mathrm{var}_{\mathrm{struct}}(\mathcal{T},K) \tag{5.2}$$

where $\mathrm{var}_{\mathrm{geom}}(\mathcal{T},K)$ is the relative geometry variance as computed in Equation 4.4, and $\beta \in [0,1]$ is a constant trade-off factor. For all experiments in this thesis, we use $\beta = 0.5$.

Thresholding for Low-Variance Classes

Now we have a way to compare two clusters, but we also need a more absolute metric. How low-variance is low enough to make a composite class good? As we discussed at the beginning of this chapter, we would like to compare the following two values:

- the variance in relative geometry of the child nodes in a cluster with respect to their composite parents; and
- the variance in relative geometry of those child nodes, but with respect to the roots of the full two-level trees for each image (i.e., their parents, if no composite parents were introduced).

We only want to introduce composite classes in which the variance with respect to the new composite parents is much lower than with respect to the original root.

To do this, we assume that each node t in a subtree stores an additional geometry vector $\mathrm{geom}_{\mathrm{abs,root}}(t)$. This is the absolute geometry vector produced from the bounding box of *all* labeled bounding boxes in the training image for node t. In other words, it is the geometry of the root node of what would have been the initial two-level tree produced for t's training image in the last chapter. (So this vector is identical for all nodes from the same image.)

The node t also stores a geometry vector $\mathrm{geom}_{\mathrm{rel,root}}(t)$, which is the node's own absolute geometry, but relative to $\mathrm{geom}_{\mathrm{abs,root}}(t)$. It is computed exactly as in Equation 4.1, but using $\mathrm{geom}_{\mathrm{abs,root}}(t)$ instead of $\mathrm{geom}_{\mathrm{abs}}(\mathrm{parent}(t))$ as the "parent" geometry.

Then, for a fixed assignment of RPIDs to a set of subtrees $\mathcal{T}$, we can compute, for each RPID k, the normalized variance in the $\mathrm{geom}_{\mathrm{rel,root}}(t)$ vectors across all nodes assigned the label k:

$$\mathrm{var}_{\mathrm{geom,root}}(\mathcal{T},k) = \frac{1}{n_k} \sum_{T\in\mathcal{T}} \sum_{\substack{t\in T \text{ s.t.} \\ \mathrm{rpid}(t)=k}} \left(\left\| \mathrm{geom}_{\mathrm{rel,root}}(t) - \mu_{\mathrm{geom,root},k} \right\|_2^* \right)^2 \tag{5.3}$$

where n_k is the number of child nodes across all trees with RPID label k, $\|\mathbf{x}\|_2^*$ is the normalized L^2 norm of a vector $\mathbf{x}$ (Equation 4.3), and

$$\mu_{\mathrm{geom,root},k} = \frac{1}{n_k} \sum_{T\in\mathcal{T}} \sum_{\substack{t\in T \text{ s.t.} \\ \mathrm{rpid}(t)=k}} \mathrm{geom}_{\mathrm{rel,root}}(t)$$

is the mean relative-to-root geometry vector for RPID k. Again, this is the equivalent of how we computed the relative geometry variance $\mathrm{var}_{\mathrm{geom}}(\mathcal{T},k)$, but using the $\mathrm{geom}_{\mathrm{rel,root}}(t)$ vectors rather than $\mathrm{geom}_{\mathrm{rel}}(t)$. (It is also computed for a single RPID k at a time.)

Now, for each RPID k in the rule for a new composite class, it seems natural to simply compare $\mathrm{var}_{\mathrm{geom}}(\mathcal{T},k)$ and $\mathrm{var}_{\mathrm{geom,root}}(\mathcal{T},k)$ to see how much reduction in variance has been provided by introducing the new composite parent nodes. However, this is not quite correct, because the x and

y values in the geometry vectors used to compute these variances are not in the same coordinate frame—they have been scaled by different parent scale factors.

So we need to define yet one more type geometry vector, $\text{geom}_{\text{rel,rootsc}}(t)$, for a node t. This vector is a modified form of the original relative geometry vector $\text{geom}_{\text{rel}}(t)$, but computed such that the scale value of t's root absolute geometry vector, rather than t's parent's absolute geometry, is used to scale the x and y values. Specifically, if we have the following absolute geometry vectors for t, t's parent, and t's hypothetical root node:

$$\begin{aligned} \text{geom}_{\text{abs}}(t) &= \langle x_{\text{c}}, y_{\text{c}}, \log s_{\text{c}} \rangle \\ \text{geom}_{\text{abs}}(\text{parent}(t)) &= \langle x_{\text{p}}, y_{\text{p}}, \log s_{\text{p}} \rangle \\ \text{geom}_{\text{abs,root}}(t) &= \langle x_{\text{r}}, y_{\text{r}}, \log s_{\text{r}} \rangle \end{aligned}$$

then

$$\text{geom}_{\text{rel,rootsc}}(t) = \left\langle \frac{s_{\text{r}}(x_{\text{c}} - x_{\text{p}})}{\kappa_{\text{geom,x}}}, \frac{s_{\text{r}}(y_{\text{c}} - y_{\text{p}})}{\kappa_{\text{geom,y}}}, \frac{\log s_{\text{c}} - \log s_{\text{p}}}{\kappa_{\text{geom,s}}} \right\rangle \tag{5.4}$$

The only difference between this expression and the definition of $\text{geom}_{\text{rel}}(t)$ in Equation 4.1 is that we have replaced s_{p} in the first two vector elements with s_{r}—so we are scaling the child's x and y values by the root's scale, rather than the parent's.

Now we can compute the variance in the $\text{geom}_{\text{rel,rootsc}}(t)$ vectors, for a specific RPID k, to have a value that is directly comparable to the relative-to-root geometry variance $\text{var}_{\text{geom,root}}(\mathcal{T}, k)$. This is given by:

$$\text{var}_{\text{geom,rootsc}}(\mathcal{T}, k) = \frac{1}{n_k} \sum_{T \in \mathcal{T}} \sum_{\substack{t \in T \text{ s.t.} \\ \text{rpid}(t) = k}} \left(\left\| \text{geom}_{\text{rel,rootsc}}(t) - \mu_{\text{geom,rootsc},k} \right\|_2^* \right)^2 \tag{5.5}$$

where

$$\mu_{\text{geom,rootsc},k} = \frac{1}{n_k} \sum_{T \in \mathcal{T}} \sum_{\substack{t \in T \text{ s.t.} \\ \text{rpid}(t) = k}} \text{geom}_{\text{rel,rootsc}}(t)$$

These expressions are identical to those before, but simply use a different geometry vector.

To decide if a cluster has low enough variance in geometry, we consider each rule part k in the template K, and ask whether the relative-to-parent geometry standard deviation (but using root scale in x and y) is lower than some percentage of the relative-to-root geometry standard deviation:

$$\text{lowvar}_{\text{geom}}(\mathcal{T}, K) = \begin{cases} \text{true} & \text{if } \forall_{k \in K} \left(\text{var}_{\text{geom,rootsc}}(\mathcal{T}, k) \right)^{1/2} \leq \gamma_{\text{geom}} \left(\text{var}_{\text{geom,root}}(\mathcal{T}, k) \right)^{1/2} \\ \text{false} & \text{otherwise} \end{cases} \tag{5.6}$$

where $\gamma_{\text{geom}} \in [0, 1]$ is a constant. For all experiments in this thesis, we use the conservative value $\gamma_{\text{geom}} = 0.25$, chosen based on preliminary experiments. This means we will only learn composite classes that reduce the geometry variance by a substantial amount, for *all* component rule parts.

Finally, we apply a simple constant threshold on structural variance:

$$\text{lowvar}_{\text{geom}}(\mathcal{T}, K) = \begin{cases} \text{true} & \text{if } \text{var}_{\text{struct}}(\mathcal{T}, K) \leq \gamma_{\text{struct}} \\ \text{false} & \text{otherwise} \end{cases} \tag{5.7}$$

where $\gamma_{\text{struct}} \in [0, 1]$ is a constant. For all experiments in this thesis, we use the conservative value $\gamma_{\text{struct}} = 0.25$, again chosen based on preliminary experiments.

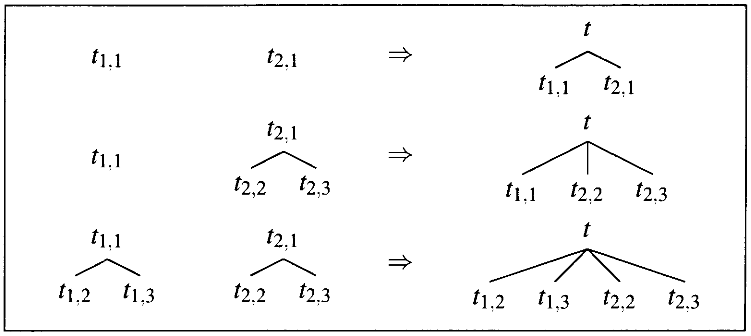


Figure 5-3: Merging subtrees.

Step 1: Clustering to Propose Candidate Classes

Phase 1 consists of iterating three steps. In Step 1, we perform clustering to find a large set Γ of good candidate composite classes.

Given a set of current subtrees $\mathcal{T}^*$, let C be the set of classes that label the root nodes of the subtrees. On the first iteration of Phase 1, this will simply be the set of object classes, since the initial subtrees are singleton nodes. On later iterations, this will include composite classes created on previous iterations.

Then, we consider each pair of classes $\langle c_i, c_j \rangle$ in C (including the case when $c_i = c_j$). For each pair of classes, we create a new subtree for each pair of existing subtrees from the same image with those two class labels on the root nodes. So, if the two classes are fork and knife, we would create a new subtree for each instance of a fork-knife pair in the same image. If the two classes are fork and fork, we would create a new subtree for each pair of forks in the same image.

Each new subtree has two levels, where the leaves are the union of the leaves of the source subtrees. If either or both of the existing subtrees are singleton nodes, these nodes become leaf nodes in the new tree. (See Figure 5-3 for examples.) We initialize the absolute geometry vector of the new tree's root to the vector produced from the bounding box of its children's bounding boxes.

This merging process on pairs of subtrees creates a set of candidate subtrees $\mathcal{T}_{c_i,c_j}$ for each pair of classes. Within $\mathcal{T}_{c_i,c_j}$, the subtrees will likely overlap with one another, since the same object may appear in multiple pairs.

Then, we perform agglomerative clustering separately within each set of new subtrees $\mathcal{T}_{c_i,c_j}$. The clustering algorithm, for a single pair of classes, is shown in Algorithm 5.1. The algorithm is very similar to the one for learning classes templates in the last chapter (Algorithm 4.4), with a few changes. First, we impose two constraints on the clustering. In Line 9, two clusters that contain overlapping subtrees may not be merged, since each cluster must be internally consistent. In Line 13, we impose thresholds on the structural and geometry variance of a cluster, as we discussed in the last section, to ensure that we only learn low-variance classes. Second, in Line 14 we use the combined geometry and structural variance metric, rather than geometry variance alone.

The last change is that, for efficiency, we do not update the parent geometry for the trees $\mathcal{T}$ in the merged cluster on each iteration. Updating the parent geometry is not as important when dealing with small sets of objects, since in that case the initial parent bounding boxes in the subtree root nodes tend to produce comparable coordinate frames across subtrees. Furthermore, the parent geometry of subtrees in merged clusters will be updated during template learning in Step 2.

For each pair of classes $\langle c_i, c_j \rangle$, the clustering results in a set of low-variance clusters Γ_{c_i,c_j}. We take the union of the cluster sets found across all pairs of classes to produce the full set of candidate clusters Γ.

Input: Set of subtrees $\mathcal{T}$, produced for the pair of classes $\langle c_1, c_2 \rangle$.
Output: Set of candidate low-variance clusters Γ.

Let $\Gamma = \emptyset$.
For each tree $T \in \mathcal{T}_{c_1,c_2}$,
    Assign each child node $t \in T$ a globally unique RPID.
    Let K be the set of RPID (and class) labels on the child nodes in T.
    Let $\Gamma = \Gamma \cup \{\langle K, \{T\} \rangle\}$.
Loop:
    Let $d^* = \infty$.
    For each pair of clusters $\langle \langle K_i, \mathcal{T}_i \rangle, \langle K_j, \mathcal{T}_j \rangle \rangle \in \Gamma$,
        If $\mathcal{T}_i$ and $\mathcal{T}_j$ have any objects in common, let $d = \infty$.
        Otherwise,
            Match templates K_i and K_j to get K (and update $\mathcal{T}_i$ and $\mathcal{T}_j$). **Algorithm 4.3**
            Let $\mathcal{T} = \mathcal{T}_i \cup \mathcal{T}_j$.
            If $\neg \text{lowvar}_{\text{geom}}(\mathcal{T}, K)$ or $\neg \text{lowvar}_{\text{geom}}(\mathcal{T}, K)$, let $d = \infty$. **Equations 5.6, 5.7**
            Otherwise, let $d = \text{var}(\mathcal{T}, K)$. **Equation 5.2**
        If $d < d^*$, let $d^* = d$, $i^* = i$, $j^* = j$, $K^* = K$, and $\mathcal{T}^* = \mathcal{T}$.
    If $d^* = \infty$, exit loop.
    Let $\Gamma = \Gamma - \{\langle K_{i^*}, \mathcal{T}_{i^*} \rangle, \langle K_{j^*}, \mathcal{T}_{j^*} \rangle\}$.
    Let $\Gamma = \Gamma \cup \{\langle K^*, \mathcal{T}^* \rangle\}$..

Algorithm 5.1: Phase 1, Step 1: An agglomerative clustering algorithm for proposing a set of candidate low-variance composite classes (for a single pair of root classes $\langle c_1, c_2 \rangle$).

Step 2: Choosing the Best Set of Consistent Classes

Step 1 produces a large set of clusters Γ, each of which are internally consistent and low-variance. But the clusters are not mutually compatible, since the same object will appear as a leaf node in multiple clusters. The new clusters may also be incompatible with clusters in Γ^* that were found on previous iterations, since the class pairs in Step 1 included these already-learned composite classes. Finally, the vast majority of the clusters in Γ will have only a few subtree members, so we would not want to keep them even if they did not conflict with any other clusters. In Step 2, we look for a subset of mutually consistent clusters from $\Gamma^* \cup \Gamma$ such that each cluster is large enough, according to a simple thresholding scheme. These clusters will be the new set of composite classes Γ^*.

Pseudocode for Step 2 is shown in Algorithm 5.2. In Line 1, we put all the new and old clusters together to form the set Γ. In Lines 2-3, we make sure the clusters in Γ are big enough. But we want a richer definition of "big enough" than a constant threshold, for several reasons. First, we need to take into account the size of the training set. Second, we want to be sensitive to the relative frequency of different object classes. If a cluster has few members, but the object classes is its template are rare, the cluster might still be a good choice for a composite class—it captures an important pattern. On the other hand, a cluster with few members but very common object classes probably should not be kept—the "pattern" may actually be coincidental.

To capture this intuition, we define the following notion of minimum cluster size:

$$\text{minclustersize}(K) = \min_{c \in \text{classes}(K)} \left(\max \left(\gamma_{\text{minsize,abs}}, \gamma_{\text{minsize}} m_c \right) \right) \tag{5.8}$$

where $\text{classes}(K)$ is the set of unique class labels on the RPIDs in K, and where m_c is the number

Input:
- Set of old clusters Γ^*, from previous iterations.
- Set of new clusters Γ, produced in Step 1.

Output: New set of mutually-consistent clusters Γ^*.

1. Let $\Gamma = \Gamma^* \cup \Gamma$.
2. For each cluster $\langle \mathcal{T}, K \rangle \in \Gamma$,
3.   If $|\mathcal{T}| < \text{minclustersize}(K)$, let $\Gamma = \Gamma - \{\langle \mathcal{T}, K \rangle\}$. **Equation 5.8**
4. Let $\Gamma^* = \emptyset$.
5. Sort the clusters in Γ by decreasing size, breaking ties to prefer low variance.
6. For each cluster $\langle \mathcal{T}, K \rangle \in \Gamma$ (in order),
7.   Let Γ_{overlap} be the set of clusters in Γ^* that share objects with $\langle \mathcal{T}, K \rangle$.
8.   If $\Gamma_{\text{overlap}} = \emptyset$, let $\Gamma^* = \Gamma^* \cup \{\langle \mathcal{T}, K \rangle\}$.
9.   Otherwise,
10.   Let $\mathcal{T}_{\text{new}} = \mathcal{T} \cup \left(\bigcup_{\langle \mathcal{T}', K' \rangle \in \Gamma_{\text{overlap}}} \mathcal{T}' \right)$, merging any subtrees that share objects.
11.   Learn a new template K_{new} for $\mathcal{T}_{\text{new}}$. **Algorithm 4.4**
12.   If $\text{lowvar}_{\text{geom}}(\mathcal{T}_{\text{new}}, K_{\text{new}})$ and $\text{lowvar}_{\text{geom}}(\mathcal{T}_{\text{new}}, K_{\text{new}})$, **Equations 5.6, 5.7**
13.   Let $\Gamma^* = \Gamma^* - \Gamma_{\text{overlap}}$.
14.   Let $\Gamma^* = \Gamma^* \cup \{\langle \mathcal{T}_{\text{new}}, K_{\text{new}} \rangle\}$.
15.   Otherwise, discard $\langle \mathcal{T}, K \rangle$ and $\langle \mathcal{T}_{\text{new}}, K_{\text{new}} \rangle$.

Algorithm 5.2: Phase 1, Step 2: An algorithm for choosing a set of mutually-consistent clusters.

of object instances with class c across the entire training set. Essentially, this definition computes the threshold on cluster size as a percentage of the number of training instances of the *rarest* object class in the cluster template K. (There is also a fixed lower bound $\gamma_{\text{minsize,abs}}$ on the minimum cluster size, which is relevant only for the smallest dataset. For all experiments in this thesis, we use $\gamma_{\text{minsize,abs}} = 10$.) The value of the percentage γ_{minsize} determines how large clusters need to be—higher values will result in fewer composite classes being learned. In the experiments in Section 5.2, we will explore the effect of varying this percentage.

In Line 5, we sort the clusters in order of decreasing size, breaking ties to prefer lower variance (using the combined geometry and structural metric). We then walk through the clusters in order. For each cluster $\langle \mathcal{T}, K \rangle$, we find all the already-accepted clusters $\Gamma_{\text{overlap}} \subseteq \Gamma^*$ that are not compatible with $\langle \mathcal{T}, K \rangle$, because they share one or more objects (Line 7). If $\langle \mathcal{T}, K \rangle$ does not conflict with any already-chosen clusters, we accept it (Line 8).

Otherwise, we have to resolve the conflict. We attempt to merge the new cluster with those it overlaps with, hoping to produce a single cluster that still has low-enough variance. To merge clusters, we take the union of their subtree sets. But the resulting set $\mathcal{T}$ will contain subsets of subtrees that overlap with one another, since otherwise the clusters would not have been incompatible. So each of these subsets of overlapping subtrees is merged to produce a single subtree, using the same technique for merging subtrees described in Step 1. This process occurs in Line 10.

Now we have non-overlapping subtrees $\mathcal{T}_{\text{new}}$ for the merged cluster, but we need a good template. To do this, we use exactly the same algorithm for learning class templates in Chapter 4, but on subtrees rather than full scene trees. This produces the template K_{new} for the merged cluster (Line 11). If the merged cluster $\langle \mathcal{T}_{\text{new}}, K_{\text{new}} \rangle$ has low enough variance, we accept it and remove all old overlapping clusters (Lines 12-14). Otherwise, we discard both the merged cluster and the cluster $\langle \mathcal{T}, K \rangle$ that prompted the merge (Line 15).

Input: Set of initial subtrees $\mathcal{T}^*$, with a single-node tree for each object in each training image.

Output: Set of clusters Γ^* and updated subtrees $\mathcal{T}^*$.

1. Let $\Gamma^* = \emptyset$.
2. Loop:
3.   Let $\Gamma = \emptyset$. — **Step 1**
4.   Let C be the set of classes labeling the root nodes of $\mathcal{T}^*$.
5.   For each pair of classes $\langle c_i, c_j \rangle \in C$ (including $c_i = c_j$),
6.   Create set of merged subtrees $\mathcal{T}_{c_i,c_j}$ from subtrees in $\mathcal{T}^*$ labeled with c_i and c_j.
7.   Perform clustering on $\mathcal{T}_{c_i,c_j}$ to get Γ_{c_i,c_j}. — **Algorithm 5.1**
8.   Let $\Gamma = \Gamma \cup \Gamma_{c_i,c_j}$.
9.   Let $\Gamma^*_{\text{old}} = \Gamma^*$. — **Step 2**
10.   Choose a new set of clusters Γ^* from Γ^*_{old} and Γ. — **Algorithm 5.2**
11.   If $\Gamma^* = \Gamma^*_{\text{old}}$, exit loop.
12.   Update the subtrees in $\mathcal{T}^*$ with those in Γ^*. — **Step 3**

Algorithm 5.3: Phase 1: An algorithm for learning a set of composite classes and subtrees.

Step 3: Updating Subtrees with New Classes

Now that we have a new set of good composite classes Γ^*, we need to update the global set of subtrees $\mathcal{T}^*$ accordingly. Since the clusters in Γ^* are mutually consistent, we can take the union of their sets of subtrees:

$$\mathcal{T}' = \bigcup_{\langle \mathcal{T}, K \rangle \in \Gamma^*} \mathcal{T}$$

We then remove any subtree from $\mathcal{T}^*$ that overlaps with a subtree in $\mathcal{T}'$, and add $\mathcal{T}'$ to $\mathcal{T}^*$.

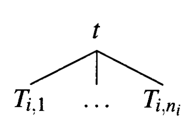
Putting It All Together

The full algorithm for Phase 1 is shown in Algorithm 5.3. It simply iterates Step 1 (Lines 3-8), Step 2 (Lines 9-10), and Step 3 (Line 12) until the set of clusters Γ^* does not change in Line 11.

Finally, we need to convert the learned clusters Γ^* and subtrees $\mathcal{T}^*$ to a set of composite classes C, each with a learned rule, and a set with a full scene trees T_i for each training images I_i. This is straightforward. For each cluster $\langle \mathcal{T}, K \rangle \in \Gamma^*$, we create a new composite class with a unique class name c_{new}, and a rule produced from the cluster template K:

$$c_{\text{new}} \rightarrow \ldots, \text{class}(k)_k \ldots \quad \forall_{k \in K}$$

Then, for each training image I_i labeled with scene class c_i, let $\{T_{i,1}, \ldots, T_{i,n_i}\}$ be the set of subtrees in $\mathcal{T}^*$ that came from image I_i. We create the full scene tree T_i for image I_i as:



where $\text{class}(t) = c_i$, and the absolute geometry vector of t is produced from the bounding box of the bounding boxes on the leaves. The RPID labels on the root nodes of $\{T_{i,1}, \ldots, T_{i,n_i}\}$ are unset.

5.1.2 Phase 2: Learning Scene Class Rules

In Phase 2, the goal is to learn the top-level rules for the scene classes S, and in the process, assign the RPID labels to the root nodes of the subtrees in each scene tree T_i created at the end of Phase 1.

In fact, we already have an algorithm to do exactly this—Algorithm 4.4 for learning class templates. For each scene class $c \in S$, we run template learning on the set of scene trees with root class c, to produce a template K_c. (The template learning uses only the root node and its children in each tree, so the fact that these are three-level trees is not a problem.) We then write down the rule for scene class c as:

$$c \rightarrow \ldots, \mathrm{class}(k)_k \ldots \quad \forall_{k \in K_c}$$

The template learning also updates the scene trees for each scene class to have RPID labels corresponding to the learned rule. And it updates the root absolute geometry vectors in each scene tree to something more appropriate than the simple bounding box of leaves set at the end of Phase 1.

Thus, by the end of Phase 2, we have learned a set of composite classes C, each with a single rule; a top-level rule for each scene class in S; and a set of scene trees for the training images.

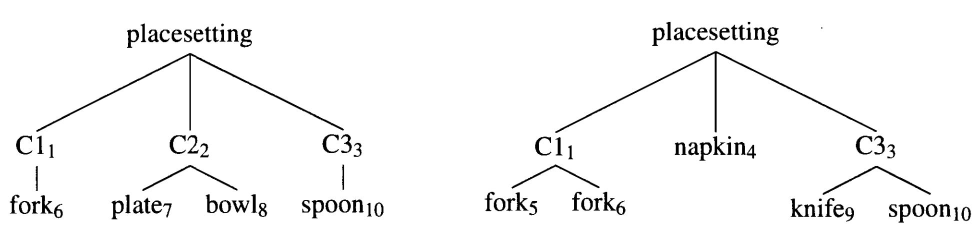
5.1.3 Assigning Single Objects to Composite Classes

In the example at the beginning of the chapter, we said that for the placesetting images in Figure 5-4, we might want to learn the following grammar:

$$C = \{\mathrm{placesetting}, \mathrm{C1}, \mathrm{C2}, \mathrm{C3}\}$$

$$R = \left\{ \begin{array}{c} \mathrm{placesetting} \rightarrow \mathrm{C1}_1\ \mathrm{C2}_2\ \mathrm{C3}_3\ \mathrm{napkin}_4 \\ \mathrm{C1} \rightarrow \mathrm{fork}_5\ \mathrm{fork}_6 \\ \mathrm{C2} \rightarrow \mathrm{plate}_7\ \mathrm{bowl}_8 \\ \mathrm{C3} \rightarrow \mathrm{knife}_9\ \mathrm{spoon}_{10} \end{array} \right\}$$

with these scene trees:



But in fact, the two-phase algorithm we have described so far would not be able to learn the tree structure on the left. Specifically, it could never assign the composite class label of C1 to a single fork, or C3 to a single spoon, because singleton object groups are not included in the clustering

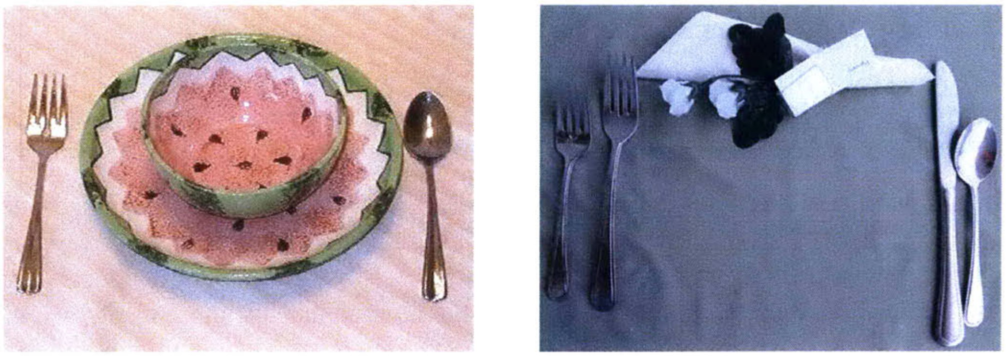


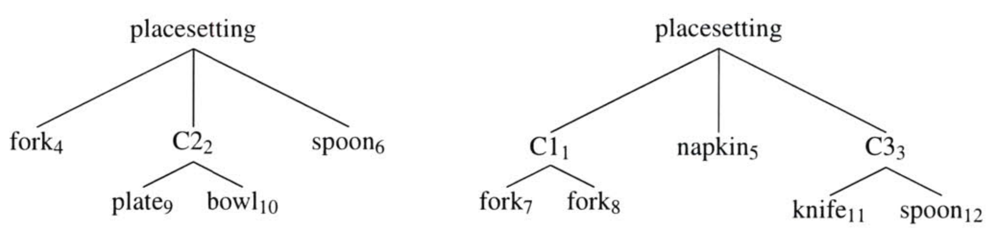
Figure 5-4: Two images of simple placesettings, again.

process on pairs of subtrees. The current algorithm might produce this structure instead:[1]

$$C = \{\text{placesetting}, \text{C1}, \text{C2}, \text{C3}\}$$

$$R = \left\{ \begin{array}{r} \text{placesetting} \rightarrow \text{C1}_1\ \text{C2}_2\ \text{C3}_3\ \text{fork}_4\ \text{napkin}_5\ \text{spoon}_6 \\ \text{C1} \rightarrow \text{fork}_7\ \text{fork}_8 \qquad\qquad\qquad \\ \text{C2} \rightarrow \text{plate}_9\ \text{bowl}_{10} \qquad\qquad\quad \\ \text{C3} \rightarrow \text{knife}_{11}\ \text{spoon}_{12} \qquad\qquad \end{array} \right\}$$

with these trees:



Although this structure is not as compact as the first one, there is nothing inherently wrong with it. In fact, it more expressive. However, it comes with the problems that traditionally accompany less compact structures. In particular, it does not allow the sharing of parameters across corresponding forks and spoons in the two trees. It does not capture that the relationship of the fork in the left image to the rest of objects is very similar to the relationship of the pair of forks in the second image to the rest of those objects.

These patterns occur at the top level. To discover them, we need to consider the geometry of objects or object groups with respect to the root of the entire scene tree, rather than focusing only on the relationships within groups as we did in Phase 1. Thus, we add an additional optional step in which we consider adding singleton objects to existing composite classes. This step can occur within Phase 1, at the beginning of Phase 2, or not at all; we will explore this choice in the experiments this chapter.

The algorithm operates separately within each scene class, and assumes an existing set of composite classes Γ^* and subtrees $\mathcal{T}^*$ (that have not yet been merged to produce full scene trees). Psuedocode is given in Algorithm 5.4.

[1] This example assumes there are other training images with similar structure as well—otherwise even the plate-bowl, fork-fork, and knife-spoon pairs wouldn't be learned.

Input: Set of composite clusters Γ^* and current subtrees $\mathcal{T}^*$, for a single scene class.

Output: Updated clusters Γ^* and subtrees $\mathcal{T}^*$, with some singleton subtrees in $\mathcal{T}^*$ assigned to clusters in Γ^*.

1. For each composite cluster $\langle K_\ell, \mathcal{T}_\ell \rangle \in \Gamma^*$,
2. Perform clustering on subtrees in $\mathcal{T}_\ell$, producing a set of groups $\{\mathcal{T}_{\ell,g}\}$.
3. For each group $\mathcal{T}_{\ell,g}$,
4. Let $\sigma^2_{\ell,g}$ be the normalized relative-to-root geometry variance for $\{\mathcal{T}_{\ell,g}\}$.
5. Loop:
6. Let $d^* = \infty$.
7. For each singleton subtree $t \in \mathcal{T}^*$ and each group $\mathcal{T}_{\ell,g}$,
8. Consider adding t to $\mathcal{T}_{\ell,g}$, producing new tree T and distance d. **Algorithm 5.5**
9. If $d < d^*$, let $d^* = d$, $T^* = T$, $\ell^* = \ell$, and $g^* = g$.
10. If $d^* = \infty$, exit loop.
11. Let class(root(T)) be the class label for composite class ℓ^*.
12. Let $\mathcal{T}_{\ell^*,g^*} = \mathcal{T}_{\ell^*,g^*} \cup \{T\}$.
13. Let $\mathcal{T}^* = \mathcal{T}^* - \{t\}$.
14. Let $\mathcal{T}^* = \mathcal{T}^* \cup \{T\}$.
15. For each composite cluster $\langle K_\ell, \mathcal{T}_\ell \rangle \in \Gamma^*$,
16. Let $\mathcal{T}_\ell = \bigcup_g \mathcal{T}_{\ell,g}$.

Algorithm 5.4: An algorithm for assigning single objects to existing composite classes (within a single scene class).

First, we perform clustering *within* each composite class, to find groups of subtrees with similar parent-to-root geometry. Formally, for each cluster $\langle K_\ell, \mathcal{T}_\ell \rangle \in \Gamma^*$, we compute the relative-to-root geometry vector $\text{geom}_{\text{rel}}(\text{root}(T))$ for each subtree $T \in \mathcal{T}_\ell$ in the same way as the relative geometry vector for a leaf node (Equation 4.1), but using $\text{geom}_{\text{abs}}(\text{root}(T))$ as the "child" vector and $\text{geom}_{\text{abs,root}}(\text{root}(T))$ as the "parent". The goal is to partition the subtrees in $\mathcal{T}_\ell$ into groups such that within each group, the $\text{geom}_{\text{rel}}(\text{root}(T))$ vectors have low variance, and each subtree T is from a different image (since these groups are proxies for top-level rule parts that might be learned later in Phase 2). We do this with agglomerative clustering on subtrees in $\mathcal{T}_\ell$, minimizing variance in the $\text{geom}_{\text{rel}}(\text{root}(T))$ vectors while not allowing clusters with subtrees from the same image to merge. This clustering-within-a-cluster occurs in Lines 1-2 of Algorithm 5.4, and produces a set of subtree groups $\{\mathcal{T}_{\ell,g}\}$ for each composite cluster. We also compute the relative-to-root geometry variance for each group (Lines 3-4), which will be used later.

Next, we perform constrained clustering to assign singleton objects to compatible composite groups. For a single-node subtree $t \in \mathcal{T}^*$, a "compatible" composite group $\mathcal{T}_{\ell,g}$ is one whose template K_ℓ contains class(t), because we need to be able to assign t an existing RPID in K_ℓ, but does not contain any subtrees $T \in \mathcal{T}_{\ell,g}$ from the same training image as t. So we perform agglomerative clustering, but the only type of cluster merges we allow are between single subtrees t and compatible groups $\mathcal{T}_{\ell,g}$ (Line 7).

In the earlier example, we might have singleton subtree t with class(t) = fork from the left image, and a group $\mathcal{T}_{\text{C1},g}$ containing this subtree:

$$\begin{array}{c} \text{C1}_g \\ \text{fork}_7 \quad \text{fork}_8 \end{array}$$

Input:
- Singleton subtree t.
- Composite class group $\mathcal{T}$ with class template K and original relative-to-root variance σ^2.

Output: New two-node subtree T with t as a child, and a distance d^* between T and $\mathcal{T}$.

1. Let $d^* = \infty$.
2. If $\text{class}(t)$ does not appear in K, return.
3. If any subtree $T \in \mathcal{T}$ is from the same training image as t, return.
4. Let μ_{root} be the mean $\text{geom}_{\text{rel}}(\text{root}(T))$ vector across subtrees $T \in \mathcal{T}$.
5. For each $k \in K$ such that $\text{class}(k) = \text{class}(t)$,
6.   Let μ_k be the mean $\text{geom}_{\text{rel}}(t_{\text{leaf}})$ vector across leaf nodes t_{leaf} in $\mathcal{T}$ with RPID label k.
7.   Let t' be a new node, with $\text{geom}_{\text{abs}}(t')$ chosen such that $\text{geom}_{\text{rel}}(t)$ with parent t' equals μ_k.
8.   Let $\text{geom}_{\text{rel}}(t')$ be computed with parent geometry $\text{geom}_{\text{abs,root}}(t)$.
9.   Let $d = \|\text{geom}_{\text{rel}}(t') - \mu_{\text{root}}\|_2^*$.
10.   If $d < d^*$, let $d^* = d$, $k^* = k$, and $t'^* = t'$.
11. Let $\text{rpid}(t) = k^*$.
12. Let T be a new two-node subtree with t'^* as root and t as the only child.
13. Let σ^2_{new} be the normalized relative-to-root geometry variance for $\mathcal{T} \cup \{T\}$.
14. If $\sigma^2_{\text{new}} > \sigma^2$, let $d^* = \infty$.

Algorithm 5.5: An algorithm for adding a single-object subtree to a composite class group.

from the right image. If we choose to add t to $\mathcal{T}_{\text{C1},g}$, we need to create a new parent node t' for t with $\text{class}(t') = \text{C1}$, and choose an RPID label $\text{rpid}(t)$ for t and an absolute geometry vector $\text{geom}_{\text{abs}}(t')$ for t' such that:

1. The relative-to-parent geometry vector $\text{geom}_{\text{rel}}(t)$ produced using $\text{geom}_{\text{abs}}(t')$ as parent is similar to the equivalent vector for corresponding leaf nodes with $\text{rpid}(t)$ in other subtrees $T \in \mathcal{T}_{\text{C1},g}$. If we assign t RPID 8, it should be on the same side with respect to its new parent t' as the fork labeled 8 is with respect to its parent in the right image.

2. The relative-to-root geometry vector $\text{geom}_{\text{rel}}(t')$ produced using $\text{geom}_{\text{abs}}(t')$ is similar to the $\text{geom}_{\text{rel}}(\text{root}(T))$ vectors on other subtrees $T \in \mathcal{T}_{\text{C1},g}$. The new parent t' should be on the left side of the other objects in the same way that the pair of forks is in the right image.

In other words, we want to pick a parent node t' with geometry that makes both its child t and its root (the bounding box of all the objects in the image) happy.

Pseudocode to do this is shown in Algorithm 5.5. For a singleton node t and a composite group $\mathcal{T}$ with template K, we first check that $\mathcal{T}$ is compatible with t (Lines 2-3). Then, we search over RPID assignments for t from those in K. For each RPID k, we choose a $\text{geom}_{\text{abs}}(t')$ vector for the new parent node such that $\text{geom}_{\text{rel}}(t)$ would be exactly equal to the mean relative-to-parent geometry vector across all leaf nodes with RPID label k in $\mathcal{T}$ (Lines 6-7). Thus, the first criteria above is satisfied perfectly. We then let the "distance" between the singleton tree t and the group $\mathcal{T}$ be the normalized L^2 distance between the relative-to-root vector $\text{geom}_{\text{rel}}(t')$ produced using $\text{geom}_{\text{abs}}(t')$, and the mean relative-to-root vector across the root nodes of the subtrees in $\mathcal{T}$ (Lines 8-9). The RPID k^* that minimizes this distance is chosen to label t (Line 11), and this distance is also what we minimize during agglomerative clustering.

This approach ensures that the *internal* geometry variance within a composite class will not increase, since we choose RPID labels and parent vectors for the singleton objects that perfectly agree with the template mean (at least within the subset of the composite class in the group). So the only concern is that the variance in the relative-to-root geometry vectors in each group $\mathcal{T}$ not increase by much. Let σ^2 be the normalized relative-to-root geometry variance for $\mathcal{T}$ before any single objects are added. Then, we will not add a singleton subtree t to $\mathcal{T}$ if adding the best two-node tree T for t to $\mathcal{T}$ would produce a new relative-to-root geometry variance that is greater than σ^2 (Lines 13-14).

Returning to Algorithm 5.4, the assigning of single objects to composite groups continues until no more assignments are valid (Line 10). The remaining Lines 11-16 simply update the global set of composite classes Γ^* and subtrees $\mathcal{T}^*$ to reflect any changes.

We can insert the step of assigning single objects to composite classes in several places in the full structure learning algorithm. First, we can perform it as Step 4 in each iteration of Phase 1 (Algorithm 5.3). In practice, this means we learn fewer composite classes, each containing many member subtrees, but many of the subtrees are single-object trees. The learned structure is thus very compact, but less expressive. Second, we can perform the step as a first step in Phase 2 (Section 5.1.2), before learning the top-level rules for each scene class. In this case, we give priority to learning composite classes for multiple-object subtrees, and only assign single object subtrees after full composite class learning has finished. Finally, we can choose to not perform the step at all, in which case we learn less compact, but more expressive, structures. In the next section, we explore these choices further.

5.2 Experiments with Three-Level Learned Structures

In this section, we present experiments evaluating three-level WGG models learned using the structure learning algorithms described in this chapter. We explore the following algorithm variations:

- the percentage γ_{minsize} of training objects used to determine minimum cluster size; and
- whether and/or when to assign single-object subtrees to composite classes (during Phase 1 or Phase 2, or not at all).

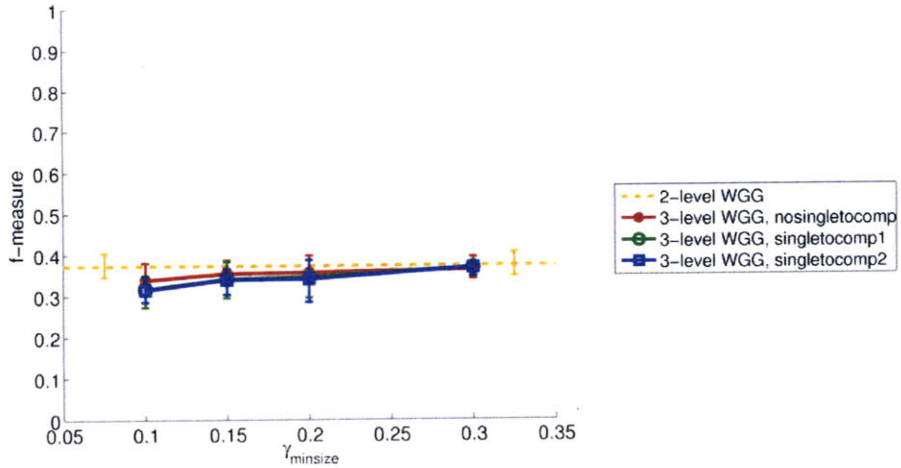
We compare these results to the DTPBM object detector and the hand-built and two-level WGG models presented in previous chapters.

For each dataset, we used the same experimental setup as for that dataset in previous chapters. For simplicity, we performed all the experiments in this chapter using pairwise weights, class-based object detector features, and fixed ground-truth internal geometry.

5.2.1 Results on the Small Placesetting Dataset

Figure 5-5 shows the performance of the DTPBM object detector, the hand-built WGG structure, the learned two-level WGG models from last chapter, and twelve variations of learned three-level WGGs. The graph plots cumulative f-measure of three-level WGGs as a function of the value of the γ_{minsize} parameter, for each version of whether or when single objects are assigned to composite classes. It also shows the average f-measure from the learned two-level WGG (dotted orange line), for comparison. Figure 5-6 shows the same graph for each class.

The most salient feature of these results is that they are roughly equivalent, or slightly worse, than the performance of the learned two-level WGGs from last chapter. Furthermore, the performance does not differ much among different variations of the structure learning algorithm. As the value of γ_{minsize} increases (meaning that clusters need to be larger in order to be accepted as composite clases), we do see a slight improvement in results. It also seems that not assigning singleton



 # targets (all models): 741.0

	recall	**precision**	**f-measure**	**# detections**	**# correct**
DTPBM, known # objs	0.380 [0.341 0.420]	0.571 [0.521 0.598]	0.456 [0.412 0.493]	493.0	280.8
DTPBM, heuristic # objs	0.415 [0.376 0.447]	0.138 [0.117 0.171]	0.205 [0.184 0.235]	2263.6	306.8
hand-built WGG	0.254 [0.244 0.264]	0.604 [0.561 0.647]	0.357 [0.353 0.362]	312.0	188.0
2-level WGG	0.278 [0.248 0.304]	0.565 [0.512 0.601]	0.372 [0.346 0.404]	363.8	205.2
3-level WGG:					
$\gamma_{minsize} = 0.1$, nosingletocomp	0.235 [0.203 0.263]	0.606 [0.563 0.682]	0.338 [0.301 0.380]	286.6	173.4
$\gamma_{minsize} = 0.1$, singletocomp1	0.215 [0.173 0.244]	0.610 [0.545 0.645]	0.316 [0.272 0.349]	262.6	159.2
$\gamma_{minsize} = 0.1$, singletocomp2	0.212 [0.180 0.238]	0.619 [0.576 0.679]	0.314 [0.285 0.346]	255.2	156.8
$\gamma_{minsize} = 0.15$, nosingletocomp	0.253 [0.222 0.282]	0.597 [0.532 0.639]	0.354 [0.330 0.386]	314.2	186.8
$\gamma_{minsize} = 0.15$, singletocomp1	0.236 [0.200 0.277]	0.614 [0.558 0.644]	0.340 [0.294 0.382]	284.6	175.0
$\gamma_{minsize} = 0.15$, singletocomp2	0.235 [0.208 0.259]	0.612 [0.562 0.644]	0.339 [0.303 0.366]	284.0	174.4
$\gamma_{minsize} = 0.2$, nosingletocomp	0.258 [0.232 0.289]	0.582 [0.524 0.637]	0.356 [0.333 0.397]	327.8	190.2
$\gamma_{minsize} = 0.2$, singletocomp1	0.245 [0.200 0.282]	0.586 [0.531 0.619]	0.345 [0.294 0.386]	309.8	181.6
$\gamma_{minsize} = 0.2$, singletocomp2	0.239 [0.189 0.282]	0.594 [0.560 0.617]	0.340 [0.284 0.386]	297.4	177.0
$\gamma_{minsize} = 0.3$, nosingletocomp	0.265 [0.241 0.292]	0.572 [0.540 0.605]	0.362 [0.340 0.393]	342.8	196.0
$\gamma_{minsize} = 0.3$, singletocomp1	0.269 [0.257 0.287]	0.579 [0.531 0.603]	0.367 [0.346 0.381]	343.8	199.0
$\gamma_{minsize} = 0.3$, singletocomp2	0.269 [0.257 0.287]	0.579 [0.531 0.603]	0.367 [0.346 0.381]	343.8	199.0

Figure 5-5: Cumulative results for the DTPBM object detector, the hand-built WGG structure and learned two-level WGG models from last chapter, and variations of learned three-level WGGs on the small placesetting dataset. $\gamma_{minsize}$ is the percentage of training objects used to determine minimum cluster size. nosingletocomp means that single-object subtrees are not assigned to composite classes. singletocomp1 means that single-object subtrees were assigned to composite classes during Phase 1, while singletocomp2 means single-object subtrees were assigned to composite classes during Phase 2. All WGGs use pairwise tree structure features, class-based object detector features, and fixed ground-truth internal geometry (pairwise, classobj, fixed).

objects to composite classes at all slightly outperforms assigning them during either Phase 1 or Phase 2. Both of these trends suggest that the models perform better with *fewer* composite classes. Put another way, hierarchy does not help, and perhaps even hurts performance—at least, hierarchy learned with the algorithms in this chapter.

Figure 5-7 visualizes the results in a different way. For each learned three-level WGG, we compute the percentage of training objects that were assigned to composite classes in the learned

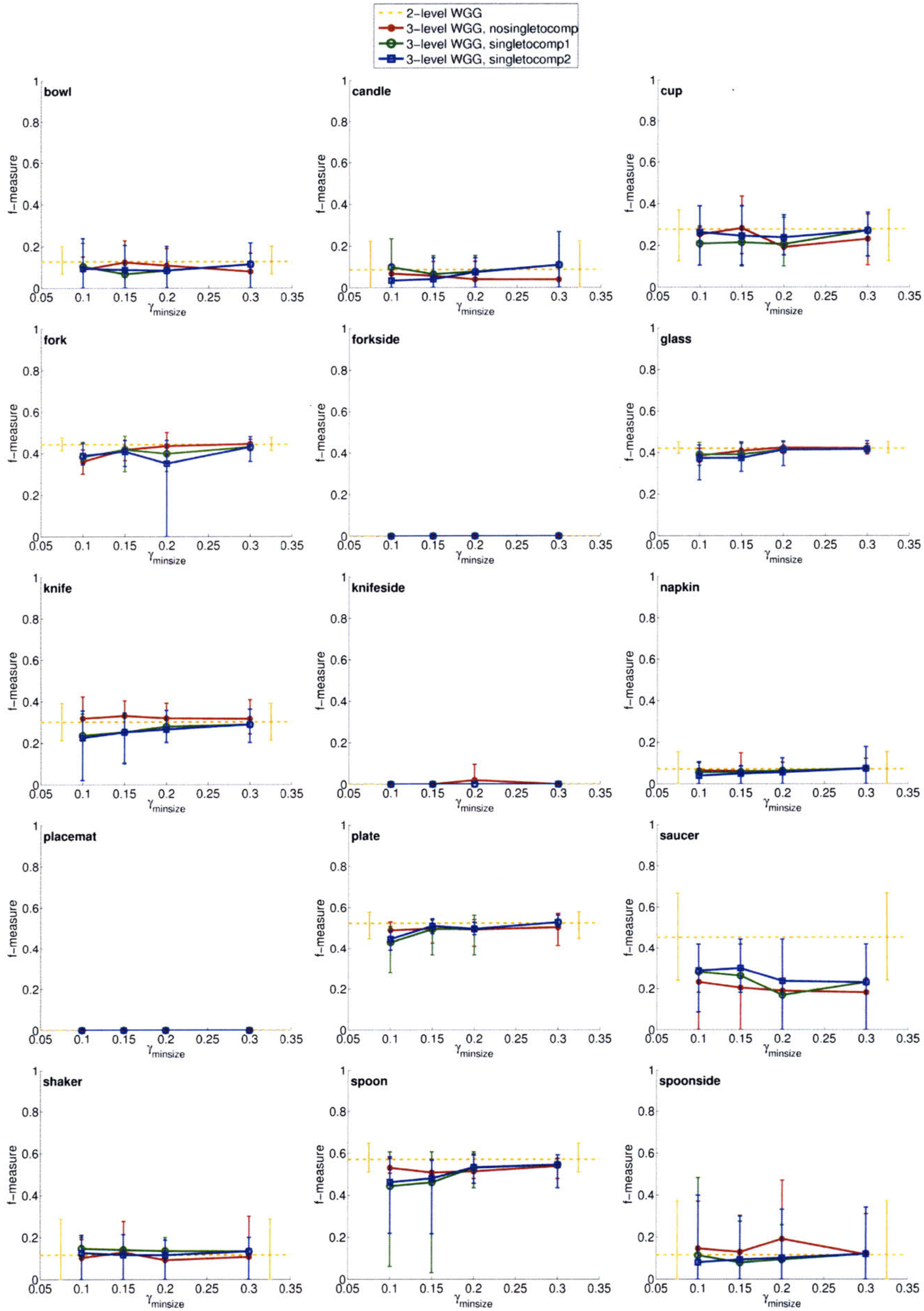


Figure 5-6: Per-class results for learned two-level and three-level WGG models on the small place-setting dataset. For numeric per-class results, see Appendix B (Tables B.12 and B.18-B.29).

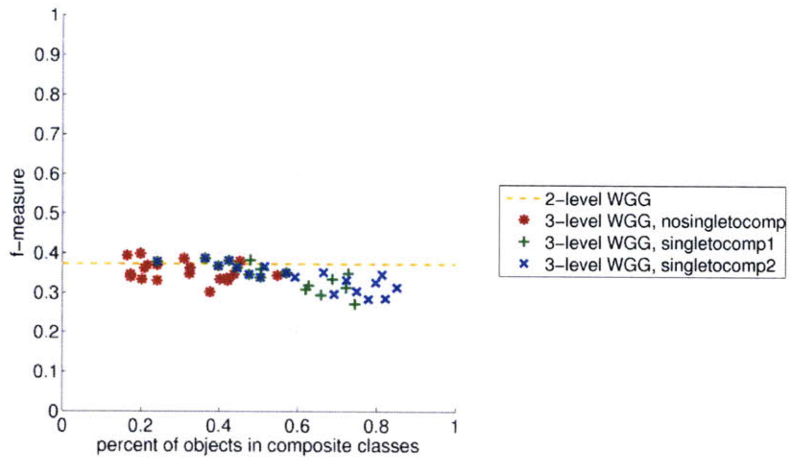


Figure 5-7: Test performance (cumulative f-measure) of each learned three-level WGG model on the small placesetting dataset, as a function of the percent of training objects assigned to composite classes during structure learning for that model.

parse trees for the training images. We then plot the cumulative f-measure of the model on the test set, as a function of that percentage. We do this for each individual experiment with each variation of the model; since we performed 5 runs for each of 12 variations, this results in 60 total points on the scatterplot. Again, we also plot the the average f-measure from the learned two-level WGG (dotted orange line), for comparison.

The scatterplot illustrates more directly the relationship between amount of learned hierarchy and test performance. It shows that, independent of which version of the algorithm learned the model, there is a slight inverse correlation between the number of objects that are assigned to composite classes, and the model's performance at test time.

These results may seem disappointing initially. We had hypothesized that by introducing composite classes, we could better capture co-occurence patterns among groups of objects. We had also hoped that the local coordinate frames associated with the composite classes would help weaken the strong conditional independence assumptions of the geometry models. However, the results do not seem to confirm these hypotheses.

In fact, these findings are consistent with the comparison between the hand-built WGG structure and the learned two-level models last chapter. There, we found that the learned models slightly outperformed the hand-built structure; this despite that we deliberately constructed the model and labeled trees to produce composite classes for co-occurring object groups with low internal geometry variance. Figure 5-8 shows several representative learned three-level structures. Notice the qualitative similarity between the learned structures and the hand-built one in Figure 3-3.

This analysis suggests that the structure learning algorithms are actually achieving our design objectives to a reasonable degree. Rather, it seems that the objectives themselves, based on our assumptions about the utility of composite classes, were incorrect.

Alternatively, we can interpret these results positively. The two-level structure learning algorithm is much simpler than its three-level counterpart, and it produces it better results. A two-level learned structure is a relatively compact model that assigns a larger number of training objects to each rule part in the single flat rule, lessening the danger of overfitting. Furthermore, the use of pairwise tree structure features means that the perceptron actually performs some "structure learning"—it learns a fairly rich model of co-occurrence among pairs of objects, without needing to search over explicit subsets of objects.

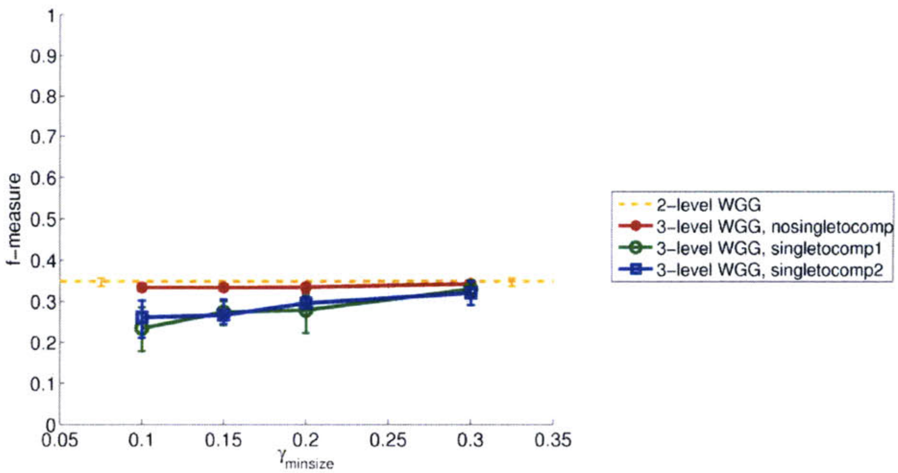
```
sceneClasses: {'placesetting'}
objClasses: {'bowl' 'candle' 'cup' 'fork' 'forkside' 'glass' 'knife' 'knifeside' 'napkin' 'placemat'
            'plate' 'saucer' 'shaker' 'spoon' 'spoonside'}
placesetting -> C00016 (23963)  C00016 (23964)  C00016 (23965)  C00017 (23966)  C00018 (23967)
                C00018 (23968)  C00019 (23969)  C00019 (23970)  C00019 (23971)  C00020 (23972)
                C00020 (23973)  C00020 (23974)  C00021 (23976)  C00021 (23977)  C00021 (23978)
                C00022 (23979)  C00023 (23980)  C00024 (23981)  C00024 (23982)  C00025 (23983)
                C00025 (23984)  bowl (23928)  bowl (23929)  bowl (23930)  candle (23931)
                candle (23932)  candle (23933)  cup (23935)  cup (23936)  fork (23937)
                fork (23938)  forkside (23939)  glass (23940)  glass (23941)  glass (23942)
                knife (23943)  knife (23944)  knifeside (23945)  napkin (23947)  napkin (23948)
                placemat (23949)  plate (23950)  plate (23951)  plate (23952)  shaker (23953)
                shaker (23954)  spoon (23955)  spoon (23956)  spoonside (23957)  spoonside (23958)
                spoonside (23959)  spoonside (23960)
C00016 -> bowl (371)  napkin (372)  plate (5173)  plate (5174)
C00017 -> knife (4805)  spoon (4806)
C00018 -> fork (1463)  fork (1464)
C00019 -> glass (3127)  glass (3128)  glass (3129)  glass (3130)
C00022 -> fork (7171)  fork (7172)  napkin (7357)
C00020 -> cup (1343)  saucer (1344)
C00023 -> knife (9016)  spoon (9017)
C00024 -> knife (8872)  plate (8873)
C00021 -> forkside (3043)  spoonside (3044)
C00025 -> glass (13659)  knife (13660)
```

$\gamma_{\text{minsize}} = 0.1$, nosingletocomp

```
sceneClasses: {'placesetting'}
objClasses: {'bowl' 'candle' 'cup' 'fork' 'forkside' 'glass' 'knife' 'knifeside' 'napkin' 'placemat'
            'plate' 'saucer' 'shaker' 'spoon' 'spoonside'}
placesetting -> C00016 (12347)  C00016 (12348)  C00016 (12349)  C00017 (12350)  C00018 (12351)
                C00018 (12352)  C00019 (12353)  C00019 (12354)  C00019 (12355)  C00020 (12356)
                C00020 (12357)  C00020 (12358)  C00020 (12359)  C00021 (12360)  C00021 (12361)
                C00021 (12362)  bowl (12318)  bowl (12319)  candle (12320)  candle (12321)
                candle (12322)  fork (12325)  forkside (12326)  glass (12327)  knife (12328)
                knife (12329)  knife (12330)  knifeside (12331)  knifeside (12332)
                napkin (12333)  napkin (12334)  placemat (12335)  plate (12336)  plate (12337)
                plate (12338)  shaker (12339)  shaker (12340)  spoon (12341)  spoonside (12342)
                spoonside (12343)  spoonside (12344)  spoonside (12345)
C00016 -> bowl (371)  napkin (372)  plate (5173)  plate (5174)
C00018 -> fork (1463)  fork (1464)
C00017 -> knife (4805)  spoon (4806)
C00019 -> glass (3127)  glass (3128)  glass (3129)  glass (3130)
C00021 -> forkside (3043)  spoonside (3044)
C00020 -> cup (1343)  saucer (1344)
```

$\gamma_{\text{minsize}} = 0.1$, singletocomp1

Figure 5-8: Representative learned WGG three-level structures for the small placesetting dataset.



 # targets (all models): 3899.0

	recall	**precision**	**f-measure**	**# detections**	**# correct**
DTPBM, known # objs	0.387 [0.381 0.390]	0.432 [0.428 0.437]	0.408 [0.403 0.412]	3494.7	1509.3
DTPBM, heuristic # objs	0.475 [0.465 0.482]	0.079 [0.075 0.085]	0.135 [0.130 0.144]	23666.3	1850.3
2-level WGG	0.267 [0.255 0.274]	0.499 [0.492 0.508]	0.348 [0.336 0.356]	2086.7	1040.7
3-level WGG:					
$\gamma_{minsize} = 0.1$, nosingletocomp	0.242 [0.237 0.250]	0.535 [0.519 0.546]	0.333 [0.325 0.343]	1766.7	944.7
$\gamma_{minsize} = 0.1$, singletocomp1	0.154 [0.114 0.193]	0.494 [0.410 0.543]	0.234 [0.178 0.285]	1204.0	601.0
$\gamma_{minsize} = 0.1$, singletocomp2	0.166 [0.132 0.194]	0.611 [0.536 0.671]	0.261 [0.211 0.301]	1052.7	647.7
$\gamma_{minsize} = 0.15$, nosingletocomp	0.245 [0.240 0.251]	0.521 [0.514 0.527]	0.333 [0.328 0.340]	1836.7	956.0
$\gamma_{minsize} = 0.15$, singletocomp1	0.189 [0.164 0.209]	0.496 [0.462 0.526]	0.273 [0.242 0.300]	1479.7	736.0
$\gamma_{minsize} = 0.15$, singletocomp2	0.177 [0.161 0.204]	0.535 [0.498 0.600]	0.266 [0.244 0.304]	1288.3	690.7
$\gamma_{minsize} = 0.2$, nosingletocomp	0.251 [0.238 0.258]	0.500 [0.487 0.516]	0.334 [0.320 0.344]	1956.3	977.7
$\gamma_{minsize} = 0.2$, singletocomp1	0.201 [0.155 0.232]	0.455 [0.393 0.492]	0.278 [0.222 0.313]	1706.7	782.0
$\gamma_{minsize} = 0.2$, singletocomp2	0.217 [0.204 0.235]	0.458 [0.456 0.461]	0.295 [0.283 0.310]	1851.0	847.7
$\gamma_{minsize} = 0.3$, nosingletocomp	0.261 [0.252 0.266]	0.498 [0.489 0.511]	0.342 [0.333 0.348]	2042.7	1016.0
$\gamma_{minsize} = 0.3$, singletocomp1	0.248 [0.229 0.274]	0.491 [0.485 0.499]	0.329 [0.313 0.350]	1972.7	968.0
$\gamma_{minsize} = 0.3$, singletocomp2	0.241 [0.211 0.271]	0.480 [0.463 0.490]	0.320 [0.290 0.348]	1958.3	940.7

Figure 5-9: Cumulative results for the DTPBM object detector, the learned two-level WGG models from last chapter, and variations of learned three-level WGGs on the big placesetting dataset. $\gamma_{minsize}$ is the percentage of training objects used to determine minimum cluster size. nosingletocomp means that single-object subtrees are not assigned to composite classes. singletocomp1 means that single-object subtrees were assigned to composite classes during Phase 1, while singletocomp2 means single-object subtrees were assigned to composite classes during Phase 2. All WGGs use pairwise tree structure features, class-based object detector features, and fixed ground-truth internal geometry (pairwise, classobj, fixed).

5.2.2 Results on the Big Placesetting Dataset

Figure 5-9 shows the performance of the DTPBM object detector, the learned two-level WGG models from last chapter, and twelve variations of learned three-level WGGs. The graph plots cumulative f-measure of three-level WGGs as a function of the value of the $\gamma_{minsize}$ parameter, for each version of whether or when single objects are assigned to composite classes. It also shows the average f-measure from the learned two-level WGG (dotted orange line), for comparison. Figure 5-10 shows the same graph for each class. And Figure 5-11 shows the same results, but visualized as a

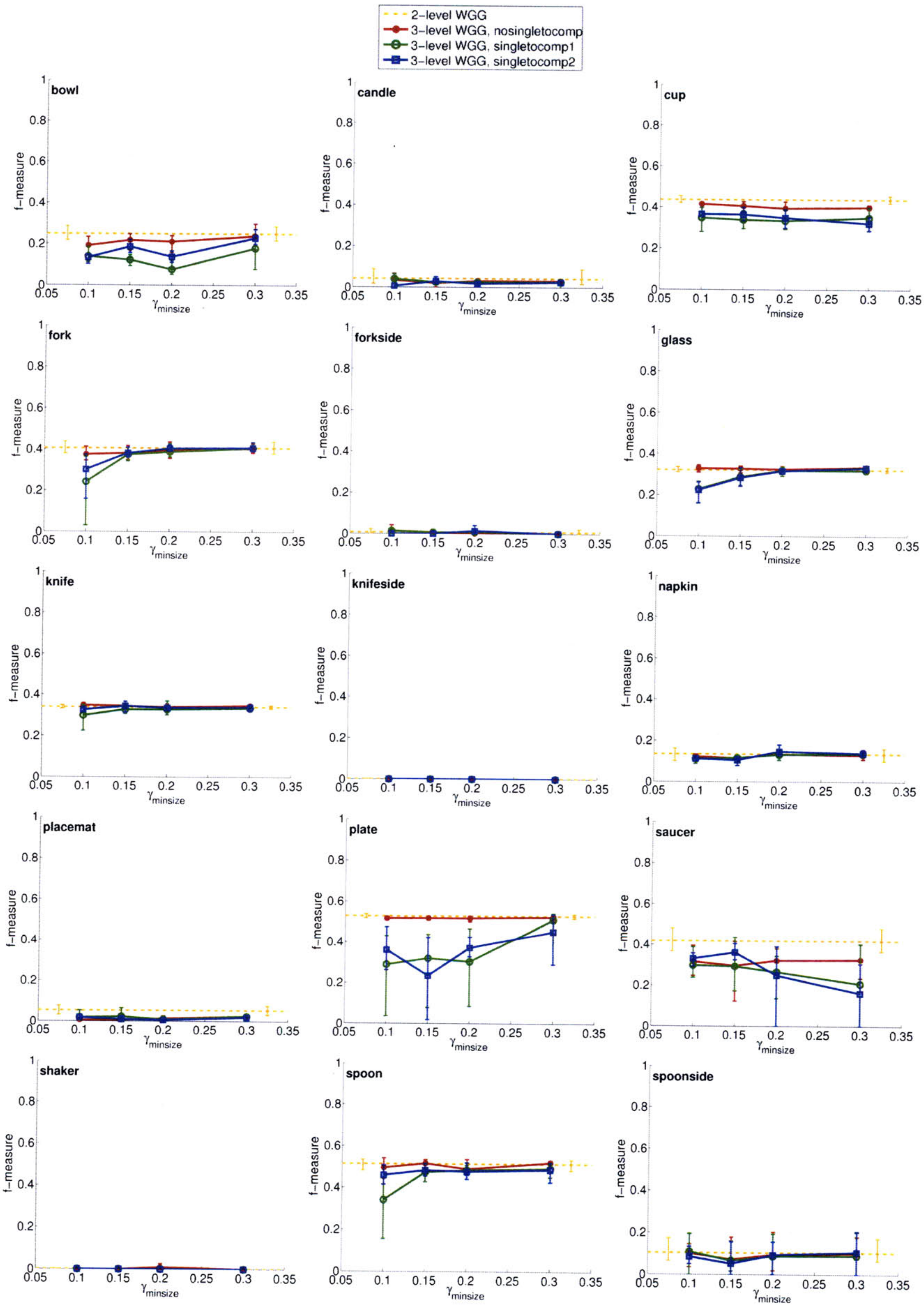


Figure 5-10: Per-class results for learned two-level and three-level WGG models on the big place-setting dataset. For numeric per-class results, see Appendix B (Tables B.32 and B.38-B.49).

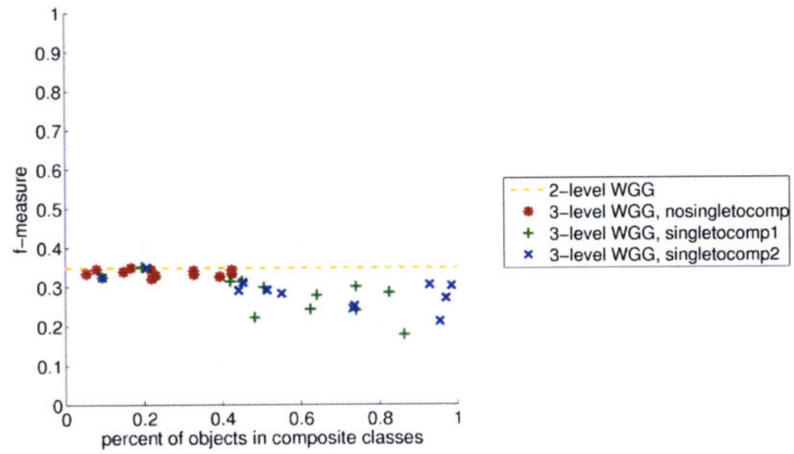


Figure 5-11: Test performance (cumulative f-measure) of each learned three-level WGG model on the big placesetting dataset, as a function of the percent of training objects assigned to composite classes during structure learning for that model.

scatterplot. (Review the description of how the plot was made in the last section.)

In this dataset, the learned models that do not assign singleton objects to composite classes at all ('nosingletocomp') outperform those that do ('singletocomp1' and 'singletocomp2') to a greater degree, particularly for lower values of γ_{minsize}. It seems that forcing single objects to participate in composite classes damages performance more in this dataset, perhaps because the spatial arrangements are more variable than in the smaller dataset.

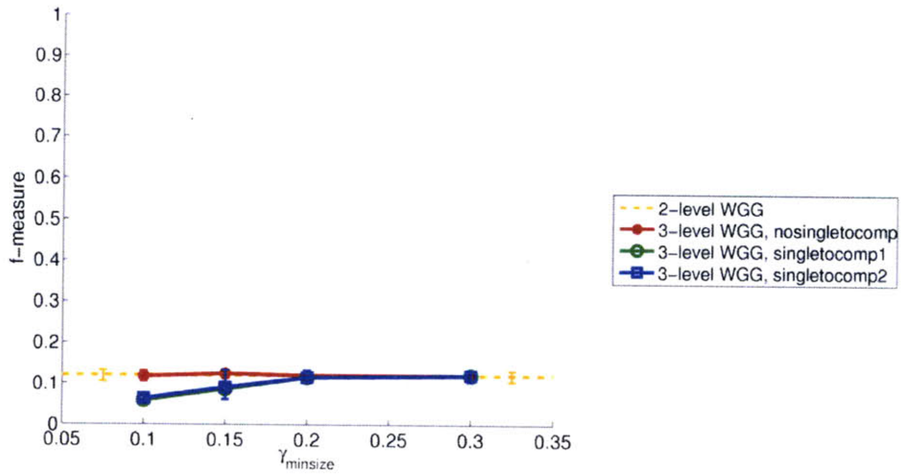
It is also important to remember that when composite classes are formed, the objects in those classes no longer have pairwise part presence features with objects not assigned to the same class. Put another way, a two-level model can actually represent a richer set of co-occurrence patterns than a three-level model, which explicitly hard-codes certain co-occurrence patterns at the expense of others. Thus, another explanation for the greater drop in performance is that, in the more complicated large placesetting dataset, there is more need for the greater expressive power offered by the two-level models' pairwise features. This theory is consistent with the experiments in the last chapter, in which we saw that removing the pairwise features from the learned two-level models resulted in a significant drop in performance on the large placesetting dataset (Figure 4-11).

5.2.3 Results on the House Dataset

The cumulative and per-class results for three-level WGGs on the house dataset are shown in Figure 5-12 and Figure 5-13, respectively. Figure 5-14 shows the cumulative results, but visualized as a scatterplot. (Again, review the description of how the scatterplot was made in Section 5.2.1.) And Figure 5-15 shows several representative learned three-level structures.

The profile of the results on this dataset is consistent with the results on both datasets in the placesetting domain. Again, we see that three-level WGGs in which only a relatively small percentage of objects are assigned to composite classes have very similar performance to two-level WGGs. And again, as the number of objects assigned to composite classes grows, the performance of the three-level WGGs degrades by a fair amount.

Overall, the experiments in this chapter have demonstrated that adding an additional level of hierarchy to WGG models does not improve performance in general, and in fact can damage performance if applied too aggressively. We observe this trend consistently across all three datasets,



 # targets (all models): 6558.0

	recall	**precision**	**f-measure**	**# detections**	**# correct**
DTPBM, known # objs	0.214 [0.206 0.223]	0.216 [0.206 0.224]	0.215 [0.206 0.223]	6521.3	1406.0
DTPBM, heuristic # objs	0.337 [0.329 0.346]	0.053 [0.049 0.058]	0.092 [0.085 0.098]	41846.3	2210.0
2-level WGG	0.082 [0.073 0.089]	0.230 [0.185 0.256]	0.120 [0.105 0.132]	2347.3	535.3
3-level WGG:					
$\gamma_{\text{minsize}} = 0.1$, nosingletocomp	0.078 [0.071 0.087]	0.248 [0.209 0.273]	0.118 [0.106 0.130]	2077.3	511.7
$\gamma_{\text{minsize}} = 0.1$, singletocomp1	0.035 [0.033 0.038]	0.172 [0.132 0.226]	0.058 [0.057 0.059]	1443.0	232.7
$\gamma_{\text{minsize}} = 0.1$, singletocomp2	0.039 [0.036 0.044]	0.198 [0.144 0.291]	0.063 [0.058 0.069]	1417.3	253.0
$\gamma_{\text{minsize}} = 0.15$, nosingletocomp	0.083 [0.077 0.088]	0.247 [0.210 0.274]	0.124 [0.113 0.133]	2218.0	544.0
$\gamma_{\text{minsize}} = 0.15$, singletocomp1	0.054 [0.051 0.056]	0.219 [0.171 0.247]	0.086 [0.084 0.088]	1651.7	351.0
$\gamma_{\text{minsize}} = 0.15$, singletocomp2	0.059 [0.037 0.088]	0.195 [0.143 0.274]	0.090 [0.061 0.133]	1963.0	385.0
$\gamma_{\text{minsize}} = 0.2$, nosingletocomp	0.081 [0.076 0.089]	0.232 [0.190 0.256]	0.120 [0.109 0.132]	2317.7	532.0
$\gamma_{\text{minsize}} = 0.2$, singletocomp1	0.076 [0.064 0.089]	0.235 [0.194 0.256]	0.115 [0.102 0.132]	2161.3	500.7
$\gamma_{\text{minsize}} = 0.2$, singletocomp2	0.076 [0.064 0.089]	0.235 [0.194 0.256]	0.115 [0.102 0.132]	2161.3	500.7
$\gamma_{\text{minsize}} = 0.3$, nosingletocomp	0.082 [0.073 0.089]	0.230 [0.185 0.256]	0.120 [0.105 0.132]	2347.3	535.3
$\gamma_{\text{minsize}} = 0.3$, singletocomp1	0.082 [0.073 0.089]	0.230 [0.185 0.256]	0.120 [0.105 0.132]	2347.3	535.3
$\gamma_{\text{minsize}} = 0.3$, singletocomp2	0.082 [0.073 0.089]	0.230 [0.185 0.256]	0.120 [0.105 0.132]	2347.3	535.3

Figure 5-12: Cumulative results for the DTPBM object detector, the learned two-level WGG models from last chapter, and variations of learned three-level WGGs on the house dataset. γ_{minsize} is the percentage of training objects used to determine minimum cluster size. nosingletocomp means that single-object subtrees are not assigned to composite classes. singletocomp1 means that single-object subtrees were assigned to composite classes during Phase 1, while singletocomp2 means single-object subtrees were assigned to composite classes during Phase 2. All WGGs use pairwise tree structure features, class-based object detector features, and fixed ground-truth internal geometry (pairwise, classobj, fixed).

and even when using a hand-built three-level structure on the small placesetting dataset (Chapter 3). Futhermore, the learned structures appear, qualitatively, to be grouping the expected types of objects into composite classes (e.g., cups and saucers in placesettings, roofs and chimneys in houses, etc.).

These findings suggests that our learning algorithms are finding reasonable hierarchical structures, given our criteria. Instead, it seems that our assumptions about the advantages provided by these structures (e.g., better modeling of co-occurrence patterns and spatial relations) were not necessarily correct. In fact, learned two-level models with pairwise features provide significant ex-

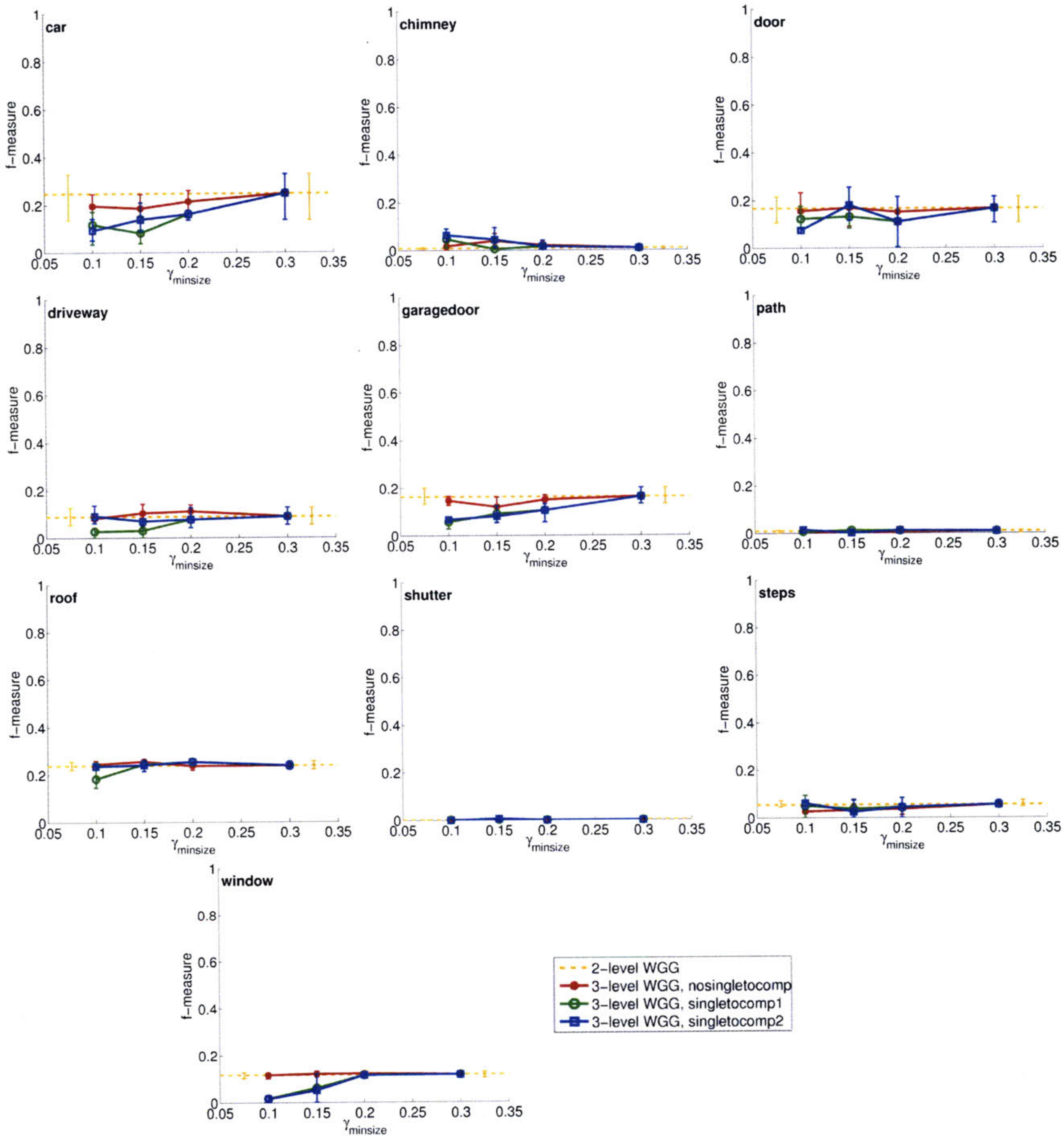


Figure 5-13: Per-class results for learned two-level and three-level WGG models on the house dataset. For numeric per-class results, see Appendix B (Tables B.52 and B.58-B.69).

pressive power, enough to capture both complex co-occurrence patterns and valuable information about expected spatial relationships among objects.

Next chapter, we consider avenues for future work in the WGG framework, and summarize the results and contributions of this thesis.

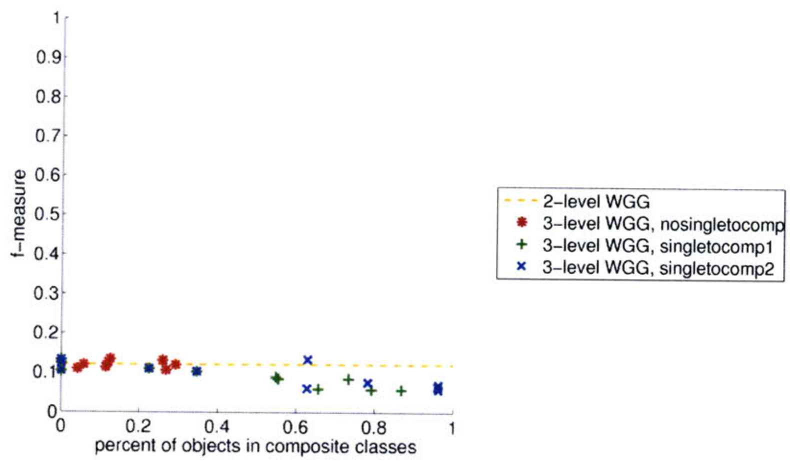


Figure 5-14: Test performance (cumulative f-measure) of each learned three-level WGG model on the house dataset, as a function of the percent of training objects assigned to composite classes during structure learning for that model.

```
sceneClasses: {'house'}
objClasses: {'car' 'chimney' 'door' 'driveway' 'garagedoor' 'path' 'roof' 'shutter' 'steps' 'window'}
house -> C00011 (182138)  C00011 (182139)  C00011 (182140)  C00011 (182141)  C00011 (182142)  C00011 (182143)
         C00011 (182144)  C00011 (182145)  C00011 (182146)  C00012 (182147)  C00012 (182148)  C00013 (182149)
         C00014 (182150)  C00014 (182151)  C00015 (182152)  C00015 (182153)  C00015 (182154)  C00016 (182155)
         C00016 (182156)  C00016 (182157)  C00016 (182158)  C00016 (182159)  C00016 (182160)  C00017 (182161)
         C00017 (182162)  C00018 (182163)  C00018 (182164)  C00019 (182165)  C00019 (182166)  C00020 (182167)
         C00020 (182168)  C00020 (182169)  C00020 (182170)  C00021 (182171)  C00022 (182172)  car (182056)
         car (182057)  chimney (182059)  chimney (182060)  chimney (182061)  door (182062)  door (182063)
         door (182064)  door (182065)  door (182066)  door (182067)  door (182068)  driveway (182069)
         driveway (182070)  garagedoor (182071)  garagedoor (182072)  garagedoor (182073)  garagedoor (182074)
         path (182075)  path (182076)  path (182077)  roof (182078)  roof (182079)  roof (182080)  roof (182081)
         roof (182082)  roof (182083)  roof (182084)  shutter (182085)  shutter (182086)  shutter (182087)
         shutter (182088)  shutter (182089)  shutter (182090)  shutter (182091)  shutter (182092)
         shutter (182093)  shutter (182094)  shutter (182095)  shutter (182096)  shutter (182097)
         shutter (182098)  shutter (182099)  shutter (182100)  shutter (182101)  shutter (182102)
         shutter (182103)  shutter (182104)  shutter (182105)  shutter (182106)  shutter (182107)
         shutter (182108)  steps (182109)  steps (182110)  window (182111)  window (182112)  window (182113)
         window (182114)  window (182115)  window (182116)  window (182117)  window (182118)  window (182119)
         window (182120)  window (182121)  window (182122)  window (182123)  window (182124)  window (182125)
         window (182126)  window (182127)  window (182128)  window (182129)  window (182130)  window (182131)
         window (182132)  window (182133)  window (182134)  window (182135)  window (182136)  window (182137)
C00011 -> door (8243)  shutter (8244)  window (23517)
C00016 -> door (35986)  shutter (35987)  window (35988)  window (35989)
C00019 -> door (69108)  steps (69109)  window (89835)
C00012 -> driveway (9385)  garagedoor (9386)  roof (11843)
C00017 -> car (28799)  driveway (28800)  garagedoor (28943)  roof (28944)
C00020 -> chimney (65707)  window (65708)
C00013 -> steps (25253)  window (25254)
C00018 -> chimney (32209)  window (32210)
C00014 -> chimney (2879)  roof (2880)
C00015 -> car (1211)  window (1212)  window (1221)
C00021 -> path (112945)  steps (112946)
C00022 -> driveway (109456)  window (109457)
```

$\gamma_{\text{minsize}} = 0.1$, nosingletocomp

```
sceneClasses: {'house'}
objClasses: {'car' 'chimney' 'door' 'driveway' 'garagedoor' 'path' 'roof' 'shutter' 'steps' 'window'}
house -> C00011 (118520)  C00011 (118521)  C00011 (118522)  C00011 (118523)  C00011 (118524)  C00011 (118525)
         C00011 (118526)  C00011 (118527)  C00011 (118528)  C00011 (118529)  C00011 (118530)  C00011 (118531)
         C00011 (118532)  C00011 (118533)  C00011 (118534)  C00011 (118535)  C00011 (118536)  C00011 (118537)
         C00011 (118538)  C00011 (118539)  C00011 (118540)  C00011 (118541)  C00011 (118542)  C00011 (118543)
         C00011 (118544)  C00011 (118545)  C00011 (118546)  C00011 (118547)  C00011 (118548)  C00011 (118549)
         C00012 (118550)  C00012 (118551)  C00012 (118552)  C00012 (118553)  C00013 (118554)  C00014 (118555)
         C00014 (118556)  C00014 (118557)  C00015 (118558)  C00015 (118559)  C00015 (118560)  C00015 (118561)
         C00015 (118562)  C00016 (118563)  car (118478)  car (118479)  chimney (118480)  chimney (118481)
         chimney (118482)  chimney (118483)  door (118484)  door (118485)  driveway (118486)  garagedoor (118487)
         garagedoor (118488)  path (118489)  path (118490)  path (118491)  roof (118492)  roof (118493)
         roof (118494)  roof (118495)  shutter (118496)  shutter (118497)  shutter (118498)  shutter (118499)
         shutter (118500)  shutter (118501)  shutter (118502)  shutter (118503)  shutter (118504)
         shutter (118505)  shutter (118506)  shutter (118507)  steps (118509)  steps (118510)
         steps (118511)  window (118512)  window (118513)  window (118514)  window (118515)  window (118516)
         window (118517)  window (118518)  window (118519)
C00011 -> door (8243)  shutter (8244)  window (23517)
C00015 -> car (1211)  window (1212)  window (1221)
C00012 -> driveway (9385)  garagedoor (9386)  roof (11843)
C00014 -> chimney (2879)  roof (2880)
C00013 -> steps (25253)  window (25254)
C00016 -> car (28639)  driveway (28640)
```

$\gamma_{\text{minsize}} = 0.1$, singletocomp1

Figure 5-15: Representative learned WGG three-level structures for the house dataset.

Chapter 6

Conclusions and Future Work

In this final chapter, we summarize the conclusions and contributions of this thesis. First, however, we consider future work.

6.1 Future Work

There are number of avenues that future work on WGG models could take.

Additional Features

A first such area would be to introduce additional types of image-based features, to reduce the model's sensitivity to noise in the object detector. For example, we saw in the experiments that class-based object detector features outperformed their rule-part-based counterparts in some object classes, sometimes dramatically, while the reverse was true in other cases. But in a feature-based discriminative model, we have the flexibility to simply include *both* class-based and rule-part-based object detector features. It would be interesting to see the effects of this simple extension.

The DTPBM object detector has the option to learn multiple components per class, rather the one per class we used in our experiments. Each component corresponds roughly to a different canonical aspect ratio for the object class, and outputs an independent score. Thus, another simple modification could use these components as additional object detector features.

Extending this idea, we could use simple clustering on visual features to learn subclasses for each object class, based on visual differences rather than just different aspect ratios. We could then define features on the scores from object detectors trained for each subclass.

But perhaps most promising category of image-based features we might add are global scene-centered features, such as gist features (Oliva & Torralba, 2001; Torralba, 2003) or global orientation histograms (Lazebnik et al., 2006). We could associate these global features with the entire image, but also with portions of the image, such as nodes corresponding to composite classes or object classes themselves. Such a representation could provide an important additional source of information about the image to overcome noise in the object detector's low-level signal. Furthermore, incorporating such features into the model framework would be simple, and would only require some thought to ensure that parsing can still be performed efficiently.

Although image-based features are crucial, we also might consider alternative forms of geometry features. The quadratic (log Gaussian) models we use in this thesis are powerful and convenient, but we demonstrated cases in which they cannot capture the spatial patterns that occur in the data. For example, they are somewhat poorly suited to localizing tiny chimneys, which can occur anywhere within a much larger roof region. In such cases, some notion of a region of uniform expectation might be a better fit. It is also worth exploring the types of binary relationships (above, below, etc.) that appear commonly in other work on contextual object detection. These relationships could mitigate some of the conditional independence effects of the quadratic features in our current model. The challenge would be incorporating or approximating such pairwise relationships while maintaining efficient inference.

Balancing Recall and Precision

As we have seen in the experiments, the structured perceptron seems to consistently find weights that result in very conservative interpretations of test images. The learned WGG models tend to make few detections, but those detections are often correct; this leads to relatively low recall but very high precision. We speculated in Section 3.2.4 that this trend occurs because the object detector produces such noisy output, while the inherent variance in spatial arrangements means that the geometry models cannot be informative enough to overcome the noise.

In most object detection systems, a per-class threshold on detector scores can be manually varied in order to trade off between recall and precision at test time. This is desirable because there are situations in which certain types of errors are more costly than others. In some contexts, false negatives may be more severe than false positives, while the reverse may be true in other cases. But the ability to perform manual tuning is also useful because it allows the system's designer to counteract tendencies in the learning algorithms to prefer one type of error over another, as seems to be happening in the perceptron for WGGs.

In a WGG model, on the other hand, there is no single global or per-class threshold to vary or manually tune in this way. All of the per-part and pairwise weights are set jointly during learning, along with the weights on geometry and object detector features, in order to minimize training error. To allow manual balancing of recall and precision, the most principled approach would incorporate into the perceptron's update step some type of asymmetric cost for each type of error.

There is precedent for using asymmetric costs in discriminative learning methods. For example, Viola and Jones (2002) used biased costs within the AdaBoost framework, with simple binary perceptrons as weak classifiers, to give one type of error (e.g., false negatives) more cost than another (false positives).

In the structured perceptron, we could achieve something similar by using, for each feature on tree structure, a different learning rate depending on the type of error made by the predicted tree for that feature on that step. The type of error (false positive or false negative) would be determined by looking at the sign of the difference between the detected and ground-truth feature values. (The updates to geometry and object detector features would be performed as usual.) By varying the magnitude of the learning rate on false positives versus false negatives, the learning process would hopefully lead to more or less conservative detections at test time, thus trading off between recall and precision.

Modeling in 3D

Finally, in the long term, we believe that effective modeling of the geometric relationships among objects requires 3D reasoning. We chose the scene classes considered in this thesis because they provided high levels of complexity while still offering a 2D framework some hope of success. However, for most scenes, the rich pattern in object interactions is most compactly captured in 3D. But again, the challenge is not how to incorporate such features into the WGG model, but rather what the features should be, and how to efficiently perform recognition from a single image using such features.

6.2 Conclusions and Contributions

This thesis has introduced a novel formalism, weighted geometric grammars, for flexibly representing and recognizing combinations of objects and their spatial relationships in scenes. The model can incorporate arbitrary image-based and geometry features, and compactly represents many combinations of possible objects through optional rule parts with per-part and pairwise structure weights. Because the geometry relationships are modeled in a tree-structured manner, we can perform parsing efficiently.

We adapted the structured perceptron algorithm to parameter learning in WGG models, and showed that it can find weights that effectively trade off among object detector output, geometric relationships, and co-occurrence patterns. Furthermore, the model's optional parts and pairwise weights means that the process of parameter learning actually performs a significant amount of "structure learning"—determining patterns of co-occurring objects—in an efficient manner.

To learn the grammar structure, we developed two sets of clustering-based algorithms. The first set learned two-level WGG structures, with no intermediate composite classes between the object and scene levels. The second set of algorithms learned these composite classes, producing hierarchical three-level WGGs.

We presented experiments using several variations on WGG models, comparing their performance to a state-of-the-art object detector. The results suggested that there is certainly value in capturing high-level co-occurrence and geometry information, but that it can often be represented well enough without hierarchy. In particular, it seems that sometimes pairwise weights are more valuable than hierarchical composite classes.

These findings are interesting, in part because they are somewhat surprising. Our instinct as computer scientists is often to throw hierarchy at every problem we encounter, and these results serve as a useful cautionary note. If there is valuable hierarchy in these datasets, our algorithms are not finding it. This may be because the current state of object detectors does not provide strong enough signal to support such deep learning yet. It may be because our algorithms for learning composite classes from the variance in spatial relationships are based on inaccurate assumptions, as we discussed last chapter. Or it may be that the domains we consider simply do not have the level of hierarchical structure we imagine.

Nonetheless, even when no hierarchy is learned, the structure learning algorithms in Chapter 4 provide a valuable function. They determine what the modes should be for the multimodal geometry models, and decide how many and which objects are present. And the perceptron algorithm learns weights that trade off between geometry and object detector scores, while learning context-specific thresholds and co-occurrence patterns. As a result, the non-hierarchical models are powerful enough to outperform the state-of-the-art DTPBM object detector.

Finally, this thesis contributes three new fully-labeled datasets, in two domains, to the scene recognition community. As the field continues to advance, we hope these challenging datasets will provide a source of inspiration to researchers in scene understanding, and that the performance levels demonstrated in this thesis can be improved upon further.

Appendix A

Datasets

This appendix contains supplementary information about the three datasets used in this thesis.

Figure A-1: Images from the small placesetting dataset.

static_silverware	001.jpg
static_silverware	04150013_g.jpg
static_silverware	2-plates.jpg
static_silverware	20020923_18.jpg
static_silverware	20020927_26.jpg
static_silverware	20020927_28.jpg
static_silverware	2004122010446.jpg
static_silverware	22758027.jpg
static_silverware	2844400500.jpg
static_silverware	3ofknifeandfork.jpg
static_silverware	3settings2.jpg
static_silverware	54256.jpg
static_silverware	IMG_0220.jpg
static_silverware	assiette.jpg
static_silverware	assiette2.jpg
static_silverware	carbonada3.jpg
static_silverware	cb001397.jpg
static_silverware	cb049907.jpg
static_silverware	curry.jpg
static_silverware	dinnerfor1_pewterplates_jan05_red.jpg
static_silverware	draw.jpg
static_silverware	dsgiants.jpg
static_silverware	dsiowa.jpg
static_silverware	fotoseptiembre.jpg
static_silverware	hayegg.jpg
static_silverware	homemade-pasta-dish.jpg
static_silverware	jdw2.jpg
static_silverware	jdw3.jpg
static_silverware	jwd1.jpg
static_silverware	mcu016.jpg
static_silverware	mcu036.jpg
static_silverware	obj95.jpg
static_silverware	obj98.jpg
static_silverware	p1010665.jpg
static_silverware	p1010666.jpg
static_silverware	p1010667.jpg
static_silverware	p1010668.jpg
static_silverware	p1010669.jpg
static_silverware	p1010670.jpg
static_silverware	p1010671.jpg
static_silverware	p1010672.jpg
static_silverware	p1010673.jpg
static_silverware	p1010674.jpg
static_silverware	p1010676.jpg
static_silverware	p1010928.jpg
static_silverware	p1010929.jpg
static_silverware	p1010930.jpg
static_silverware	p9100035.jpg
static_silverware	p9110007.jpg
static_silverware	photo-steak-assiette.jpg
static_silverware	placesetting.jpg
static_silverware	plate.jpg
static_silverware	plate1.jpg
static_silverware	plate2.jpg
static_silverware	plateandsilverware.jpg
static_silverware	plates-01.jpg

Figure A-2: LabelMe directories and filenames for the small placesetting dataset.

static_silverware	platocubiertos.jpg
static_silverware	proto-91-887082.jpg
static_silverware	set1.jpg
static_silverware	set2.jpg
static_silverware	set3.jpg
static_silverware	set4.jpg
static_silverware	set5.jpg
static_silverware	set6.jpg
static_silverware	set7.jpg
static_silverware	shoe_plates_jun_04.jpg
static_silverware	silverware.jpg
static_silverware	simpsonstable.jpg
static_silverware	stainlessware.jpg
static_silverware	stor_tallrik.jpg
static_silverware	vajilla.jpg
static_silverware	vajillas_4.jpg
static_silverware	whitechina.jpg
static_web_submitted_meg_placesettings	placesetting_0001.jpg
static_web_submitted_meg_placesettings	placesetting_0002.jpg
static_web_submitted_meg_placesettings	placesetting_0003.jpg
static_web_submitted_meg_placesettings	placesetting_0004.jpg
static_web_submitted_meg_placesettings	placesetting_0005.jpg
static_web_submitted_meg_placesettings	placesetting_0006.jpg
static_web_submitted_meg_placesettings	placesetting_0007.jpg
static_web_submitted_meg_placesettings	placesetting_0008.jpg
static_web_submitted_meg_placesettings	placesetting_0010.jpg
static_web_submitted_meg_placesettings	placesetting_0013.jpg
static_web_submitted_meg_placesettings	placesetting_0014.jpg
static_web_submitted_meg_placesettings	placesetting_0015.jpg
static_web_submitted_meg_placesettings	placesetting_0018.jpg
static_web_submitted_meg_placesettings	placesetting_0019.jpg
static_web_submitted_meg_placesettings	placesetting_0021.jpg
static_web_submitted_meg_placesettings	placesetting_0022.jpg
static_web_submitted_meg_placesettings	placesetting_0028.jpg
static_web_submitted_meg_placesettings	placesetting_0029.jpg
static_web_submitted_meg_placesettings	placesetting_0030.jpg
static_web_submitted_meg_placesettings	placesetting_0033.jpg
static_web_submitted_meg_placesettings	placesetting_0034.jpg
static_web_submitted_meg_placesettings	placesetting_0036.jpg
static_web_submitted_meg_placesettings	placesetting_0037.jpg
static_web_submitted_meg_placesettings	placesetting_0038.jpg
static_web_submitted_meg_placesettings	placesetting_0039.jpg
static_web_submitted_meg_placesettings	placesetting_0043.jpg
static_web_submitted_meg_placesettings	placesetting_0046.jpg
static_web_submitted_meg_placesettings	placesetting_0047.jpg
static_web_submitted_meg_placesettings	placesetting_0048.jpg
static_web_submitted_meg_placesettings	placesetting_0049.jpg
static_web_submitted_meg_placesettings	placesetting_0052.jpg
static_web_submitted_meg_placesettings	placesetting_0053.jpg
static_web_submitted_meg_placesettings	placesetting_0054.jpg
static_web_submitted_meg_placesettings	placesetting_0056.jpg
static_web_submitted_meg_placesettings	placesetting_0059.jpg
static_web_submitted_meg_placesettings	placesetting_0060.jpg
static_web_submitted_meg_placesettings	placesetting_0063.jpg
static_web_submitted_meg_placesettings	placesetting_0066.jpg
static_web_submitted_meg_placesettings	placesetting_0067.jpg

Figure A-3: LabelMe directories and filenames for the small placesetting dataset *(cont.)*.

static_web_submitted_meg_placesettings	placesetting_0068.jpg
static_web_submitted_meg_placesettings	placesetting_0071.jpg
static_web_submitted_meg_placesettings	placesetting_0072.jpg
static_web_submitted_meg_placesettings	placesetting_0073.jpg
static_web_submitted_meg_placesettings	placesetting_0074.jpg
static_web_submitted_meg_placesettings	placesetting_0075.jpg
static_web_submitted_meg_placesettings	placesetting_0076.jpg
static_web_submitted_meg_placesettings	placesetting_0077.jpg
static_web_submitted_meg_placesettings	placesetting_0078.jpg
static_web_submitted_meg_placesettings	placesetting_0080.jpg
static_web_submitted_meg_placesettings	placesetting_0081.jpg
static_web_submitted_meg_placesettings	placesetting_0082.jpg
static_web_submitted_meg_placesettings	placesetting_0083.jpg
static_web_submitted_meg_placesettings	placesetting_0084.jpg
static_web_submitted_meg_placesettings	placesetting_0087.jpg
static_web_submitted_meg_placesettings	placesetting_0088.jpg
static_web_submitted_meg_placesettings	placesetting_0089.jpg
static_web_submitted_meg_placesettings	placesetting_0090.jpg
static_web_submitted_meg_placesettings	placesetting_0092.jpg
static_web_submitted_meg_placesettings	placesetting_0094.jpg
static_web_submitted_meg_placesettings	placesetting_0096.jpg
static_web_submitted_meg_placesettings	placesetting_0097.jpg
static_web_submitted_meg_placesettings	placesetting_0098.jpg
static_web_submitted_meg_placesettings	placesetting_0099.jpg
static_web_submitted_meg_placesettings	placesetting_0100.jpg
static_web_submitted_meg_placesettings	placesetting_0101.jpg
static_web_submitted_meg_placesettings	placesetting_0102.jpg
static_web_submitted_meg_placesettings	placesetting_0103.jpg
static_web_submitted_meg_placesettings	placesetting_0104.jpg
static_web_submitted_meg_placesettings	placesetting_0105.jpg
static_web_submitted_meg_placesettings	placesetting_0106.jpg
static_web_submitted_meg_placesettings	placesetting_0107.jpg
static_web_submitted_meg_placesettings	placesetting_0108.jpg
static_web_submitted_meg_placesettings	placesetting_0109.jpg
static_web_submitted_meg_placesettings	placesetting_0112.jpg
static_web_submitted_meg_placesettings	placesetting_0113.jpg
static_web_submitted_meg_placesettings	placesetting_0114.jpg
static_web_submitted_meg_placesettings	placesetting_0115.jpg
static_web_submitted_meg_placesettings	placesetting_0117.jpg
static_web_submitted_meg_placesettings	placesetting_0118.jpg
static_web_submitted_meg_placesettings	placesetting_0121.jpg
static_web_submitted_meg_placesettings	placesetting_0122.jpg
static_web_submitted_meg_placesettings	placesetting_0124.jpg
static_web_submitted_meg_placesettings	placesetting_0126.jpg
static_web_submitted_meg_placesettings	placesetting_0129.jpg
static_web_submitted_meg_placesettings	placesetting_0130.jpg
static_web_submitted_meg_placesettings	placesetting_0137.jpg
static_web_submitted_meg_placesettings	placesetting_0145.jpg
static_web_submitted_meg_placesettings	placesetting_0159.jpg
static_web_submitted_meg_placesettings	placesetting_0189.jpg
static_web_submitted_meg_placesettings	placesetting_0196.jpg
static_web_submitted_meg_placesettings	placesetting_0201.jpg
static_web_submitted_meg_placesettings	placesetting_0202.jpg
static_web_submitted_meg_placesettings	placesetting_0203.jpg
static_web_submitted_meg_placesettings	placesetting_0204.jpg

Figure A-4: LabelMe directories and filenames for the small placesetting dataset *(cont.)*.

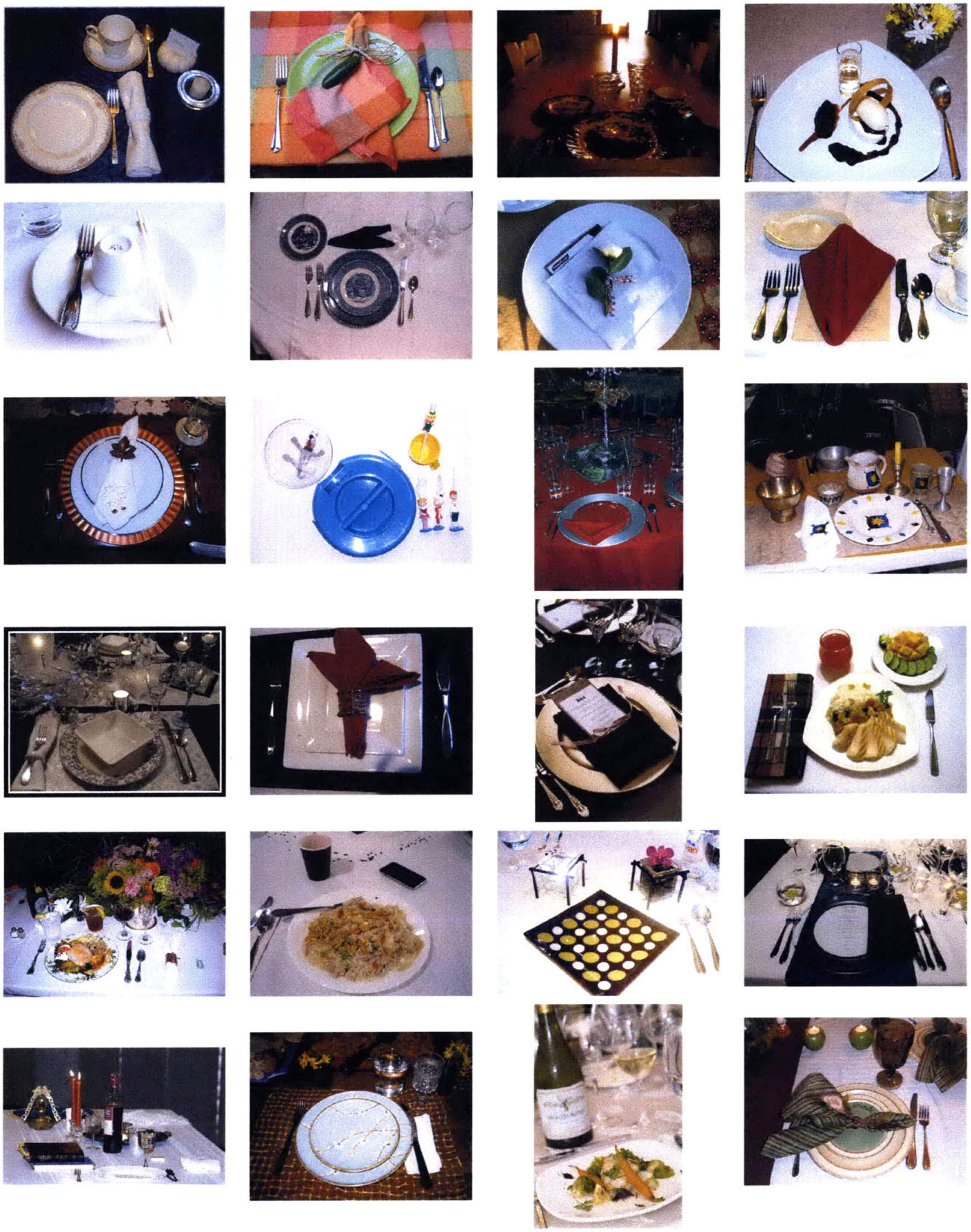


Figure A-5: Images from the big placesetting dataset.

Figure A-6: Images from the house dataset.

Appendix B

Results

This appendix lists the complete averaged results for all sets of experiments presented in this thesis. The first three tables (B.1, B.2, B.3) summarize for each dataset the cumulative results across *all* models. Each line in these tables then points to the table presenting the per-class results for that individual set of experiments.

All results on the small placesetting dataset were averaged over 5 runs, while all results on the big placesetting and house datasets were averaged over 3 runs. In all cases, the minimum and maximum scores across runs are reported, in addition to the mean.

The following abbreviations are used in the tables:

DTPBM object detector	
known # objects	The ground-truth number of objects of each class in each image is used in selecting detections. (Remember that this is cheating!)
heuristic # objects	The maximum number of objects of each class seen in the training images is used in selecting detections.
Hand-built and 2-level WGGs	
pairwise	Pairwise tree structure weights are used.
nopairwise	Pairwise tree structure weights are not used.
classobj	Class-based object detector features are used.
rpobj	Rule-part-based object detector features are used.
fixed	Fixed ground-truth internal geometry is used in the perceptron.
free	Free ground-truth internal geometry is used in the perceptron.
3-level WGGs	
$\gamma_{minsize}$	The percentage of training objects used to determine minimum cluster size.
nosingletocomp	Single-object subtrees are not assigned to composite classes.
singletocomp1	Single-object subtrees are assigned to composite classes during Phase 1.
singletocomp2	Single-object subtrees are assigned to composite classes during Phase 2.

All experiments with three-level WGG structures were run with pairwise weights, class-based object detector features, and fixed ground-truth internal geometry (pairwise, classobj, fixed).

		recall	precision	f-measure	Table
DTPBM obj detector	known # objects	0.380 [0.341 0.420]	0.571 [0.521 0.598]	0.456 [0.412 0.493]	B.4
	heuristic # objects	0.415 [0.376 0.447]	0.138 [0.117 0.171]	0.205 [0.184 0.235]	B.5
WGG, hand-built	pairwise, classobj, fixed	0.254 [0.244 0.264]	0.604 [0.561 0.647]	0.357 [0.353 0.362]	B.6
	pairwise, rpobj, fixed	0.235 [0.221 0.253]	0.659 [0.623 0.705]	0.346 [0.332 0.359]	B.7
	pairwise, classobj, free	0.278 [0.237 0.306]	0.561 [0.547 0.571]	0.371 [0.334 0.397]	B.8
	pairwise, rpobj, free	0.282 [0.256 0.304]	0.608 [0.576 0.637]	0.385 [0.357 0.409]	B.9
	nopairwise, classobj, fixed	0.242 [0.218 0.260]	0.623 [0.584 0.643]	0.348 [0.325 0.361]	B.10
	nopairwise, rpobj, fixed	0.217 [0.208 0.227]	0.698 [0.627 0.741]	0.331 [0.321 0.348]	B.11
WGG, 2-level	pairwise, classobj, fixed	0.278 [0.248 0.304]	0.565 [0.512 0.601]	0.372 [0.346 0.404]	B.12
	pairwise, rpobj, fixed	0.276 [0.258 0.294]	0.596 [0.571 0.621]	0.377 [0.360 0.399]	B.13
	pairwise, classobj, free	0.267 [0.233 0.289]	0.569 [0.513 0.614]	0.362 [0.338 0.383]	B.14
	pairwise, rpobj, free	0.273 [0.241 0.290]	0.612 [0.592 0.631]	0.377 [0.346 0.395]	B.15
	nopairwise, classobj, fixed	0.264 [0.231 0.284]	0.593 [0.549 0.631]	0.365 [0.333 0.387]	B.16
	nopairwise, rpobj, fixed	0.248 [0.226 0.272]	0.654 [0.587 0.681]	0.359 [0.338 0.385]	B.17
WGG, 3-level	$\gamma_{minsize} = 0.1$, nosingletocomp	0.235 [0.203 0.263]	0.606 [0.563 0.682]	0.338 [0.301 0.380]	B.18
	$\gamma_{minsize} = 0.1$, singletocomp1	0.215 [0.173 0.244]	0.610 [0.545 0.645]	0.316 [0.272 0.349]	B.19
	$\gamma_{minsize} = 0.1$, singletocomp2	0.212 [0.180 0.238]	0.619 [0.576 0.679]	0.314 [0.285 0.346]	B.20
	$\gamma_{minsize} = 0.15$, nosingletocomp	0.253 [0.222 0.282]	0.597 [0.532 0.639]	0.354 [0.330 0.386]	B.21
	$\gamma_{minsize} = 0.15$, singletocomp1	0.236 [0.200 0.277]	0.614 [0.558 0.644]	0.340 [0.294 0.382]	B.22
	$\gamma_{minsize} = 0.15$, singletocomp2	0.235 [0.208 0.259]	0.612 [0.562 0.644]	0.339 [0.303 0.366]	B.23
	$\gamma_{minsize} = 0.2$, nosingletocomp	0.258 [0.232 0.289]	0.582 [0.524 0.637]	0.356 [0.333 0.397]	B.24
	$\gamma_{minsize} = 0.2$, singletocomp1	0.245 [0.200 0.282]	0.586 [0.531 0.619]	0.345 [0.294 0.386]	B.25
	$\gamma_{minsize} = 0.2$, singletocomp2	0.239 [0.189 0.282]	0.594 [0.560 0.617]	0.340 [0.284 0.386]	B.26
	$\gamma_{minsize} = 0.3$, nosingletocomp	0.265 [0.241 0.292]	0.572 [0.540 0.605]	0.362 [0.340 0.393]	B.27
	$\gamma_{minsize} = 0.3$, singletocomp1	0.269 [0.257 0.287]	0.579 [0.531 0.603]	0.367 [0.346 0.381]	B.28
	$\gamma_{minsize} = 0.3$, singletocomp2	0.269 [0.257 0.287]	0.579 [0.531 0.603]	0.367 [0.346 0.381]	B.29

Table B.1: Cumulative results on the **small placesetting dataset**.

		recall	precision	f-measure	Table
DTPBM obj detector	known # objects	0.387 [0.381 0.390]	0.432 [0.428 0.437]	0.408 [0.403 0.412]	B.30
	heuristic # objects	0.475 [0.465 0.482]	0.079 [0.075 0.085]	0.135 [0.130 0.144]	B.31
WGG, 2-level	pairwise, classobj, fixed	0.267 [0.255 0.274]	0.499 [0.492 0.508]	0.348 [0.336 0.356]	B.32
	pairwise, rpobj, fixed	0.212 [0.201 0.220]	0.579 [0.569 0.587]	0.311 [0.297 0.320]	B.33
	pairwise, classobj, free	0.249 [0.231 0.264]	0.504 [0.471 0.562]	0.332 [0.327 0.338]	B.34
	pairwise, rpobj, free	0.214 [0.205 0.219]	0.595 [0.588 0.603]	0.314 [0.305 0.322]	B.35
	nopairwise, classobj, fixed	0.209 [0.206 0.212]	0.580 [0.550 0.612]	0.307 [0.303 0.315]	B.36
	nopairwise, rpobj, fixed	0.178 [0.172 0.184]	0.682 [0.618 0.722]	0.282 [0.278 0.284]	B.37
WGG, 3-level	$\gamma_{minsize} = 0.1$, nosingletocomp	0.242 [0.237 0.250]	0.535 [0.519 0.546]	0.333 [0.325 0.343]	B.38
	$\gamma_{minsize} = 0.1$, singletocomp1	0.154 [0.114 0.193]	0.494 [0.410 0.543]	0.234 [0.178 0.285]	B.39
	$\gamma_{minsize} = 0.1$, singletocomp2	0.166 [0.132 0.194]	0.611 [0.536 0.671]	0.261 [0.211 0.301]	B.40
	$\gamma_{minsize} = 0.15$, nosingletocomp	0.245 [0.240 0.251]	0.521 [0.514 0.527]	0.333 [0.328 0.340]	B.41
	$\gamma_{minsize} = 0.15$, singletocomp1	0.189 [0.164 0.209]	0.496 [0.462 0.526]	0.273 [0.242 0.300]	B.42
	$\gamma_{minsize} = 0.15$, singletocomp2	0.177 [0.161 0.204]	0.535 [0.498 0.600]	0.266 [0.244 0.304]	B.43
	$\gamma_{minsize} = 0.2$, nosingletocomp	0.251 [0.238 0.258]	0.500 [0.487 0.516]	0.334 [0.320 0.344]	B.44
	$\gamma_{minsize} = 0.2$, singletocomp1	0.201 [0.155 0.232]	0.455 [0.393 0.492]	0.278 [0.222 0.313]	B.45
	$\gamma_{minsize} = 0.2$, singletocomp2	0.217 [0.204 0.235]	0.458 [0.456 0.461]	0.295 [0.283 0.310]	B.46
	$\gamma_{minsize} = 0.3$, nosingletocomp	0.261 [0.252 0.266]	0.498 [0.489 0.511]	0.342 [0.333 0.348]	B.47
	$\gamma_{minsize} = 0.3$, singletocomp1	0.248 [0.229 0.274]	0.491 [0.485 0.499]	0.329 [0.313 0.350]	B.48
	$\gamma_{minsize} = 0.3$, singletocomp2	0.241 [0.211 0.271]	0.480 [0.463 0.490]	0.320 [0.290 0.348]	B.49

Table B.2: Cumulative results on the **big placesetting dataset**.

		recall	precision	f-measure	Table
DTPBM obj detector	known # objects	0.214 [0.206 0.223]	0.216 [0.206 0.224]	0.215 [0.206 0.223]	B.50
	heuristic # objects	0.337 [0.329 0.346]	0.053 [0.049 0.058]	0.092 [0.085 0.098]	B.51
WGG, 2-level	pairwise, classobj, fixed	0.082 [0.073 0.089]	0.230 [0.185 0.256]	0.120 [0.105 0.132]	B.52
	pairwise, rpobj, fixed	0.053 [0.051 0.054]	0.215 [0.177 0.237]	0.085 [0.082 0.088]	B.53
	pairwise, classobj, free	0.080 [0.074 0.086]	0.234 [0.191 0.269]	0.119 [0.107 0.130]	B.54
	pairwise, rpobj, free	0.050 [0.049 0.052]	0.203 [0.161 0.231]	0.080 [0.078 0.082]	B.55
	nopairwise, classobj, fixed	0.043 [0.038 0.051]	0.392 [0.355 0.421]	0.078 [0.069 0.089]	B.56
	nopairwise, rpobj, fixed	0.038 [0.037 0.039]	0.418 [0.410 0.426]	0.069 [0.067 0.071]	B.57
WGG, 3-level	$\gamma_{minsize} = 0.1$, nosingletocomp	0.078 [0.071 0.087]	0.248 [0.209 0.273]	0.118 [0.106 0.130]	B.58
	$\gamma_{minsize} = 0.1$, singletocomp1	0.035 [0.033 0.038]	0.172 [0.132 0.226]	0.058 [0.057 0.059]	B.59
	$\gamma_{minsize} = 0.1$, singletocomp2	0.039 [0.036 0.044]	0.198 [0.144 0.291]	0.063 [0.058 0.069]	B.60
	$\gamma_{minsize} = 0.15$, nosingletocomp	0.083 [0.077 0.088]	0.247 [0.210 0.274]	0.124 [0.113 0.133]	B.61
	$\gamma_{minsize} = 0.15$, singletocomp1	0.054 [0.051 0.056]	0.219 [0.171 0.247]	0.086 [0.084 0.088]	B.62
	$\gamma_{minsize} = 0.15$, singletocomp2	0.059 [0.037 0.088]	0.195 [0.143 0.274]	0.090 [0.061 0.133]	B.63
	$\gamma_{minsize} = 0.2$, nosingletocomp	0.081 [0.076 0.089]	0.232 [0.190 0.256]	0.120 [0.109 0.132]	B.64
	$\gamma_{minsize} = 0.2$, singletocomp1	0.076 [0.064 0.089]	0.235 [0.194 0.256]	0.115 [0.102 0.132]	B.65
	$\gamma_{minsize} = 0.2$, singletocomp2	0.076 [0.064 0.089]	0.235 [0.194 0.256]	0.115 [0.102 0.132]	B.66
	$\gamma_{minsize} = 0.3$, nosingletocomp	0.082 [0.073 0.089]	0.230 [0.185 0.256]	0.120 [0.105 0.132]	B.67
	$\gamma_{minsize} = 0.3$, singletocomp1	0.082 [0.073 0.089]	0.230 [0.185 0.256]	0.120 [0.105 0.132]	B.68
	$\gamma_{minsize} = 0.3$, singletocomp2	0.082 [0.073 0.089]	0.230 [0.185 0.256]	0.120 [0.105 0.132]	B.69

Table B.3: Cumulative results on the **house dataset**.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.103 [0.037 0.250]	0.775 [0.429 1.000]	0.154 [0.071 0.320]	25.4	5.6	2.8
candle	0.314 [0.111 0.455]	0.314 [0.111 0.455]	0.314 [0.111 0.455]	14.6	14.6	4.6
cup	0.387 [0.310 0.429]	0.675 [0.429 0.900]	0.476 [0.429 0.545]	21.2	12.4	8.0
fork	0.604 [0.563 0.639]	0.749 [0.705 0.848]	0.667 [0.655 0.677]	121.6	98.8	73.4
forkside	0.027 [0.000 0.059]	0.027 [0.000 0.059]	0.027 [0.000 0.059]	21.4	21.4	0.6
glass	0.522 [0.477 0.612]	0.597 [0.486 0.768]	0.555 [0.481 0.681]	100.0	89.0	52.2
knife	0.271 [0.202 0.330]	0.339 [0.310 0.400]	0.299 [0.259 0.361]	90.8	73.0	24.6
knifeside	0.013 [0.000 0.067]	0.013 [0.000 0.067]	0.013 [0.000 0.067]	13.8	13.6	0.2
napkin	0.060 [0.000 0.136]	0.693 [0.364 1.000]	0.100 [0.000 0.214]	46.8	5.8	2.8
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.364 [0.270 0.497]	0.948 [0.910 1.000]	0.519 [0.425 0.643]	154.8	59.0	55.6
saucer	0.494 [0.333 0.643]	0.878 [0.714 1.000]	0.618 [0.467 0.750]	13.6	7.4	6.4
shaker	0.324 [0.176 0.389]	0.324 [0.176 0.389]	0.324 [0.176 0.389]	17.8	17.8	5.8
spoon	0.626 [0.516 0.704]	0.785 [0.706 0.896]	0.691 [0.649 0.775]	60.0	48.2	37.4
spoonside	0.269 [0.030 0.500]	0.273 [0.030 0.500]	0.271 [0.030 0.500]	27.4	26.4	6.4
cumulative	0.380 [0.341 0.420]	0.571 [0.521 0.598]	0.456 [0.412 0.493]	741.0	493.0	280.8

Table B.4: DTPBM object detector, small placesetting dataset, known # objects.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.109 [0.037 0.281]	0.367 [0.097 1.000]	0.107 [0.069 0.200]	25.4	19.4	3.0
candle	0.314 [0.111 0.455]	0.016 [0.003 0.022]	0.030 [0.006 0.042]	14.6	294.4	4.6
cup	0.438 [0.310 0.643]	0.259 [0.058 0.600]	0.268 [0.106 0.409]	21.2	60.0	8.8
fork	0.645 [0.611 0.683]	0.421 [0.345 0.510]	0.504 [0.450 0.558]	121.6	192.4	78.4
forkside	0.045 [0.000 0.074]	0.004 [0.000 0.008]	0.008 [0.000 0.015]	21.4	252.4	1.0
glass	0.591 [0.484 0.657]	0.275 [0.141 0.464]	0.358 [0.232 0.531]	100.0	274.0	59.4
knife	0.357 [0.261 0.437]	0.164 [0.130 0.216]	0.218 [0.195 0.241]	90.8	209.8	32.4
knifeside	0.013 [0.000 0.067]	0.003 [0.000 0.013]	0.004 [0.000 0.021]	13.8	108.6	0.2
napkin	0.060 [0.000 0.136]	0.419 [0.000 1.000]	0.091 [0.000 0.179]	46.8	9.4	2.8
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.374 [0.270 0.524]	0.913 [0.843 1.000]	0.522 [0.425 0.647]	154.8	63.4	57.2
saucer	0.506 [0.286 0.875]	0.351 [0.096 0.667]	0.343 [0.173 0.519]	13.6	31.0	6.2
shaker	0.324 [0.176 0.389]	0.041 [0.025 0.063]	0.072 [0.044 0.107]	17.8	143.2	5.8
spoon	0.647 [0.516 0.722]	0.449 [0.277 0.635]	0.511 [0.400 0.614]	60.0	94.4	38.6
spoonside	0.362 [0.061 0.636]	0.025 [0.003 0.066]	0.041 [0.006 0.087]	27.4	511.2	8.4
cumulative	0.415 [0.376 0.447]	0.138 [0.117 0.171]	0.205 [0.184 0.235]	741.0	2263.6	306.8

Table B.5: DTPBM object detector, small placesetting dataset, heuristic # objects.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.045 [0.000 0.094]	0.599 [0.143 1.000]	0.077 [0.000 0.162]	25.4	3.4	1.2
candle	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	14.6	0.0	0.0
cup	0.082 [0.034 0.143]	1.000 [1.000 1.000]	0.150 [0.067 0.250]	21.2	1.6	1.6
fork	0.349 [0.325 0.381]	0.514 [0.417 0.592]	0.414 [0.369 0.455]	121.6	83.6	42.4
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.275 [0.243 0.297]	0.660 [0.578 0.738]	0.388 [0.342 0.413]	100.0	41.6	27.4
knife	0.257 [0.170 0.330]	0.348 [0.242 0.425]	0.295 [0.200 0.371]	90.8	66.6	23.4
knifeside	0.000 [0.000 0.000]	0.600 [0.000 1.000]	0.000 [0.000 0.000]	13.8	0.8	0.0
napkin	0.034 [0.000 0.068]	1.000 [1.000 1.000]	0.064 [0.000 0.128]	46.8	1.6	1.6
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.371 [0.305 0.413]	0.907 [0.855 0.981]	0.524 [0.465 0.566]	154.8	63.0	57.0
saucer	0.135 [0.048 0.250]	1.000 [1.000 1.000]	0.232 [0.091 0.400]	13.6	1.6	1.6
shaker	0.074 [0.000 0.130]	0.650 [0.500 1.000]	0.125 [0.000 0.222]	17.8	2.4	1.4
spoon	0.499 [0.422 0.574]	0.673 [0.544 0.805]	0.568 [0.524 0.635]	60.0	45.0	29.8
spoonside	0.030 [0.000 0.105]	0.933 [0.667 1.000]	0.054 [0.000 0.182]	27.4	0.8	0.6
cumulative	0.254 [0.244 0.264]	0.604 [0.561 0.647]	0.357 [0.353 0.362]	741.0	312.0	188.0

Table B.6: Hand-built WGG structure, small placesetting dataset, pairwise, classobj, fixed.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.033 [0.000 0.048]	0.533 [0.333 1.000]	0.059 [0.000 0.083]	25.4	2.0	0.8
candle	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	14.6	0.0	0.0
cup	0.120 [0.059 0.174]	0.840 [0.400 1.000]	0.202 [0.111 0.286]	21.2	3.4	2.6
fork	0.317 [0.260 0.365]	0.543 [0.453 0.636]	0.396 [0.354 0.430]	121.6	72.6	38.6
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.192 [0.148 0.242]	0.686 [0.656 0.708]	0.298 [0.244 0.355]	100.0	27.8	19.0
knife	0.237 [0.182 0.297]	0.430 [0.356 0.488]	0.304 [0.241 0.351]	90.8	50.2	21.6
knifeside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	13.8	0.0	0.0
napkin	0.033 [0.000 0.057]	0.633 [0.000 1.000]	0.063 [0.000 0.102]	46.8	2.6	1.6
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.373 [0.310 0.413]	0.951 [0.900 1.000]	0.535 [0.474 0.573]	154.8	60.4	57.4
saucer	0.181 [0.100 0.267]	0.900 [0.500 1.000]	0.287 [0.182 0.421]	13.6	2.8	2.4
shaker	0.050 [0.000 0.130]	0.750 [0.500 1.000]	0.087 [0.000 0.222]	17.8	1.6	1.0
spoon	0.468 [0.438 0.500]	0.709 [0.659 0.789]	0.562 [0.549 0.594]	60.0	39.6	28.0
spoonside	0.048 [0.000 0.136]	0.880 [0.400 1.000]	0.081 [0.000 0.240]	27.4	1.6	1.0
cumulative	0.235 [0.221 0.253]	0.659 [0.623 0.705]	0.346 [0.332 0.359]	741.0	264.6	174.0

Table B.7: Hand-built WGG structure, small placesetting dataset, pairwise, rpobj, fixed.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.047 [0.000 0.080]	0.292 [0.000 0.667]	0.080 [0.000 0.143]	25.4	4.0	1.2
candle	0.000 [0.000 0.000]	0.800 [0.000 1.000]	0.000 [0.000 0.000]	14.6	1.4	0.0
cup	0.118 [0.069 0.143]	0.830 [0.400 1.000]	0.201 [0.129 0.231]	21.2	3.2	2.4
fork	0.354 [0.328 0.380]	0.472 [0.382 0.537]	0.404 [0.353 0.438]	121.6	91.8	43.0
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.360 [0.315 0.388]	0.656 [0.574 0.741]	0.464 [0.417 0.510]	100.0	54.8	35.8
knife	0.276 [0.202 0.319]	0.304 [0.232 0.341]	0.289 [0.216 0.330]	90.8	82.0	25.0
knifeside	0.000 [0.000 0.000]	0.800 [0.000 1.000]	0.000 [0.000 0.000]	13.8	0.2	0.0
napkin	0.029 [0.000 0.057]	0.787 [0.333 1.000]	0.054 [0.000 0.103]	46.8	2.6	1.4
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.400 [0.270 0.469]	0.890 [0.842 0.959]	0.547 [0.422 0.612]	154.8	69.2	61.2
saucer	0.192 [0.095 0.250]	0.830 [0.400 1.000]	0.297 [0.174 0.353]	13.6	3.2	2.4
shaker	0.086 [0.000 0.188]	0.750 [0.500 1.000]	0.135 [0.000 0.273]	17.8	2.8	1.6
spoon	0.499 [0.452 0.574]	0.624 [0.596 0.667]	0.554 [0.523 0.588]	60.0	47.8	29.8
spoonside	0.059 [0.000 0.158]	0.520 [0.000 1.000]	0.098 [0.000 0.250]	27.4	2.4	1.2
cumulative	0.278 [0.237 0.306]	0.561 [0.547 0.571]	0.371 [0.334 0.397]	741.0	365.4	205.0

Table B.8: Hand-built WGG structure, small placesetting dataset, pairwise, classobj, free.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.055 [0.000 0.094]	0.263 [0.000 0.500]	0.090 [0.000 0.158]	25.4	4.4	1.4
candle	0.000 [0.000 0.000]	0.800 [0.000 1.000]	0.000 [0.000 0.000]	14.6	0.4	0.0
cup	0.196 [0.130 0.261]	0.805 [0.500 1.000]	0.304 [0.231 0.400]	21.2	5.2	4.0
fork	0.344 [0.244 0.420]	0.505 [0.455 0.562]	0.405 [0.319 0.481]	121.6	82.8	41.8
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.325 [0.280 0.379]	0.758 [0.627 0.868]	0.451 [0.417 0.506]	100.0	43.2	32.4
knife	0.298 [0.234 0.368]	0.345 [0.275 0.411]	0.319 [0.253 0.370]	90.8	78.4	27.0
knifeside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	13.8	0.0	0.0
napkin	0.043 [0.000 0.091]	0.467 [0.000 1.000]	0.075 [0.000 0.154]	46.8	4.6	2.0
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.393 [0.328 0.441]	0.946 [0.903 1.000]	0.554 [0.494 0.606]	154.8	64.0	60.4
saucer	0.280 [0.143 0.400]	0.900 [0.500 1.000]	0.405 [0.250 0.571]	13.6	4.0	3.4
shaker	0.129 [0.000 0.222]	0.633 [0.500 1.000]	0.192 [0.000 0.308]	17.8	4.4	2.4
spoon	0.523 [0.476 0.593]	0.658 [0.564 0.705]	0.580 [0.530 0.615]	60.0	47.8	31.2
spoonside	0.119 [0.000 0.368]	0.727 [0.000 1.000]	0.167 [0.000 0.467]	27.4	3.6	2.4
cumulative	0.282 [0.256 0.304]	0.608 [0.576 0.637]	0.385 [0.357 0.409]	741.0	342.8	208.4

Table B.9: Hand-built WGG structure, small placesetting dataset, pairwise, rpobj, free.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.044 [0.000 0.125]	0.612 [0.143 1.000]	0.072 [0.000 0.211]	25.4	3.4	1.2
candle	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	14.6	0.0	0.0
cup	0.052 [0.000 0.087]	1.000 [1.000 1.000]	0.097 [0.000 0.160]	21.2	1.0	1.0
fork	0.335 [0.303 0.361]	0.520 [0.483 0.550]	0.407 [0.385 0.427]	121.6	78.8	40.8
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.301 [0.287 0.341]	0.611 [0.540 0.721]	0.402 [0.383 0.440]	100.0	49.4	30.0
knife	0.200 [0.159 0.245]	0.447 [0.359 0.500]	0.274 [0.220 0.324]	90.8	41.0	18.2
knifeside	0.000 [0.000 0.000]	0.800 [0.000 1.000]	0.000 [0.000 0.000]	13.8	0.2	0.0
napkin	0.038 [0.000 0.068]	0.950 [0.750 1.000]	0.072 [0.000 0.128]	46.8	2.0	1.8
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.364 [0.287 0.400]	0.868 [0.829 0.926]	0.511 [0.439 0.541]	154.8	64.8	56.0
saucer	0.087 [0.000 0.143]	1.000 [1.000 1.000]	0.156 [0.000 0.250]	13.6	1.0	1.0
shaker	0.051 [0.000 0.130]	0.830 [0.400 1.000]	0.082 [0.000 0.222]	17.8	1.8	1.0
spoon	0.460 [0.419 0.556]	0.645 [0.526 0.794]	0.530 [0.486 0.557]	60.0	43.6	27.4
spoonside	0.020 [0.000 0.053]	0.800 [0.000 1.000]	0.037 [0.000 0.100]	27.4	0.6	0.4
cumulative	0.242 [0.218 0.260]	0.623 [0.584 0.643]	0.348 [0.325 0.361]	741.0	287.6	178.8

Table B.10: Hand-built WGG structure, small placesetting dataset, nopairwise, classobj, fixed.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.045 [0.000 0.094]	0.670 [0.250 1.000]	0.080 [0.000 0.162]	25.4	2.4	1.2
candle	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	14.6	0.0	0.0
cup	0.075 [0.059 0.087]	0.833 [0.500 1.000]	0.136 [0.111 0.160]	21.2	2.0	1.6
fork	0.302 [0.260 0.341]	0.579 [0.471 0.686]	0.394 [0.366 0.428]	121.6	65.0	36.8
forkside	0.000 [0.000 0.000]	0.800 [0.000 1.000]	0.000 [0.000 0.000]	21.4	0.2	0.0
glass	0.183 [0.131 0.242]	0.686 [0.647 0.737]	0.285 [0.222 0.352]	100.0	26.4	18.0
knife	0.177 [0.068 0.275]	0.508 [0.261 0.615]	0.260 [0.108 0.376]	90.8	30.8	16.2
knifeside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	13.8	0.0	0.0
napkin	0.034 [0.000 0.091]	0.680 [0.000 1.000]	0.062 [0.000 0.163]	46.8	2.4	1.6
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.365 [0.310 0.413]	0.935 [0.895 1.000]	0.524 [0.474 0.571]	154.8	60.2	56.2
saucer	0.119 [0.095 0.143]	1.000 [1.000 1.000]	0.213 [0.174 0.250]	13.6	1.6	1.6
shaker	0.078 [0.000 0.188]	0.700 [0.500 1.000]	0.128 [0.000 0.273]	17.8	2.4	1.4
spoon	0.432 [0.365 0.500]	0.694 [0.667 0.719]	0.531 [0.484 0.581]	60.0	37.2	25.8
spoonside	0.030 [0.000 0.105]	1.000 [1.000 1.000]	0.055 [0.000 0.190]	27.4	0.6	0.6
cumulative	0.217 [0.208 0.227]	0.698 [0.627 0.741]	0.331 [0.321 0.348]	741.0	231.2	161.0

Table B.11: Hand-built WGG structure, small placesetting dataset, nopairwise, rpobj, fixed.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.076 [0.037 0.125]	0.556 [0.111 1.000]	0.127 [0.067 0.200]	25.4	5.0	2.0
candle	0.071 [0.000 0.182]	0.111 [0.000 0.286]	0.086 [0.000 0.222]	14.6	7.2	0.8
cup	0.208 [0.069 0.294]	0.566 [0.308 0.800]	0.278 [0.125 0.370]	21.2	8.0	4.0
fork	0.370 [0.328 0.420]	0.552 [0.494 0.585]	0.442 [0.413 0.476]	121.6	81.8	45.0
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.335 [0.282 0.352]	0.570 [0.500 0.659]	0.419 [0.395 0.451]	100.0	59.8	33.4
knife	0.269 [0.205 0.363]	0.344 [0.220 0.423]	0.301 [0.212 0.391]	90.8	72.0	24.4
knifeside	0.000 [0.000 0.000]	0.800 [0.000 1.000]	0.000 [0.000 0.000]	13.8	0.2	0.0
napkin	0.042 [0.000 0.089]	0.400 [0.000 1.000]	0.071 [0.000 0.154]	46.8	8.2	2.0
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.373 [0.293 0.434]	0.891 [0.841 0.944]	0.523 [0.447 0.577]	154.8	64.4	57.2
saucer	0.312 [0.143 0.500]	0.877 [0.750 1.000]	0.453 [0.240 0.667]	13.6	4.4	3.8
shaker	0.075 [0.000 0.188]	0.256 [0.000 0.600]	0.116 [0.000 0.286]	17.8	3.8	1.4
spoon	0.488 [0.413 0.611]	0.695 [0.667 0.744]	0.570 [0.510 0.647]	60.0	41.8	29.0
spoonside	0.107 [0.000 0.421]	0.558 [0.125 1.000]	0.115 [0.000 0.372]	27.4	7.2	2.2
cumulative	0.278 [0.248 0.304]	0.565 [0.512 0.601]	0.372 [0.346 0.404]	741.0	363.8	205.2

Table B.12: Learned 2-level WGGs, small placesetting dataset, pairwise, classobj, fixed.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.080 [0.000 0.120]	0.509 [0.118 1.000]	0.121 [0.000 0.188]	25.4	6.8	2.0
candle	0.047 [0.000 0.125]	0.090 [0.000 0.250]	0.062 [0.000 0.167]	14.6	4.8	0.6
cup	0.227 [0.103 0.294]	0.608 [0.375 0.750]	0.317 [0.182 0.387]	21.2	7.8	4.6
fork	0.373 [0.336 0.429]	0.550 [0.537 0.566]	0.444 [0.419 0.477]	121.6	82.6	45.4
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.315 [0.282 0.330]	0.699 [0.654 0.744]	0.433 [0.403 0.455]	100.0	45.2	31.4
knife	0.271 [0.202 0.341]	0.358 [0.271 0.437]	0.308 [0.232 0.383]	90.8	69.4	24.6
knifeside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	13.8	0.0	0.0
napkin	0.038 [0.000 0.067]	0.455 [0.000 1.000]	0.066 [0.000 0.120]	46.8	7.4	1.8
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.386 [0.333 0.420]	0.900 [0.853 0.983]	0.538 [0.498 0.565]	154.8	66.2	59.4
saucer	0.303 [0.095 0.500]	0.820 [0.667 1.000]	0.428 [0.167 0.615]	13.6	4.4	3.6
shaker	0.094 [0.000 0.174]	0.627 [0.333 1.000]	0.152 [0.000 0.286]	17.8	3.4	1.8
spoon	0.453 [0.365 0.519]	0.696 [0.651 0.800]	0.545 [0.479 0.577]	60.0	39.0	27.0
spoonside	0.084 [0.000 0.211]	0.600 [0.333 1.000]	0.121 [0.000 0.258]	27.4	5.6	2.0
cumulative	0.276 [0.258 0.294]	0.596 [0.571 0.621]	0.377 [0.360 0.399]	741.0	342.6	204.2

Table B.13: Learned 2-level WGGs, small placesetting dataset, pairwise, rpobj, fixed.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.093 [0.074 0.125]	0.548 [0.167 1.000]	0.151 [0.121 0.205]	25.4	6.0	2.4
candle	0.050 [0.000 0.111]	0.440 [0.000 1.000]	0.067 [0.000 0.167]	14.6	5.4	0.6
cup	0.222 [0.172 0.304]	0.601 [0.308 0.833]	0.306 [0.250 0.412]	21.2	8.4	4.6
fork	0.363 [0.339 0.382]	0.539 [0.453 0.652]	0.432 [0.396 0.465]	121.6	83.2	44.2
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.310 [0.253 0.363]	0.619 [0.534 0.707]	0.407 [0.371 0.440]	100.0	51.6	31.0
knife	0.250 [0.160 0.341]	0.328 [0.221 0.443]	0.282 [0.185 0.385]	90.8	70.0	22.6
knifeside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	13.8	0.0	0.0
napkin	0.042 [0.000 0.089]	0.382 [0.000 1.000]	0.070 [0.000 0.148]	46.8	6.8	2.0
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.368 [0.299 0.434]	0.885 [0.827 0.946]	0.517 [0.452 0.569]	154.8	64.4	56.6
saucer	0.284 [0.143 0.500]	0.850 [0.667 1.000]	0.400 [0.240 0.571]	13.6	4.2	3.4
shaker	0.041 [0.000 0.118]	0.507 [0.000 1.000]	0.057 [0.000 0.148]	17.8	3.4	0.8
spoon	0.466 [0.397 0.537]	0.673 [0.592 0.735]	0.547 [0.515 0.571]	60.0	41.6	27.8
spoonside	0.070 [0.000 0.211]	0.637 [0.286 1.000]	0.105 [0.000 0.276]	27.4	4.0	1.6
cumulative	0.267 [0.233 0.289]	0.569 [0.513 0.614]	0.362 [0.338 0.383]	741.0	349.0	197.6

Table B.14: Learned 2-level WGGs, small placesetting dataset, pairwise, classobj, free.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.058 [0.000 0.143]	0.417 [0.167 1.000]	0.081 [0.000 0.154]	25.4	6.4	1.4
candle	0.036 [0.000 0.182]	0.067 [0.000 0.333]	0.047 [0.000 0.235]	14.6	5.8	0.4
cup	0.256 [0.176 0.348]	0.637 [0.300 0.800]	0.361 [0.222 0.485]	21.2	8.6	5.4
fork	0.367 [0.336 0.405]	0.555 [0.517 0.590]	0.441 [0.419 0.480]	121.6	80.4	44.6
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.295 [0.259 0.352]	0.720 [0.652 0.842]	0.417 [0.383 0.496]	100.0	40.8	29.2
knife	0.287 [0.213 0.374]	0.400 [0.268 0.540]	0.333 [0.259 0.442]	90.8	66.2	26.0
knifeside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	13.8	0.0	0.0
napkin	0.050 [0.021 0.111]	0.502 [0.091 1.000]	0.086 [0.036 0.185]	46.8	6.8	2.4
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.375 [0.293 0.420]	0.900 [0.853 0.944]	0.527 [0.447 0.580]	154.8	64.2	57.6
saucer	0.277 [0.190 0.375]	0.860 [0.667 1.000]	0.414 [0.296 0.545]	13.6	4.4	3.6
shaker	0.110 [0.067 0.130]	0.620 [0.333 1.000]	0.181 [0.125 0.214]	17.8	3.8	2.0
spoon	0.459 [0.419 0.519]	0.712 [0.605 0.778]	0.556 [0.495 0.596]	60.0	38.8	27.4
spoonside	0.072 [0.000 0.211]	0.533 [0.000 1.000]	0.115 [0.000 0.320]	27.4	3.8	1.8
cumulative	0.273 [0.241 0.290]	0.612 [0.592 0.631]	0.377 [0.346 0.395]	741.0	330.0	201.8

Table B.15: Learned 2-level WGGs, small placesetting dataset, pairwise, rpobj, free.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.058 [0.000 0.156]	0.529 [0.100 1.000]	0.095 [0.000 0.256]	25.4	4.4	1.6
candle	0.018 [0.000 0.091]	0.467 [0.000 1.000]	0.029 [0.000 0.143]	14.6	1.8	0.2
cup	0.277 [0.138 0.412]	0.633 [0.357 0.800]	0.362 [0.235 0.519]	21.2	9.2	5.4
fork	0.339 [0.294 0.380]	0.501 [0.451 0.536]	0.402 [0.378 0.443]	121.6	82.8	41.2
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.370 [0.301 0.407]	0.565 [0.461 0.660]	0.442 [0.413 0.471]	100.0	67.6	37.0
knife	0.212 [0.160 0.319]	0.506 [0.386 0.652]	0.292 [0.256 0.392]	90.8	39.2	19.2
knifeside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	13.8	0.0	0.0
napkin	0.038 [0.000 0.068]	0.550 [0.000 1.000]	0.069 [0.000 0.125]	46.8	3.6	1.8
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.360 [0.276 0.413]	0.807 [0.747 0.923]	0.493 [0.425 0.536]	154.8	69.2	55.2
saucer	0.306 [0.143 0.500]	0.821 [0.500 1.000]	0.412 [0.240 0.545]	13.6	5.0	3.8
shaker	0.100 [0.056 0.188]	0.576 [0.200 1.000]	0.160 [0.091 0.261]	17.8	3.8	1.8
spoon	0.435 [0.365 0.481]	0.678 [0.639 0.727]	0.529 [0.465 0.565]	60.0	38.4	26.0
spoonside	0.087 [0.000 0.316]	0.620 [0.200 1.000]	0.110 [0.000 0.353]	27.4	4.8	1.8
cumulative	0.264 [0.231 0.284]	0.593 [0.549 0.631]	0.365 [0.333 0.387]	741.0	329.8	195.0

Table B.16: Learned 2-level WGGs, small placesetting dataset, nopairwise, classobj, fixed.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.058 [0.000 0.156]	0.308 [0.000 0.714]	0.093 [0.000 0.256]	25.4	5.4	1.6
candle	0.032 [0.000 0.111]	0.300 [0.000 1.000]	0.048 [0.000 0.167]	14.6	2.8	0.4
cup	0.286 [0.172 0.357]	0.681 [0.417 1.000]	0.382 [0.294 0.444]	21.2	9.2	5.8
fork	0.339 [0.285 0.395]	0.538 [0.448 0.581]	0.411 [0.378 0.452]	121.6	78.0	41.2
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.278 [0.233 0.341]	0.707 [0.614 0.756]	0.397 [0.353 0.470]	100.0	39.2	27.6
knife	0.218 [0.160 0.374]	0.570 [0.429 0.704]	0.309 [0.233 0.463]	90.8	35.2	19.8
knifeside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	13.8	0.0	0.0
napkin	0.033 [0.000 0.057]	0.455 [0.000 1.000]	0.060 [0.000 0.094]	46.8	4.6	1.6
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.346 [0.293 0.407]	0.918 [0.871 0.980]	0.500 [0.447 0.555]	154.8	58.2	53.2
saucer	0.278 [0.143 0.500]	0.853 [0.714 1.000]	0.399 [0.240 0.615]	13.6	4.2	3.4
shaker	0.078 [0.000 0.188]	0.667 [0.250 1.000]	0.128 [0.000 0.300]	17.8	3.0	1.4
spoon	0.425 [0.333 0.468]	0.701 [0.618 0.765]	0.528 [0.433 0.571]	60.0	36.2	25.4
spoonside	0.102 [0.000 0.273]	0.648 [0.125 1.000]	0.142 [0.000 0.400]	27.4	5.4	2.2
cumulative	0.248 [0.226 0.272]	0.654 [0.587 0.681]	0.359 [0.338 0.385]	741.0	281.4	183.6

Table B.17: Learned 2-level WGGs, small placesetting dataset, nopairwise, rpobj, fixed.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.053 [0.000 0.094]	0.495 [0.100 1.000]	0.088 [0.000 0.150]	25.4	4.8	1.4
candle	0.059 [0.000 0.182]	0.085 [0.000 0.333]	0.067 [0.000 0.235]	14.6	8.0	0.6
cup	0.159 [0.130 0.176]	0.620 [0.400 0.833]	0.252 [0.211 0.286]	21.2	5.4	3.4
fork	0.256 [0.203 0.314]	0.617 [0.553 0.674]	0.360 [0.301 0.418]	121.6	50.2	31.0
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.279 [0.269 0.297]	0.620 [0.509 0.718]	0.383 [0.354 0.419]	100.0	46.0	27.8
knife	0.254 [0.182 0.319]	0.427 [0.250 0.630]	0.317 [0.211 0.423]	90.8	55.8	23.0
knifeside	0.000 [0.000 0.000]	0.200 [0.000 1.000]	0.000 [0.000 0.000]	13.8	1.2	0.0
napkin	0.033 [0.000 0.057]	0.800 [0.000 1.000]	0.064 [0.000 0.107]	46.8	2.6	1.6
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.347 [0.276 0.387]	0.842 [0.800 0.941]	0.489 [0.427 0.530]	154.8	63.8	53.4
saucer	0.140 [0.000 0.267]	0.960 [0.800 1.000]	0.232 [0.000 0.421]	13.6	2.4	2.2
shaker	0.064 [0.000 0.111]	0.473 [0.200 1.000]	0.102 [0.000 0.190]	17.8	4.0	1.2
spoon	0.428 [0.403 0.463]	0.697 [0.649 0.758]	0.530 [0.505 0.575]	60.0	36.8	25.6
spoonside	0.097 [0.000 0.263]	0.547 [0.182 1.000]	0.145 [0.000 0.370]	27.4	5.6	2.2
cumulative	0.235 [0.203 0.263]	0.606 [0.563 0.682]	0.338 [0.301 0.380]	741.0	286.6	173.4

Table B.18: Learned 3-level WGGs, small placesetting dataset, $\gamma_{\text{minsize}} = 0.1$, nosingletocomp.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.060 [0.000 0.125]	0.577 [0.250 1.000]	0.103 [0.000 0.216]	25.4	3.2	1.6
candle	0.078 [0.000 0.182]	0.333 [0.000 1.000]	0.098 [0.000 0.235]	14.6	5.8	1.0
cup	0.121 [0.059 0.174]	0.760 [0.500 1.000]	0.208 [0.105 0.294]	21.2	3.4	2.8
fork	0.301 [0.235 0.387]	0.565 [0.414 0.718]	0.382 [0.346 0.455]	121.6	69.2	36.6
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.272 [0.241 0.330]	0.700 [0.565 0.828]	0.390 [0.338 0.448]	100.0	39.2	27.0
knife	0.191 [0.011 0.299]	0.403 [0.310 0.500]	0.236 [0.022 0.340]	90.8	46.0	17.4
knifeside	0.000 [0.000 0.000]	0.400 [0.000 1.000]	0.000 [0.000 0.000]	13.8	2.2	0.0
napkin	0.029 [0.000 0.057]	0.900 [0.500 1.000]	0.055 [0.000 0.102]	46.8	2.0	1.4
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.292 [0.166 0.364]	0.869 [0.803 0.960]	0.430 [0.280 0.507]	154.8	52.0	44.8
saucer	0.170 [0.100 0.267]	0.920 [0.600 1.000]	0.282 [0.182 0.421]	13.6	2.8	2.4
shaker	0.090 [0.059 0.125]	0.513 [0.333 1.000]	0.147 [0.100 0.200]	17.8	3.8	1.6
spoon	0.357 [0.032 0.526]	0.777 [0.675 1.000]	0.443 [0.062 0.606]	60.0	29.0	21.0
spoonside	0.082 [0.000 0.368]	0.540 [0.000 1.000]	0.113 [0.000 0.483]	27.4	4.0	1.6
cumulative	0.215 [0.173 0.244]	0.610 [0.545 0.645]	0.316 [0.272 0.349]	741.0	262.6	159.2

Table B.19: Learned 3-level WGGs, small placesetting dataset, $\gamma_{\text{minsize}} = 0.1$, singletocomp1.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.058 [0.000 0.156]	0.500 [0.167 1.000]	0.093 [0.000 0.238]	25.4	4.2	1.6
candle	0.032 [0.000 0.111]	0.038 [0.000 0.111]	0.034 [0.000 0.111]	14.6	6.8	0.4
cup	0.158 [0.059 0.241]	0.820 [0.500 1.000]	0.263 [0.105 0.389]	21.2	4.2	3.6
fork	0.280 [0.235 0.361]	0.661 [0.589 0.691]	0.389 [0.347 0.448]	121.6	52.0	34.0
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.255 [0.159 0.308]	0.726 [0.643 0.850]	0.372 [0.268 0.434]	100.0	35.4	25.2
knife	0.193 [0.011 0.275]	0.298 [0.143 0.500]	0.225 [0.021 0.355]	90.8	55.4	17.6
knifeside	0.000 [0.000 0.000]	0.600 [0.000 1.000]	0.000 [0.000 0.000]	13.8	0.4	0.0
napkin	0.020 [0.000 0.057]	0.800 [0.000 1.000]	0.039 [0.000 0.107]	46.8	1.2	1.0
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.307 [0.247 0.353]	0.843 [0.791 0.935]	0.447 [0.391 0.488]	154.8	56.4	47.2
saucer	0.176 [0.048 0.267]	0.900 [0.500 1.000]	0.289 [0.087 0.421]	13.6	2.4	2.2
shaker	0.076 [0.000 0.125]	0.600 [0.333 1.000]	0.127 [0.000 0.211]	17.8	3.0	1.4
spoon	0.357 [0.127 0.491]	0.736 [0.696 0.800]	0.462 [0.219 0.583]	60.0	29.4	21.4
spoonside	0.063 [0.000 0.316]	0.509 [0.000 1.000]	0.080 [0.000 0.400]	27.4	4.4	1.2
cumulative	0.212 [0.180 0.238]	0.619 [0.576 0.679]	0.314 [0.285 0.346]	741.0	255.2	156.8

Table B.20: Learned 3-level WGGs, small placesetting dataset, $\gamma_{\text{minsize}} = 0.1$, singletocomp2.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.081 [0.037 0.156]	0.367 [0.143 0.500]	0.123 [0.069 0.227]	25.4	6.8	2.2
candle	0.050 [0.000 0.111]	0.076 [0.000 0.200]	0.056 [0.000 0.125]	14.6	8.4	0.6
cup	0.181 [0.087 0.294]	0.812 [0.429 1.000]	0.282 [0.160 0.435]	21.2	4.8	3.6
fork	0.330 [0.260 0.395]	0.586 [0.522 0.632]	0.417 [0.366 0.463]	121.6	69.2	40.0
forkside	0.000 [0.000 0.000]	0.800 [0.000 1.000]	0.000 [0.000 0.000]	21.4	0.2	0.0
glass	0.312 [0.262 0.343]	0.598 [0.507 0.692]	0.407 [0.380 0.449]	100.0	53.2	31.2
knife	0.269 [0.213 0.330]	0.431 [0.308 0.532]	0.330 [0.252 0.403]	90.8	58.0	24.4
knifeside	0.000 [0.000 0.000]	0.200 [0.000 1.000]	0.000 [0.000 0.000]	13.8	1.6	0.0
napkin	0.034 [0.000 0.089]	0.472 [0.000 1.000]	0.061 [0.000 0.148]	46.8	3.6	1.6
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.350 [0.276 0.406]	0.877 [0.784 0.966]	0.497 [0.427 0.538]	154.8	61.8	53.8
saucer	0.121 [0.000 0.267]	0.933 [0.667 1.000]	0.204 [0.000 0.421]	13.6	2.0	1.8
shaker	0.082 [0.000 0.174]	0.517 [0.250 1.000]	0.129 [0.000 0.276]	17.8	3.8	1.6
spoon	0.400 [0.339 0.481]	0.701 [0.684 0.724]	0.507 [0.462 0.571]	60.0	34.0	23.8
spoonside	0.103 [0.000 0.263]	0.424 [0.000 1.000]	0.129 [0.000 0.303]	27.4	6.8	2.2
cumulative	0.253 [0.222 0.282]	0.597 [0.532 0.639]	0.354 [0.330 0.386]	741.0	314.2	186.8

Table B.21: Learned 3-level WGGs, small placesetting dataset, $\gamma_{\text{minsize}} = 0.15$, nosingletocomp.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.039 [0.000 0.062]	0.442 [0.125 1.000]	0.066 [0.000 0.111]	25.4	3.8	1.0
candle	0.050 [0.000 0.111]	0.136 [0.000 0.500]	0.064 [0.000 0.154]	14.6	6.0	0.6
cup	0.126 [0.059 0.241]	0.767 [0.333 1.000]	0.214 [0.100 0.389]	21.2	3.6	3.0
fork	0.340 [0.228 0.429]	0.564 [0.490 0.677]	0.418 [0.313 0.483]	121.6	74.0	41.2
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.269 [0.241 0.319]	0.727 [0.617 0.846]	0.390 [0.366 0.443]	100.0	37.6	26.8
knife	0.204 [0.057 0.286]	0.460 [0.304 0.833]	0.253 [0.106 0.335]	90.8	49.8	18.6
knifeside	0.000 [0.000 0.000]	0.800 [0.000 1.000]	0.000 [0.000 0.000]	13.8	0.8	0.0
napkin	0.030 [0.000 0.045]	1.000 [1.000 1.000]	0.057 [0.000 0.087]	46.8	1.4	1.4
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.356 [0.227 0.407]	0.853 [0.753 0.969]	0.494 [0.366 0.540]	154.8	65.6	55.0
saucer	0.156 [0.100 0.267]	0.950 [0.750 1.000]	0.263 [0.182 0.421]	13.6	2.4	2.2
shaker	0.085 [0.000 0.130]	0.620 [0.500 1.000]	0.140 [0.000 0.214]	17.8	3.0	1.6
spoon	0.380 [0.016 0.526]	0.779 [0.682 1.000]	0.461 [0.031 0.606]	60.0	31.0	22.4
spoonside	0.057 [0.000 0.182]	0.543 [0.000 1.000]	0.079 [0.000 0.276]	27.4	5.6	1.2
cumulative	0.236 [0.200 0.277]	0.614 [0.558 0.644]	0.340 [0.294 0.382]	741.0	284.6	175.0

Table B.22: Learned 3-level WGGs, small placesetting dataset, $\gamma_{\text{minsize}} = 0.15$, singletocomp1.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.052 [0.000 0.125]	0.521 [0.200 1.000]	0.087 [0.000 0.205]	25.4	3.4	1.4
candle	0.028 [0.000 0.091]	0.081 [0.000 0.333]	0.040 [0.000 0.143]	14.6	7.2	0.4
cup	0.149 [0.059 0.241]	0.733 [0.500 1.000]	0.246 [0.105 0.389]	21.2	4.2	3.4
fork	0.333 [0.228 0.429]	0.573 [0.418 0.677]	0.408 [0.339 0.462]	121.6	74.8	40.4
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.255 [0.184 0.319]	0.744 [0.609 0.950]	0.373 [0.309 0.450]	100.0	35.2	25.2
knife	0.198 [0.057 0.264]	0.402 [0.275 0.500]	0.252 [0.101 0.346]	90.8	47.6	18.0
knifeside	0.000 [0.000 0.000]	0.600 [0.000 1.000]	0.000 [0.000 0.000]	13.8	0.8	0.0
napkin	0.025 [0.000 0.044]	1.000 [1.000 1.000]	0.049 [0.000 0.085]	46.8	1.2	1.2
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.361 [0.327 0.401]	0.875 [0.800 0.969]	0.510 [0.473 0.545]	154.8	64.0	56.0
saucer	0.182 [0.100 0.286]	0.950 [0.750 1.000]	0.300 [0.182 0.444]	13.6	2.6	2.4
shaker	0.074 [0.000 0.130]	0.503 [0.250 1.000]	0.117 [0.000 0.214]	17.8	3.6	1.4
spoon	0.388 [0.127 0.484]	0.698 [0.675 0.727]	0.480 [0.216 0.566]	60.0	33.2	23.0
spoonside	0.081 [0.000 0.316]	0.457 [0.000 1.000]	0.093 [0.000 0.300]	27.4	6.2	1.6
cumulative	0.235 [0.208 0.259]	0.612 [0.562 0.644]	0.339 [0.303 0.366]	741.0	284.0	174.4

Table B.23: Learned 3-level WGGs, small placesetting dataset, $\gamma_{\text{minsize}} = 0.15$, singletocomp2.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.067 [0.037 0.125]	0.342 [0.143 0.500]	0.108 [0.069 0.190]	25.4	5.6	1.8
candle	0.028 [0.000 0.091]	0.065 [0.000 0.200]	0.039 [0.000 0.125]	14.6	8.0	0.4
cup	0.115 [0.087 0.143]	0.733 [0.333 1.000]	0.191 [0.154 0.242]	21.2	3.6	2.4
fork	0.347 [0.285 0.420]	0.590 [0.495 0.636]	0.434 [0.393 0.500]	121.6	72.4	42.2
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.332 [0.301 0.361]	0.590 [0.523 0.660]	0.423 [0.395 0.449]	100.0	57.2	33.2
knife	0.267 [0.213 0.308]	0.399 [0.308 0.538]	0.319 [0.252 0.392]	90.8	61.8	24.2
knifeside	0.013 [0.000 0.067]	0.233 [0.000 1.000]	0.019 [0.000 0.095]	13.8	4.0	0.2
napkin	0.033 [0.000 0.057]	0.477 [0.000 1.000]	0.061 [0.000 0.103]	46.8	3.6	1.6
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.350 [0.264 0.406]	0.856 [0.763 0.966]	0.493 [0.411 0.541]	154.8	63.4	53.8
saucer	0.109 [0.000 0.143]	0.950 [0.750 1.000]	0.189 [0.000 0.250]	13.6	1.8	1.6
shaker	0.053 [0.000 0.087]	0.617 [0.250 1.000]	0.091 [0.000 0.148]	17.8	2.4	1.0
spoon	0.410 [0.365 0.519]	0.693 [0.686 0.700]	0.513 [0.479 0.596]	60.0	35.2	24.4
spoonside	0.158 [0.000 0.364]	0.519 [0.143 1.000]	0.191 [0.000 0.471]	27.4	8.8	3.4
cumulative	0.258 [0.232 0.289]	0.582 [0.524 0.637]	0.356 [0.333 0.397]	741.0	327.8	190.2

Table B.24: Learned 3-level WGGs, small placesetting dataset, $\gamma_{\text{minsize}} = 0.2$, nosingletocomp.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.052 [0.000 0.125]	0.425 [0.125 1.000]	0.083 [0.000 0.200]	25.4	4.8	1.4
candle	0.059 [0.000 0.111]	0.160 [0.000 0.500]	0.076 [0.000 0.154]	14.6	6.4	0.8
cup	0.124 [0.059 0.214]	0.693 [0.333 1.000]	0.205 [0.100 0.333]	21.2	3.8	2.6
fork	0.324 [0.228 0.412]	0.526 [0.440 0.597]	0.398 [0.313 0.462]	121.6	75.2	39.4
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.303 [0.231 0.363]	0.657 [0.610 0.725]	0.413 [0.336 0.455]	100.0	45.8	30.0
knife	0.231 [0.170 0.308]	0.355 [0.250 0.424]	0.279 [0.203 0.357]	90.8	59.6	21.0
knifeside	0.000 [0.000 0.000]	0.600 [0.000 1.000]	0.000 [0.000 0.000]	13.8	1.2	0.0
napkin	0.034 [0.000 0.067]	0.850 [0.500 1.000]	0.063 [0.000 0.125]	46.8	2.0	1.6
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.357 [0.227 0.441]	0.864 [0.778 0.952]	0.496 [0.366 0.562]	154.8	64.8	55.0
saucer	0.094 [0.000 0.143]	1.000 [1.000 1.000]	0.168 [0.000 0.250]	13.6	1.4	1.4
shaker	0.085 [0.000 0.130]	0.567 [0.333 1.000]	0.135 [0.000 0.200]	17.8	3.8	1.6
spoon	0.424 [0.317 0.526]	0.724 [0.690 0.765]	0.531 [0.435 0.606]	60.0	34.8	25.2
spoonside	0.068 [0.000 0.182]	0.567 [0.143 1.000]	0.093 [0.000 0.258]	27.4	6.2	1.6
cumulative	0.245 [0.200 0.282]	0.586 [0.531 0.619]	0.345 [0.294 0.386]	741.0	309.8	181.6

Table B.25: Learned 3-level WGGs, small placesetting dataset, $\gamma_{\text{minsize}} = 0.2$, singletocomp1.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.052 [0.000 0.125]	0.475 [0.125 1.000]	0.084 [0.000 0.200]	25.4	4.4	1.4
candle	0.059 [0.000 0.111]	0.126 [0.000 0.333]	0.073 [0.000 0.143]	14.6	6.4	0.8
cup	0.147 [0.087 0.217]	0.643 [0.500 0.833]	0.238 [0.154 0.345]	21.2	4.8	3.2
fork	0.302 [0.000 0.438]	0.634 [0.473 1.000]	0.351 [0.000 0.462]	121.6	68.8	36.6
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.296 [0.231 0.330]	0.681 [0.610 0.763]	0.412 [0.336 0.453]	100.0	43.2	29.4
knife	0.222 [0.170 0.308]	0.332 [0.254 0.424]	0.266 [0.204 0.357]	90.8	60.8	20.2
knifeside	0.000 [0.000 0.000]	0.600 [0.000 1.000]	0.000 [0.000 0.000]	13.8	1.4	0.0
napkin	0.029 [0.000 0.067]	0.950 [0.750 1.000]	0.055 [0.000 0.125]	46.8	1.6	1.4
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.345 [0.308 0.380]	0.899 [0.841 0.978]	0.496 [0.468 0.528]	154.8	59.8	53.4
saucer	0.143 [0.000 0.286]	1.000 [1.000 1.000]	0.237 [0.000 0.444]	13.6	2.0	2.0
shaker	0.074 [0.000 0.130]	0.633 [0.333 1.000]	0.116 [0.000 0.188]	17.8	3.6	1.4
spoon	0.426 [0.333 0.481]	0.719 [0.675 0.765]	0.533 [0.457 0.591]	60.0	35.4	25.4
spoonside	0.082 [0.000 0.316]	0.721 [0.250 1.000]	0.101 [0.000 0.333]	27.4	5.2	1.8
cumulative	0.239 [0.189 0.282]	0.594 [0.560 0.617]	0.340 [0.284 0.386]	741.0	297.4	177.0

Table B.26: Learned 3-level WGGs, small placesetting dataset, $\gamma_{\text{minsize}} = 0.2$, singletocomp2.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.045 [0.000 0.094]	0.500 [0.167 1.000]	0.078 [0.000 0.167]	25.4	3.4	1.2
candle	0.032 [0.000 0.111]	0.054 [0.000 0.143]	0.038 [0.000 0.118]	14.6	7.6	0.4
cup	0.143 [0.071 0.235]	0.683 [0.200 1.000]	0.230 [0.105 0.348]	21.2	4.6	3.0
fork	0.375 [0.328 0.420]	0.548 [0.518 0.608]	0.444 [0.413 0.467]	121.6	83.6	45.6
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.331 [0.291 0.370]	0.581 [0.468 0.682]	0.419 [0.391 0.440]	100.0	58.8	33.2
knife	0.267 [0.213 0.330]	0.389 [0.279 0.536]	0.316 [0.244 0.408]	90.8	63.0	24.2
knifeside	0.000 [0.000 0.000]	0.400 [0.000 1.000]	0.000 [0.000 0.000]	13.8	1.8	0.0
napkin	0.043 [0.000 0.068]	0.566 [0.000 1.000]	0.075 [0.000 0.122]	46.8	5.4	2.0
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.357 [0.264 0.434]	0.875 [0.822 0.958]	0.503 [0.414 0.569]	154.8	63.2	54.8
saucer	0.104 [0.000 0.143]	0.950 [0.750 1.000]	0.181 [0.000 0.250]	13.6	1.8	1.6
shaker	0.067 [0.000 0.188]	0.467 [0.000 1.000]	0.107 [0.000 0.300]	17.8	3.0	1.2
spoon	0.447 [0.365 0.537]	0.692 [0.617 0.788]	0.538 [0.479 0.574]	60.0	38.8	26.6
spoonside	0.104 [0.000 0.368]	0.512 [0.125 1.000]	0.117 [0.000 0.311]	27.4	7.8	2.2
cumulative	0.265 [0.241 0.292]	0.572 [0.540 0.605]	0.362 [0.340 0.393]	741.0	342.8	196.0

Table B.27: Learned 3-level WGGs, small placesetting dataset, $\gamma_{\text{minsize}} = 0.3$, nosingletocomp.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.068 [0.000 0.125]	0.577 [0.250 1.000]	0.114 [0.000 0.216]	25.4	3.6	1.8
candle	0.081 [0.000 0.182]	0.242 [0.000 0.500]	0.108 [0.000 0.267]	14.6	6.4	1.0
cup	0.166 [0.087 0.217]	0.770 [0.500 1.000]	0.271 [0.148 0.357]	21.2	4.4	3.4
fork	0.359 [0.306 0.412]	0.538 [0.440 0.581]	0.430 [0.361 0.480]	121.6	81.2	43.6
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.327 [0.291 0.363]	0.576 [0.468 0.630]	0.415 [0.397 0.455]	100.0	57.8	32.6
knife	0.249 [0.170 0.309]	0.345 [0.250 0.439]	0.289 [0.203 0.363]	90.8	64.8	22.6
knifeside	0.000 [0.000 0.000]	0.800 [0.000 1.000]	0.000 [0.000 0.000]	13.8	0.8	0.0
napkin	0.042 [0.000 0.111]	0.741 [0.455 1.000]	0.074 [0.000 0.179]	46.8	3.6	2.0
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.388 [0.345 0.441]	0.834 [0.765 0.952]	0.527 [0.506 0.562]	154.8	72.2	59.8
saucer	0.143 [0.000 0.286]	0.893 [0.667 1.000]	0.231 [0.000 0.421]	13.6	2.4	2.0
shaker	0.085 [0.000 0.130]	0.636 [0.250 1.000]	0.134 [0.000 0.200]	17.8	3.8	1.6
spoon	0.445 [0.317 0.509]	0.715 [0.638 0.784]	0.545 [0.435 0.592]	60.0	37.4	26.6
spoonside	0.100 [0.000 0.316]	0.364 [0.000 1.000]	0.120 [0.000 0.343]	27.4	5.4	2.0
cumulative	0.269 [0.257 0.287]	0.579 [0.531 0.603]	0.367 [0.346 0.381]	741.0	343.8	199.0

Table B.28: Learned 3-level WGGs, small placesetting dataset, $\gamma_{\text{minsize}} = 0.3$, singletocomp1.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.068 [0.000 0.125]	0.577 [0.250 1.000]	0.114 [0.000 0.216]	25.4	3.6	1.8
candle	0.081 [0.000 0.182]	0.242 [0.000 0.500]	0.108 [0.000 0.267]	14.6	6.4	1.0
cup	0.166 [0.087 0.217]	0.770 [0.500 1.000]	0.271 [0.148 0.357]	21.2	4.4	3.4
fork	0.359 [0.306 0.412]	0.538 [0.440 0.581]	0.430 [0.361 0.480]	121.6	81.2	43.6
forkside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	21.4	0.0	0.0
glass	0.327 [0.291 0.363]	0.576 [0.468 0.630]	0.415 [0.397 0.455]	100.0	57.8	32.6
knife	0.249 [0.170 0.309]	0.345 [0.250 0.439]	0.289 [0.203 0.363]	90.8	64.8	22.6
knifeside	0.000 [0.000 0.000]	0.800 [0.000 1.000]	0.000 [0.000 0.000]	13.8	0.8	0.0
napkin	0.042 [0.000 0.111]	0.741 [0.455 1.000]	0.074 [0.000 0.179]	46.8	3.6	2.0
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	11.8	0.0	0.0
plate	0.388 [0.345 0.441]	0.834 [0.765 0.952]	0.527 [0.506 0.562]	154.8	72.2	59.8
saucer	0.143 [0.000 0.286]	0.893 [0.667 1.000]	0.231 [0.000 0.421]	13.6	2.4	2.0
shaker	0.085 [0.000 0.130]	0.636 [0.250 1.000]	0.134 [0.000 0.200]	17.8	3.8	1.6
spoon	0.445 [0.317 0.509]	0.715 [0.638 0.784]	0.545 [0.435 0.592]	60.0	37.4	26.6
spoonside	0.100 [0.000 0.316]	0.364 [0.000 1.000]	0.120 [0.000 0.343]	27.4	5.4	2.0
cumulative	0.269 [0.257 0.287]	0.579 [0.531 0.603]	0.367 [0.346 0.381]	741.0	343.8	199.0

Table B.29: Learned 3-level WGGs, small placesetting dataset, $\gamma_{\text{minsize}} = 0.3$, singletocomp2.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.367 [0.356 0.379]	0.369 [0.356 0.387]	0.368 [0.356 0.383]	202.7	201.3	74.3
candle	0.229 [0.161 0.286]	0.230 [0.165 0.286]	0.229 [0.163 0.286]	105.0	104.3	24.3
cup	0.556 [0.527 0.577]	0.567 [0.527 0.610]	0.562 [0.527 0.593]	178.3	175.0	99.3
fork	0.471 [0.447 0.484]	0.561 [0.482 0.705]	0.506 [0.482 0.547]	459.0	401.3	216.3
forkside	0.085 [0.062 0.106]	0.085 [0.062 0.106]	0.085 [0.062 0.106]	86.3	85.7	7.3
glass	0.471 [0.456 0.484]	0.476 [0.460 0.490]	0.473 [0.458 0.487]	682.3	676.0	321.7
knife	0.328 [0.314 0.344]	0.343 [0.330 0.354]	0.336 [0.322 0.349]	345.3	330.3	113.3
knifeside	0.035 [0.013 0.056]	0.054 [0.013 0.094]	0.041 [0.013 0.056]	77.0	61.0	2.7
napkin	0.132 [0.117 0.150]	0.149 [0.131 0.167]	0.140 [0.130 0.158]	295.7	263.3	39.0
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	141.0	0.0	0.0
plate	0.482 [0.454 0.497]	0.560 [0.554 0.566]	0.518 [0.504 0.527]	802.0	690.3	386.7
saucer	0.495 [0.455 0.536]	0.505 [0.470 0.536]	0.500 [0.462 0.536]	116.3	114.0	57.7
shaker	0.112 [0.077 0.167]	0.112 [0.077 0.167]	0.112 [0.077 0.167]	77.7	77.7	8.7
spoon	0.563 [0.549 0.589]	0.608 [0.564 0.658]	0.584 [0.558 0.599]	223.3	207.7	125.7
spoonside	0.304 [0.252 0.337]	0.305 [0.252 0.337]	0.305 [0.252 0.337]	107.0	106.7	32.3
cumulative	0.387 [0.381 0.390]	0.432 [0.428 0.437]	0.408 [0.403 0.412]	3899.0	3494.7	1509.3

Table B.30: DTPBM object detector, big placesetting dataset, known # objects.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.529 [0.507 0.564]	0.059 [0.048 0.078]	0.105 [0.087 0.135]	202.7	1931.0	107.3
candle	0.310 [0.226 0.371]	0.016 [0.014 0.017]	0.031 [0.027 0.033]	105.0	2009.0	33.0
cup	0.696 [0.632 0.728]	0.064 [0.046 0.085]	0.118 [0.087 0.151]	178.3	2056.0	124.0
fork	0.548 [0.497 0.584]	0.223 [0.123 0.385]	0.296 [0.202 0.434]	459.0	1470.0	251.7
forkside	0.144 [0.097 0.176]	0.008 [0.007 0.009]	0.015 [0.013 0.018]	86.3	1494.7	12.3
glass	0.581 [0.572 0.587]	0.110 [0.108 0.113]	0.185 [0.183 0.189]	682.3	3601.7	396.7
knife	0.419 [0.397 0.443]	0.101 [0.090 0.123]	0.162 [0.148 0.188]	345.3	1460.3	144.7
knifeside	0.048 [0.037 0.070]	0.006 [0.002 0.013]	0.009 [0.003 0.019]	77.0	1274.7	3.7
napkin	0.234 [0.179 0.274]	0.064 [0.052 0.074]	0.098 [0.087 0.104]	295.7	1143.3	69.3
placemat	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	141.0	0.0	0.0
plate	0.557 [0.517 0.583]	0.318 [0.282 0.366]	0.403 [0.378 0.429]	802.0	1427.0	446.7
saucer	0.610 [0.554 0.660]	0.039 [0.034 0.045]	0.073 [0.064 0.083]	116.3	1848.0	70.7
shaker	0.180 [0.128 0.221]	0.009 [0.006 0.011]	0.017 [0.012 0.020]	77.7	1599.7	14.0
spoon	0.623 [0.598 0.636]	0.152 [0.111 0.233]	0.238 [0.188 0.335]	223.3	1040.7	139.0
spoonside	0.350 [0.313 0.375]	0.029 [0.023 0.033]	0.054 [0.042 0.061]	107.0	1310.3	37.3
cumulative	0.475 [0.465 0.482]	0.079 [0.075 0.085]	0.135 [0.130 0.144]	3899.0	23666.3	1850.3

Table B.31: DTPBM object detector, big placesetting dataset, heuristic # objects.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.212 [0.193 0.222]	0.303 [0.248 0.398]	0.248 [0.217 0.285]	202.7	147.0	43.0
candle	0.023 [0.009 0.048]	0.347 [0.167 0.625]	0.042 [0.017 0.088]	105.0	6.0	2.3
cup	0.366 [0.355 0.379]	0.538 [0.513 0.566]	0.436 [0.420 0.454]	178.3	121.3	65.3
fork	0.318 [0.282 0.358]	0.579 [0.479 0.698]	0.406 [0.379 0.437]	459.0	260.7	146.0
forkside	0.004 [0.000 0.012]	0.048 [0.000 0.143]	0.007 [0.000 0.022]	86.3	3.0	0.3
glass	0.261 [0.249 0.272]	0.417 [0.395 0.441]	0.321 [0.311 0.336]	682.3	427.3	178.0
knife	0.283 [0.274 0.290]	0.424 [0.417 0.432]	0.339 [0.331 0.347]	345.3	230.3	97.7
knifeside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	77.0	0.0	0.0
napkin	0.111 [0.083 0.133]	0.168 [0.127 0.203]	0.134 [0.100 0.161]	295.7	194.3	32.7
placemat	0.028 [0.016 0.041]	0.556 [0.500 0.667]	0.053 [0.030 0.076]	141.0	7.0	4.0
plate	0.408 [0.400 0.416]	0.745 [0.683 0.816]	0.527 [0.517 0.537]	802.0	442.3	327.7
saucer	0.357 [0.306 0.388]	0.516 [0.430 0.662]	0.418 [0.366 0.480]	116.3	81.7	41.3
shaker	0.000 [0.000 0.000]	0.000 [0.000 0.000]	0.000 [0.000 0.000]	77.7	1.7	0.0
spoon	0.429 [0.407 0.448]	0.638 [0.588 0.693]	0.513 [0.481 0.533]	223.3	150.7	96.0
spoonside	0.060 [0.035 0.108]	0.504 [0.440 0.571]	0.103 [0.065 0.173]	107.0	13.3	6.3
cumulative	0.267 [0.255 0.274]	0.499 [0.492 0.508]	0.348 [0.336 0.356]	3899.0	2086.7	1040.7

Table B.32: Learned 2-level WGGs, big placesetting dataset, pairwise, classobj, fixed.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.207 [0.183 0.241]	0.357 [0.280 0.480]	0.262 [0.222 0.321]	202.7	120.7	42.0
candle	0.013 [0.009 0.019]	0.389 [0.167 0.667]	0.025 [0.016 0.037]	105.0	4.0	1.3
cup	0.313 [0.277 0.331]	0.579 [0.513 0.680]	0.402 [0.394 0.412]	178.3	98.3	55.7
fork	0.255 [0.237 0.267]	0.633 [0.530 0.709]	0.361 [0.349 0.380]	459.0	188.7	117.0
forkside	0.008 [0.000 0.012]	0.389 [0.000 1.000]	0.015 [0.000 0.023]	86.3	2.7	0.7
glass	0.035 [0.019 0.063]	0.772 [0.667 0.929]	0.065 [0.037 0.116]	682.3	31.7	23.3
knife	0.246 [0.227 0.264]	0.568 [0.536 0.592]	0.343 [0.326 0.365]	345.3	150.7	85.3
knifeside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	77.0	0.0	0.0
napkin	0.102 [0.092 0.113]	0.218 [0.210 0.226]	0.138 [0.130 0.150]	295.7	137.7	30.0
placemat	0.035 [0.027 0.047]	0.607 [0.571 0.667]	0.066 [0.052 0.088]	141.0	8.3	5.0
plate	0.426 [0.414 0.439]	0.698 [0.634 0.735]	0.528 [0.519 0.539]	802.0	492.0	341.7
saucer	0.274 [0.223 0.359]	0.640 [0.509 0.833]	0.375 [0.310 0.443]	116.3	51.0	31.3
shaker	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	77.7	0.0	0.0
spoon	0.399 [0.388 0.420]	0.688 [0.570 0.874]	0.499 [0.477 0.537]	223.3	134.0	89.0
spoonside	0.054 [0.017 0.077]	0.570 [0.400 0.727]	0.098 [0.033 0.139]	107.0	9.3	5.7
cumulative	0.212 [0.201 0.220]	0.579 [0.569 0.587]	0.311 [0.297 0.320]	3899.0	1429.0	828.0

Table B.33: Learned 2-level WGGs, big placesetting dataset, pairwise, classobj, fixed.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.169 [0.148 0.207]	0.306 [0.200 0.365]	0.215 [0.174 0.264]	202.7	118.3	34.3
candle	0.013 [0.000 0.029]	0.617 [0.250 1.000]	0.024 [0.000 0.055]	105.0	3.0	1.3
cup	0.343 [0.290 0.379]	0.549 [0.486 0.583]	0.419 [0.387 0.443]	178.3	113.3	61.3
fork	0.297 [0.273 0.315]	0.567 [0.467 0.651]	0.387 [0.376 0.399]	459.0	248.3	136.7
forkside	0.000 [0.000 0.000]	0.333 [0.000 1.000]	0.000 [0.000 0.000]	86.3	1.7	0.0
glass	0.237 [0.197 0.262]	0.417 [0.394 0.461]	0.299 [0.276 0.315]	682.3	392.0	161.3
knife	0.280 [0.277 0.284]	0.444 [0.427 0.469]	0.343 [0.339 0.349]	345.3	218.0	96.7
knifeside	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	77.0	1.0	0.0
napkin	0.100 [0.076 0.126]	0.161 [0.138 0.179]	0.121 [0.106 0.143]	295.7	188.0	29.7
placemat	0.026 [0.016 0.041]	0.589 [0.500 0.667]	0.049 [0.030 0.076]	141.0	6.0	3.7
plate	0.390 [0.371 0.416]	0.733 [0.668 0.806]	0.507 [0.500 0.513]	802.0	430.0	312.3
saucer	0.326 [0.289 0.368]	0.536 [0.427 0.600]	0.405 [0.345 0.451]	116.3	72.0	38.0
shaker	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	77.7	0.7	0.0
spoon	0.409 [0.375 0.444]	0.636 [0.595 0.683]	0.496 [0.484 0.509]	223.3	144.7	91.3
spoonside	0.036 [0.000 0.108]	0.167 [0.000 0.500]	0.059 [0.000 0.177]	107.0	9.0	3.7
cumulative	0.249 [0.231 0.264]	0.504 [0.471 0.562]	0.332 [0.327 0.338]	3899.0	1946.0	970.3

Table B.34: Learned 2-level WGGs, big placesetting dataset, pairwise, classobj, free.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.194 [0.172 0.232]	0.333 [0.259 0.448]	0.244 [0.211 0.305]	202.7	121.3	39.3
candle	0.019 [0.011 0.029]	0.472 [0.333 0.750]	0.036 [0.021 0.055]	105.0	4.3	2.0
cup	0.293 [0.245 0.319]	0.623 [0.477 0.882]	0.385 [0.379 0.392]	178.3	92.0	52.0
fork	0.263 [0.246 0.280]	0.667 [0.564 0.727]	0.376 [0.358 0.401]	459.0	184.0	120.7
forkside	0.008 [0.000 0.025]	0.697 [0.091 1.000]	0.013 [0.000 0.039]	86.3	7.3	0.7
glass	0.045 [0.034 0.066]	0.783 [0.632 1.000]	0.083 [0.064 0.121]	682.3	40.3	30.0
knife	0.244 [0.224 0.267]	0.591 [0.540 0.618]	0.346 [0.329 0.373]	345.3	144.0	84.7
knifeside	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	77.0	0.3	0.0
napkin	0.105 [0.083 0.117]	0.240 [0.212 0.274]	0.146 [0.119 0.164]	295.7	129.0	31.0
placemat	0.040 [0.027 0.061]	0.589 [0.500 0.667]	0.074 [0.052 0.110]	141.0	9.7	5.7
plate	0.420 [0.394 0.446]	0.717 [0.647 0.757]	0.528 [0.519 0.538]	802.0	473.7	337.0
saucer	0.297 [0.231 0.388]	0.645 [0.538 0.791]	0.401 [0.324 0.473]	116.3	53.7	34.0
shaker	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	77.7	0.0	0.0
spoon	0.405 [0.397 0.415]	0.714 [0.600 0.836]	0.514 [0.491 0.538]	223.3	129.0	90.3
spoonside	0.052 [0.009 0.077]	0.432 [0.200 0.727]	0.091 [0.017 0.139]	107.0	11.7	5.3
cumulative	0.214 [0.205 0.219]	0.595 [0.588 0.603]	0.314 [0.305 0.322]	3899.0	1400.3	832.7

Table B.35: Learned 2-level WGGs, big placesetting dataset, pairwise, classobj, free.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.171 [0.139 0.197]	0.405 [0.301 0.476]	0.241 [0.190 0.279]	202.7	86.3	34.7
candle	0.016 [0.011 0.019]	0.236 [0.125 0.333]	0.029 [0.020 0.036]	105.0	7.3	1.7
cup	0.304 [0.296 0.313]	0.628 [0.564 0.718]	0.409 [0.397 0.427]	178.3	87.3	54.3
fork	0.263 [0.236 0.292]	0.610 [0.504 0.741]	0.362 [0.359 0.369]	459.0	206.0	120.7
forkside	0.004 [0.000 0.011]	0.833 [0.500 1.000]	0.007 [0.000 0.021]	86.3	0.7	0.3
glass	0.003 [0.000 0.009]	0.867 [0.600 1.000]	0.006 [0.000 0.018]	682.3	3.3	2.0
knife	0.252 [0.234 0.281]	0.564 [0.536 0.596]	0.348 [0.325 0.382]	345.3	154.3	87.3
knifeside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	77.0	0.0	0.0
napkin	0.095 [0.066 0.123]	0.210 [0.206 0.217]	0.129 [0.100 0.157]	295.7	132.7	28.0
placemat	0.026 [0.016 0.034]	0.595 [0.500 0.714]	0.049 [0.030 0.065]	141.0	6.0	3.7
plate	0.440 [0.417 0.464]	0.684 [0.649 0.722]	0.534 [0.517 0.545]	802.0	516.7	352.3
saucer	0.321 [0.264 0.388]	0.618 [0.519 0.765]	0.416 [0.362 0.444]	116.3	61.3	37.0
shaker	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	77.7	0.0	0.0
spoon	0.366 [0.276 0.420]	0.687 [0.632 0.762]	0.470 [0.405 0.515]	223.3	120.3	81.3
spoonside	0.101 [0.070 0.147]	0.582 [0.326 0.727]	0.161 [0.127 0.203]	107.0	23.3	10.7
cumulative	0.209 [0.206 0.212]	0.580 [0.550 0.612]	0.307 [0.303 0.315]	3899.0	1405.7	814.0

Table B.36: Learned 2-level WGGs, big placesetting dataset, nopairwise, classobj, fixed.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.150 [0.099 0.192]	0.467 [0.278 0.591]	0.226 [0.146 0.290]	202.7	66.0	30.3
candle	0.013 [0.009 0.019]	0.192 [0.091 0.286]	0.024 [0.016 0.036]	105.0	7.7	1.3
cup	0.235 [0.207 0.280]	0.703 [0.614 0.816]	0.350 [0.310 0.397]	178.3	60.3	42.0
fork	0.227 [0.200 0.260]	0.666 [0.496 0.754]	0.333 [0.315 0.342]	459.0	167.0	104.3
forkside	0.007 [0.000 0.022]	0.778 [0.333 1.000]	0.013 [0.000 0.040]	86.3	2.0	0.7
glass	0.011 [0.000 0.034]	0.933 [0.800 1.000]	0.022 [0.000 0.065]	682.3	10.0	8.0
knife	0.202 [0.187 0.227]	0.704 [0.693 0.721]	0.314 [0.295 0.346]	345.3	99.3	70.0
knifeside	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	77.0	0.3	0.0
napkin	0.080 [0.059 0.102]	0.262 [0.247 0.273]	0.122 [0.098 0.148]	295.7	90.7	23.7
placemat	0.030 [0.023 0.041]	0.640 [0.571 0.750]	0.058 [0.045 0.077]	141.0	6.7	4.3
plate	0.380 [0.368 0.398]	0.841 [0.748 0.901]	0.522 [0.519 0.528]	802.0	365.3	304.7
saucer	0.243 [0.132 0.350]	0.727 [0.679 0.775]	0.354 [0.224 0.462]	116.3	38.3	27.7
shaker	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	77.7	0.0	0.0
spoon	0.297 [0.220 0.344]	0.783 [0.658 0.895]	0.423 [0.353 0.464]	223.3	87.3	66.0
spoonside	0.102 [0.052 0.147]	0.576 [0.366 0.750]	0.163 [0.098 0.210]	107.0	22.3	10.7
cumulative	0.178 [0.172 0.184]	0.682 [0.618 0.722]	0.282 [0.278 0.284]	3899.0	1023.3	693.7

Table B.37: Learned 2-level WGGs, big placesetting dataset, nopairwise, classobj, fixed.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.135 [0.079 0.167]	0.325 [0.208 0.386]	0.191 [0.115 0.234]	202.7	83.0	27.3
candle	0.019 [0.009 0.038]	0.156 [0.083 0.286]	0.034 [0.016 0.067]	105.0	12.0	2.0
cup	0.315 [0.296 0.326]	0.604 [0.566 0.649]	0.414 [0.407 0.420]	178.3	94.0	56.3
fork	0.272 [0.244 0.317]	0.616 [0.538 0.720]	0.374 [0.345 0.412]	459.0	206.7	124.7
forkside	0.008 [0.000 0.024]	0.061 [0.000 0.182]	0.014 [0.000 0.042]	86.3	6.0	0.7
glass	0.271 [0.254 0.282]	0.416 [0.400 0.437]	0.328 [0.310 0.343]	682.3	444.0	184.7
knife	0.272 [0.264 0.278]	0.481 [0.471 0.497]	0.348 [0.339 0.353]	345.3	195.7	94.0
knifeside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	77.0	0.0	0.0
napkin	0.092 [0.078 0.109]	0.183 [0.172 0.195]	0.122 [0.110 0.133]	295.7	150.3	27.3
placemat	0.002 [0.000 0.007]	0.667 [0.000 1.000]	0.005 [0.000 0.014]	141.0	0.7	0.3
plate	0.394 [0.383 0.400]	0.752 [0.728 0.777]	0.516 [0.513 0.520]	802.0	420.3	315.7
saucer	0.199 [0.149 0.264]	0.777 [0.720 0.826]	0.315 [0.247 0.395]	116.3	30.0	23.3
shaker	0.000 [0.000 0.000]	0.333 [0.000 1.000]	0.000 [0.000 0.000]	77.7	1.3	0.0
spoon	0.367 [0.308 0.397]	0.769 [0.680 0.844]	0.494 [0.442 0.540]	223.3	107.3	82.0
spoonside	0.058 [0.019 0.087]	0.391 [0.200 0.538]	0.101 [0.035 0.145]	107.0	15.3	6.3
cumulative	0.242 [0.237 0.250]	0.535 [0.519 0.546]	0.333 [0.325 0.343]	3899.0	1766.7	944.7

Table B.38: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.1$, nosingletocomp.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.089 [0.074 0.118]	0.310 [0.227 0.414]	0.138 [0.112 0.184]	202.7	58.7	18.0
candle	0.021 [0.000 0.034]	0.655 [0.364 1.000]	0.039 [0.000 0.063]	105.0	5.3	2.3
cup	0.252 [0.179 0.314]	0.590 [0.541 0.635]	0.347 [0.280 0.397]	178.3	77.0	44.7
fork	0.188 [0.015 0.296]	0.648 [0.468 1.000]	0.241 [0.030 0.362]	459.0	179.7	86.0
forkside	0.008 [0.000 0.012]	0.078 [0.000 0.200]	0.013 [0.000 0.022]	86.3	12.0	0.7
glass	0.148 [0.091 0.178]	0.550 [0.500 0.625]	0.228 [0.159 0.266]	682.3	188.0	100.3
knife	0.205 [0.134 0.259]	0.602 [0.532 0.705]	0.298 [0.225 0.348]	345.3	124.7	71.7
knifeside	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	77.0	2.0	0.0
napkin	0.087 [0.055 0.117]	0.181 [0.125 0.232]	0.110 [0.088 0.121]	295.7	162.3	25.7
placemat	0.009 [0.000 0.027]	0.889 [0.667 1.000]	0.018 [0.000 0.053]	141.0	2.0	1.3
plate	0.202 [0.020 0.316]	0.552 [0.219 0.762]	0.288 [0.036 0.430]	802.0	243.0	161.0
saucer	0.207 [0.136 0.311]	0.676 [0.516 0.944]	0.297 [0.238 0.388]	116.3	39.0	23.3
shaker	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	77.7	0.0	0.0
spoon	0.258 [0.089 0.444]	0.657 [0.592 0.701]	0.341 [0.157 0.507]	223.3	93.0	58.7
spoonside	0.068 [0.000 0.127]	0.290 [0.000 0.450]	0.109 [0.000 0.195]	107.0	17.3	7.3
cumulative	0.154 [0.114 0.193]	0.494 [0.410 0.543]	0.234 [0.178 0.285]	3899.0	1204.0	601.0

Table B.39: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.1$, singletocomp1.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.086 [0.059 0.109]	0.369 [0.232 0.474]	0.133 [0.103 0.149]	202.7	54.3	17.3
candle	0.003 [0.000 0.009]	0.389 [0.000 1.000]	0.005 [0.000 0.016]	105.0	3.0	0.3
cup	0.247 [0.245 0.249]	0.703 [0.634 0.763]	0.365 [0.353 0.373]	178.3	63.0	44.0
fork	0.204 [0.088 0.267]	0.698 [0.603 0.796]	0.301 [0.159 0.374]	459.0	142.3	94.3
forkside	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	86.3	3.7	0.0
glass	0.144 [0.094 0.177]	0.542 [0.502 0.588]	0.224 [0.162 0.262]	682.3	183.0	97.3
knife	0.222 [0.209 0.234]	0.627 [0.564 0.673]	0.326 [0.319 0.330]	345.3	122.3	76.3
knifeside	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	77.0	1.3	0.0
napkin	0.071 [0.061 0.083]	0.277 [0.231 0.300]	0.112 [0.102 0.122]	295.7	78.3	21.0
placemat	0.010 [0.007 0.014]	0.833 [0.500 1.000]	0.018 [0.013 0.027]	141.0	1.7	1.3
plate	0.239 [0.170 0.329]	0.742 [0.569 0.846]	0.360 [0.262 0.474]	802.0	255.7	191.7
saucer	0.204 [0.174 0.223]	0.881 [0.844 0.913]	0.331 [0.292 0.357]	116.3	27.0	23.7
shaker	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	77.7	0.3	0.0
spoon	0.335 [0.290 0.379]	0.734 [0.706 0.765]	0.459 [0.415 0.507]	223.3	102.0	75.0
spoonside	0.051 [0.026 0.088]	0.575 [0.281 1.000]	0.085 [0.051 0.134]	107.0	14.7	5.3
cumulative	0.166 [0.132 0.194]	0.611 [0.536 0.671]	0.261 [0.211 0.301]	3899.0	1052.7	647.7

Table B.40: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.1$, singletocomp2.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.168 [0.134 0.202]	0.311 [0.225 0.386]	0.217 [0.168 0.248]	202.7	111.7	34.0
candle	0.010 [0.000 0.019]	0.217 [0.000 0.400]	0.019 [0.000 0.036]	105.0	4.7	1.0
cup	0.311 [0.272 0.335]	0.588 [0.530 0.622]	0.405 [0.379 0.426]	178.3	95.7	55.7
fork	0.289 [0.262 0.328]	0.570 [0.471 0.667]	0.381 [0.349 0.417]	459.0	238.7	132.7
forkside	0.004 [0.000 0.011]	0.400 [0.000 1.000]	0.007 [0.000 0.020]	86.3	2.7	0.3
glass	0.271 [0.245 0.284]	0.414 [0.406 0.425]	0.327 [0.307 0.340]	682.3	445.3	184.3
knife	0.268 [0.264 0.277]	0.481 [0.436 0.530]	0.344 [0.339 0.353]	345.3	193.3	92.7
knifeside	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	77.0	0.7	0.0
napkin	0.092 [0.086 0.100]	0.153 [0.145 0.158]	0.114 [0.108 0.122]	295.7	176.7	27.0
placemat	0.002 [0.000 0.007]	0.667 [0.000 1.000]	0.005 [0.000 0.014]	141.0	0.7	0.3
plate	0.390 [0.378 0.396]	0.771 [0.758 0.786]	0.517 [0.507 0.525]	802.0	405.0	312.3
saucer	0.192 [0.066 0.288]	0.815 [0.735 0.889]	0.296 [0.123 0.414]	116.3	28.7	22.3
shaker	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	77.7	0.0	0.0
spoon	0.397 [0.348 0.422]	0.747 [0.672 0.848]	0.515 [0.494 0.533]	223.3	120.7	88.7
spoonside	0.046 [0.009 0.118]	0.403 [0.333 0.500]	0.072 [0.017 0.179]	107.0	12.3	4.7
cumulative	0.245 [0.240 0.251]	0.521 [0.514 0.527]	0.333 [0.328 0.340]	3899.0	1836.7	956.0

Table B.41: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.15$, nosingletocomp.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.076 [0.059 0.089]	0.317 [0.214 0.409]	0.122 [0.093 0.146]	202.7	49.7	15.3
candle	0.013 [0.010 0.017]	0.245 [0.200 0.286]	0.024 [0.018 0.032]	105.0	5.3	1.3
cup	0.229 [0.190 0.284]	0.668 [0.578 0.765]	0.337 [0.295 0.381]	178.3	62.3	40.7
fork	0.275 [0.257 0.304]	0.587 [0.486 0.651]	0.373 [0.342 0.409]	459.0	218.7	126.3
forkside	0.004 [0.000 0.012]	0.006 [0.000 0.019]	0.005 [0.000 0.015]	86.3	19.0	0.3
glass	0.212 [0.162 0.269]	0.467 [0.412 0.502]	0.287 [0.246 0.325]	682.3	317.0	144.3
knife	0.255 [0.227 0.284]	0.492 [0.389 0.690]	0.327 [0.308 0.342]	345.3	195.3	88.3
knifeside	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	77.0	0.7	0.0
napkin	0.088 [0.055 0.106]	0.207 [0.150 0.302]	0.115 [0.092 0.128]	295.7	148.3	26.0
placemat	0.011 [0.000 0.034]	0.875 [0.625 1.000]	0.022 [0.000 0.065]	141.0	2.7	1.7
plate	0.224 [0.047 0.315]	0.568 [0.253 0.743]	0.318 [0.079 0.438]	802.0	279.3	179.0
saucer	0.197 [0.096 0.330]	0.730 [0.642 0.857]	0.292 [0.173 0.436]	116.3	32.0	22.0
shaker	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	77.7	0.0	0.0
spoon	0.388 [0.286 0.444]	0.662 [0.516 0.842]	0.473 [0.427 0.514]	223.3	140.3	86.7
spoonside	0.039 [0.009 0.098]	0.568 [0.250 1.000]	0.066 [0.017 0.161]	107.0	9.0	4.0
cumulative	0.189 [0.164 0.209]	0.496 [0.462 0.526]	0.273 [0.242 0.300]	3899.0	1479.7	736.0

Table B.42: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.15$, singletocomp1.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.129 [0.099 0.148]	0.359 [0.262 0.429]	0.186 [0.157 0.220]	202.7	76.3	26.0
candle	0.017 [0.009 0.032]	0.205 [0.115 0.250]	0.028 [0.017 0.050]	105.0	11.3	1.7
cup	0.248 [0.213 0.280]	0.701 [0.648 0.766]	0.364 [0.333 0.398]	178.3	64.0	44.3
fork	0.279 [0.257 0.308]	0.612 [0.536 0.691]	0.381 [0.360 0.409]	459.0	212.3	128.0
forkside	0.000 [0.000 0.000]	0.333 [0.000 1.000]	0.000 [0.000 0.000]	86.3	1.0	0.0
glass	0.207 [0.158 0.281]	0.492 [0.409 0.553]	0.283 [0.242 0.333]	682.3	301.3	141.3
knife	0.270 [0.243 0.290]	0.474 [0.443 0.500]	0.344 [0.314 0.367]	345.3	197.0	93.7
knifeside	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	77.0	0.0	0.0
napkin	0.073 [0.048 0.096]	0.210 [0.168 0.244]	0.105 [0.079 0.122]	295.7	109.0	21.7
placemat	0.005 [0.000 0.014]	1.000 [1.000 1.000]	0.009 [0.000 0.027]	141.0	0.7	0.7
plate	0.151 [0.010 0.280]	0.658 [0.444 0.855]	0.234 [0.019 0.422]	802.0	157.0	120.3
saucer	0.234 [0.207 0.272]	0.800 [0.758 0.852]	0.361 [0.325 0.405]	116.3	34.3	27.3
shaker	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	77.7	0.0	0.0
spoon	0.370 [0.344 0.401]	0.699 [0.605 0.770]	0.482 [0.455 0.515]	223.3	119.3	82.7
spoonside	0.029 [0.000 0.088]	0.564 [0.000 1.000]	0.052 [0.000 0.157]	107.0	4.7	3.0
cumulative	0.177 [0.161 0.204]	0.535 [0.498 0.600]	0.266 [0.244 0.304]	3899.0	1288.3	690.7

Table B.43: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.15$, singletocomp2.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.156 [0.124 0.177]	0.322 [0.217 0.382]	0.210 [0.158 0.239]	202.7	100.7	31.7
candle	0.017 [0.009 0.022]	0.252 [0.071 0.400]	0.030 [0.015 0.040]	105.0	8.7	1.7
cup	0.292 [0.237 0.324]	0.611 [0.586 0.625]	0.393 [0.343 0.426]	178.3	86.0	52.3
fork	0.318 [0.255 0.371]	0.526 [0.467 0.585]	0.392 [0.355 0.435]	459.0	283.7	146.3
forkside	0.000 [0.000 0.000]	0.000 [0.000 0.000]	0.000 [0.000 0.000]	86.3	5.0	0.0
glass	0.273 [0.266 0.284]	0.400 [0.389 0.421]	0.324 [0.316 0.339]	682.3	465.3	186.0
knife	0.271 [0.255 0.281]	0.450 [0.406 0.500]	0.338 [0.314 0.355]	345.3	208.7	93.7
knifeside	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	77.0	1.0	0.0
napkin	0.112 [0.102 0.117]	0.164 [0.152 0.177]	0.133 [0.122 0.140]	295.7	201.7	33.0
placemat	0.007 [0.000 0.014]	0.833 [0.500 1.000]	0.013 [0.000 0.027]	141.0	1.3	1.0
plate	0.393 [0.373 0.404]	0.753 [0.741 0.760]	0.517 [0.500 0.528]	802.0	418.7	315.3
saucer	0.202 [0.165 0.248]	0.780 [0.690 0.833]	0.321 [0.267 0.380]	116.3	30.3	23.7
shaker	0.004 [0.000 0.013]	0.417 [0.000 1.000]	0.008 [0.000 0.025]	77.7	2.0	0.3
spoon	0.387 [0.339 0.457]	0.674 [0.642 0.724]	0.488 [0.462 0.534]	223.3	129.7	86.7
spoonside	0.057 [0.010 0.127]	0.596 [0.308 1.000]	0.094 [0.019 0.202]	107.0	13.7	6.0
cumulative	0.251 [0.238 0.258]	0.500 [0.487 0.516]	0.334 [0.320 0.344]	3899.0	1956.3	977.7

Table B.44: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.2$, nosingletocomp.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.046 [0.030 0.079]	0.263 [0.214 0.333]	0.075 [0.052 0.119]	202.7	37.3	9.3
candle	0.013 [0.010 0.017]	0.219 [0.125 0.333]	0.023 [0.019 0.031]	105.0	7.0	1.3
cup	0.213 [0.178 0.264]	0.789 [0.716 0.833]	0.332 [0.293 0.386]	178.3	49.0	38.0
fork	0.308 [0.257 0.362]	0.534 [0.468 0.613]	0.386 [0.362 0.427]	459.0	271.7	141.7
forkside	0.004 [0.000 0.012]	0.167 [0.000 0.500]	0.008 [0.000 0.023]	86.3	2.0	0.3
glass	0.264 [0.245 0.275]	0.400 [0.362 0.433]	0.318 [0.292 0.336]	682.3	452.0	180.0
knife	0.274 [0.268 0.281]	0.415 [0.347 0.535]	0.327 [0.302 0.369]	345.3	235.3	94.7
knifeside	0.000 [0.000 0.000]	0.333 [0.000 1.000]	0.000 [0.000 0.000]	77.0	0.7	0.0
napkin	0.109 [0.086 0.130]	0.166 [0.137 0.189]	0.131 [0.105 0.154]	295.7	192.7	32.0
placemat	0.005 [0.000 0.007]	0.833 [0.500 1.000]	0.009 [0.000 0.014]	141.0	1.0	0.7
plate	0.213 [0.050 0.338]	0.545 [0.288 0.748]	0.302 [0.085 0.466]	802.0	280.0	171.3
saucer	0.166 [0.074 0.233]	0.802 [0.649 0.900]	0.265 [0.137 0.343]	116.3	25.0	19.0
shaker	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	77.7	0.3	0.0
spoon	0.394 [0.344 0.440]	0.634 [0.600 0.694]	0.482 [0.460 0.507]	223.3	140.3	88.0
spoonside	0.054 [0.000 0.127]	0.469 [0.000 1.000]	0.087 [0.000 0.194]	107.0	12.3	5.7
cumulative	0.201 [0.155 0.232]	0.455 [0.393 0.492]	0.278 [0.222 0.313]	3899.0	1706.7	782.0

Table B.45: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.2$, singletocomp1.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.089 [0.064 0.119]	0.339 [0.276 0.394]	0.137 [0.110 0.166]	202.7	56.3	18.0
candle	0.009 [0.000 0.017]	0.089 [0.000 0.143]	0.016 [0.000 0.030]	105.0	10.0	1.0
cup	0.223 [0.183 0.253]	0.787 [0.729 0.836]	0.347 [0.298 0.388]	178.3	51.0	40.0
fork	0.336 [0.289 0.366]	0.518 [0.435 0.601]	0.403 [0.390 0.421]	459.0	308.3	154.7
forkside	0.008 [0.000 0.024]	0.048 [0.000 0.143]	0.013 [0.000 0.040]	86.3	6.0	0.7
glass	0.266 [0.252 0.275]	0.388 [0.375 0.410]	0.316 [0.302 0.329]	682.3	466.0	181.0
knife	0.280 [0.273 0.289]	0.421 [0.367 0.462]	0.335 [0.324 0.343]	345.3	233.0	96.7
knifeside	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	77.0	0.3	0.0
napkin	0.130 [0.116 0.157]	0.170 [0.138 0.211]	0.147 [0.126 0.180]	295.7	227.0	38.3
placemat	0.002 [0.000 0.007]	1.000 [1.000 1.000]	0.005 [0.000 0.014]	141.0	0.3	0.3
plate	0.261 [0.237 0.305]	0.648 [0.527 0.725]	0.371 [0.327 0.424]	802.0	327.0	209.3
saucer	0.160 [0.000 0.248]	0.887 [0.750 1.000]	0.249 [0.000 0.390]	116.3	22.0	18.3
shaker	0.000 [0.000 0.000]	0.333 [0.000 1.000]	0.000 [0.000 0.000]	77.7	1.0	0.0
spoon	0.370 [0.332 0.431]	0.676 [0.654 0.716]	0.476 [0.441 0.519]	223.3	123.3	83.0
spoonside	0.059 [0.000 0.098]	0.555 [0.281 1.000]	0.093 [0.000 0.156]	107.0	19.3	6.3
cumulative	0.217 [0.204 0.235]	0.458 [0.456 0.461]	0.295 [0.283 0.310]	3899.0	1851.0	847.7

Table B.46: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.2$, singletocomp2.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.195 [0.153 0.251]	0.304 [0.203 0.367]	0.237 [0.175 0.298]	202.7	133.3	39.7
candle	0.017 [0.009 0.022]	0.223 [0.125 0.400]	0.030 [0.016 0.037]	105.0	9.0	1.7
cup	0.303 [0.290 0.310]	0.587 [0.566 0.606]	0.399 [0.389 0.410]	178.3	92.0	54.0
fork	0.326 [0.275 0.369]	0.542 [0.491 0.626]	0.402 [0.382 0.428]	459.0	283.7	150.0
forkside	0.000 [0.000 0.000]	0.000 [0.000 0.000]	0.000 [0.000 0.000]	86.3	3.3	0.0
glass	0.279 [0.269 0.287]	0.415 [0.408 0.423]	0.334 [0.324 0.339]	682.3	458.3	190.3
knife	0.286 [0.284 0.287]	0.435 [0.409 0.455]	0.345 [0.337 0.352]	345.3	226.7	98.7
knifeside	0.000 [0.000 0.000]	0.333 [0.000 1.000]	0.000 [0.000 0.000]	77.0	1.7	0.0
napkin	0.110 [0.092 0.143]	0.152 [0.132 0.174]	0.128 [0.109 0.157]	295.7	213.3	32.7
placemat	0.012 [0.007 0.016]	0.611 [0.333 1.000]	0.023 [0.013 0.030]	141.0	3.0	1.7
plate	0.398 [0.380 0.417]	0.761 [0.752 0.778]	0.523 [0.505 0.537]	802.0	420.0	319.7
saucer	0.209 [0.140 0.272]	0.776 [0.739 0.815]	0.325 [0.236 0.402]	116.3	31.3	24.3
shaker	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	77.7	0.3	0.0
spoon	0.432 [0.415 0.457]	0.650 [0.627 0.679]	0.519 [0.513 0.529]	223.3	149.0	96.7
spoonside	0.063 [0.000 0.118]	0.262 [0.000 0.400]	0.100 [0.000 0.180]	107.0	17.7	6.7
cumulative	0.261 [0.252 0.266]	0.498 [0.489 0.511]	0.342 [0.333 0.348]	3899.0	2042.7	1016.0

Table B.47: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.3$, nosingletocomp.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.136 [0.044 0.188]	0.317 [0.248 0.383]	0.178 [0.078 0.242]	202.7	91.7	27.7
candle	0.014 [0.009 0.022]	0.270 [0.143 0.500]	0.024 [0.016 0.038]	105.0	7.0	1.3
cup	0.235 [0.207 0.286]	0.687 [0.650 0.731]	0.348 [0.322 0.397]	178.3	61.7	42.0
fork	0.328 [0.287 0.353]	0.544 [0.459 0.635]	0.404 [0.395 0.419]	459.0	285.7	150.7
forkside	0.000 [0.000 0.000]	0.000 [0.000 0.000]	0.000 [0.000 0.000]	86.3	2.0	0.0
glass	0.264 [0.258 0.275]	0.400 [0.390 0.409]	0.318 [0.311 0.329]	682.3	450.0	180.0
knife	0.273 [0.261 0.289]	0.426 [0.396 0.443]	0.333 [0.320 0.350]	345.3	221.0	94.3
knifeside	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	77.0	0.7	0.0
napkin	0.114 [0.102 0.123]	0.165 [0.158 0.179]	0.135 [0.124 0.146]	295.7	203.7	33.7
placemat	0.012 [0.007 0.016]	0.667 [0.500 1.000]	0.023 [0.013 0.030]	141.0	2.7	1.7
plate	0.404 [0.319 0.461]	0.706 [0.656 0.752]	0.509 [0.448 0.541]	802.0	463.0	323.7
saucer	0.132 [0.000 0.272]	0.553 [0.000 0.882]	0.207 [0.000 0.403]	116.3	18.0	14.3
shaker	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	77.7	0.0	0.0
spoon	0.416 [0.348 0.466]	0.608 [0.574 0.634]	0.491 [0.450 0.514]	223.3	154.0	93.0
spoonside	0.054 [0.000 0.127]	0.383 [0.000 0.667]	0.089 [0.000 0.202]	107.0	11.7	5.7
cumulative	0.248 [0.229 0.274]	0.491 [0.485 0.499]	0.329 [0.313 0.350]	3899.0	1972.7	968.0

Table B.48: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.3$, singletocomp1.

	recall	precision	f-measure	# targets	# detections	# correct
bowl	0.179 [0.133 0.217]	0.335 [0.248 0.397]	0.228 [0.199 0.271]	202.7	114.3	36.3
candle	0.013 [0.009 0.019]	0.209 [0.083 0.400]	0.024 [0.016 0.036]	105.0	8.0	1.3
cup	0.203 [0.172 0.231]	0.778 [0.731 0.853]	0.320 [0.286 0.353]	178.3	47.3	36.3
fork	0.331 [0.287 0.367]	0.542 [0.458 0.635]	0.406 [0.389 0.434]	459.0	288.7	152.0
forkside	0.000 [0.000 0.000]	0.000 [0.000 0.000]	0.000 [0.000 0.000]	86.3	3.0	0.0
glass	0.275 [0.258 0.288]	0.414 [0.401 0.426]	0.331 [0.314 0.344]	682.3	453.3	187.7
knife	0.274 [0.267 0.287]	0.438 [0.396 0.460]	0.337 [0.320 0.353]	345.3	216.3	94.7
knifeside	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	77.0	0.7	0.0
napkin	0.114 [0.102 0.131]	0.173 [0.158 0.193]	0.138 [0.124 0.156]	295.7	194.0	33.7
placemat	0.010 [0.000 0.016]	0.500 [0.000 1.000]	0.019 [0.000 0.030]	141.0	2.3	1.3
plate	0.362 [0.203 0.452]	0.619 [0.509 0.711]	0.452 [0.290 0.538]	802.0	458.3	290.0
saucer	0.094 [0.000 0.184]	0.635 [0.000 1.000]	0.162 [0.000 0.306]	116.3	11.3	10.3
shaker	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	77.7	0.3	0.0
spoon	0.403 [0.295 0.478]	0.652 [0.572 0.767]	0.486 [0.426 0.521]	223.3	143.7	90.0
spoonside	0.066 [0.000 0.127]	0.292 [0.000 0.542]	0.107 [0.000 0.206]	107.0	16.7	7.0
cumulative	0.241 [0.211 0.271]	0.480 [0.463 0.490]	0.320 [0.290 0.348]	3899.0	1958.3	940.7

Table B.49: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.3$, singletocomp2.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.373 [0.360 0.388]	0.376 [0.360 0.398]	0.374 [0.360 0.393]	110.0	109.0	41.0
chimney	0.164 [0.132 0.219]	0.164 [0.132 0.219]	0.164 [0.132 0.219]	344.0	344.0	55.7
door	0.283 [0.220 0.319]	0.283 [0.220 0.319]	0.283 [0.220 0.319]	497.0	497.0	141.0
driveway	0.170 [0.121 0.230]	0.227 [0.173 0.273]	0.188 [0.165 0.233]	145.3	112.3	24.7
garagedoor	0.327 [0.306 0.349]	0.328 [0.306 0.349]	0.327 [0.306 0.349]	142.7	142.3	46.7
path	0.055 [0.052 0.058]	0.055 [0.052 0.058]	0.055 [0.052 0.058]	199.7	199.7	11.0
roof	0.262 [0.260 0.264]	0.263 [0.260 0.266]	0.263 [0.260 0.265]	891.3	889.0	234.0
shutter	0.237 [0.225 0.249]	0.237 [0.225 0.249]	0.237 [0.225 0.249]	819.7	819.7	194.3
steps	0.121 [0.100 0.135]	0.121 [0.100 0.135]	0.121 [0.100 0.135]	272.0	272.0	33.0
window	0.199 [0.186 0.212]	0.199 [0.186 0.212]	0.199 [0.186 0.212]	3136.3	3136.3	624.7
cumulative	0.214 [0.206 0.223]	0.216 [0.206 0.224]	0.215 [0.206 0.223]	6558.0	6521.3	1406.0

Table B.50: DTPBM object detector, house dataset, known # objects.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.420 [0.400 0.440]	0.030 [0.025 0.036]	0.057 [0.047 0.067]	110.0	1535.3	46.3
chimney	0.263 [0.241 0.304]	0.027 [0.023 0.032]	0.050 [0.043 0.056]	344.0	3333.3	90.0
door	0.510 [0.441 0.548]	0.071 [0.060 0.078]	0.124 [0.105 0.136]	497.0	3599.0	254.0
driveway	0.209 [0.121 0.277]	0.088 [0.062 0.133]	0.111 [0.098 0.127]	145.3	417.7	30.3
garagedoor	0.439 [0.431 0.452]	0.047 [0.042 0.057]	0.085 [0.076 0.101]	142.7	1348.3	62.7
path	0.100 [0.073 0.116]	0.019 [0.018 0.019]	0.031 [0.028 0.033]	199.7	1066.7	20.0
roof	0.335 [0.332 0.338]	0.103 [0.083 0.119]	0.158 [0.133 0.176]	891.3	2946.0	299.0
shutter	0.390 [0.374 0.417]	0.027 [0.026 0.028]	0.049 [0.048 0.052]	819.7	12000.0	318.3
steps	0.185 [0.157 0.204]	0.038 [0.034 0.044]	0.063 [0.056 0.072]	272.0	1333.3	50.3
window	0.331 [0.312 0.347]	0.075 [0.064 0.093]	0.122 [0.108 0.143]	3136.3	14266.7	1039.0
cumulative	0.337 [0.329 0.346]	0.053 [0.049 0.058]	0.092 [0.085 0.098]	6558.0	41846.3	2210.0

Table B.51: DTPBM object detector, house dataset, heuristic # objects.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.157 [0.080 0.219]	0.614 [0.471 0.714]	0.248 [0.137 0.329]	110.0	27.7	17.7
chimney	0.004 [0.003 0.006]	0.238 [0.071 0.500]	0.008 [0.005 0.012]	344.0	12.3	1.3
door	0.149 [0.099 0.193]	0.191 [0.115 0.244]	0.167 [0.106 0.215]	497.0	392.0	74.0
driveway	0.059 [0.036 0.087]	0.177 [0.104 0.232]	0.088 [0.053 0.127]	145.3	48.3	8.7
garagedoor	0.103 [0.083 0.130]	0.396 [0.324 0.439]	0.163 [0.133 0.201]	142.7	37.0	14.7
path	0.007 [0.000 0.010]	0.010 [0.000 0.016]	0.008 [0.000 0.012]	199.7	102.3	1.3
roof	0.172 [0.161 0.185]	0.383 [0.352 0.414]	0.237 [0.221 0.256]	891.3	399.7	153.3
shutter	0.000 [0.000 0.000]	0.000 [0.000 0.000]	0.000 [0.000 0.000]	819.7	2.0	0.0
steps	0.042 [0.025 0.054]	0.118 [0.044 0.200]	0.054 [0.045 0.072]	272.0	149.0	11.3
window	0.081 [0.071 0.088]	0.215 [0.185 0.237]	0.117 [0.103 0.128]	3136.3	1177.0	253.0
cumulative	0.082 [0.073 0.089]	0.230 [0.185 0.256]	0.120 [0.105 0.132]	6558.0	2347.3	535.3

Table B.52: Learned 2-level WGGs, house dataset, pairwise, classobj, fixed.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.191 [0.120 0.246]	0.555 [0.394 0.750]	0.274 [0.195 0.324]	110.0	42.0	21.3
chimney	0.003 [0.000 0.009]	0.792 [0.375 1.000]	0.006 [0.000 0.018]	344.0	2.7	1.0
door	0.166 [0.136 0.184]	0.202 [0.151 0.251]	0.182 [0.143 0.209]	497.0	416.3	82.7
driveway	0.113 [0.074 0.137]	0.204 [0.179 0.220]	0.143 [0.111 0.167]	145.3	81.7	16.3
garagedoor	0.194 [0.181 0.219]	0.302 [0.276 0.329]	0.234 [0.226 0.244]	142.7	92.7	27.7
path	0.018 [0.010 0.025]	0.018 [0.012 0.027]	0.016 [0.015 0.016]	199.7	260.7	3.7
roof	0.187 [0.185 0.189]	0.418 [0.415 0.425]	0.259 [0.256 0.261]	891.3	400.0	167.3
shutter	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	819.7	2.0	0.0
steps	0.086 [0.062 0.100]	0.072 [0.064 0.081]	0.077 [0.070 0.084]	272.0	328.7	23.3
window	0.001 [0.000 0.003]	0.763 [0.289 1.000]	0.002 [0.000 0.007]	3136.3	12.7	3.7
cumulative	0.053 [0.051 0.054]	0.215 [0.177 0.237]	0.085 [0.082 0.088]	6558.0	1639.3	347.0

Table B.53: Learned 2-level WGGs, house dataset, pairwise, classobj, fixed.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.078 [0.000 0.149]	0.427 [0.000 0.714]	0.130 [0.000 0.236]	110.0	15.0	9.0
chimney	0.003 [0.000 0.009]	0.059 [0.000 0.176]	0.006 [0.000 0.018]	344.0	7.3	1.0
door	0.153 [0.111 0.205]	0.203 [0.125 0.279]	0.174 [0.117 0.236]	497.0	383.0	76.0
driveway	0.055 [0.036 0.081]	0.253 [0.238 0.280]	0.088 [0.062 0.121]	145.3	32.0	8.0
garagedoor	0.077 [0.062 0.097]	0.357 [0.341 0.385]	0.126 [0.105 0.151]	142.7	31.0	11.0
path	0.003 [0.000 0.010]	0.009 [0.000 0.027]	0.005 [0.000 0.015]	199.7	39.0	0.7
roof	0.169 [0.163 0.179]	0.377 [0.357 0.400]	0.233 [0.224 0.247]	891.3	399.0	150.7
shutter	0.000 [0.000 0.000]	0.000 [0.000 0.000]	0.000 [0.000 0.000]	819.7	4.0	0.0
steps	0.036 [0.007 0.054]	0.104 [0.053 0.182]	0.042 [0.014 0.058]	272.0	149.3	9.7
window	0.083 [0.074 0.089]	0.215 [0.186 0.232]	0.119 [0.106 0.127]	3136.3	1209.7	260.0
cumulative	0.080 [0.074 0.086]	0.234 [0.191 0.269]	0.119 [0.107 0.130]	6558.0	2269.3	526.0

Table B.54: Learned 2-level WGGs, house dataset, pairwise, classobj, free.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.156 [0.070 0.224]	0.493 [0.318 0.591]	0.236 [0.115 0.325]	110.0	33.7	17.7
chimney	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	344.0	0.0	0.0
door	0.173 [0.159 0.193]	0.195 [0.115 0.262]	0.179 [0.136 0.222]	497.0	485.3	85.7
driveway	0.102 [0.060 0.137]	0.298 [0.260 0.333]	0.147 [0.102 0.179]	145.3	51.0	14.7
garagedoor	0.157 [0.111 0.185]	0.366 [0.247 0.485]	0.210 [0.181 0.245]	142.7	68.0	22.3
path	0.015 [0.005 0.030]	0.023 [0.010 0.044]	0.014 [0.007 0.020]	199.7	186.0	3.0
roof	0.178 [0.172 0.182]	0.396 [0.388 0.400]	0.245 [0.238 0.251]	891.3	400.0	158.3
shutter	0.003 [0.000 0.007]	0.345 [0.015 1.000]	0.004 [0.000 0.009]	819.7	127.3	2.0
steps	0.072 [0.029 0.104]	0.072 [0.057 0.084]	0.066 [0.043 0.087]	272.0	288.3	19.7
window	0.002 [0.000 0.006]	0.753 [0.260 1.000]	0.004 [0.000 0.012]	3136.3	25.7	6.7
cumulative	0.050 [0.049 0.052]	0.203 [0.161 0.231]	0.080 [0.078 0.082]	6558.0	1665.3	330.0

Table B.55: Learned 2-level WGGs, house dataset, pairwise, classobj, free.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.127 [0.060 0.175]	0.656 [0.588 0.750]	0.206 [0.111 0.270]	110.0	23.0	14.3
chimney	0.002 [0.000 0.006]	0.889 [0.667 1.000]	0.004 [0.000 0.012]	344.0	1.0	0.7
door	0.070 [0.027 0.122]	0.384 [0.351 0.432]	0.115 [0.050 0.190]	497.0	86.7	34.7
driveway	0.076 [0.065 0.081]	0.194 [0.176 0.226]	0.109 [0.095 0.119]	145.3	57.0	11.0
garagedoor	0.128 [0.116 0.151]	0.492 [0.425 0.579]	0.203 [0.185 0.239]	142.7	37.3	18.3
path	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	199.7	0.0	0.0
roof	0.184 [0.176 0.193]	0.416 [0.405 0.437]	0.256 [0.246 0.268]	891.3	395.7	164.3
shutter	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	819.7	0.3	0.0
steps	0.015 [0.011 0.019]	0.379 [0.263 0.444]	0.028 [0.021 0.036]	272.0	11.7	4.0
window	0.012 [0.000 0.030]	0.595 [0.284 1.000]	0.021 [0.000 0.054]	3136.3	118.0	35.7
cumulative	0.043 [0.038 0.051]	0.392 [0.355 0.421]	0.078 [0.069 0.089]	6558.0	730.7	283.0

Table B.56: Learned 2-level WGGs, house dataset, nopairwise, classobj, fixed.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.133 [0.070 0.190]	0.666 [0.579 0.778]	0.215 [0.128 0.286]	110.0	24.0	15.0
chimney	0.003 [0.000 0.009]	0.917 [0.750 1.000]	0.006 [0.000 0.019]	344.0	1.3	1.0
door	0.054 [0.029 0.083]	0.416 [0.377 0.494]	0.095 [0.053 0.142]	497.0	63.0	27.0
driveway	0.078 [0.068 0.086]	0.241 [0.235 0.250]	0.118 [0.105 0.126]	145.3	47.0	11.3
garagedoor	0.119 [0.111 0.130]	0.495 [0.457 0.528]	0.192 [0.179 0.209]	142.7	34.3	17.0
path	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	199.7	0.0	0.0
roof	0.187 [0.181 0.196]	0.419 [0.410 0.430]	0.258 [0.251 0.269]	891.3	398.3	166.7
shutter	0.000 [0.000 0.000]	1.000 [1.000 1.000]	0.000 [0.000 0.000]	819.7	0.0	0.0
steps	0.008 [0.004 0.014]	0.323 [0.125 0.444]	0.016 [0.007 0.028]	272.0	7.3	2.3
window	0.002 [0.000 0.006]	0.571 [0.250 1.000]	0.004 [0.000 0.011]	3136.3	18.3	7.3
cumulative	0.038 [0.037 0.039]	0.418 [0.410 0.426]	0.069 [0.067 0.071]	6558.0	593.7	247.7

Table B.57: Learned 2-level WGGs, house dataset, nopairwise, classobj, fixed.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.119 [0.060 0.149]	0.583 [0.462 0.739]	0.195 [0.106 0.245]	110.0	22.3	13.3
chimney	0.010 [0.000 0.022]	0.283 [0.000 0.750]	0.017 [0.000 0.036]	344.0	29.7	3.3
door	0.121 [0.080 0.201]	0.229 [0.170 0.274]	0.155 [0.109 0.232]	497.0	254.0	60.0
driveway	0.048 [0.036 0.054]	0.273 [0.211 0.381]	0.081 [0.062 0.094]	145.3	27.0	7.0
garagedoor	0.089 [0.082 0.097]	0.437 [0.293 0.538]	0.147 [0.128 0.165]	142.7	30.7	12.7
path	0.005 [0.000 0.015]	0.007 [0.000 0.021]	0.006 [0.000 0.018]	199.7	55.3	1.0
roof	0.177 [0.164 0.187]	0.395 [0.360 0.417]	0.244 [0.226 0.258]	891.3	399.0	157.7
shutter	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	819.7	0.7	0.0
steps	0.019 [0.000 0.054]	0.038 [0.000 0.093]	0.025 [0.000 0.068]	272.0	67.3	5.0
window	0.080 [0.069 0.086]	0.211 [0.197 0.233]	0.116 [0.102 0.125]	3136.3	1191.3	251.7
cumulative	0.078 [0.071 0.087]	0.248 [0.209 0.273]	0.118 [0.106 0.130]	6558.0	2077.3	511.7

Table B.58: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.1$, nosingletocomp.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.064 [0.018 0.095]	0.906 [0.800 1.000]	0.117 [0.034 0.172]	110.0	8.0	7.0
chimney	0.043 [0.005 0.066]	0.060 [0.049 0.072]	0.046 [0.010 0.069]	344.0	224.0	14.3
door	0.093 [0.033 0.142]	0.367 [0.116 0.525]	0.122 [0.062 0.176]	497.0	243.7	46.0
driveway	0.014 [0.000 0.022]	0.310 [0.000 0.500]	0.026 [0.000 0.041]	145.3	5.0	2.0
garagedoor	0.030 [0.014 0.041]	0.561 [0.417 0.667]	0.056 [0.028 0.077]	142.7	8.3	4.3
path	0.003 [0.000 0.010]	0.010 [0.000 0.029]	0.005 [0.000 0.015]	199.7	28.7	0.7
roof	0.127 [0.096 0.182]	0.324 [0.262 0.413]	0.182 [0.145 0.253]	891.3	345.3	113.7
shutter	0.001 [0.000 0.003]	0.002 [0.000 0.005]	0.001 [0.000 0.004]	819.7	228.0	1.0
steps	0.055 [0.000 0.102]	0.044 [0.000 0.087]	0.049 [0.000 0.094]	272.0	242.3	15.3
window	0.009 [0.005 0.015]	0.270 [0.234 0.325]	0.017 [0.009 0.028]	3136.3	109.7	28.3
cumulative	0.035 [0.033 0.038]	0.172 [0.132 0.226]	0.058 [0.057 0.059]	6558.0	1443.0	232.7

Table B.59: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.1$, singletocomp1.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.050 [0.026 0.080]	0.738 [0.600 1.000]	0.092 [0.050 0.142]	110.0	7.7	5.3
chimney	0.049 [0.028 0.063]	0.095 [0.064 0.156]	0.063 [0.039 0.090]	344.0	199.7	17.3
door	0.042 [0.032 0.051]	0.430 [0.223 0.667]	0.074 [0.062 0.083]	497.0	63.7	21.0
driveway	0.057 [0.036 0.087]	0.237 [0.175 0.310]	0.091 [0.062 0.136]	145.3	34.7	8.3
garagedoor	0.037 [0.034 0.042]	0.363 [0.250 0.545]	0.067 [0.060 0.077]	142.7	16.0	5.3
path	0.020 [0.000 0.040]	0.010 [0.000 0.020]	0.014 [0.000 0.027]	199.7	266.3	4.0
roof	0.169 [0.163 0.174]	0.384 [0.370 0.399]	0.235 [0.226 0.240]	891.3	393.3	151.0
shutter	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	819.7	32.0	0.0
steps	0.061 [0.043 0.077]	0.063 [0.032 0.091]	0.061 [0.037 0.074]	272.0	289.3	16.3
window	0.008 [0.000 0.017]	0.144 [0.000 0.222]	0.015 [0.000 0.032]	3136.3	114.7	24.3
cumulative	0.039 [0.036 0.044]	0.198 [0.144 0.291]	0.063 [0.058 0.069]	6558.0	1417.3	253.0

Table B.60: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.1$, singletocomp2.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.115 [0.050 0.155]	0.510 [0.417 0.581]	0.185 [0.089 0.245]	110.0	24.3	13.0
chimney	0.025 [0.011 0.050]	0.196 [0.116 0.238]	0.039 [0.022 0.070]	344.0	58.7	8.3
door	0.141 [0.078 0.227]	0.214 [0.112 0.291]	0.169 [0.092 0.255]	497.0	326.7	70.3
driveway	0.062 [0.047 0.088]	0.321 [0.259 0.371]	0.103 [0.080 0.142]	145.3	27.7	9.0
garagedoor	0.076 [0.043 0.110]	0.305 [0.273 0.333]	0.120 [0.075 0.162]	142.7	35.7	11.0
path	0.000 [0.000 0.000]	0.000 [0.000 0.000]	0.000 [0.000 0.000]	199.7	40.3	0.0
roof	0.185 [0.182 0.190]	0.412 [0.400 0.422]	0.255 [0.251 0.262]	891.3	400.0	164.7
shutter	0.000 [0.000 0.000]	0.000 [0.000 0.000]	0.000 [0.000 0.000]	819.7	5.0	0.0
steps	0.022 [0.000 0.042]	0.044 [0.000 0.087]	0.030 [0.000 0.057]	272.0	98.0	6.0
window	0.083 [0.074 0.092]	0.217 [0.200 0.230]	0.121 [0.108 0.132]	3136.3	1201.7	261.7
cumulative	0.083 [0.077 0.088]	0.247 [0.210 0.274]	0.124 [0.113 0.133]	6558.0	2218.0	544.0

Table B.61: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.15$, nosingletocomp.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.044 [0.020 0.086]	0.590 [0.500 0.769]	0.081 [0.038 0.155]	110.0	7.7	5.0
chimney	0.002 [0.000 0.003]	0.045 [0.000 0.077]	0.004 [0.000 0.006]	344.0	10.3	0.7
door	0.114 [0.045 0.176]	0.320 [0.141 0.629]	0.132 [0.083 0.183]	497.0	308.7	57.0
driveway	0.016 [0.000 0.027]	0.194 [0.000 0.333]	0.029 [0.000 0.050]	145.3	13.0	2.3
garagedoor	0.051 [0.029 0.069]	0.475 [0.400 0.526]	0.091 [0.055 0.123]	142.7	15.7	7.3
path	0.012 [0.005 0.020]	0.015 [0.013 0.018]	0.012 [0.007 0.016]	199.7	162.7	2.3
roof	0.178 [0.171 0.191]	0.390 [0.376 0.405]	0.244 [0.236 0.260]	891.3	406.3	158.7
shutter	0.002 [0.000 0.004]	0.004 [0.000 0.009]	0.002 [0.000 0.005]	819.7	203.3	1.3
steps	0.025 [0.000 0.051]	0.072 [0.000 0.151]	0.037 [0.000 0.076]	272.0	69.7	7.0
window	0.035 [0.030 0.041]	0.248 [0.209 0.294]	0.061 [0.053 0.069]	3136.3	454.3	109.3
cumulative	0.054 [0.051 0.056]	0.219 [0.171 0.247]	0.086 [0.084 0.088]	6558.0	1651.7	351.0

Table B.62: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.15$, singletocomp1.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.077 [0.050 0.121]	0.732 [0.417 1.000]	0.138 [0.089 0.209]	110.0	12.3	8.7
chimney	0.053 [0.006 0.138]	0.270 [0.072 0.500]	0.044 [0.011 0.095]	344.0	212.0	17.0
door	0.154 [0.111 0.227]	0.222 [0.124 0.291]	0.180 [0.117 0.255]	497.0	358.7	76.7
driveway	0.039 [0.027 0.047]	0.313 [0.259 0.364]	0.069 [0.050 0.080]	145.3	19.0	5.7
garagedoor	0.046 [0.027 0.076]	0.529 [0.333 0.800]	0.081 [0.053 0.124]	142.7	16.3	6.7
path	0.002 [0.000 0.005]	0.001 [0.000 0.004]	0.001 [0.000 0.004]	199.7	105.7	0.3
roof	0.172 [0.147 0.190]	0.404 [0.390 0.422]	0.240 [0.215 0.262]	891.3	378.7	153.0
shutter	0.003 [0.000 0.006]	0.007 [0.000 0.014]	0.005 [0.000 0.009]	819.7	293.7	3.0
steps	0.023 [0.000 0.068]	0.026 [0.000 0.078]	0.024 [0.000 0.073]	272.0	94.7	6.3
window	0.034 [0.000 0.084]	0.246 [0.222 0.265]	0.052 [0.001 0.122]	3136.3	472.0	107.7
cumulative	0.059 [0.037 0.088]	0.195 [0.143 0.274]	0.090 [0.061 0.133]	6558.0	1963.0	385.0

Table B.63: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.15$, singletocomp2.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.137 [0.080 0.167]	0.519 [0.452 0.633]	0.214 [0.137 0.260]	110.0	29.7	15.3
chimney	0.012 [0.003 0.022]	0.231 [0.073 0.500]	0.020 [0.005 0.034]	344.0	43.7	4.0
door	0.128 [0.088 0.193]	0.187 [0.125 0.244]	0.151 [0.103 0.215]	497.0	336.0	63.7
driveway	0.070 [0.036 0.088]	0.287 [0.232 0.317]	0.110 [0.065 0.138]	145.3	37.7	10.3
garagedoor	0.093 [0.083 0.110]	0.402 [0.324 0.500]	0.150 [0.133 0.170]	142.7	34.3	13.3
path	0.005 [0.000 0.010]	0.006 [0.000 0.013]	0.005 [0.000 0.011]	199.7	137.0	1.0
roof	0.171 [0.160 0.185]	0.382 [0.350 0.414]	0.236 [0.219 0.256]	891.3	400.0	152.7
shutter	0.000 [0.000 0.000]	0.667 [0.000 1.000]	0.000 [0.000 0.000]	819.7	1.3	0.0
steps	0.025 [0.007 0.042]	0.097 [0.034 0.200]	0.035 [0.012 0.049]	272.0	95.3	6.7
window	0.085 [0.078 0.088]	0.220 [0.197 0.237]	0.122 [0.112 0.128]	3136.3	1202.7	265.0
cumulative	0.081 [0.076 0.089]	0.232 [0.190 0.256]	0.120 [0.109 0.132]	6558.0	2317.7	532.0

Table B.64: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.2$, nosingletocomp.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.093 [0.080 0.105]	0.577 [0.471 0.688]	0.161 [0.137 0.178]	110.0	18.0	10.3
chimney	0.009 [0.000 0.025]	0.202 [0.000 0.500]	0.015 [0.000 0.041]	344.0	26.0	3.0
door	0.104 [0.002 0.193]	0.200 [0.107 0.250]	0.110 [0.004 0.215]	497.0	308.0	51.0
driveway	0.048 [0.022 0.087]	0.347 [0.208 0.600]	0.076 [0.042 0.127]	145.3	28.3	7.0
garagedoor	0.062 [0.029 0.083]	0.461 [0.324 0.667]	0.105 [0.056 0.133]	142.7	23.7	9.0
path	0.010 [0.005 0.015]	0.024 [0.013 0.045]	0.011 [0.009 0.014]	199.7	132.3	2.0
roof	0.184 [0.182 0.185]	0.410 [0.404 0.414]	0.254 [0.252 0.256]	891.3	400.0	164.0
shutter	0.000 [0.000 0.000]	0.333 [0.000 1.000]	0.000 [0.000 0.000]	819.7	2.0	0.0
steps	0.031 [0.000 0.068]	0.102 [0.000 0.200]	0.043 [0.000 0.083]	272.0	72.7	8.7
window	0.078 [0.071 0.088]	0.214 [0.191 0.237]	0.115 [0.107 0.128]	3136.3	1150.3	245.7
cumulative	0.076 [0.064 0.089]	0.235 [0.194 0.256]	0.115 [0.102 0.132]	6558.0	2161.3	500.7

Table B.65: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.2$, singletocomp1.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.093 [0.080 0.105]	0.577 [0.471 0.688]	0.161 [0.137 0.178]	110.0	18.0	10.3
chimney	0.009 [0.000 0.025]	0.202 [0.000 0.500]	0.015 [0.000 0.041]	344.0	26.0	3.0
door	0.104 [0.002 0.193]	0.200 [0.107 0.250]	0.110 [0.004 0.215]	497.0	308.0	51.0
driveway	0.048 [0.022 0.087]	0.347 [0.208 0.600]	0.076 [0.042 0.127]	145.3	28.3	7.0
garagedoor	0.062 [0.029 0.083]	0.461 [0.324 0.667]	0.105 [0.056 0.133]	142.7	23.7	9.0
path	0.010 [0.005 0.015]	0.024 [0.013 0.045]	0.011 [0.009 0.014]	199.7	132.3	2.0
roof	0.184 [0.182 0.185]	0.410 [0.404 0.414]	0.254 [0.252 0.256]	891.3	400.0	164.0
shutter	0.000 [0.000 0.000]	0.333 [0.000 1.000]	0.000 [0.000 0.000]	819.7	2.0	0.0
steps	0.031 [0.000 0.068]	0.102 [0.000 0.200]	0.043 [0.000 0.083]	272.0	72.7	8.7
window	0.078 [0.071 0.088]	0.214 [0.191 0.237]	0.115 [0.107 0.128]	3136.3	1150.3	245.7
cumulative	0.076 [0.064 0.089]	0.235 [0.194 0.256]	0.115 [0.102 0.132]	6558.0	2161.3	500.7

Table B.66: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.2$, singletocomp2.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.157 [0.080 0.219]	0.614 [0.471 0.714]	0.248 [0.137 0.329]	110.0	27.7	17.7
chimney	0.004 [0.003 0.006]	0.238 [0.071 0.500]	0.008 [0.005 0.012]	344.0	12.3	1.3
door	0.149 [0.099 0.193]	0.191 [0.115 0.244]	0.167 [0.106 0.215]	497.0	392.0	74.0
driveway	0.059 [0.036 0.087]	0.177 [0.104 0.232]	0.088 [0.053 0.127]	145.3	48.3	8.7
garagedoor	0.103 [0.083 0.130]	0.396 [0.324 0.439]	0.163 [0.133 0.201]	142.7	37.0	14.7
path	0.007 [0.000 0.010]	0.010 [0.000 0.016]	0.008 [0.000 0.012]	199.7	102.3	1.3
roof	0.172 [0.161 0.185]	0.383 [0.352 0.414]	0.237 [0.221 0.256]	891.3	399.7	153.3
shutter	0.000 [0.000 0.000]	0.000 [0.000 0.000]	0.000 [0.000 0.000]	819.7	2.0	0.0
steps	0.042 [0.025 0.054]	0.118 [0.044 0.200]	0.054 [0.045 0.072]	272.0	149.0	11.3
window	0.081 [0.071 0.088]	0.215 [0.185 0.237]	0.117 [0.103 0.128]	3136.3	1177.0	253.0
cumulative	0.082 [0.073 0.089]	0.230 [0.185 0.256]	0.120 [0.105 0.132]	6558.0	2347.3	535.3

Table B.67: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.3$, nosingletocomp.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.157 [0.080 0.219]	0.614 [0.471 0.714]	0.248 [0.137 0.329]	110.0	27.7	17.7
chimney	0.004 [0.003 0.006]	0.238 [0.071 0.500]	0.008 [0.005 0.012]	344.0	12.3	1.3
door	0.149 [0.099 0.193]	0.191 [0.115 0.244]	0.167 [0.106 0.215]	497.0	392.0	74.0
driveway	0.059 [0.036 0.087]	0.177 [0.104 0.232]	0.088 [0.053 0.127]	145.3	48.3	8.7
garagedoor	0.103 [0.083 0.130]	0.396 [0.324 0.439]	0.163 [0.133 0.201]	142.7	37.0	14.7
path	0.007 [0.000 0.010]	0.010 [0.000 0.016]	0.008 [0.000 0.012]	199.7	102.3	1.3
roof	0.172 [0.161 0.185]	0.383 [0.352 0.414]	0.237 [0.221 0.256]	891.3	399.7	153.3
shutter	0.000 [0.000 0.000]	0.000 [0.000 0.000]	0.000 [0.000 0.000]	819.7	2.0	0.0
steps	0.042 [0.025 0.054]	0.118 [0.044 0.200]	0.054 [0.045 0.072]	272.0	149.0	11.3
window	0.081 [0.071 0.088]	0.215 [0.185 0.237]	0.117 [0.103 0.128]	3136.3	1177.0	253.0
cumulative	0.082 [0.073 0.089]	0.230 [0.185 0.256]	0.120 [0.105 0.132]	6558.0	2347.3	535.3

Table B.68: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.3$, singletocomp1.

	recall	precision	f-measure	# targets	# detections	# correct
car	0.157 [0.080 0.219]	0.614 [0.471 0.714]	0.248 [0.137 0.329]	110.0	27.7	17.7
chimney	0.004 [0.003 0.006]	0.238 [0.071 0.500]	0.008 [0.005 0.012]	344.0	12.3	1.3
door	0.149 [0.099 0.193]	0.191 [0.115 0.244]	0.167 [0.106 0.215]	497.0	392.0	74.0
driveway	0.059 [0.036 0.087]	0.177 [0.104 0.232]	0.088 [0.053 0.127]	145.3	48.3	8.7
garagedoor	0.103 [0.083 0.130]	0.396 [0.324 0.439]	0.163 [0.133 0.201]	142.7	37.0	14.7
path	0.007 [0.000 0.010]	0.010 [0.000 0.016]	0.008 [0.000 0.012]	199.7	102.3	1.3
roof	0.172 [0.161 0.185]	0.383 [0.352 0.414]	0.237 [0.221 0.256]	891.3	399.7	153.3
shutter	0.000 [0.000 0.000]	0.000 [0.000 0.000]	0.000 [0.000 0.000]	819.7	2.0	0.0
steps	0.042 [0.025 0.054]	0.118 [0.044 0.200]	0.054 [0.045 0.072]	272.0	149.0	11.3
window	0.081 [0.071 0.088]	0.215 [0.185 0.237]	0.117 [0.103 0.128]	3136.3	1177.0	253.0
cumulative	0.082 [0.073 0.089]	0.230 [0.185 0.256]	0.120 [0.105 0.132]	6558.0	2347.3	535.3

Table B.69: Learned 3-level WGGs, big placesetting dataset, $\gamma_{\text{minsize}} = 0.3$, singletocomp2.

Bibliography

Allen, J. (1995). *Natural Language Understanding*. The Benjamin/Cummings Publishing Company, Inc.

Aycinena, M. A. (2005). Probabilistic geometric grammars for object recognition. Master's thesis, Massachusetts Institute of Technology.

Biederman, I. (1985). Human image understanding: Recent research and a theory. *Computer Vision, Graphics, and Image Processing*, *32*, 29–73.

Biederman, I. (1987). Recognition-by-components: A theory of human image understanding. *Psychological Review*, *94*(2).

Biederman, I., Mezzanotte, R. J., & Rabinowitz, J. C. (1982). Scene perception: detecting and judging objects undergoing relational violations. *Cognitive Psychology*, *14*, 143–177.

Binford, T. (1971). Visual perception by computer. In *Proc. IEEE Conf. on Systems and Control*.

Bobick, A. F., & Pinhanez, C. S. (1995). Using approximate models as source of contextual information for vision processing. In *Proc. IEEE of the ICCV Workshop on Context-Based Vision*.

Carbonetto, P., de Freitas, N., & Barnard, K. (2004). A statistical model for general contextual object recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Chen, S. F. (1995). Bayesian grammar induction for language modeling. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Choi, M. J., Lim, J. J., Torralba, A., & Willsky, A. S. (2010). Exploiting hierarchical context on a large database of object categories. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Collins, M. (2004). Parameter estimation for statistical parsing models: Theory and practice of distribution-free methods. In Bunt, H., Carroll, J., & Satta, G. (Eds.), *New Developments in Parsing Technology*. Kluwer.

Crandall, D., Felzenszwalb, P., & Huttenlocher, D. (2005). Spatial priors for part-based recognition using statistical models. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Crandall, D. J., & Huttenlocher, D. P. (2006). Weakly supervised learning of part-based spatial models for visual object recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Crandall, D. J., & Huttenlocher, D. P. (2007). Composite models of objects and scenes for category recognition. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Csurka, G., Dance, C. R., Fan, L., Willamowski, J., & Bray, C. (2004). Visual categorization with bags of keypoints. In *Proc. Workshop on Statistical Learning in Computer Vision, (ECCV)*.

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Davis, J., & Goadrich, M. (2006). The relationship between precision-recall and ROC curves. In *Proceedings of the International Conference on Machine Learning (ICML)*.

de Marcken, C. G. (1996). *Unsupervised Language Acquisition*. Ph.D. thesis, Massachusetts Institute of Technology.

Desai, C., Ramanan, D., & Fowlkes, C. (2009). Discriminative models for multi-class object layout. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Draper, B. A., Collins, R. T., Brolio, J., Hanson, A. R., & Riseman, E. M. (1989). The schema system. *International Journal of Computer Vision (IJCV)*, *2*(3), 209–205.

Epshtein, B., & Ullman, S. (2005a). Feature hierarchies for object classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Epshtein, B., & Ullman, S. (2005b). Identifying semantically equivalent object fragments. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2008). The PASCAL Visual Object Classes Challenge 2008 (VOC2008) Results. http://www.pascal-network.org/challenges/VOC/voc2008/workshop/index.html.

Fei-Fei, L., & Perona, P. (2005). A bayesian hierarchical model for learning natural scene categories. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Felzenszwalb, P., Girshick, R., McAllester, D., & Ramanan, D. (2010). Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), to appear*.

Felzenszwalb, P., McAllester, D., & Ramanan, D. (2008). A discriminatively trained, multiscale, deformable part model. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Felzenszwalb, P., & Schwartz, J. (2007). Hierarchical matching of deformable shapes. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Felzenszwalb, P. F., & Huttenlocher, D. P. (2000). Efficient matching of pictorial structures. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Felzenszwalb, P. F., & Huttenlocher, D. P. (2004). Distance transforms of sampled functions. Tech. rep., Cornell Computing and Information Science.

Felzenszwalb, P. F., & Huttenlocher, D. P. (2005). Pictorial structures for object recognition. *International Journal of Computer Vision (IJCV)*, *61*(1), 55–79.

Fergus, R., Perona, P., & Zisserman, A. (2003). Object class recognition by unsupervised scale-invariant learning. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Fergus, R., Perona, P., & Zisserman, A. (2005). A sparse object category model for efficient learning and exhaustive recognition. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Fidler, S., & Leonardis, A. (2007). Towards scalable representations of object categories: Learning a hierarchy of parts. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Fink, M., & Perona, P. (2003). Mutual boosting for contextual inference. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*.

Fischler, M., & Elschlager, R. (1973). The representation and matching of pictorial structures. *IEEE Transactions on Computers*, *C-22*, 67–92.

Freund, Y., & Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, *37*(3), 277–296.

Friedman, A. (1979). Framing pictures: the role of knowledge in automatized encoding and memory of gist. *Journal of Experimental Psychology: General*, *108*, 316–355.

Galleguillos, C., Rabinovich, A., & Belongie, S. (2008). Object categorization using Co-Occurrence, location, and appearance. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Gould, S., Rodgers, J., Cohen, D., Elidan, G., & Koller, D. (2008). Multi-class segmentation with relative location prior. *International Journal of Computer Vision (IJCV)*, *80*(3), 300–316.

Grauman, K., & Darrell, T. (2005). The pyramid match kernel: Discriminative classification with sets of image features. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Grauman, K., & Darrell, T. (2006). Unsupervised learning of categories from sets of partially matching image features. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Grünwald, P. (2005). A tutorial introduction to the minimum description length principle. In Grünwald, P., Myung, I. J., & Pitt, M. A. (Eds.), *Advances in Minimum Description Length: Theory and Applications*. MIT Press.

Haghighi, A., & Klein, D. (2006). Prototype-driven grammar induction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Han, F., & Zhu, S.-C. (2005). Bottom-up/top-down image parsing by attribute graph grammar. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Hanson, A. R., & Riseman, E. M. (1978). Visions: A computer system for interpreting scenes. In Hanson, A. R., & Riseman, E. M. (Eds.), *Computer Vision Systems*, pp. 303–333. Academic Press: New York.

He, Z., Zemel, R. S., & Carreira-Perpiñán, M. A. (2004). Multiscale conditional random fields for image labeling. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Heckerman, D. (1999). A tutorial on learning with bayesian networks. In Jordan, M. (Ed.), *Learning in Graphical Models*, pp. 301–354. MIT Press.

Hess, R., Fern, A., & Mortensen, E. (2007). Mixture-of-parts pictorial structures for objects with variable part sets. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Jurafsky, D., & Martin, J. H. (2000). *Speech and Language Processing*. Prentice Hall.

Kivinen, J. J., Sudderth, E. B., & Jordan, M. I. (2007). Learning multiscale representations of natural scenes using dirichlet processes. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Klein, D., & Manning, C. D. (2001). Natural language grammar induction using a constituent-context model. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*.

Klein, D., & Manning, C. D. (2002). A generative constituent-context model for improved grammar induction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Klein, D., & Manning, C. D. (2004). Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Lazebnik, S., Schmid, C., & Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Lippow, M. A., Kaelbling, L. P., & Lozano-Pérez, T. (2008). Learning grammatical models for object recognition. In Cohn, A. G., Hogg, D. C., Möller, R., & Neumann, B. (Eds.), *Logic and Probability for Scene Interpretation*, No. 08091 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

Manning, C. D., & Schütze, H. (2002). *Foundations of Statistical Natural Language Processing*. The MIT Press.

Minsky, M. (1974). A framework for representing knowledge. Tech. rep., MIT AI Laboratory Memo 306.

Mohri, M., & Roark, B. (2006). Probabilistic context-free grammar induction based on structural zeros. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Moore, D. J., Essa, I. A., & Hayes, M. H. (1999). Exploiting human actions and object context for recognition tasks. In *Proc. IEEE International Conference on Image Processing*.

Murphy, K., Torralba, A., & Freeman, W. T. (2003). Using the forest to see the trees: A graphical model relating features, objects, and scenes. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*.

Navon, D. (1977). Forest before trees: The precedence of global features in visual perception. *Cognitive Psychology*, *9*, 353–383.

Nevill-Manning, C. G., & Witten, I. H. (1997). Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research (JAIR)*, *7*, 67–82.

Oliva, A., & Schyns, P. G. (2000). Diagnostic colors mediate scene recognition. *Cognitive Psychology*, *41*, 176–210.

Oliva, A., & Torralba, A. (2001). Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision (IJCV)*, *42*(3), 145–175.

Oliva, A., & Torralba, A. (2006). Building the gist of a scene: the role of global image features in recognition. *Progress in Brain Research, 155.*

Ommer, B., & Buhmann, J. M. (2005). Object categorization by compositional graphical models. In *Proceedings of the International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR).*

Ommer, B., & Buhmann, J. M. (2006). Learning compositional categorization models. In *Proceedings of the European Conference on Computer Vision (ECCV).*

Ommer, B., & Buhmann, J. M. (2007). Learning the compositional nature of visual objects. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR).*

Palmer, S. E. (1975). The effects of contextual scenes on the identification of objects. *Memory and Cognition, 3*(5), 519–526.

Papageorgiou, C., & Poggio, T. (2000). A trainable system for object detection. *International Journal of Computer Vision (IJCV), 38*(1), 15–33.

Pollak, I., Siskind, J. M., Harper, M. P., & Bouman, C. A. (2003). Parameter estimation for spatial random trees using the EM algorithm. In *Proc. IEEE International Conf. on Image Processing (ICIP).*

Potter, M. C. (1975). Meaning in visual search. *Science, 187*, 965–966.

Quattoni, A., Collins, M., & Darrell, T. (2004). Conditional random fields for object recognition. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS).*

Quattoni, A., Wang, S., Morency, L., Collins, M., & Darrell, T. (2007). Hidden conditional random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 29*(10).

Ramanan, D., & Sminchisescu, C. (2006). Training deformable models for localization. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR).*

Ranzato, M. A., Huang, F., Boureau, Y., & LeCun, Y. (2007). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR).*

Rasiwasia, N., & Vasconcelos, N. (2008). Scene classification with low-dimensional semantic spaces and weak supervision. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR).*

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, 65*(6), 386–408.

Rosenblatt, F. (1962). *Principles of Neurodynamics.* Spartan.

Rosenfeld, A. (1973). Progress in picture processing: 1969-71. *ACM Computing Surveys, 5*(2).

Russell, B. C., Torralba, A., Liu, C., Fergus, R., & Freeman, W. T. (2007). Object recognition by scene alignment. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS).*

Russell, B. C., Torralba, A., Murphy, K. P., & Freeman, W. T. (2008). LabelMe: a database and web-based tool for image annotation. *International Journal of Computer Vision (IJCV), 77*(1-3), 157–173.

Siskind, J., Sherman, Jr., J., Pollak, I., Harper, M., & Bouman, C. (2007). Spatial random trees grammars for modeling hierarchical structure in images with regions of arbitrary shape. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, *29*(9), 1504–1519.

Sivic, J., Russell, B. C., Efros, A. A., Zisserman, A., & Freeman, W. T. (2005). Discovering objects and their location in images. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Strat, T. M., & Fischler, M. A. (1991). Context-based vision: Recognizing objects using information from both 2D and 3D imagery. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, *13*(10), 1050–1065.

Sudderth, E., Torralba, A., Freeman, W. T., & Willsky, A. (2008). Describing visual scenes using transformed objects and parts. *International Journal of Computer Vision (IJCV)*, *77*(1-3), 291–330.

Sudderth, E. B., Torralba, A., Freeman, W. T., & Willsky, A. S. (2005a). Describing visual scenes using transformed dirichlet processes. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*.

Sudderth, E. B., Torralba, A., Freeman, W. T., & Willsky, A. S. (2005b). Learning hierarchical models of scenes, objects, and parts. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Sudderth, E. B., Torralba, A., Freeman, W. T., & Willsky, A. S. (2006). Depth from familiar objects: A hierarchical model for 3d scenes. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Torralba, A. (2003). Contextual priming for object detection. *International Journal of Computer Vision (IJCV)*, *53*(2), 169–191.

Torralba, A., Murphy, K. P., & Freeman, W. T. (2004). Sharing features: efficient boosting procedures for multiclass object detection. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Torralba, A., Murphy, K. P., & Freeman, W. T. (2005). Contextual models for object detection using boosted random fields. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*.

Torralba, A., Murphy, K. P., & Freeman, W. T. (2007). Sharing visual features for multiclass and multiview object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, *29*(5), 854–869.

Torralba, A., Murphy, K. P., Freeman, W. T., & Rubin, M. A. (2003). Context-based vision system for place and object recognition. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Tu, Z., Chen, X., Yuille, A. L., & Zhu, S.-C. (2003). Image parsing: Unifying segmentation, detection, and recognition. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Tu, Z., Chen, X., Yuille, A. L., & Zhu, S.-C. (2005). Image parsing: Unifying segmentation, detection, and recognition. *International Journal of Computer Vision (IJCV)*, *63*(2).

Tu, Z., & Zhu, S.-C. (2006). Parsing images into regions, curves, and curve groups. *International Journal of Computer Vision (IJCV)*, *69*(2).

Turk, M., & Pentland, A. (1991). Face recognition using eigenfaces. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ullman, S., & Epshtein, B. (2006). Visual classification by a hierarchy of extended fragments. In *Towards Category-Level Object Recognition, Lecture Notes on Computer Science*. Springer-Verlag.

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Viola, P., & Jones, M. (2002). Fast and robust classification using asymmetric adaboost and a detector cascade. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*.

Wang, Y., & Mori, G. (2007). Boosted multiple deformable trees for parsing human poses. In *2nd Workshop on HUMAN MOTION Understanding, Modeling, Capture and Animation (at ICCV)*.

Weber, M., Welling, M., & Perona, P. (2000). Unsupervised learning of models for recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Zettlemoyer, L., & Collins, M. (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*.

Zettlemoyer, L., & Collins, M. (2007). Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Zhu, L., Chen, Y., Lin, Y., Lin, C., & Yuille, A. (2008a). Recursive segmentation and recognition templates for 2D parsing. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*.

Zhu, L., Chen, Y., Ye, X., & Yuille, A. (2008b). Structure-perceptron learning of a hierarchical log-linear model. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhu, L., & Yuille, A. (2005). A hierarchical compositional system for rapid object detection. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*.

Zhu, L. L., Chen, Y., & Yuille, A. (2006). Unsupervised learning of a probabilistic grammar for object detection and parsing. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*.

Zhu, S.-C., & Mumford, D. (2006). A stochastic grammar of images. *Foundations and Trends in Computer Graphics and Vision*, *2*(4), 259–362.