HADES: Hierarchical Approximate Decoding for Structured Prediction

Master Thesis
Tribhuvanesh Orekondy
September 30, 2015

Advisors: Dr. Martin Jaggi, Dr. Aurelien Lucchi
Supervisor: Prof. Dr. Thomas Hofmann
Department of Computer Science, ETH Zürich
Abstract

Many fundamental tasks in machine learning require predicting complex objects rather than a simple yes-no answer or a number. Structured Output Prediction deals with learning such complex objects, which model an inherent structure between interdependent variables. Recently, large margin methods like Structured SVMs (SSVM) have gained popularity to solve this task due to efficient and generalized optimization techniques. These optimization algorithms typically rely on solving an inference or decoding sub-problem every iteration, which is computationally expensive. Moreover, little is known on learning a structured model by solving this sub-problem approximately.

To address these issues, this thesis introduces a generalized technique to learn from a series of coarse-to-fine approximate candidates based on the recent Block-Coordinate Frank-Wolfe algorithm for Structured SVMs. The core observation we use is that one can learn to a reasonable degree, even from approximate solutions. This technique is presented in the context of a popular Computer Vision problem – Semantic Image Segmentation. We pose the decoding sub-problem to that of solving a series of increasingly complex surrogate Conditional Random Fields (CRF) in search for a candidate which meets the required approximation quality.

We evaluate our technique in the context of natural scene image segmentation on the MSRC-21 dataset. Our experiments indicate that even extremely approximate solutions, which are 50x faster to decode, contribute to learning using our strategy. We achieve the same accuracy as our baseline and in addition, we reach a reasonable accuracy 1.5x-4x as quickly.
I am profoundly grateful to Martin Jaggi, Aurelien Lucchi and Thomas Hofmann. They have introduced, kindled and supported my interest in the field, even before I began with this thesis. My learning experience was only further amplified by their boundless energy and tireless pursuit for high-quality work. I am indebted to them aiding me complete this labyrinth – by helping me identify and formulate the problem, helping me ask the right questions, connecting me to people who knew the answers, keeping me motivated and constantly providing constructive feedback.

I would also like to thank many wonderful people: Ruben Wolfe, Simone Forte, Bettina Messmer and Daniel Sim. They have kept me on my feet with extremely helpful insights and fruitful discussions.
# Contents

## Contents

1 **Introduction**  
   1.1 Learning Structured Outputs  
   1.1.1 Semantic Image Segmentation  
   1.2 Motivation  
   1.3 Contribution  
   1.4 Thesis Outline  

2 **Related Work**  

3 **Preliminaries**  
   3.1 Structured Prediction  
   3.1.1 Structured Support Vector Machines  
   3.1.2 Optimization  
   3.2 Semantic Image Segmentation  
   3.2.1 Data, Superpixels and Features  
   3.2.2 Learning to Semantically Segment  

4 **dissolve**  
   4.1 Motivation  
   4.2 Method Description  
   4.2.1 Distributing BCFW  
   4.3 Implementation Details  
   4.3.1 CoDBcrw on Spark  
   4.4 Library Usage Guide  

5 **Approximate Hierarchical Decoding**  
   5.1 Hierarchical Decomposition  
   5.2 HADES-BCFW  
   5.3 Interpretation  
   5.3.1 Motivation  

6 **Hierarchical Image Segmentation**  
   6.1 Model sketch  
   6.1.1 Recap of Image Segmentation  

---

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Learning Structured Outputs</td>
<td>1</td>
</tr>
<tr>
<td>Semantic Image Segmentation</td>
<td>2</td>
</tr>
<tr>
<td>Motivation</td>
<td>3</td>
</tr>
<tr>
<td>Contribution</td>
<td>4</td>
</tr>
<tr>
<td>Thesis Outline</td>
<td>5</td>
</tr>
<tr>
<td>Related Work</td>
<td>6</td>
</tr>
<tr>
<td>Preliminaries</td>
<td>9</td>
</tr>
<tr>
<td>Structured Prediction</td>
<td>9</td>
</tr>
<tr>
<td>Structured Support Vector Machines</td>
<td>10</td>
</tr>
<tr>
<td>Optimization</td>
<td>11</td>
</tr>
<tr>
<td>Semantic Image Segmentation</td>
<td>13</td>
</tr>
<tr>
<td>Data, Superpixels and Features</td>
<td>13</td>
</tr>
<tr>
<td>Learning to Semantically Segment</td>
<td>14</td>
</tr>
<tr>
<td>dissolve</td>
<td>19</td>
</tr>
<tr>
<td>Motivation</td>
<td>19</td>
</tr>
<tr>
<td>Method Description</td>
<td>20</td>
</tr>
<tr>
<td>Distributing BCFW</td>
<td>20</td>
</tr>
<tr>
<td>Implementation Details</td>
<td>22</td>
</tr>
<tr>
<td>CoDBcrw on Spark</td>
<td>23</td>
</tr>
<tr>
<td>Library Usage Guide</td>
<td>25</td>
</tr>
<tr>
<td>Approximate Hierarchical Decoding</td>
<td>27</td>
</tr>
<tr>
<td>Hierarchical Decomposition</td>
<td>28</td>
</tr>
<tr>
<td>HADES-BCFW</td>
<td>29</td>
</tr>
<tr>
<td>Interpretation</td>
<td>30</td>
</tr>
<tr>
<td>Motivation</td>
<td>30</td>
</tr>
<tr>
<td>Hierarchical Image Segmentation</td>
<td>33</td>
</tr>
<tr>
<td>Model sketch</td>
<td>33</td>
</tr>
<tr>
<td>Recap of Image Segmentation</td>
<td>34</td>
</tr>
</tbody>
</table>
6.1.2 Hierarchical Decomposition for Image Segmentation . . 36
6.2 Surrogate CRF . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
6.3 Theory . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
6.4 Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
   6.4.1 Unary-only model . . . . . . . . . . . . . . . . . . . . . . 45
   6.4.2 Potts model . . . . . . . . . . . . . . . . . . . . . . . . . . . 46

7 Experimental Results 48
   7.1 Implementation Details and Setup . . . . . . . . . . . . . . . . 48
      7.1.1 Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
      7.1.2 Inference . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
      7.1.3 Features . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
      7.1.4 Experimental Setup . . . . . . . . . . . . . . . . . . . . . 50
   7.2 Models . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
      7.2.1 CRF . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
      7.2.2 Hierarchies . . . . . . . . . . . . . . . . . . . . . . . . . . 51
      7.2.3 Level schedule . . . . . . . . . . . . . . . . . . . . . . . . 51
      7.2.4 Nomenclature . . . . . . . . . . . . . . . . . . . . . . . . . 52
   7.3 Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53
      7.3.1 The MSRC-21 Dataset . . . . . . . . . . . . . . . . . . . 53
      7.3.2 Evaluation Metrics . . . . . . . . . . . . . . . . . . . . . . 53
      7.3.3 Baselines . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
      7.3.4 Parameters . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
   7.4 Key Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
      7.4.1 Performance . . . . . . . . . . . . . . . . . . . . . . . . . . 55
   7.5 Additional results . . . . . . . . . . . . . . . . . . . . . . . . . . 63
      7.5.1 Influence of parameters . . . . . . . . . . . . . . . . . . . 63

8 Conclusion 67

A Notation 68

B Interfaces 70
   B.1 HADES-BCFW . . . . . . . . . . . . . . . . . . . . . . . . . . . . 70

C Reproducing results 71

D Additional Experiments 74

Bibliography 76
Chapter 1

Introduction

1.1 Learning Structured Outputs

Understanding the relation between input-output variables is one of the goals in supervised machine learning. Whether it is classifying an email as spam or forecasting power demands from historical trends – uncovering such relations solve numerous day-to-day problems efficiently. Although the previous examples can be simply answered with a yes-no or a number response, we deal with many real-world problems which require a more complex answer. For instance, consider Optical Character Recognition (OCR) which requires understanding the relation between an image and its textual representation.

The solution to predict complex answers lies in the area of Structured Output Prediction. Here, the input-output variables are modeled as structured objects. These variables are structured in a sense that they capture the underlying information in a structured manner, like in the form of a sequence, trees or graphs. This makes responses capable of producing answers as complex as their queries themselves.

The key idea behind structured models is that they capture dependencies of variables within a model. In the case of OCR, instead of predicting each character or word independently, models can additionally learn patterns exhibited between the characters and words to improve accuracy. Learning such interactions is pivotal to truly understand the relations, because as humans, we tend to use such contextual cues to fine-tune predictions.

Structured Prediction methods are currently used to solve problems in numerous areas. For example, in Natural Language Processing, one can formulate the task of part-of-speech tagging as a structured prediction task as shown in Figure 1.1a. This can be generally looked at as a problem of sequence-labeling in which each word in an input sequence can take one out of a finite set of labels.

Many Computer Vision applications – such as background-foreground estimation, semantic image segmentation and human-pose estimation – can also be naturally formulated as a structured prediction task. In these cases, the (input) image is modeled as graph (where the vertices represent pixels) and the task becomes that of categorizing the nodes – either as background/fore-
1.1. Learning Structured Outputs

Notice that unlike binary classification or regression, the input-output variables both lie in a structured space instead of a Euclidean space. Techniques to solve this problem involves modeling the structure in the form of a probabilistic graphical model. Graphical models makes this approach ideal since they encode the relationships between variables using a well-defined language and introduce conditional indecencies between variables to drastically the problem size. Furthermore, models such as a Factor Graph, encodes a joint probability distribution over the variables and hence expressing a full probability distribution over all feasible solutions. When these graphical models are further parametrized, the learning problem simply reduces to estimating the optimal parameter.

1.1.1 Semantic Image Segmentation

One of the fundamental challenges in Computer Vision is the task of Semantic Image Segmentation, also referred to as Object-Class recognition. The task is to categorize/label each pixel in the image into a predefined number of classes (such as airplane, dog, etc) as shown in Figure 1.2.

We are motivated to solve this problem because:

- Images are a rich source of information, which allows comprehension in numerous ways. Furthermore, objects within a single image exhibit natural relations, such as cars often appear on top of roads. Although these might seem really simple to a human, they are unarguably non-trivial to model and learn efficiently.

- They provide an innumerable number of possibilities to semantically segment. As a result, randomly guessing is useless.

- Solving the optimal solution from the graphical model which encodes the input is tough. Apart from being intractable, even finding a good approximate solution is time consuming.
1.2. Motivation

Figure 1.2: Examples for Semantic Image Segmentation. The task is to label each pixel of the image (left) and produce a semantic segmentation (right). The images and their ground-truth annotation here are from the Pascal VOC 2010[7] and MSRC-21 datasets[40].

Being able to efficiently solve this problem is important for many reasons. Firstly, it models how humans perceive visual scenes – which finds applications in multiple areas. Secondly, it could be used for a variety of industrial applications. For instance, for a self-driving car to simply understand an image and detect where the road and obstacles are. Most importantly, this problem is also the underbelly for many common challenges faced in the field – like solving it in reasonable time, associated computational costs and memory constraints, or the general intractability.

1.2 Motivation

Many methods in machine learning are posed as a minimization of a convex objective, which is a function of a regularization and loss term:

\[
\min \frac{\lambda}{2} \|w\|^2 + \frac{1}{M} \sum_{m=1}^{M} Q(x^m, y^m; w)
\]

Here, \(D = \{(x^m, y^m)\}_{m=1}^{M}\) are the labeled training examples, where \(x^m \in \mathbb{R}^d\), \(y \pm 1\) and \(w \in \mathbb{R}^d\) is a parameter which we need to estimate. Let us assume \(Q = \Delta(h_w(x^m), y^m)\), where \(h_w(x)\) is a hypothesis function which predicts an output \(y\) some input \(x^m\) using the parameter \(w\) and \(\Delta(y', y^m)\) is a loss function which measures the penalty for predicting \(y'\) instead of \(y^m\).

A popular class of optimization technique to optimize are gradient (or subgradient) descent based methods. Generally, in these methods one optimizes (assuming stochastic optimization) \(w\) via the following update in each iteration:

\[
w_{t+1} := w_t - \eta_t \nabla_w Q(x^m, y^m)
\]

for some randomly picked example \((x^m, y^m)\) and learning rate \(\eta_t \in \mathbb{R}\).
In many cases, the gradient $\nabla_w Q(x^m, y^m)$ can be computed quickly. Take for example a Support Vector Machine (SVM) for binary classification:

$$\nabla_w Q(x^m, y^m) = \begin{cases} 
\lambda w & \text{if } y^m w x^m > 1 \\
\lambda w - y^m x^m & \text{otherwise}
\end{cases}$$

where $y = \pm 1$ and $x \in \mathbb{R}^d$.

In contrast, computing the subgradient for a large margin Structured Prediction problem involves computing an equation of the form:

$$\nabla_w Q(x^m, y^m) = \lambda w + (\phi(x^m, y^m) - \phi(x^m, \hat{y}))$$

where

$$\hat{y} = \arg\max_{y \in \mathcal{Y}^m} \Delta^m(y) + \langle w, \phi(x^m, y) \rangle$$

(1.1)

where $x^m \in \mathcal{X}^m$, $y^m \in \mathcal{Y}^m$ are the complex structured input-output pairs. $\mathcal{Y}^m$ here can be exponentially large – for example, in image segmentation we have $|\mathcal{Y}^m| = 21^{320 \times 213}$ for a $320 \times 213$ pixel image over 21 object classes. Solving Equation (1.1) is referred to as the loss-augmented decoding sub-problem. This sub-problem if often application-specific and many optimization algorithms assume access to a solver referred as Maximization Oracle or max-oracle. These oracles traditionally use tricks and application-specific knowledge to quickly and efficiently solve the sub-problem.

This loss-augmented decoding primarily motivates the thesis because it can be frustrating:

1. **Max-oracles are expensive**: Due to the complex nature of the problem, obtaining $\hat{y}$ is a challenge because the output domain $\mathcal{Y}^m$ can be exponentially huge. We will see later that this step can be formulated to finding an optimal assignment in a Factor graph. While this is efficient in some cases (in case of chain or tree topologies), it remains intractable otherwise (in case of loops). Given that this problem needs to be solved in each iteration further makes the max-oracle calls extremely expensive. In case the graph has loops, this is an NP-hard problem like in the case of Semantic Image Segmentation and hence people resort to approximate decodings. This brings us to the next point.

2. **The Approximation Decoding paradox**: Max-oracle based solvers generally assume Equation (1.1) is exactly solvable. Such as seen previously, this is no more the case for a topology with loops. As a result, people resort to approximate methods to solve the decoding problem. This raises an issue as studied in [17] where they found that learning can fail even with rigorous approximation guarantees. But if one had access to good approximate decodings, one can iterate faster to solve the problem mentioned in point 1.

### 1.3 Contribution

The thesis contains the following contributions:
1.4. Thesis Outline

- Extending the Block-Coordinate Frank Wolfe algorithm for Structured SVMs [18] (BCFW) into a generalized algorithm “HADES-BCFW” which learns from decodings generated over a series of hierarchical subspaces of the original domain.

- Formulating the problem of decoding on Conditional Random Fields (CRF), which is very often encountered in the field of Computer Vision, as a series of coarse-to-fine hierarchical surrogate CRFs.

- Analyzing the coarse-to-fine surrogate CRFs in terms of approximation qualities obtained between subspaces.

- Demonstrating all the above in the context of dissolvestruct, a scalable structured prediction solver. As a result, we were able to run experiments on large datasets which would not be possible otherwise on a single machine.

- To the best of our knowledge, we are the first to demonstrate and study how approximate inference can be used to speed-up the training of Structured SVM solvers.

1.4 Thesis Outline

The rest of the thesis is structured as follows. We place our contributions in the context of related work in Chapter 2. Chapter 3 introduces the reader to the notation and formalizes the background required to understand the following chapters.

In Chapter 4, we introduce the library dissolvestruct and provide brief documentation. This library was used during the course of the thesis and also contains the implementation of the approximate structured learning algorithm which was mentioned previously.

We elaborate on the HADES-BCFW algorithm in Chapter 5 and also provide an interpretation. Considering this algorithm is general, we introduce the notion of hierarchical approximate decodings in the context of Potts model CRF for Semantic Image Segmentation in Chapter 6.

The experimental setup, details about the implementation, few extensions to the model and the results are discussed in Chapter 7. Finally, Chapter 8 contains concluding notes and remarks.
The contribution of this work lies in an intersecting area of various domains – Structured Output Prediction, Computer Vision and Distributed Optimization. So, in this chapter we will briefly review the literature individually in each of these areas and place our contribution in context of these works.

**Learning Structured Models**

One of the earliest methods for training Structured Models dates back to [4]. This work presented the Perceptron algorithm to learn the linear chain CRF parameters for part-of-speech tagging. Given that learning structured models is an extremely huge topic, we will limit ourselves to the domain of large margin learning. The large margin methods can be broadly divided into primal and primal-dual methods.

**Primal methods** The research by [41] and [43] presented the formulation of structured output learning as a maximum-margin formulation. To solve the constrained objective, [43] introduced the Cutting Plane algorithm which could achieve convergence after $O\left(\frac{1}{\epsilon}\right)$ steps. Subgradient methods like Stochastic Subgradient methods [36] are extremely popular too, mainly because of their simplicity. The proposed algorithm Pegasos is capable of solving the unconstrained primal objective and guarantee convergence in $O\left(\frac{1}{\epsilon}\right)$ steps. Structured Primal-Dual methods

**Primal-Dual methods** A popular method to minimize the dual objective is the Sequential Minimal Optimization (SMO) algorithm. This was generalized to Structured SVMs by [41], although the convergence rates scaled badly with the size of the output space. Our main work, using hierarchical decoding scheme, is instead based on another primal-dual scheme – the Block Coordinate Frank Wolfe (Bcfw) algorithm [18]. The Bcfw algorithm provides guarantees even with approximate oracles under some conditions. This algorithm also computes the optimal step-size for a given decoding and as a result, can be used to evaluate the quality of a decoding. Other popular methods which solve the SVM objective in its dual include the Exponentiated Gradient [5] and the Proximal SDCA method [37].
Semantic Image Segmentation

Given that we exhibit our contribution to solve the task of Semantic Image Segmentation using the MSRC-21 [40] dataset, we will focus only on similar work. A majority of the work in this field focuses on obtaining good accuracy on the standard dataset. They achieve this in multiple ways. Some of them use more sophisticated models. A simple extension involves introducing global consistency [15, 33, 9] because it captures the “big picture” details of a scene. [42, 15] introduced a P^N model which jointly inferred pixel labels and image-level preferences in order to encourage consistency between labels of local and global variables. [9] use “harmony potentials” to model global preferences using the power set over all semantic classes. In our work however, we demonstrate our contributions using a simple Potts model CRF.

Another technique to obtain good accuracy on the dataset involves the art of feature engineering, which is a popular research topic. A popular set of features “TextonBoost” were described in [16], which uses color, HOG and pixel-location features following the works of [38, 33]. In contrast, we solely rely on features from Convolutional Neural Nets from various layers extracted by OverFeat [34], making our approach related to [3].

Apart from sophisticated models and features, another breakthrough to perform better include using kernelized methods [35, 2, 24] and latent variables [46].

Approximate Structured Output Prediction

As we’ve seen, the max-oracle based solvers involve computing a MAP assignment on a factor graph containing loops. This makes the problem intractable and therefore learning intractable. This leads to a popular use of approximate inference techniques. [17] however show that a learning algorithm can fail, even if the approximate inference algorithm has strong approximate guarantees. They argue the expressivity of the model decreases and can vary greatly from that of exact inference. The learning algorithm can moreover misinterpret the feedback received from such methods.

[8] introduced the notion of under-generating and over-generating techniques. The former searches makes the prediction tractable by only searching for a subset of the labels \(Y_{\text{under}} \subset Y\). Over-generating techniques achieves the same with LP relaxations and search over a set larger than the original output space \(Y_{\text{over}} \supset Y\). They conclude that over-generating techniques were more suitable and confirmed it with experimental results. In contrast, we rely on an under-generating technique by searching over a sequence of hierarchical subspaces. We use this in conjunction with the Bcrw algorithm [18] which guarantees convergence provided an oracle with additive or multiplicative accuracy which improves over time.

Hierarchies and Structured Prediction

Numerous works have captured high-level information from an image using a hierarchy. In most of these cases, the goal is to learn an image modeled as a hierarchical CRF. In other words, they model the input-output objects as a
hierarchical model containing a spectrum of low to high level features. The
approach proposed by [33] defines such a CRF on multiple discrete quanti-
zations of the image space. As a result, they obtain features for each such
level and provide a unified approach for solving the same. [30] present a
tree-structured CRF which learns the image in a hierarchical representation.
Because of this structure, they have the benefit of solving a tractable inference
problem. The work in [20] suggest a similar approach, defining “region an-
cesty” by encoding the image in different levels. Other works, where the the
core idea is to use features at different scales to increase the discriminative
power during prediction, include [28, 47].

The research by [14] suggest a grouping strategy similar to ours. However,
they mainly use it to speed-up inference time during predictions rather than
use it to make approximate predictions during training. Moreover, we leave
it to the user to formulate the hierarchical groups making this approach a
suitable candidate to construct the hierarchy.

Alternately, cascades have also been used to speed-up training of higher order
models like in the case of [45, 44]. Here, they use a hierarchy of low-to-high
order models. Inferring then entails filtering out states in this sequence and
hence making the computation tractable and efficient even in case of higher
order models.

Unlike these approaches, ours is vastly different, in a sense that we use a
particular level in the hierarchy to achieve a certain approximate guarantee.
We use this approximate inference in conjunction with our algorithm to speed
up training. Moreover, unlike other methods where the features are extracted
per level, we define the hierarchy solely based on the final level. As a result,
our hierarchical definition naturally extends any existing implementation.
Chapter 3

Preliminaries

3.1 Structured Prediction

Formally, Structured Prediction deals with learning to predict structured objects, taking the form:

$$\hat{y} = h(x) = \arg\max_{y \in \mathcal{Y}} F(x, y)$$ (3.1)

The task is to make a prediction using a hypothesis function $h(x)$ for some input $x \in \mathcal{X}$. $F(\cdot, \cdot)$ is referred to as a discriminant function, compatibility function or scoring function. Finding $y$ in this formulation is referred to as the inference problem. Often, this is expressed as a linear model parametrized by $w \in \mathbb{R}^d$ and a joint feature map $\phi(x, y) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^d$. Hence, the prediction function (3.1) can be rewritten as:

$$\hat{y} = h(x; w) = \arg\max_{y \in \mathcal{Y}} \langle w, \phi(x, y) \rangle$$ (3.2)

Using this form, the task of learning a structured model becomes that of learning the parameters $w$. The feature map $\phi$ can be thought of as a combined representation of the input-output pairs in the Euclidean space.

To learn the mapping between the input-output pairs as given in (3.2), there are two main challenges. Firstly, given some parameter $w$ the inference problem is in most cases computationally prohibitive or intractable since it involves computing the argmax over an exponentially large space $\mathcal{Y}$. For instance, consider the previously mentioned part-of-speech tagging problem. Solving the inference problem over a sentence with 10 words, and 36 tags (like in the Penn Treebank) involves $36^{10}$ possibilities. Fortunately, the inference task can sometimes be expressed equivalently to that of solving a maximum a posteriori estimated assignment in a graphical model. Although often this can comparatively more efficient in solving problem (3.2), some models contain loops (as we will see ahead in Image Segmentation) making this an NP-hard problem. Secondly, given a number of training examples and a tractable technique to solve (3.2), we use these two to fit the model parameter $w$. This is relatively straight-forward and is formulated as a minimization problem of an objective function, defined using $w$ (regularization) and a loss-function over the examples.
3.1. Structured Prediction

3.1.1 Structured Support Vector Machines

One of the most popular methods to estimate \( w \), as a means for computing the discriminant function \( h(x; w) \), is given by the maximum-margin formulation. In this section, we briefly introduce the notation and formulation of the structured SVM problem, which uses this formulation.

Given a labeled training data set \( D = \{ (x^m, y^m) \}_{m=1}^M \), \( w \) is estimated by solving the following constrained objective:

\[
\min_{w, \xi} \frac{\lambda}{2} \| w \|^2 + \frac{1}{M} \sum_{m=1}^M \xi^m \\
\text{s.t.} \quad \langle w, \psi^m(y) \rangle \geq \Delta^m(y) - \xi^m \\
\forall m, \forall y \in Y(x^m)
\] (3.3)

Here, \( \psi^m := \phi(x^m, y^m) - \phi(x^m, y) \) and \( \Delta^m(y) := \Delta(y^m, y) \) is the task-dependent structured error of predicting \( y \) instead of the ground-truth \( y^m \). \( Y^m = Y(x^m) \) represents the set of all output possible outputs for an input \( x^m \in X^m \). \( \xi^m \) is the slack variable which measures the surrogate loss for the \( m \)-th example and \( \lambda \) is the regularization parameter.

**Loss-augmented decoding** Notice that in Equation (3.3), there are \( \sum_m |Y^m| \) constraints due to the combinatorial nature of \( Y^m \). Since this can be much larger than what can fit in memory, we replace the potential exponential number of constraints with \( M \) piecewise-linear ones by defining structured hinge-loss:

\[
H^m(w) := \max_{y \in Y^m} \Delta^m(y) - \langle w, \psi^m(y) \rangle
\] (3.4)

Due to this, we now can now reformulate optimization problem (3.3) with one constraint per training example \( \xi^m \geq H^m(w) \). Doing so effectively replaces \( |Y^m| \) constraints with the single ‘most violating’ constraint as defined by (3.4) for each training example. The problem of finding this most-violating output is known as the loss-augmented decoding sub-problem and the solver which performs this task is termed as the maximization oracle. Generally, solvers assume this maximization oracle performs the decoding exactly.

Using this definition, we can reformulate the objective (3.3) into an equivalent non-smooth unconstrained formulation:

\[
\min_{w} \frac{\lambda}{2} \| w \|^2 + \frac{1}{M} \sum_{m=1}^M H^m(w)
\] (3.5)

**The Dual** The \( M \)-slack formulation given in (3.3) takes a dual form with \( k := \sum_m |Y^m| \) variables or ‘support vectors’. The dual problem is given by:

\[
\min_{\alpha \in \mathbb{R}^k} \frac{\lambda}{2} \| A\alpha \|^2 - b^T \alpha \\
\text{s.t.} \quad \sum_{y \in Y^m} a^m(y) = 1 \quad \forall m \in [M]
\] (3.6)
where \( a^m(y) \) is the dual variable corresponding to training example \( m \) and potential output \( y \in \mathcal{Y}^m \). The constraints \( \sum_{y \in \mathcal{Y}^m} a^m(y) = 1 \) imply that each \( a^m \) is a part in a simplex. Matrix \( A \in \mathbb{R}^{d \times k} \) consists of \( k \) columns

\[
A := \left\{ \frac{1}{\lambda M} \psi^m(y) \in \mathbb{R}^d \mid m \in [M], y \in \mathcal{Y}^m \right\}
\]

and vector \( b \) is given by

\[
b := \left( \frac{1}{M} \Delta^m(y) \right)_{m \in [M], y \in \mathcal{Y}^m}
\]

Given a dual variable vector \( \alpha \), we can obtain the corresponding primal variable as:

\[
w = A \alpha = \sum_{m, y \in \mathcal{Y}^m} a^m(y) \frac{\psi^m(y)}{\lambda M}
\]

### 3.1.2 Optimization

One popular solution to solve the unconstrained primal objective given in Equation (3.5) is using Stochastic Subgradient methods (SGD) [31, 36]. This is because each iteration involves calling the computationally expensive maximization oracle just once and additionally, they achieve a \( O(1/\epsilon) \) convergence rate. We will briefly look into this since it provides a generalized form of maximization-oracle based solvers.

In order to use the stochastic subgradient method, the subgradient of the unconstrained objective needs to be calculated in each iteration. For a training example \((x_m, y_m)\), with \( y^* = \max_{y \in \mathcal{Y}^m} \Delta^m(y) + \langle w, \psi^m(y) \rangle \), the subgradient amounts to

\[
\frac{d}{dw} \left[ \frac{\lambda}{2} \|w\|^2 + \frac{1}{M} \sum_{m=1}^{M} \langle w, \psi^m(y^*) \rangle \right] = \lambda w + \psi^m(y^*)
\]

An update at iteration \( t \) for this example then corresponds to:

\[
w_t := w_{t-1} - \frac{\eta}{t} \left[ \lambda w_{t-1} + \psi^m(y^*) \right]
\]

In this class of algorithms, the maximization oracle is typically the computational bottleneck, even though it needs to decode only a single training example per iteration and not all of them, as batch gradient descent would. Moreover, sometimes one can never find the exact decoding and cannot converge to the minimum, but only to an \( \epsilon \)-ball [23, 31].

**Block-Coordinate Frank-Wolfe Algorithm**

While the stochastic subgradient descent algorithm is straight-forward and easy to implement, the main focus henceforth is going to be on the Block-Coordinate Frank-Wolfe (BCFW) algorithm [18]. The BCFW algorithm (Algorithm 1) removes a number of pain points from the previously seen Stochastic Subgradient Descent.
3.1. Structured Prediction

The BCFW algorithm is an online maximization oracle-based solver which solves the dual Structured SVM problem. Although, the dual problem requires keeping track of \( \sum_{m} |Y_m| \) dual variables, BCFW overcomes this by storing these variables as a function of the sparse iterates during optimization. The BCFW algorithm also maintains the \( O(1/\epsilon) \) convergence rate and requires decoding of only a single training example per iteration, same as in SGD.

The BCFW algorithm provides three main benefits over stochastic subgradient methods:

1. **Optimal Step-size.** Unlike the previous method, where the step-size \( \eta \) is fixed, BCFW analytically calculates the optimal step-size for a given decoding in closed form.

2. **Duality Gap guarantee.** Since BCFW solves the dual problem and the primal-dual gap can be computed, the duality gap serves as a proper stopping criterion.

3. **Approximate Oracles.** BCFW provides guarantees for approximate maximization oracles with the same convergence rate as long as the quality of the approximate decoding can be controlled.

\[
\text{Algorithm 2: Bcfw: Block-Coordinate Frank-Wolfe algorithm for Structured SVM}
\]

\[
\begin{align*}
\text{Input: Data } D &= \{(x^m, y^m)\}_{m=1}^M \\
\text{Initialize: } w(0) &\leftarrow 0, w^m(0) &\leftarrow 0, b(0) &\leftarrow 0, b^m(0) &\leftarrow 0 \\
\text{for } t = 1 \ldots T \\
&\text{Choose } m \in \{1, 2, \ldots, M\} \text{ uniformly at random}
&\text{Solve } \hat{y} \leftarrow \arg \max_{y \in Y} H^m(y; w(t))
&\text{Let } w_s \leftarrow \frac{1}{M} \Psi^m(\hat{y}) \text{ and } b_s \leftarrow \frac{1}{M} \Delta^m(\hat{y})
&\text{Let } \gamma \leftarrow \frac{\lambda (w^m_{(t-1)} - w_s)^T w_{(t-1)} - b^m_{(t-1)} + b_s}{\lambda \| w^m_{(t-1)} - w_s \|^2} \text{ and clip to } [0, 1]
&\text{Update } w^m_{(t)} \leftarrow (1 - \gamma) w^m_{(t-1)} + \gamma w_s \text{ and } b^m_{(t)} \leftarrow (1 - \gamma) b^m_{(t-1)} + \gamma b_s
&\text{Update } w_{(t)} \leftarrow w_{(t-1)} + w^m_{(t)} - w^m_{(t-1)} \text{ and } b_{(t)} \leftarrow b_{(t-1)} + b^m_{(t)} - b^m_{(t-1)}
\end{align*}
\]

\[
\text{Output: } w(T) \text{ and } b(T)
\]

We will later see how we use these specific advantages to learn from approximate decodings.
3.2. Semantic Image Segmentation

Recall the task of image segmentation previously described in Section 1.1.1. In semantic image segmentation, the task is to label each pixel in an input image as one of a finite number of classes (for example, 21 in case of the MSRC-21 dataset).

A popular method of solving this is by formulating the task as that of parameter estimation of a Conditional Random Field (CRF). CRFs are a class of graphical models, which models a conditional distribution \( p(y|x) \) with an associated graphical structure. The images or visual scenes are first broken down into nodes/regions (which could be pixels or groups of neighboring pixels). These nodes form a graphical structure over which the CRF is defined, with neighboring regions represented by edges. By doing so, the primary benefit is a label of a particular node in the image is conditionally dependent on the neighboring labels. This is contrasting to a multiclass classifier, which predicts labels without taking into account the context provided by the neighboring regions. Hence, CRF-based models can additionally capture relations like cow often appears next to grass or cars appears on top of roads.

In the remainder of this section, we will further explore the implementation and formalism involved with the problem.

3.2.1 Data, Superpixels and Features

In this section, we will introduce some practical aspects related to the problem. We will take a closer look into these later in the experimental results section.

**Data** One of the popular datasets used for the task of Semantic Image Segmentation is the MSRC-21 dataset [38]. This dataset contains 591 images with objects from 21 categories, along with annotations for all pixels for each training image. An example of the input-output image pair is shown in figures 3.2a and 3.2b.

**Superpixels** In order to represent the image as a graphical structure, we first need to decide what comprises a node or region. The most straightforward option is to simply treat each pixel as a node, with neighboring pixels forming edges. But, this would translate to a quadratic blow-up in the number of edges, drastically increasing the inference time on the CRF. Alternatively,
3.2. Semantic Image Segmentation

an effective option is to reduce the number of nodes by collectively grouping them into superpixels. Throughout this report, we take such an approach, by constructing superpixels for each image using the SLIC algorithm [1]. SLIC uses a k-means clustering approach to efficiently generate super-pixels which adheres well to boundaries. The greatest benefit is that the size of the CRF remains independent of the size of the image. The superpixels generated by SLIC on the MSRC-21 dataset can be seen in Figure 3.1. For the remainder of the report, we will only consider dealing with superpixels, rather than pixels.

Features  Engineering features for images is an active research topic and one might consider it a separate challenge all together. However, we will briefly look into it. Features for the images in our case is represented per super-pixel. Each is feature is a vector and is referred to as the data or unary term. A simple technique is to represent each superpixel in terms of a d-dimensional RGB histogram vector. In order to incorporate additional context, this vector is sometimes also concatenated with the RGB histogram over the node’s neighbors at different scales.

3.2.2 Learning to Semantically Segment

Now, that we have our dataset, the superpixels for the images and the features per superpixel, we describe how to approach solving the problem of semantically segmenting images.

To describe this, we use a two-step approach. First, we formulate the task of Image Segmentation as that of a parametrized CRF model. Next, we show how we learn these parameters using a Structured Support Vector Machine (SSVM).
3.2. Semantic Image Segmentation

But, before we start, let us fix the notation used to represent the images. Take for instance, the image and its ground truth labels as seen in Figure 3.2a and 3.2b.

We begin by treating images as a composition over superpixels (See Figure 3.2c). Each superpixel in the image can then be treated as a vertex, with the edges representing neighboring vertices. Each node has an associated ground-truth label.

As for the notation, we represent our labeled training image as \((x, y)\). This is defined over a graph \(G = (V, E)\), such that each node \(i \in V\) represents a superpixel and \((i, j) \in E\) represents an edge. Each (superpixel) vertex is denoted by \(x_i \in \mathbb{R}^d\) associated with a label \(y_i \in \{1, \ldots, K\}\).

The Image Segmentation CRF Model

Factor graphs are a powerful tool to represent joint distribution over random variables. A factor graph \(F = (V_F, C_F, E_F)\) contains two kinds of nodes – random variable nodes \(V_F\) and factor nodes \(C_F\). Diagrammatically, the random variable nodes are illustrated by a circle. The interaction between the random variables are captured by factors, and are illustrated by solid squares. The factor graph is hence just a bipartite graph, with random variable nodes in one group and factors in the other. The Energy \(E(y)\) of a configuration is given by

\[
E(y) = \sum_{c \in C_F} E_c(y_c)
\]

\(E_c(y_c) : \mathcal{V}_C \rightarrow \mathbb{R}\) is a factor, which captures the potential or individual energy that arises from the joint configuration \(y_c\).

We apply this to formulate the max-oracle or the decoding problem (as given in Equation (3.4)), as that of solving the maximum-a-posteriori assignment on a factor graph. We construct the factor graph for the image segmentation task as follows. Given an input image \(x^m\), we first break it down into its superpixel as shown in Figure 3.3a. If an annotation \(y^m\) exists (like during training), we transfer the superpixel boundaries to it and obtain the ground-truth label per super-pixel. This is illustrated in Figure 3.3b. We create a blueprint model of the image in the form of a graph \(G = (V, E)\) shown in 3.3c. Here, the vertex \(i \in V\) is used to index the superpixels in the image and edge \((i, j) \in E\) to represent neighboring superpixels. This lets us denote the superpixel feature vector as \(x_i^m \in \mathbb{R}^d\), an assignment of a label to each superpixel as \(y_i \in \{1, \ldots, K\}\) and the random variables for the factor graph as \(Y_i\). To construct the factor graph, we use the last idea.

The factor graph, as illustrated in Figure 3.3d, is known as the Potts model. The model contains the the random variables \(Y_i\) illustrated by circular nodes. It also contains two types of factors. The gray factors are the data or unary factors with energy \(E(Y_i = y; x_i^m, w)\), which measures the compatibility between the assignment and the superpixel features, parametrized by \(w\). The black factors are the pairwise factors with energy \(E(Y_i = y_i, Y_j = y_j; x_i^m, x_j^m, w)\), which captures the penalty for the label transition \(y_i \rightarrow y_j\).
Additionally, we will drop the factor graph notation used above because of its unnecessary notational complexity. We will restrict to using the simple graph structure $G = (V, E)$ to represent the Potts model factor graph.

We assume that this graphical model is a parametrized conditional probability distribution (and hence the term Conditional Random Field) given by

$$p(y|x; w) = \frac{1}{Z(x, w)} \exp(-E(y|x; w))$$  \hfill (3.7)

where $Z(x, w) = \sum_{y \in Y} E(y|x; w)$, $E$ is the energy function (or the negative discriminant function $F$ earlier seen in Equation (3.1)) and $w$ is the unknown parameter which we wish to learn. The core idea is that the energy of a configuration is proportional to its probability

$$p(y|x; w) \propto -E(y|x; w)$$

The energy function $E: Y \rightarrow \mathbb{R}$ for our problem is given by:

$$E(Y = y|X = x; w) = \sum_{i \in V} E^U(Y_i = y_i|X_i = x_i; w^U)$$ \hfill (3.8)

$$+ \sum_{(i,j) \in E} E^P(Y_i = y_i, Y_j = y_j|X_i = x_i, X_j = x_j; w^P)$$

where $w = (w^U, w^P)$.  

Figure 3.3: Data representation for an input-output pair
Alternately, we sometimes use the shorthand:

\[ E_w(y) = \sum_{i \in V} E^{U}_i(y) + \sum_{(i,j) \in E} E^{P}_{ij}(y) \]  

As mentioned before, \( E^{U}_{i}(y) \) is the *unary*-factor and encourages agreement between the node’s label \( y_i \) and the local image evidence \( x_i \). \( E^{P}_{ij}(y) \) is the *pairwise*-factor which captures the cost of transition from label \( y_i \) to \( y_j \).

Some approaches [42, 9], in addition to the unaries and pairwise factors, also consider a global factor. However, similar to [22] we directly introduce the global features in to the data term.

Notice that, as a result of Equation (3.7), for some parameter \( w \), we want the a label assignment with the highest probability, which is equivalent to one with the least energy.

\[ \arg \max_{y \in \mathcal{Y}} p(y|x; w) = \arg \min_{y \in \mathcal{Y}} E(y|x; w) \]

As a result, the inference step given some \( w \) is formulated as an *Energy Minimization* problem. This can often be solved efficiently using Belief Propagation or Mean-Field.

### Parameter Learning using SSVM

Now that we have a CRF model for Image Segmentation, we want to learn the parameter \( w^* \), which makes \( p(y|x; w) \) as close as possible to the true distribution. In order to do this, we use a Structured SVM for parameter estimation.

The SSVM uses the labeled training data \( \mathcal{D} = \{(x^m, y^m)\}_{m=1}^{M} \) to estimate the optimal parameter \( w^* \). To measure the closeness to the true distribution, it uses a loss function \( \Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+ \). A straight-forward loss function would be the 0-1 structured Hamming loss

\[ \Delta(y^m, y) = \sum_{i \in V} \delta(y^m_i, y_i) \]

However, the image segmentation datasets are often imbalanced with respect to the categories. Hence, we consider the structured hamming-loss re-weighted by the class frequency

\[ \delta(y^m_i, y_i) = \begin{cases} \frac{1}{\text{frequency}(y^m_i)}, & \text{if } y^m_i \neq y_i \\ 0, & \text{otherwise} \end{cases} \]

where frequency \( (y^m_i) \) refers to the frequency of label \( y^m_i \) occurring in the training dataset. Recall, Maximization-oracle based SSVM solvers require a loss-augmented decoding for a training example \( (x^m, y^m) \) for some parameter \( w \) in each iteration i.e.,

\[ H^m(w) = \arg \max_{y \in \mathcal{Y}^m} \Delta^m(y) - \langle w, \psi^m(y) \rangle \]
3.2. Semantic Image Segmentation

In our case, we formulate this decoding as equivalent to the Energy minimization problem

\[ H^m(w) = \arg\min_{y \in \mathcal{Y}^m} E(y|x^m; w) - \Delta^m(y) \]  \hspace{1cm} (3.11)

The Energy function can be expressed as the inner product between the weight vector and the joint feature map

\[ E_U(y) = \langle w^U, \phi_U(x, y) \rangle \]  \hspace{1cm} (3.12)
\[ E_P(y) = \langle w^P, \phi_P(x, y) \rangle \]  \hspace{1cm} (3.13)

If we let the feature map of an image be the sum of the feature maps over the individual nodes

\[ \phi_U(x, y) = \sum_{i \in \mathcal{V}} \phi_{i}^U(x, y) \]  \hspace{1cm} (3.14)
\[ \phi_P(x, y) = \sum_{(ij) \in \mathcal{E}} \phi_{ij}^P(x, y) \]  \hspace{1cm} (3.15)

and

\[ w = [w^U \atop w^P] \]  \hspace{1cm} \[ \phi(y) = [\phi_U(y) \atop \phi_P(y)] \]

we can represent the total energy as the sum of the individual unary and pairwise energy components as:

\[ E_w(y) = \langle w, \phi(y) \rangle \]  \hspace{1cm} (3.16)

Let \( x_i \in \mathbb{R}^d \) be the feature vector the \( i \)-th node. We then define the unary feature vector for this node as

\[ \phi_i^U(x, y) = \begin{bmatrix} \mathbb{1}[y_i = 1] x_i \\ \mathbb{1}[y_i = 2] x_i \\ \vdots \\ \mathbb{1}[y_i = K] x_i \end{bmatrix} \]

where \( K \) is the number of classes and \( y_i \in \{1, \ldots, K\} \). Observe that the unary feature map places the feature vector for node \( i \) in the \( y_i \)-th block. Hence, we can rewrite the unary term as

\[ E_i^U(y) = \langle w^U, \phi_i^U(x, y) \rangle = \langle w^U, x_i \rangle \]  \hspace{1cm} (3.17)

Similarly, we define the pairwise joint feature map as:

\[ \phi_{ij}^P(x, y) = \begin{bmatrix} \mathbb{1}[y_i = a, y_j = b] \end{bmatrix}_{(a,b) \in \{1, \ldots, K\}^2} \]

Because of this sparse notation, the pairwise energy is:

\[ E_{ij}^P(y) = \langle w^P, \phi_{ij}^P(x, y) \rangle = w_{y_i,y_j}^P \]  \hspace{1cm} (3.18)
In this chapter, we introduce dissolve\textsuperscript{struct}, a distributed solver for structured output prediction tasks. We will first cover the motivation behind developing the framework, followed by the algorithm, implementation details and brief documentation.

4.1 Motivation

Structured Prediction is a popular and widely applicable supervised technique for solving complex problems nowadays. Unlike binary or multiclass classifiers, structured prediction methods can be used to efficiently predict structured output labels. Structured outputs here refers to structures like strings, sequences, graphs and trees. Additionally, they provide the advantage of taking the context into account.

A popular technique used for structured prediction is the Structured Support Vector Machine (SSVM) as described in Section 3.1.1. These are a generalization of large margin methods trained to predict structured responses. As [27] points it out, they provide multiple advantages over other traditional approaches. Most importantly, SSVMs train on reducing the Bayesian empirical risk, by minimizing over a user-specified loss function. Additionally, because of the strongly convex objective, they are amenable to efficient optimization algorithms.

Oracle-based SSVM optimization algorithms, like SGD or Bcfw, are common techniques to minimize the SSVM objective. These solvers require calls to a maximization oracle in each iteration. This oracle, generally involves computing the MAP assignment on a non-trivial factor graph. As a result, each such call is expensive and is the computational bottleneck in this class of algorithms.

An interesting property of this oracle is that it depends only on a single training example for a given set of parameters. Hence, in an approach such as Stochastic Subgradient Descent using mini-batches, one could distribute the decodings on different machines. While this would be ideal, the main challenge to overcome is the communication overhead involved between iterations. In traditional distributed frameworks like Map-Reduce, a communi-
cation round is expensive because of the batch-processing premise on which it is designed. This translates to each round of communication triggering data to exchange among the nodes and incurring a significant penalty. Given that often hundreds of iterations are made by the solver, more time would be spent communicating rather than computing.

Being able to distribute a structured prediction task is important for many reasons. Firstly, problems of this nature are memory-intensive. In case of image segmentation, a training example is represented using a graph structure with each node comprising of a possibly large feature vector. Furthermore, datasets like Pascal VOC [7] and Microsoft COCO [21] contain thousands of images, each image represented by hundreds of such vectors. Hence, for even moderately large datasets, this would require special hardware, which might be inaccessible to many. Secondly, because the problems and tasks are huge, it is common for training a structured prediction model to take days. Given that many optimization techniques are often embarrassingly parallel in theory, it is extremely beneficial to implement a solution which takes advantage of the parallelism to speed-up training.

4.2 Method Description

To solve the distributed structured prediction problem, we use the recently proposed CoCoA framework [10, 25]. The CoCoA framework takes advantage of the primal-dual structure of optimization problem to reduce the amount of the communication required. It does this by solving a dual optimization method simultaneously on multiple (worker) machines in rounds. To solve the local optimization problem, each machine needs access only to its local data. After each round, a single weight vector is communicated from the workers to the master node.

While CoCoA is defined for any local dual optimization method, dissolve\textsuperscript{struct} is designed to use the Block-Coordinate Frank Wolfe (BCFW) algorithm [18] as the local optimization procedure. BCFW is an online optimization algorithm which solves the SSVM in its dual form. Because each iteration of BCFW involves a call to the maximization oracle, which in Structured Prediction applications are the bottleneck, it’s useful to solve multiple such oracles simultaneously.

The BCFW algorithm makes an ideal choice for the task at hand because of many advantages over other subgradient descent methods. BCFW optimally calculates the step-size in each iteration, freeing the user from a parameter that greatly influences performance. Secondly, it provides the primal-dual gap which serves as a proper stopping criterion.

In the remainder of this section, we will delve into the algorithm used by dissolve\textsuperscript{struct} for performing the distributed optimization. In the next section, we will proceed to explain how this is implemented using Apache Spark.

4.2.1 Distributing BCFW

We begin by assuming our data $D = \{(x^m, y^m)\}_{m=1}^{M}$ is distributed over $K$ machines. We also assume another machine, the master, in addition to these
K workers which initializes and initiates the local optimization procedure on each machine. Each datapoint \((x^m, y^m)\) is associated with a primal parameter vector \(w^m\) and a primal value \(b^m\) which compactly represents the dual variables, distributed in the same way. This was previously described in Section 3.1.2. Due to this correspondence, we will interchangeably use dual variables and the respective primal mapping \((w^m, b^m)\). The key idea is the use the dual variables to merge the parallel updates from the K workers. By allowing each worker to optimize on a disjoint set of dual variable, the CoCoA framework allows to combine updates from the different machines in a communication efficient way, without conflicting each other.

**Algorithm 3: CoDBcfw: Communication-Efficient Distributed BCFW**

**Input:** \(T \geq 1, \) scaling parameter \(1 \leq \beta_K \leq K\) (default: \(\beta_K := 1\)).

**Data:** \(\{(x^m, y^m)\}_{m=1}^M\) distributed over K machines

**Initialize:** \(w^{(0)}_k \leftarrow 0, b^{(0)}_k \leftarrow 0 \) for all machines \(k\) and \(w^{(0)} \leftarrow 0, b^{(0)} \leftarrow 0\)

1. for \(t = 1, 2, \ldots, T\)
   2. for all machines \(k = 1, 2, \ldots, K\) in parallel
   3. \((\Delta w^k, \Delta b^k) \leftarrow \text{LocalBCfw}(w^{(t-1)}_k, w^{(t-1)}_k)\)
   4. \(w^{(t)}_k \leftarrow w^{(t)}_k + \frac{\beta_K}{K} \Delta w^k\)
   5. end
   6. reduce \(w^{(t)} \leftarrow w^{(t-1)} + \frac{\beta_K}{K} \sum_{k=1}^K \Delta w^k\)
   7. and \(b^{(t)} \leftarrow b^{(t-1)} + \frac{\beta_K}{K} \sum_{k=1}^K \Delta b^k\)
   8. end

The framework CoDBcfw which performs this distributed optimization is presented in Algorithm 2. In each round, all the workers run procedure LocalBCfw (given in Procedure A) on the local copy of data and dual variables. At the end of each round, each worker communicates a single weight vector \(\Delta w^k \in \mathbb{R}^d\) to the master. The master combines these updates (as shown in line 6) and initiates the next round of optimization. Notice that the \(\Delta w^k\) compactly represent the updates in the dual variables. This circumvents the need to communicate the dual variables or the data between machines.

CoDBcfw follows the same routine as CoCoA, but with one twist. CoCoA works by each local machine keep track of all dual variables \(\alpha\), performing updates on it in each iteration. However, in case of Bcfw, this is unfeasible to implement due to the exponentially large number of constraints \((= \sum_m |Y^m|)\). As a result, the dual variables \(\alpha^{[m]}\) corresponding to each block \(m\) are compactly represented using a single weight vector \(w^m = A\alpha^{[m]}\) and a value \(b^m = b\alpha^{[m]}\). Each \(w^m\) has a sparsity pattern, which is the combination of the corresponding joint feature vector representations \(\psi(y)\) for this \(m\).
4.3 Implementation Details

Procedure A: LocalBCFW: BCFW iterations on machine $k$

**Input:** $f \in (0, 1]$, $w^{[k]} \in \mathbb{R}^{M_k \times d}$ and $w \in \mathbb{R}^d$ consistent with other coordinate blocks of $\alpha$ s.t. $w = A\alpha$

**Data:** Local $\{\{x^m, y^m\}\}_{m=1}^{M_k}$

**Initialize:** $w^{(0)} \leftarrow w$, $\Delta w^{[k]} \leftarrow 0 \in \mathbb{R}^{M_k \times d}$, $H \leftarrow f M_k$

1. for $h = 1, 2, \ldots, H$
   2. choose $m \in \{1, 2, \ldots, M_k\}$ uniformly at random
   3. Solve $\hat{y} \leftarrow \arg\max_{y \in \mathcal{Y}^m} H^m(y; w^{(h)})$
   4. $w_s \leftarrow \frac{1}{\lambda} H^m(\hat{y})$ and $b_s \leftarrow \frac{1}{\pi} \Lambda^m(\hat{y})$
   5. $\gamma \leftarrow \frac{\lambda(w_m^{(h-1)} - w_s)^T w_m^{(h-1)} - b_m^{(h-1)} + b_s}{\lambda||w_m^{(h-1)} - w_s||^2}$ and clip to $[0, 1]$
   6. Update $w_m^{(h)} \leftarrow (1 - \gamma)w_m^{(h-1)} + \gamma w_s$
   7. and $b_m^{(h)} \leftarrow (1 - \gamma)b_m^{(h-1)} + \gamma b_s$
   8. Update $w^{(h)} \leftarrow w^{(h-1)} + w_m^{(h)} - w_m^{(h-1)}$
   9. and $b^{(h)} \leftarrow b^{(h-1)} + b_m^{(h)} - b_m^{(h-1)}$

10. end

**Output:** $\Delta w^{[k]}$ and $\Delta w^k$

Keeping track of these dual variables is central to using BCFW for two reasons. Firstly, it is used to calculate the optimal step size using line-search in each iteration (see line 5 of procedure A). Secondly, the duality gap is defined in terms of the dual variables, which is used by BCFW as a sound stopping criterion.

CoCoA inherits the convergence guarantee of any algorithm run locally, such as LocalBCFW. Since CoDBCFW is simply CoCoA applied to BCFW, it follows that we inherit the same convergence guarantee.

4.3 Implementation Details

dissolve\textit{struct} is designed as a Structured Prediction library, completely written in Scala 2.10. In order to develop a structured prediction application, the user merely needs to provide three application-specific functions, just like in SVM\textit{struct}[43], a popular non-distributed solver to solve the same task.

The library internally uses the popular Apache Spark framework in order to utilize the communication-efficient scheme. The Spark-specific details have been abstracted and the user can package an application and execute it anywhere -- locally or remotely using multiple cores or a cluster -- with minimal knowledge of Spark. The target scenario is a distributed setting, because structured prediction tasks are memory and computationally intensive. Often, one deals with more data than that can fit on a single machine’s memory.
But with dissolve_{struct}, the application has access to the entirety of the cluster’s memory. This way, larger applications can be simply handled by adding additional nodes to the cluster.

We choose Spark for dissolve_{struct} for multiple reasons. Firstly, because of its in-memory computation architecture. While alternatives like Hadoop pay an exorbitant communication penalty between rounds, this can be largely avoided in Spark. Resilient Distributed Datasets (RDD) in Spark have been designed such that nodes can retain and restrict data locally, and yet be fault-tolerant. This is pivotal, since our intention is to keep the data and the dual variables local to a machine. Secondly, Spark is simple and abstracts the distributed mechanism away from the user. By doing so, one can use a variety of common functional programming operations like map, filter, reduce or groupBy on an RDD treating it just like any other Scala collection object.

For the remainder of the section, we will dig deeper into the pipeline used by dissolve_{struct}.

### 4.3.1 CoDBcfw on Spark

In this section, we will see how Spark is used to implement CoDBcfw. For this, we assume that the reader has a basic understanding of Spark, RDDs, partitioning. We will merely bridge the gap between the CoDBcfw algorithm (refer to algorithm 2) and the Spark implementation. Note that this knowledge is not necessary for a practitioner to use dissolve_{struct}.

The core idea of using Spark for CoDBcfw is to prevent data and the dual variables being communicated between nodes during computation. This is done through the use of Pair RDDs and their corresponding functions. Pair RDD is just an RDD composed of tuples. Internally, Spark treats the first value as the key and the rest as the value, essentially converting this into a key-value data structure. As an added benefit, by using the HashPartitioner strategy, the RDD gets partitioned such that each key-value pair’s location is deterministically determined as a function of the key. Since each partition resides on a different machine, it effectively means that the RDD and its transformations (derivations) preserves the partitioning, hence keeping it restricted to the same machine.

We use this observation to construct two Pair RDDs, using the index $m$ as the key – one for the dataset $\{(x^m, y^m)\}$ and another for the associated primal parameters per training example $\{(w^m_{(t)}, b^m_{(t)})\}$. We denote the set of these key-value pairs as $\{m \rightarrow (x^m, y^m)\}$ and $\{m \rightarrow (w^m_{(t)}, b^m_{(t)})\}$. As a result, it is ensured that the training example is always on the same machine as their duals. Notice that dual variables need to be updated in each round. Since RDDs are immutable, we have a different (pair) RDD for the dual variables at each round $t$.

Given this as the premise – the data and dual variables partitioned by the index and distributed on multiple machines – we now dig deeper into how a single round of CoDBcfw looks like. Figure 4.1 describes this flow in detail. Note that all the operations except the reduce, requires no communication and so are performed in parallel by the worker machines. Take for instance,
4.3. Implementation Details

the `mapValues` transformation which produces the updated set of dual variables like in Algorithm 2 line 4. Initially, we begin with a set of indexed output produced by CoDBcfw (which are partitioned as previously mentioned). For all $m$, each machine has access to all the information it needs to compute the updated set locally. As a result, Spark respects the partitioning and implements the `mapValues` function without shuffling or involving any inter-machine communication. Notice that the same applies to all the operations seen in the figure, except the `reduce`. The `reduce` step does the update as seen in Algorithm 2 line 6. Using this, each machine communicates a single weight vector to the master, which merges these updates to produce the next global weight vector $w(t)$ used in the next round by all the workers.

Inference caching

As `dissolveStruct` is based on Bcfw, which supports approximate oracles, we include the feature to reuse previously violated constraints (i.e., labels from previous decodings). The idea is similar to the caching technique in the 1-Slack Cutting Plane algorithm [12]. Since the maximization oracle is very expensive, it is useful to reduce the number of calls made. We reduce it by reusing the previously decoded labels. Through inference caching, we store the latest $c$ (typically 10) loss-augmented decodings $\hat{y}^m$ per training example $(x^m, y^m)$. Let us denote the cache for training example $m$ as $C^m$. Hence, in each iteration, we first query the cache for a good-quality decoding.

$$\hat{y}^m = \arg \max_{y \in C^m} H^m(y, w)$$

We determine the quality of a decoding by calculating the optimal step-size obtained using line-search. If step-size is non-positive, the algorithm will call
the max-oracle for a new decoding.

In terms of implementation, we store an additional RDD \( \{ m \mapsto C^m \} \), which is once again partitioned based on the index. Hence, the cache for the \( m \)-th training example is co-located with the respective dual variables and data. For \textsc{LocalBcfw} to have access to this data, we merely perform an additional join with this RDD with minimal overhead.

4.4 Library Usage Guide

dissolve\textsc{struct} is an open source library under the Apache 2.0 license and the code is publicly hosted on the Internet\textsuperscript{1}. The relevant documentation and usage guide is found on the dissolve\textsc{struct} website\textsuperscript{2}. Although the website serves as the up-to-date and detailed guide, this section aims to summarize the basic usage.

**Platform Requirements**

- Linux or Mac OS X
- Scala 2.10
- Scala Build Tool (sbt) (Tested on 0.13.8)
- Apache Spark (Tested on 1.4.1)

**Package overview**

The source code of dissolve\textsc{struct} is composed of three separate packages, with the contents as described as follows:

1. **dissolve-struct-lib**: Source code for the implementation as seen in the previous section.
2. **dissolve-struct-examples**: Applications of dissolve\textsc{struct} to solve problems from various domains, ranging from Binary classification to text OCR. More of this will be covered in the next section.
3. **dissolve-struct-application**: A starter-kit/template provided to help bootstrap the user to a new application. This contains the necessary build files and skeleton code for an application tagged with placeholders and comments to guide the user.

**Obtaining the library**

To obtain the library, the user has three options.

1. Clone the source code and locally build the library.
2. Include a dependency on the library in the sbt build file as:

   ```
   libraryDependencies += "ch.ethz.dalab" % "dissolvestruct" % "0.1"
   ```

\textsuperscript{1}\url{https://github.com/dalab/dissolve-struct}
\textsuperscript{2}\url{http://dalab.github.io/dissolve-struct/}
3. Directly use the packaged binary jar file found on the releases page.³

Using the library

Once the library is visible in the application classpath, the user needs to implement four functions specified by the trait (analogous to a Java interface) `ch.ethz.dalab.dissolve.optimization.DissolveFunctions`. These functions are provided in Listing 4.1

```scala
trait DissolveFunctions[X, Y] extends Serializable {
  def featureFn(x: X, y: Y): Vector[Double]
  def lossFn(yPredicted: Y, yTruth: Y): Double
  def oracleFn(model: StructSVMModel[X, Y], x: X, y: Y): Y
  def predictFn(model: StructSVMModel[X, Y], x: X): Y
}
```

Listing 4.1: DissolveFunctions interface

These correspond to their mathematical equivalents:

1. `featureFn`: The joint feature map \( \phi(x, y) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^d \), which embeds the complex input-output pair into a Euclidean space.sub-problem
2. `lossFn`: The structured error term \( \Delta(y, y^m) : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_{\geq 0} \), which measures the penalty incurred by predicting a label \( y \) instead of \( y^m \).
3. `oracleFn`: The Maximization Oracle \( H^m(y; w) : \mathcal{X} \times \mathcal{Y} \times \mathbb{R}^d \rightarrow \mathcal{Y} \), which solves the loss-augmented decoding sub-problem.
4. `predictFn`: The discriminant function \( F(x; w) : \mathcal{X} \times \mathbb{R}^d \rightarrow \mathcal{Y} \), which predicts the most compatible label for the given input and parameters. This is optional, since the behavior as the same as the max-oracle with the zero loss.

Equipped with these application-specific functions (say in a singleton object `DSApp`), the user can run CoDBcfw by simply calling it like so:

```scala
val model: StructSVMModel[X, Y] =
  new StructSVMWithDBCFW[X, Y](trainDataRDD,
  DSApp,
  solverOptions).trainModel()
```

where `trainDataRDD` is an RDD of the training examples and `solverOptions` contains parameters for the solver.

For more information, refer to the dissolve-struct website⁴.

---

³https://github.com/dalab/dissolve-struct/releases
⁴http://dalab.github.io/dissolve-struct/
Chapter 5

Approximate Hierarchical Decoding

In Section 3.1.1, we discussed the Structured SVM formulation. We saw that common optimization algorithms are maximization-oracle based, which amounts to solving the follow loss-augmented decoding sub-problem every iteration:

$$H_m(w) = \max_{y \in \mathcal{Y}_m} \Delta^m(y) - \delta F_m(y; w) = \langle w, \psi^m(y) \rangle$$

Alternately, with $F_m(y; w) = \langle w, \phi^m(x^m, y) \rangle$, this can be rewritten as:

$$\hat{y}^m = \arg\max_{y \in \mathcal{Y}_m} \Delta^m(y) + F_m(y; w) = \langle w, \phi^m(y) \rangle$$ (5.1)

Often, (5.1) involves solving an NP-hard problem. For instance, in case of Semantic Image Segmentation, the problem is formulated as solving the maximum a posteriori (MAP) assignment of a 2nd order factor graph. Hence, calling the oracle in each iteration is expensive and this cost often dominates the remaining operations during optimization.

The dependence on the oracle raises two looming problems in such prediction-based solvers. Let us look at this problem in context of the Semantic Image Segmentation application. Firstly, it is impossible to find the exact solution to problem (5.1) because the MAP needs to be performed over a factor graph containing loops, which is intractable. Secondly, the complexity of the problem makes a call to the oracle very expensive.

As a result, one resorts to using approximate oracles for learning. [17] however express the severity of the problem. They show that learning with approximate oracles can fail even with rigorous approximation guarantees in a structured perceptron. This occurs because traditionally, the optimization routines do not have access to the quality of a solution and as a result can misinterpret feedback received from approximate solutions.

In this section, we introduce an approximate-oracle based learning strategy to overcome this problem. Our idea is based on the observation that the Block-coordinate Frank Wolfe algorithm calculates the optimal step-size for
5.1 Hierarchical Decomposition

The main idea behind our approach is defining a sequence of oracle candidates, using a notion of increasing "quality". We will soon see how this is measured.

Notice in Equation (5.1), the domain for the decoding is $\mathcal{Y}(x^m) = \mathcal{Y}^m$ for training example $(x^m, y^m)$. We formalize the above notion by introducing hierarchical decomposability of this domain.

**Definition 5.1 (Hierarchical Decomposability)** Assume $\mathcal{Y}^m$, the space of all possible output labels for $x^m$, be decomposable as

$$\mathcal{Y}^m = \mathcal{Y}^m_1 \cup \mathcal{Y}^m_2 \cup \cdots \cup \mathcal{Y}^m_L$$

The decomposition of the output domains are **hierarchical** if

$$\mathcal{Y}^m_l \subseteq \mathcal{Y}^m_{l+1} \subseteq \mathcal{Y}^m$$

holds. Furthermore, we term $\mathcal{Y}^m_l$ as the **level** $l$ subspace of $\mathcal{Y}^m$.

**Definition 5.2 ($\rho_l$-Maximization Oracle)** Assume $\mathcal{Y}^m$, the space of all possible output labels for $x^m$ be decomposable as

$$\mathcal{Y}^m = \mathcal{Y}^m_1 \cup \mathcal{Y}^m_2 \cup \cdots \cup \mathcal{Y}^m_L$$

A $\rho_l$-Maximization Oracle obtains the best decoding in a subspace of the output domain $\mathcal{Y}^m_l \subseteq \mathcal{Y}^m$ such that

$$H^m_l(w) := \max_{y \in \mathcal{Y}^m_l} \Delta^m(y) - \langle w, \psi^m(y) \rangle$$

Alternatively,

$$H^m_l(w) := \arg\max_{y \in \mathcal{Y}^m_l} \Delta^m(y) + F^m(y; w)$$

**Definition 5.3 (Coarse-to-Fine Maximization Oracle)** Assuming $\mathcal{Y}^m$ is hierarchically decomposable and $\mathcal{Y}^m = \mathcal{Y}^m_1 \cup \mathcal{Y}^m_2 \cup \cdots \cup \mathcal{Y}^m_L$, we define a Coarse-to-Fine Maximization Oracle as an ordered sequence of $\rho_l$-Maximization Oracles

$$C^m(w) := (H^m_1(w), H^m_2(w), \ldots, H^m_L(w))$$

As a consequence of these definitions, we have created a scheme with the following properties

**Increasing sizes of domain** The coarse-to-fine oracle define candidates sequentially in an increasing size of output domains, since $|\mathcal{Y}^m_l| \leq |\mathcal{Y}^m_{l+1}|$. Because of this, we claim that in lower levels it is easier to decode and obtain coarser decodings.
Monotonically increasing quality of max-oracle decodings

**Proposition 5.4** If $Y^m$ can be hierarchically decomposed, we have:

$$\max_{y \in Y^m} F(x^m, y; w) \leq \max_{y \in Y^m_{l+1}} F(x^m, y; w)$$

**Proof** Let us prove by contradiction. Say, this is not the case and hence:

$$\max_{y \in Y^m} F(x^m, y; w) > \max_{y \in Y^m_{l+1}} F(x^m, y; w)$$

$$= \max \left\{ \max_{y \in Y^m} F(x^m, y; w), \max_{y \in Y^m_{l+1}} F(x^m, y; w) \right\}$$

Which is not possible.

As a consequence, we also have:

$$H^m_f(w) \leq H^m_{f+1}(w)$$

Which means, decoding a coarse-to-fine oracle at a higher level always yields a better – or in the worst case, an equivalent – candidate.

### 5.2 HADES-BCFW

**Algorithm 4**: HADES-BCFW: Hierarchical Approximate Decoding using Block-Coordinate Frank-Wolfe algorithm for Structured SVM

**Input**: Data $\mathcal{D} = \{(x^m, y^m)\}_{m=1}^M$

**Initialize**: $w(0) \leftarrow 0$, $w^m(0) \leftarrow 0$, $b(0) \leftarrow 0$, $b^m(0) \leftarrow 0$

1. **for** $t = 1 \ldots T$
   2. Choose $m \in \{1, 2, \ldots, M\}$ uniformly at random
   3. Let $\hat{y} \leftarrow \text{Hades}(x^m, y^m, w_{(t-1)}, b_{(t-1)}, w^m_{(t-1)}, b^m_{(t-1)}, \lambda)$
   4. Let $w_s \leftarrow \frac{1}{M_m} \Psi^m(\hat{y})$ and $b_s \leftarrow \frac{1}{M_m} \Delta^m(\hat{y})$
   5. Let $\gamma \leftarrow \frac{\lambda (w^m_{(t-1)} - w_s) + b^m_{(t-1)} + b_s}{\lambda ||w^m_{(t-1)} - w_s||^2}$ and clip to $[0, 1]$
   6. Update $w^m_{(t)} \leftarrow (1 - \gamma)w^m_{(t-1)} + \gamma w_s$
   7. and $b^m_{(t)} \leftarrow (1 - \gamma) b^m_{(t-1)} + \gamma b_s$
   8. Update $w_{(t)} \leftarrow w_{(t-1)} + w^m_{(t)} - w^m_{(t-1)}$
   9. and $b_{(t)} \leftarrow b_{(t-1)} + b^m_{(t)} - b^m_{(t-1)}$

10. **end**

**Output**: $w_{(T)}$ and $b_{(T)}$
5.3 Interpretation

Procedure B: HADES: Hierarchical Approximate Decoding for a training example

Initialize: \( k \leftarrow 1, \gamma \leftarrow 0 \)

Input: Training example \((x^m, y^m), w, b, w^m, b^m, \lambda\)

1. while \( \gamma < \epsilon \) and \( \gamma^m_k \subseteq Y^m \) do
   2. \( \hat{y}^m_k \leftarrow \text{argmax}_{y \in Y^m_w} \Delta^m(y) + F^m(y; w) \)
   3. Let \( w_s \leftarrow \frac{1}{M} \psi^m(\hat{y}_k) \) and \( b_s \leftarrow \frac{1}{M} \Delta^m(\hat{y}_k) \)
   4. Let \( \gamma \leftarrow \frac{\lambda (w^m - w_s)^T w - b^m + b_s}{\lambda \|w^m - w_s\|^2} \) and clip to \([0, 1]\)
   5. \( k \leftarrow k + 1 \)

Output: \( \hat{y}^m_k \)

Given access to such a coarse-to-fine oracle, we now look into how to learn from approximate decodings. The HADES-BCFW algorithm as seen in Algorithm 4 contains the BCFW variant which works via the coarse-to-fine max-oracle.

The HADES-BCFW algorithm is very similar to BCFW. The main difference being the decoding step, as seen in line 3 of Algorithm 4. Method HADE as seen in Procedure B is responsible for the decoding step.

The key idea of the algorithm is to determine the quality of the decoding, i.e., its capacity to decrease the loss, using the optimal step-size \( \gamma \) calculated by line-search. For some decoding \( y \in Y \), line search minimizes a quadratic objective, which is a function of the current iterate \( a^m \) and the decoding (the search corner). Hence, the magnitude of the step-size is proportional to the quality of the decoding, with the step-size being the highest for the exact decoding at the current iterate. For a ‘bad’ candidate, which results in increasing the objective (rather than decrease it), we expect \( \gamma \leq 0 \).

5.3 Interpretation

5.3.1 Motivation

The main motivation behind this algorithm is with respect to applications in Computer Vision.

Consider the problem of Semantic Image Segmentation in Figure 5.1 in the context of Structured SVMs. Typically, the structure to solve this problem is a Conditional Random Field as we saw in Section 3.2.2. Recall, a CRF consists of variables and factors defined on a graph. The variables refer to disjoint regions (pixels or superpixels) of the image. Each variable takes one of possible \( K \) assignments or labels, such as a “bike”, “car” or “human”. A factor is a function of these variables and evaluates the relation among them. In case of CRF, we have two factors – Unary and Pairwise. Unary
5.3. Interpretation

(a) Image  (b) Ground Truth annotation

Figure 5.1: An example from the Microsoft COCO[21] dataset

factors determine the correspondence between the label and the local image evidence. Pairwise factors capture the penalty for transition between two labels.

To begin, we need to first design this CRF per image, which involves breaking the image down into regions and representing each neighboring region with a pairwise factor. But, what defines a region? As previously discussed in Section 3.2.1, the most simple approach would be to treat each pixel as a region. Given that computing the MAP assignment of a CRF gets quadratically more difficult with more variables and factors, this would become infeasible for even moderately sized images.

Alternatively, we could break the image down into disjoint superpixels, with each superpixel representing a single variable. Therefore, this strategy groups individual decisions into joint decisions. Formally, we constrain the original output space $\mathcal{Y}$ to $\mathcal{Y}^\prime \subset \mathcal{Y}$. $\mathcal{Y}$ refers to an output space, where each pixel in the input image corresponds to a label. Using the SLIC algorithm [1], the number of superpixels per image can be configured. Figure 5.2 displays the effect of this parameter on the image.

However, this raises another dilemma – what should be the size of the superpixels? Ideally, we would want the superpixels defined similar to the annotation, like in Figure 5.1b. But, this is unavailable at prediction time for new images. So on one hand, we could use smaller superpixels which provide finer boundaries, at the expense of the context and making the inference more expensive. On the other hand, we could use bigger superpixels to provide more information within a superpixel, although this could very suboptimal for predicting labels which are defined in a tiny region (Like the purple annotated backpack of Figure 5.1b).

Our motivation behind Hades-Bcfw stems from this observation. The core idea is to use multiple such superpixel sizes to learn from the underlying image. Another way of looking at it is each superpixel assignment in Figure 5.2 corresponds a subspace $\mathcal{Y}^\prime \subset \mathcal{Y}$ (Note that these subspaces are however not hierarchical). Learning from a smaller $\mathcal{Y}^\prime$ provides two benefits. It greatly reduces the inference time, which means we can iterate more quickly. Moreover, it enables us to take advantage of patterns within the structure itself to learn from features encapsulating larger regions.
Figure 5.2: Breaking down the inference problem of image of Figure 5.1 using superpixels. The four images here contain 50, 100, 200 and 300 superpixels respectively.
Chapter 6

Hierarchical Image Segmentation

In the previous section, we looked at Hades-Bcfw, a Bcfw-based algorithm which learns using approximate max-oracle decodings. In this section, we will define the notion of hierarchical subspaces for the task of Semantic Image Segmentation.

6.1 Model sketch

Recall that the decoding problem involves:

\[ \hat{y}_m = \arg \max_{y \in \mathcal{Y}_m} F_m(y; w) \]

and its loss-augmented version:

\[ \hat{y}_m = \arg \max_{y \in \mathcal{Y}_m} \Delta_m(y) + F_m(y; w) \]

Generally, this problem is equivalent to solving the MAP assignment on a Factor graph.

Now, we want to combine the ideas from Hades-Bcfw and apply it to this problem, but in the context of a factor graph. We will specifically look at the Potts Model CRF as the factor graph which was previously described in Section 3.2.2. In order to use the Hades-Bcfw algorithm, we first need to define how to:

1. Decompose the output space \( \mathcal{Y}_m \) of an image \( x_m \) into subspaces.
2. Maintain the notion of hierarchy between these subspaces.

Equipped with this, we still have the problem of computing the MAP assignment on a subspace of the the original output domain. If the above decoding were performed using brute force, the notion of sub-spaces would be enough to perform the approximate inference. But, this is rarely possible due to the combinatorial nature of the problem. Hence, this is modeled as a Factor graph, on which we can run approximate probabilistic inference algorithms (such as Belief Propagation) to compute the MAP assignment. This
6.1. Model sketch

(a) Input $x^m$

(b) Annotation/Output $y^m$

(c) Graph structure representing both $x^m$ and $y^m$

Figure 6.1: Data representation for an input-output pair

then raises two questions: a) how do we use inference algorithms to produce candidates from a subspace $\mathcal{Y}^m \subset \mathcal{Y}^m$; and b) what kind of label configurations belong to a subspace. But, before we step into this, let us take a slight digression to review the problem of Image Segmentation in the context of large-margin Structured Prediction methods.

6.1.1 Recap of Image Segmentation

In this section, we briefly summarize the content previously covered in Section 3.2.2, so as to extend the notation and ideas to our contribution in Section 6.1.2.

Data Representation For the task of Image Segmentation, we are given training examples of images and their respective annotations. Since pixel-based images contain minimal information per-pixel and is computationally inefficient, the images and their annotations are then broken down into superpixels as shown in Figure 6.1a and 6.1b. These superpixels additionally decompose into a graph structure.

The dataset $\mathcal{D} = \{(x^m, y^m)\}_{m=1}^M$ consists of input-output pairs, both treated in terms of their superpixels. Both $x^m$ and $y^m$ decomposes over a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as shown in Figure 6.1c. The data term or nodes corresponding to the superpixel is given by $x^m = \{x^m_i \in \mathbb{R}^d | i \in \mathcal{V}\}$, where $x^m_i$ is a feature vector for the superpixel. Similarly, the output is $y^m = \{y^m_i \in \{1, \ldots, K\} | i \in \mathcal{V}\}$, and $y^m_i$ is the label of the superpixel.
6.1. Model sketch

Figure 6.2: The CRF representation for solving Equation (6.1) for the image in 6.1. The circular nodes represent the random variables $Y_i$, or the variables. The coloring indicates the ground truth annotation (which is during inference). The square nodes represent the two kinds of factors – gray ones indicate the unary factors and blacks indicate pairwise factors.

**CRF and the Maximization Oracle** The loss-augmented decoding sub-problem for Image Segmentation is posed as an Energy Minimization problem of a loss-augmented CRF:

$$H^m(w) := \text{argmin}_{y \in \mathcal{Y}^m} E^m(y; w) - \Delta^m(y)$$  \hspace{1cm} (6.1)

and the discriminant function is just the label configuration which minimizes the Energy:

$$\hat{y}^m := \text{argmin}_{y \in \mathcal{Y}^m} E^m(y; w)$$  \hspace{1cm} (6.2)

where

$$E^m(y; w) = E^U_{i}(y_i; x^m, w) + E^P_{ij}(y_i = y_j; x^m, w) = \sum_{i \in \mathcal{V}} E^U_{i}(Y_i = y_i; x^m, w) + \sum_{(i,j) \in \mathcal{E}} E^P_{ij}(Y_i = y_i, Y_j = y_j; x^m, w)$$

$$= \sum_{i \in \mathcal{V}} \langle w^U_{i}, \phi^U_{i}(x^m, y) \rangle + \sum_{(i,j) \in \mathcal{E}} \langle w^P_{ij}, \phi^P_{ij}(x^m, y) \rangle$$  \hspace{1cm} (6.3)

In the previous section, we also showed that solving Equation (6.1) is equivalent to solving the MAP assignment on a CRF. The CRF for this problem is given in Figure 6.2. However, for the sake of brevity and simplicity of notation, we will use Figure 6.1c to illustrate the CRF and $G = (\mathcal{V}, \mathcal{E})$ to denote the same.

In terms of a factor graph, each term $E^U_{i} : \mathcal{Y}_i \to \mathbb{R}$ in the unary summation is the potential of unary factor (the gray square nodes). Each term $E^P_{ij} : \mathcal{Y}_i \times \mathcal{Y}_j \to \mathbb{R}$ in the pairwise summation is the potential of the pairwise factors (the black square nodes). Notice that the unary factor depends only on a single random variable and the pairwise factor is defined on two random variables.
6.1. Model sketch

(a) Image and its ground-truth annotation
(b) Grouping of nodes (denoted by green boxes)

Figure 6.3: Subspace of $\mathcal{Y}$. The factors have been left out for readability.

6.1.2 Hierarchical Decomposition for Image Segmentation

Given that the max oracle for Image Segmentation is equivalent to solving MAP on the corresponding CRF, we now define hierarchical decomposition of subspaces for the problem. For this, we will first model subspaces $\mathcal{Y}^m_l \subset \mathcal{Y}^m$. Then, we will introduce the notion of hierarchy between successive subspaces.

Subspace of $\mathcal{Y}$

We model a subspace $\mathcal{Y}' \subset \mathcal{Y}$ by constraining multiple random variables in the factor graph to take the same label. In other words, we jointly assign a label to multiple nodes, where the nodes form a subgraph of the original graph $\mathcal{G}' \subset \mathcal{G}$.

For instance, consider this scheme in a previous example, as shown in Figure 6.3. The figure shows a grouping of variables. By jointly assigning a group a single label, all the possible labels on this graph belong to a subspace $\mathcal{Y}' \subset \mathcal{Y}$.

In Figure 6.3, also notice that the grouping shown is suboptimal. Therefore, the ground-truth assignment as seen in Figure 6.3a does not lie in the solution space defined by this subspace. The max oracle hence predicts at best, an approximate decoding for the problem.

Introducing hierarchy

To make the subspaces hierarchical, we do so by making the groupings or joint assignments hierarchical. Consider the groupings in Figure 6.4. Notice that all label configurations in $\mathcal{Y}_l$ through joint assignments are also possible in $\mathcal{Y}_{l+1}$. We will later show techniques on how to obtain these hierarchical groupings.

For terminology, we refer to $\mathcal{Y}_l$ as the $l$-th level subspace of the output domain $\mathcal{Y}$, or sometimes just the $l$-th level. Additionally, we also design these subspaces such that higher levels contain smaller or finer groupings and lower levels contain coarser groupings.
6.2 Surrogate CRF

By creating configurations of labels on groups of variables, rather than on the variables themselves we just established the notion of (hierarchical) subspace for a CRF. Now, we will look at how we use this to construct a Coarse-to-Fine oracle. This oracle by solving the MAP assignment provides the minimum energy configuration in $\mathcal{Y}_l$ at the $l$-th level.

We introduce a general solution by defining CRFs per level, and hence making this technique compatible with any inference procedure. The idea is solving the CRF in a subspace is equivalent to solving a compact CRF in another output domain. We do so by assigning a supernode to each group as shown in Figure 6.5.

In the process, the coarse-to-fine oracle solving the $l$-th level decoding problem reduces to solving a surrogate CRF defined for this level.

Recall, previously our input-output pair $(x^m, y^m)$ was defined over a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ like in Figure 6.4. The energy of some configuration $y \in \mathcal{Y}$ is then given by Equation (6.3):

$$E^m(y; w) = \sum_{i \in \mathcal{V}} E^U_i(Y_i = y_i; x^m, w) + \sum_{(i,j) \in \mathcal{E}} E^P_{ij}(Y_i = y_i, Y_j = y_j; x^m, w)$$

where $E^U_i$ and $E^P_{ij}$ are the unary and pairwise factors respectively, which map an assignment to a real-valued potential. The unary factor is associated with each variable (or a vertex in $\mathcal{V}$), and the pairwise factor is associated with each edge between variables (or an edge in $\mathcal{E}$).

Let us assume access to an $l$-th level grouping on $\mathcal{G}$ (Figure 6.6a) as shown in Figure 6.6b. Let this grouping be represented in form of a graph of
6.2. Surrogate CRF

Figure 6.6: Obtaining a surrogate CRF

supernodes \( \tilde{G}_l = (\tilde{V}_l, \tilde{E}_l) \). This \( l \)-th level “super-graph” is illustrated in Figure 6.6c where vertices in each group are represented using supernodes \( \{u \in \tilde{V}_l\} \). The edges between the groups are represented as supernode-transitions \( \{(u, v) \in \tilde{E}_l\} \) between the supernodes. As a result, based on this grouping we obtain the graph \( \tilde{G}_l = (\tilde{V}_l, \tilde{E}_l) \) shown in Figure 6.6c.

We also establish a node-supernode relation between \( G \) and \( \tilde{G}_l \) by defining the notation as shown in Table 6.1.

**Constraints on \( \tilde{G}_l \)** We assume the following constraints on \( \tilde{G}_l \) with respect to \( G = (V, E) \)

**Assumption 6.1 (No vertex left behind)** \( V = \bigcup_{u \in \tilde{V}_l} \text{atm}_U(u) \)

**Assumption 6.2 (One parent per vertex)** \( \text{atm}_U(u) \cap \text{atm}_U(v) = \emptyset, \forall u, v \in \tilde{V}_l \)

**Assumption 6.3 (No edge left behind)** \( E = \bigcup_{u \in \tilde{V}_l} \text{atms}(u) \cup \bigcup_{(u,v) \in \tilde{E}_l} \text{atm}_P(u) \)

**Assumption 6.4 (Hierarchy)** Consider two graphs \( \tilde{G}_l = (\tilde{V}_l, \tilde{E}_l) \) and \( \tilde{G}_{l+1} = (\tilde{V}_{l+1}, \tilde{E}_{l+1}) \). We assume these are *hierarchical* with \( \tilde{G}_l \) and \( \tilde{G}_{l+1} \) forming a parent-child relation with:

- \( \tilde{V}_{l+1} = \bigcup_{u \in \tilde{V}_l} \text{atm}_U(u) \)
- \( \text{atm}_U(u) \cap \text{atm}_U(v) = \emptyset, \forall u, v \in \tilde{V}_l \)
- \( \tilde{E}_{l+1} = \bigcup_{u \in \tilde{V}_l} \text{atms}(u) \cup \bigcup_{(u,v) \in \tilde{E}_l} \text{atm}_P(u) \)
6.2. Surrogate CRF

<table>
<thead>
<tr>
<th>Notation</th>
<th>Terminology</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>atm(_U(u))</td>
<td>Atoms</td>
<td>{i : i ∈ \mathcal{V}} Set of vertices in (\mathcal{G}) for a supernode (u ∈ \tilde{\mathcal{V}}_l).</td>
</tr>
<tr>
<td>atm(_S(u))</td>
<td>Static transitions</td>
<td>{(i, j) : (i, j) ∈ \mathcal{E}} Set of edges contained within group represented by supernode (u ∈ \tilde{\mathcal{V}}_l).</td>
</tr>
<tr>
<td>atm(_P(u, v))</td>
<td>Supernode transitions</td>
<td>{(i, j) : (i, j) ∈ \mathcal{E}} Set of edges between groups represented by supernodes (u, v ∈ \tilde{\mathcal{V}}_l).</td>
</tr>
<tr>
<td>chi(_U(u))</td>
<td>Children</td>
<td>{i : i ∈ \tilde{\mathcal{V}}_{l+1}} Set of child supernodes in level (l + 1) for supernode (u ∈ \tilde{\mathcal{V}}_l).</td>
</tr>
<tr>
<td>chi(_S(u))</td>
<td>Static transitions</td>
<td>{(i, j) : (i, j) ∈ \tilde{\mathcal{E}}_{l+1}} Set of edges at level (l + 1) contained within group represented by supernode (u ∈ \tilde{\mathcal{V}}_l).</td>
</tr>
<tr>
<td>chi(_P(u, v))</td>
<td>Supernode transitions</td>
<td>{(i, j) : (i, j) ∈ \tilde{\mathcal{E}}_{l+1}} Set of edges at level (l + 1) between groups represented by supernodes (u, v ∈ \tilde{\mathcal{V}}_l).</td>
</tr>
</tbody>
</table>

Table 6.1: Graph Notation

Subgraph interpretation Consider some supernode \(u ∈ \tilde{\mathcal{V}}_l\), which denotes a set of vertices, say \(\mathcal{V}_u ⊂ \mathcal{V}\). Let \(\mathcal{V}_u\) induce a subgraph on \(G_u = (\mathcal{V}_u, \mathcal{E}_u) ∈ \mathcal{G}\). Then, the vertices \(\mathcal{V}_u\) form the atoms, the edges \(\mathcal{E}_u\) within the subgraph form the static transitions and the edges connecting two subgraphs \(\mathcal{V}_u\) and \(\mathcal{V}_v\) are the supernode transitions.

Using the concept discussed previously and the notation in Table 6.1, we can now represent all labels \(y ∈ \mathcal{Y}_l\) compactly using the surrogate graph structure \(\tilde{\mathcal{G}}_l = (\tilde{\mathcal{V}}_l, \tilde{\mathcal{E}}_l)\). An assignment \(Y_u = k, u ∈ \tilde{\mathcal{V}}_l\) implies the joint assignment \(Y_i = k, ∀i ∈ \text{atm}_U(u)\). Additionally, let the super-node of the data term \(\tilde{x}_m = \{\tilde{x}_m^u : u ∈ \tilde{\mathcal{V}}_l\}\) be defined as the sum over the respective data-term atoms:

\[
\tilde{x}_m^u = \sum_{i ∈ \text{atm}_U(u)} x_i^m
\]  

(6.4)

Given this construct, let us define a surrogate CRF for level \(l\) (assuming access to some grouping \(\tilde{\mathcal{G}}_l = (\tilde{\mathcal{V}}_l, \tilde{\mathcal{E}}_l)\)), solving which is equivalent to solving the
6.2. Surrogate CRF

$l$-th level max oracle. Let the energy (potential) of the surrogate CRF be

$$E_l(y; x^m, w) = \sum_{u \in \tilde{V}_l} \tilde{E}^U(y_u = y_{u}; x^m, w) + \sum_{u \in \tilde{V}_l} \tilde{E}^S(y_u = y_{u}; x^m, w) + \sum_{(u,v) \in \tilde{E}_l} \tilde{E}^P(y_u = y_{u}, y_v = y_{v}; x^m, w) \quad (6.5)$$

with unary factors

$$\tilde{E}^U(y_u = y_{u}; x^m, w) = \sum_{i \in \text{atm}_U(u)} E^U_i(y_i = y_{u}; x^m, w) \quad (6.6)$$

and pairwise factors

$$\tilde{E}^P(y_u = y_{u}, y_v = y_{v}; x^m, w) = \sum_{(i,j) \in \text{atm}_P(u,v)} E^P_{ij}(y_i = y_{u}, y_j = y_{v}; x^m, w) \quad (6.8)$$

Because of the construction of the unary feature map, we can rewrite Equation (6.6) as

$$\tilde{E}^U(y_u = y_{u}; x^m, w) = \sum_{i \in \text{atm}_U(u)} E^U_i(y_i = y_{u}; x^m, w) = \sum_{i \in \text{atm}_U(u)} \langle w^U_i, \phi^U_i(x^m, y) \rangle = \sum_{i \in \text{atm}_U(u)} \langle w^U_{y_{u}}, x^m \rangle \quad (6.9)$$

Figure 6.6d depicts the CRF obtained from the above. The gray squares indicate the sum of unary energies $\tilde{E}^U$ and $\tilde{E}^S$, from equations (6.6) and (6.7) respectively. The black squares indicate the pairwise factors $\tilde{E}^P$ from Equation (6.8).

Using these definitions, the decoding sub-problem in a subspace $Y_l$ defined by $\tilde{G}_l = (\tilde{V}_l, \tilde{E}_l)$ amounts to solving:

$$\hat{y}^m = \arg\min_{y \in Y^m} E^m(y; w) = \arg\min_{y \in \tilde{Y}^m} E_l(y; x^m, w) \quad (6.10)$$

**Loss augmented CRF**

Assume a loss function $\Delta : Y \to \mathbb{R}$ which decomposes over the nodes as $\delta$

$$\Delta(y, y^m) = \frac{1}{|V|} \sum_{i \in V} \delta(y_i, y^m_i)$$

Let $h_u^m \in \mathbb{R}^K$ be a vector containing the normalized counts of classes occurring in the group such that

$$h_u^m(k) = \frac{1}{M} \sum_{i \in \text{atm}_U(u)} \mathbb{1}[y_i^m = k]$$
The loss-augmented energy function then distributes over the nodes:

\[ E_l(y; x^m, w) = \sum_{u \in \tilde{V}} \tilde{E}_U(Y_u = y_u; x^m, w) + \sum_{u \in \tilde{V}} \tilde{E}_S(Y_u = y_u; x^m, w) + \sum_{(u, v) \in \tilde{E}} \delta(y_u, y_v)\tilde{h}_{uv}^m(k) \]  

(6.12)

As a result of the new formulation, we obtain a generalized technique to compute the \(l\)-th level max oracle. In summary, we begin with a predefined sequence of hierarchical groupings in the form of surrogate graphs \(\tilde{G}_l\) (shown in Figure 6.6) adhering Assumptions 6.1–6.3. We establish a relation between \(\tilde{G}_l\) and the original graph \(G\), using the notation described in Table 6.1. By doing so, we construct a surrogate CRF for the subspace \(\tilde{Y}^m_l\), solving which is equivalent to solving equations (6.1) or (6.2). The surrogate CRF is defined in equations (6.5) and (6.11), over the output domain \(\tilde{Y}^m_l\). The assignment over this domain \(\tilde{Y}^m_l\) can be expressed in the subspace \(Y^m_l\) without conflicts because of the constraints on \(\tilde{G}_l\).

In the next section, we will cover the reasoning behind this formulation.

6.3 Theory

Proposition 6.5 (Equivalence of CRFs) For all \(w\), we have

\[ \arg\min_{y \in Y^m_l} E(y; x^m, w) \equiv \arg\min_{y \in \tilde{Y}^m_l} E(y; \tilde{x}^m, w) \]

where \(Y^m_l \subseteq Y^m\), \(\tilde{Y}_l\) is the output domain of the surrogate CRF for over some graph \(\tilde{G}_l\) satisfying the assumptions 6.1–6.4.

Proof The input-output pair of training examples are defined on a graph \(G = (V, E)\). Let us create \(q\) disjoint sets from \(V\), to obtain the following properties:

\[ V = \bigcup_{u=1}^q V_u \]  

(6.13)

\[ V_u \cap V_v = \emptyset, \quad V_u, V_v \subseteq V, \quad \forall u, v \in [1, q] \]  

(6.14)

\[ E = \left( \bigcup_{u=1}^q E_u \right) \cup \left( \bigcup_{(u, v) \in \tilde{E}} \tilde{E}_{uv} \right) \]  

(6.15)

\(E_u\) is the edges contained in the vertex-induced subgraph by \(V_u\): \(G(V_u) = (V_u, E_u)\). \(E_{uv}\) is the set of edges connecting subgraphs \(G(V_u)\) and \(G(V_v)\). We term \(E_u\) as static-transitions and \(E_{uv}\) as supernode-transitions.
Notice that these are same as the assumptions stated in Assumptions 6.1–6.3. We will now go ahead and derive the energy definition of the surrogate CRF previously seen in Equation (6.5).

Recall, we had defined the energy of a configuration $y \in \mathcal{Y}$ as the individual sums of the unary and pairwise energies:

$$E_m(y; w) = E_U(y; x^m, w) + E_P(y; x^m, w)$$

$$= \sum_{i \in V} E_{U_i}(Y_i = y_i; x^m_i, w) + \sum_{(i,j) \in E} E_{P_{ij}}(Y_i = y_i, Y_j = y_j; x^m, w)$$

Let us break this down into individual components.

**Unary Energy**

$$E_U(y; x^m, w) = \sum_{i \in V} E_{U_i}(Y_i = y_i; x^m_i, w)$$

- **Definition**
- **Assumption 6.1**
- **Collective assignment**

$$\sum_{i = 1}^{q} \sum_{i \in V_u} E_{U_i}(Y_u = y_u; x^m_i, w)$$

$$= \sum_{i = 1}^{q} \sum_{i \in V_u} \langle w^{U_i}, \phi_{\{u\}}(x^m, y) \rangle$$

$$= \sum_{i = 1}^{q} \sum_{i \in V_u} \langle w_y, x^m \rangle$$

$$= \sum_{i = 1}^{q} \langle w^{y_u}, \sum_{i \in V_u} x^m \rangle$$

$$\hat{x}^m = \sum_{i \in V_u} x^m_i$$

We can represent this energy from the joint assignments in a surrogate CRF with $q$ variables. Assigning a label to a variable $Y_u$ in the factor graph represents the exact unary energy as that of assigning it to the variables in $\{Y_i : i \in V_u\}$.

**Pairwise Energy** Notice that the pairwise energy is the sum of energy over each individual edge:

$$E_P(y; x^m, w) = \sum_{(i,j) \in E} E_{P_{ij}}(Y_i = y_i, Y_j = y_j; x^m, w)$$

As stated in Equation (6.15), let us categorize these energies into static transitions and supernode transitions. Given, for some vertex set $V_u$, the static transitions are given by the edges contained in the vertex-induced subgraph...
6.3. Theory

Figure 6.7: A subgraph interpretation of a surrogate CRF. A supernode \( u \in \tilde{V} \) can be thought of as a subset of vertices \( V_u \subset V \) which induces a graph on \( \mathcal{G} \). The green and yellow edges are termed as static transitions for their respective supernodes and the red edges are termed as supernode transitions.

\( \mathcal{G}(V_u) = (V_u, \mathcal{E}_u) \). In Figure 6.7, this corresponds to the green edges within the group represented as \( u \). The energy over such static transitions is given by:

\[
\tilde{E}^{S}(y; x^m, w) = \sum_{u=1}^{q} \sum_{(i,j) \in \mathcal{E}_u} E_{ij}^{P}(Y_i = y_u; Y_j = y_u; x^m, w) \tag{6.16}
\]

Similarly, for supernode transitions, we have two vertex sets \( V_u \) and \( V_v \), each inducing subgraphs \( \mathcal{G}(V_u) = (V_u, \mathcal{E}_u) \) and \( \mathcal{G}(V_v) = (V_v, \mathcal{E}_v) \). Let \( \mathcal{E}_{uv} \) be the set of edges between \( \mathcal{G}(V_u) \) and \( \mathcal{G}(V_v) \) as shown by the red edges in Figure 6.7. The energy over the supernode transitions then become:

\[
\tilde{E}^{P}(y; x^m, w) = \sum_{\forall u,v} \sum_{(i,j) \in \mathcal{E}_{uv}} E_{ij}^{P}(Y_i = y_u; Y_j = y_v; x^m, w) \tag{6.17}
\]

Equations (6.16) and (6.17) together represent the pairwise energy for some configuration \( y \in \mathcal{Y} \).

Therefore, for some grouping which satisfies equations (6.13), (6.14) and (6.15), the energy takes the same value as in the surrogate CRF.

\[
\begin{align*}
\text{Proposition 6.6 (Equivalent loss augmented decoding)} & \quad \text{For all } w, \text{ we have} \\
& \quad \arg\min_{y \in \mathcal{Y}} E(y; x^m, w) - \Delta(y, y^m) = \arg\min_{y \in \tilde{\mathcal{Y}}} E(y; \tilde{x}^m, w) - \Delta(y, \tilde{y}^m)
\end{align*}
\]

where \( \mathcal{Y}^m \subseteq \mathcal{Y} \), \( \tilde{\mathcal{Y}} \) is the output domain of the surrogate CRF for over some graph \( \tilde{\mathcal{G}} \) satisfying the assumptions 6.1–6.4.
6.3. Theory

**Proof** Let the arbitrary loss function $\Delta$ additively decompose over the nodes using $\delta$, such as Hamming loss. In our case, we assume it to be of the form:

$$\Delta(y, y^m) = \frac{1}{|V|} \sum_{i \in V} \delta(y_i, y_i^m)$$

Hence, we have

$$\Delta(y, y^m) = \frac{1}{|V|} \sum_{u=1}^{q} \sum_{v \in V_u} \delta(y_u, y_u^m)$$

**Definition**

$$\sum_{u=1}^{q} \sum_{v \in V_u} I[y_u^m = k] \delta(y_u, k)$$

**Definition, Joint assignment**

$$\sum_{u=1}^{q} \sum_{v \in V_u} \sum_{k=1}^{K} I[y_u^m = k] \delta(y_u, k)$$

**Rearrangement**

$$\sum_{u=1}^{q} \sum_{v \in V_u} \sum_{k=1}^{K} \delta(y_u, k) h_{u}^m(k)$$

where $h_{u}^m \in \mathbb{R}^K_+$ is a vector containing the normalized counts of classes occurring in the group such that:

$$h_{u}^m(k) = \frac{1}{M} \sum_{i \in V_u} \mathbb{1}[y_i^m = k]$$

As a result, for some assignment $y$, the loss function decomposes also decomposes over the vertex sets $V_u, \forall u$ if

$$y_i = y_{u}, \quad \forall u \in V_u, i \in V, V_u \subseteq V$$

**Proposition 6.7 (Hierarchical Decoding)** In case of hierarchical decoding, for any $w$, we have

$$\min_{y \in \mathcal{Y}_{l+1}} E(y; x, w) \leq \min_{y \in \mathcal{Y}_l} E(y; x, w)$$

**Proof** This can be clearly seen, since:

$$\mathcal{Y}_l \subseteq \mathcal{Y}_{l+1}$$

which holds, since by searching for candidates in the next level, we are increasing the size of the subspace (or lifting some restrictions).

In the case of the our CRF, a level increment induces additional configurations of label assignments, including all assignments at the current level.

An alternate representation is:

$$\min \left\{ \min_{y \in \mathcal{Y}_{l+1}} E(y; x, w), \min_{y \in \mathcal{Y}_l} E(y; x, w) \right\} \leq \min_{y \in \mathcal{Y}_l} E(y; x, w)$$

□
6.4 Analysis

[8] and [29] mention quality of approximation, when using approximate oracles for Structured Prediction tasks. This is given by:

\[ \rho F(\hat{y}; x, w) \leq F(\tilde{y}; x, w) \leq F(\hat{y}; x, w) \]

In our setting (with the Energy minimization task and hierarchical decomposition), this is equivalent to:

\[ E(\hat{y}; x, w) \leq \tilde{E}(\hat{y}; x, w) \leq \rho E(\hat{y}; x, w) \]

We want to go ahead and show that our energy at level \( l \), for some MAP assignment is bounded by some additive factor \( \rho \geq 0 \) of the original minimum energy configuration.

Let us use this shorthand notation for this section:

\[ \hat{E} \leq \hat{E}_l \leq \hat{E} + \rho(l) \]

Recall, we have

\[
E_{\text{min}} = \sum_{i \in V} \langle w_{y_i, x_i} \rangle_{E_{\text{min}}} + \sum_{(i,j) \in E} w_{y_i,y_j}
\]

and

\[
E_l = \sum_{u \in \tilde{V}_l} \sum_{i \in \text{atm}_U(u)} \langle w_{y_i, x_i} \rangle_{E_{\text{min}}} + \sum_{(u,v) \in \tilde{E}_l} \sum_{(i,j) \in \text{atm}_P(u,v)} w_{y_i,y_j} + \sum_{u \in \tilde{V}_l} \sum_{(i,j) \in \text{atm}_S(u)} w_{y_i,y_j}
\]

Let us rewrite \( \hat{E}_l \) as:

\[ \hat{E}_l = \hat{E} + (\hat{E}_l - \hat{E}) \geq 0 \]

6.4.1 Unary-only model

Let:

- \( P = |V| \)
- \( N_l = \text{No. of supernodes at level } l \)
- \( a_l = \text{No. of atoms per supernode, at level } l \)
- each supernode shares a label with at least one of its pixels

As a result of the first three points, we have: \( P = N_l \times a_l \).

We can now write the additive error as:

\[
\hat{E}_l - \hat{E} \leq N_l \cdot (a_l - 1) \cdot 2BR_U
= N_l \cdot \left( \frac{P}{N_l} - 1 \right) \cdot 2BR_U
= (P - N_l) \cdot 2BR_U
\] (6.18)
6.4. Analysis

6.4.2 Potts model

Let

- \( T_l = |\tilde{E}_l| \)
- \( t_l = \) No. of supernode transitions per super-edge
- \( N_l = \) No. of supernodes at level \( l \)
- \( s_l = \) No. of static transitions contained within supernode

Additionally,

- \( Z = \) No. of transitions = \( |E| \)
- \( Z = T_l \cdot t_l + N_l \cdot s_l \)

\[
E^p(y_a, y_b; w^p) - E^p(y_c, y_d; w^p) \leq 2 \cdot B \cdot R_P(y_a, y_b) \neq (y_c, y_d)
\]

- Per super-node, at least a single atom-atom transition is correctly assigned. Hence, no. of “bad” edges per super node = \( s_l - 1 \)
- Per super-edge, at least a single atom-atom transition between supernodes is correctly assigned. Hence, no. of “bad” edges per super-edge = \( t_l - 1 \)

Hence, we have:

\[
\tilde{E}_l - \hat{E}_l \leq N_l \cdot (\frac{P}{N_l} - 1) \cdot 2BR_U + T_l \cdot (t_l - 1) \cdot 2BR_P + N_l \cdot (s_l - 1) \cdot 2BR_P
\]

\[
= N_l \cdot (\frac{P}{N_l} - 1) \cdot 2BR_U + (Z - (T_l + N_l)) \cdot 2BR_P
\]

\[
= (P - N_l) \cdot 2BR_U + (Z - (T_l + N_l)) \cdot 2BR_P
\]

\[
\leq (P - N_l) \cdot 2BR_U + (Z - T_l) \cdot 2BR_P
\]  

(6.19)

Consequently, if we assume a 2D grid structure, we can add at most 2 super-edges to the graph by creating a super-node. Hence, \( T_l \leq 2 \cdot N_l \). As a result, for 2D grid structures, we have:

\[
\tilde{E}_l - \hat{E} \leq (P - N_l) \cdot 2BR_U + (Z - 2N_l) \cdot 2BR_P
\]  

(6.20)

Assuming, we have \( ||x_i|| = \phi^p(y_a, y_b) = 1 \), we get:

\[
\tilde{E}_l - \hat{E} \leq 2B \cdot (P + Z - (N_l + T_l))
\]  

(6.21)

The difference of Energies between MAP assignments of two levels then is:

\[
\hat{E}_i - \hat{E}_{i+1} \leq 2B \cdot ((N_{i+1} - N_l) + (T_{i+1} - T_l))
\]  

(6.22)
In case of the **Quad-tree**, we have:

\[ N_{l+1} = 4 \cdot N_l \]

\[ \Rightarrow T_{l+1} \leq 2 \cdot N_{l+1} = 8 \cdot N_l \]

Consequently,

\[ \hat{E}_l - \tilde{E}_{l+1}^* \leq 18 \cdot N_l \cdot B \quad (6.23) \]
Experimental Results

In this section, we will delve deeper into implementation details of Hades-Bcfw we discussed in Chapter 5. We will apply this technique to construct a coarse-to-fine sequence of Surrogate CRFs from Chapter 6 to solve the problem of Semantic Image Segmentation. The rest of the section focuses on this specific application.

7.1 Implementation Details and Setup

7.1.1 Overview

Hades-Bcfw is developed by extending the Bcfw algorithm in dissolve\textsuperscript{struct}. Building on top of dissolve\textsuperscript{struct} provides benefits which is useful in the current context. Primarily, we are not limited by the size of the problem. As we will see later on, the size of features and training examples can be a limiting factor to perform experiments on a single machine. This enables us to simply create a cluster on EC2 and perform the experiments, instead of requiring special hardware.

Recall the interface provided by dissolve\textsuperscript{struct} for the user to provide the application-specific functions (Listing 4.1). For Hades-Bcfw, the critical change required is to extend the oracle function to return multiple candidates as a sequence. While the algorithm requires these candidates to be in increasing order of quality, we left this constraint to be implemented by the user and hence keeping the interface generic. To use multiple candidates, we extend the previous interface to additionally accept a Stream of candidates as shown in Listing B.1. By using a Scala Stream we perform the expensive decodings lazily, or only when required. Unlike an Iterator, the values in Stream remains computed and can be reused. Hence, the oracle can compute a candidate with access to all previous decodings.

For the task of Semantic Image Segmentation, we also provide a generic implementation. The main ingredient required to use it is a Quantize trait as shown in Listing B.2. The QLevel corresponds to a data structure representing the relation between $\tilde{G}_l = (\tilde{V}_l, \tilde{E}_l)$ and $G = (V, E)$ for some level $l$. 
7.1. Implementation Details and Setup

7.1.2 Inference

The main task is to learn the optimal parameters \( \hat{w} \) of a Conditional Random Field. To do this, we need to perform a loss-augmented inference step (computing the MAP assignment) every iteration for some parameters \( w \) of the CRF.

It is extremely common for the Image Segmentation tasks to require the MAP to be performed on a grid-like CRF containing loops. This makes computing the MAP assignment NP-hard. To solve this problem, people generally use approximate inference techniques such as Loopy Belief Propagation or Mean Field.

We perform Loopy Belief Propagation to compute the MAP assignment on a CRF using the Factorie [26] library.

7.1.3 Features

In Section 3.2.2, we briefly discussed the feature representation. The main focus was on how to embed the complex structured object pair \((x^m, y^m)\) in an Euclidean space. This was a result of the unary and pairwise feature functions:

\[
\phi_U(x, y) = \sum_{i \in V} \phi_{ij}^U(x, y)
\]
\[
\phi_P(x, y) = \sum_{(i, j) \in E} \phi_{ij}^P(x, y)
\]

Now we will look into what constitutes these features.

Data Features

The data-dependent feature \( \phi_{ij}^U(x, y) \) \( \in \mathbb{R}^d \) jointly represents the feature vector \( x^m \) and the label \( y^m\) \( \in \{1, \ldots, K\} \) for some vertex \( i \) (region or superpixel). Now we focus on how we obtain the \( x^m_i \) \( \in \mathbb{R}^d \) vectors for each superpixel.

Throughout the thesis, we experimented with multiple features per superpixel:

1. **Local Color Histograms**: An \( 8^3 \) histogram capturing each of RGB intensities using 8 levels.

2. **Global HOG features**: A 5376-dimensional vector containing global Histogram of Gradient (HOG) features [6] for an image \( x^m \). Each such global vector is concatenated with the local data term \( x^m_i \) like in [22]. This feature was constructed by using [13] toolbox, which is based on [19] and [32].

3. **Context**: To incorporate context for a superpixel, we additionally concatenate the sum of the local feature vectors of its neighbors to \( x^m_i \).

4. **CNN-based features**: Using Deep Learning features has become popular recently, attributed to its performance. The CNN-based features are extracted per image from various layers, using OverFeat [34]. Each superpixel is then represented as a combination of the features per window interpolated to its closest superpixel.
7.2. Models

Transitional Features
The transitional feature map measures the penalty paid to transition from a label \( y_{m_i} \) to \( y_{m_j} \). The idea behind this feature is to learn the interactions between labels – which is central to using a CRF.

Previously, we looked at a data-independent model which assigned the same weight for this transition irrespective of the data terms \( x_{m_i} \) and \( x_{m_j} \). This results in maintaining a matrix \( w^p \in \mathbb{R}^{K \times K} \), where \( w^p_{ij} \) denotes the weight for the transition \( y_{m_i} \rightarrow y_{m_j} \).

Although this works in practice, we extended this by maintaining one such matrix for different ranges of edge intensity and orientation.

Edge intensity Given two assignments \( y_{m_i} \) to \( y_{m_j} \), where \( (i, j) \in E \), we calculate edge intensity as \( \|e_i - e_j\| \), where \( e_i, e_j \in \mathbb{R}^3 \) are the individual intensities represented as vectors in the CIELab color space. For each transition, we bin these values into 10 equally sized ranges.

Orientation Once again, given two assignments \( y_{m_i} \) to \( y_{m_j} \), where \( (i, j) \in E \), we classify the orientation as one of: top-bottom, bottom-top and lateral (left-right and right-left). This is done by maintaining centroids of the superpixels \( c_i, c_j \in \mathbb{R}^2 \) in the Euclidean space and assigning the orientation accordingly with respect to a horizontal line passing through \( c_i \) and parallel to the horizontal axis of the image.

7.1.4 Experimental Setup
All the experiments presented in this section have been run on Amazon EC2. We mainly used two different configurations depending on the usage.

Distributed Uses the Spark standalone mode with 1 master instance (m3.xlarge – 4 cores and 15 GB memory) with 4 slave instances (m3.large – 2 cores and 7.5 GB memory). Since each worker instance has 2 cores, we partition our RDDs into 16 partitions (maintaining at least 2 partitions per core).

Single machine Uses the Spark local mode with 1 instance (c3.2xlarge – 8 cores and 15 GB memory), containing both the driver and executors reside on the same machine. This is especially useful to run experiments without using the network and to obtain experimental metrics from the executors. In this case, our RDDs are partitioned to use 6 cores and thus maintaining a total of 12 partitions.

7.2 Models
We encountered and conceptualized various models throughout the thesis to reflect experimental data. These models are a combination formed out of the following categories.
7.2. Models

7.2.1 CRF

The core idea for solving the Semantic Image Segmentation problem lies in learning the weights for a CRF. Here, we explore the two types of CRF models to learn.

**Unaries and Pairwise**  This is the traditional Potts model CRF, as has been extensively discussed so far.

**Unaries only**  We consider only the unary features and ignore the interactions. Doing so exposes two interesting properties. Firstly, this gives rise to exact decodings (wherein the MAP assignment is a unique solution). Secondly, the decodings are more than 200x faster and vary very little with levels.

7.2.2 Hierarchies

Recall, we defined surrogate CRFs defined on a graph $\tilde{G}_l = (\tilde{V}_l, \tilde{E}_l)$ for some level $l$ in Section 6.2. Additionally, this was implemented via an extended oracle function in dissolve which provided a stream of coarse-to-fine decodings. Now we discuss various schemes to construct this hierarchy which adheres assumptions 6.1–6.4.

**Quadtree**  For the Quadtree approach, we first represent the images as a grid of $10 \times 10$-sized square superpixels. In the first level, we split the entire image into four equally sized quadrants containing their respective superpixels. We recursively construct the next level splitting each of these areas into four more quadrants.

**Hierarchical SLIC**  The Hierarchical SLIC is simply a hierarchy constructed with SLIC superpixels forming the leaves. Using SLIC allows us to solve a problem which works well with boundaries. To create the hierarchy we first create the SLIC superpixels as a graph. Then we proceed to agglomeratively cluster them in a bottom-top clustering manner. In each level from the bottom, we eliminate half the number of superpixels by grouping adjacent vertices based on a distance metric. The distance metric used is a combination of – the edge intensity in CIELab color space and the number of superpixels in a supernode. The latter is used to maintain uniform sizes among the supernodes.

7.2.3 Level schedule

The Hades-Bcfw algorithm (Algorithm 4) as outlined in Section 5.2 performs optimization on a set of datapoints. In this optimization procedure, the Bcfw algorithm was extended to search for candidates from a series of subspaces based on a simple scheme – whether the step-size using line-search is slightly positive ($\gamma > \epsilon$) for the current subspace. If this condition was not met, the Hade strategy increased the size of the subspace (i.e., proceeded to the next level). Ideally this the level is expected to increase once Bcfw cannot optimize any more on the current level using the existing search corners. This might
Table 7.1: Model Nomenclature, as discussed in 7.2. The default models are marked with a †.

<table>
<thead>
<tr>
<th>CRF</th>
<th>Unaries &amp; Pairwise</th>
<th>Unaries only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchy</td>
<td>Hierarchical SLIC</td>
<td>HSLIC†</td>
</tr>
<tr>
<td></td>
<td>Quad Tree</td>
<td>QUAD</td>
</tr>
<tr>
<td>Level schedule</td>
<td>Fixed Schedule</td>
<td>FIXS</td>
</tr>
<tr>
<td></td>
<td>Stub Repetitions</td>
<td>STUBR</td>
</tr>
<tr>
<td></td>
<td>Resume Previous Level</td>
<td>RPL</td>
</tr>
</tbody>
</table>

however be slow in practice and therefore, we will discuss some additional strategies.

**Bare** The Naive strategy works as described previously by checking if step-size is slightly positive ($\gamma > \epsilon$). In experiments, we set $\epsilon = 2.2204 \times 10^{-16}$, which the minimum distance between two floating point numbers that can be typically represented by a machine.

**$\epsilon$-threshold** This schedule varies the value of $\epsilon$ which was discussed in the previous paragraph.

**Fixed Schedule** A Fixed Schedule for levels is of the form: $l = \lfloor k^m / \nu \rfloor$, where $k^m$ is the number of times this training example has been decoded and $\nu > 0$ is a parameter to the schedule

**Resume Previous Level** The expected trend of levels encountered with respect to time is expected to increase. This is only natural, since the optimization strategy cannot decrease the objective (and yield a descent) in the same subspace. As a result, in the latter part of the optimization strategy, an example can be unnecessarily and repeatedly decoded at coarser levels. Using this strategy, the decoding resumes at the level of the previous decoding.

**Stub Repetitions** Often, the max-oracle yields the same decoding multiple times with an $\epsilon$-positive step-size. This happens due to the optimization procedure re-optimizing the duals in terms of the existing corners, albeit with a smaller step-size. Stub Repetitions maintains a cache of $n$ previous decodings (10 in our case). If a decoding has already been suggested, the search continues over the next level.
7.3 Evaluation

7.3.1 The MSRC-21 Dataset

The MSRC-21\cite{40} is a popular dataset used for the task of Semantic Image Segmentation. It consists of 591 examples across 21 categories, split into 45% training, 10% validation and 45% test sets. Each example contains an image and a ground-truth annotation as shown in 7.1. The images contain many real-world objects from both indoor and outdoor scenes. The dataset contains these in either a $320 \times 213$ landscape or $213 \times 320$ portrait format.

The ground-truth annotations were hand-segmented and are often imperfect. Moreover, many pixels are labeled as ‘void’ to denote unavailability. These pixels are ignored during training and testing.

![Figure 7.1: The MSRC-21 dataset. Top row displays the images and the bottom row the respective ground-truth annotations](image)

During training, recall that the image is fragmented into superpixels. Our loss function $\Delta(y^{m}, y)$ measures the penalty between the labels of two superpixels and the superpixel often contain multiple pixel-based labels. Hence, each superpixel is attributed with the ground-truth label in the following way. If multiple pixel-level labels exist within a superpixel, the majority label is considered. Suppose the majority label is ‘void’ and another label covers more than a quarter of the pixels, the second candidate is used instead.

7.3.2 Evaluation Metrics

**Quantitative evaluation** The results are quantitatively evaluated using per-pixel accuracy for each class and is summarized as an average per-class accuracy (denoted as Average). Additionally, we also collect the global per-pixel accuracy (denoted as Global). Unless otherwise mentioned, this section by default reports the global accuracy.

**Performance evaluation** Considering the decoding time is lesser with HADES-BcFW due to approximate inference, an important criteria is performance. To
7.4 Key Results

In this section, we will walk through the main results obtained during the course of the thesis. First, we evaluate the performance of HADES-Bcfw.

Considering the numerous combinations of possible with the models and parameters, we restrict the experiments with HADES-Bcfw applied to SLIC superpixels (called “HSLIC”). In HSLIC, we use a 6-level coarse-to-fine hierarchy with the superpixels present on the final level. As a baseline, we use the same application on the final level (called “SLIC”). This yields the traditional approach to optimizing the parameters for a Potts model CRF.

We also restrict ourselves to analyze the algorithm in terms of training accuracy for majority of the section. The validation set was used only for tuning the parameters.

verify this, we plot the accuracy vs. time curve to analyze the learning rate.

7.3.3 Baselines

We compare our methods to [9], [22], [39], [11] and [33] to evaluate the quantitative accuracy. To realize if any speed-up is obtained, the performance graphs of HADES-Bcfw is compared to the Bcfw algorithm on the same set of superpixels and tested in the same experimental setup.

7.3.4 Parameters

Parameters to the experiment are provided through a configuration file. The default one used is displayed in appendix and is listed in Table 7.2.

Table 7.2: Default experimental parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoDBcfw</td>
<td></td>
</tr>
<tr>
<td>$\lambda$</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>Duration</td>
<td>2 hours</td>
</tr>
<tr>
<td>$f$ (No. of examples sampled per round)</td>
<td>0.4</td>
</tr>
<tr>
<td>Weighted averaging</td>
<td>Enabled</td>
</tr>
<tr>
<td>HSLIC</td>
<td></td>
</tr>
<tr>
<td>No. of superpixels</td>
<td>200–400</td>
</tr>
<tr>
<td>No. of levels</td>
<td>6</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>$2.2204 \times 10^{-16}$</td>
</tr>
<tr>
<td>Inference</td>
<td></td>
</tr>
<tr>
<td>Loopy BP iterations</td>
<td>20</td>
</tr>
<tr>
<td>Features</td>
<td></td>
</tr>
<tr>
<td>Unary feature type</td>
<td>OverFeat CNN Layers 4, 12 &amp; 19</td>
</tr>
<tr>
<td>Edge intensity difference bins</td>
<td>10</td>
</tr>
<tr>
<td>Orientation bins</td>
<td>3</td>
</tr>
</tbody>
</table>
7.4. Key Results

7.4.1 Performance

Performance of HSLIC

In order to demonstrate the performance of HSLIC, we first analyze the Bare (BARE) model. In this model, we use the coarse-to-fine hierarchy as described in Section 6.2. The optimization method is Algorithm 4 with the HADE procedure (B). If the procedure cannot optimize further with respect to the current level, it decodes the CRF at the next level. The following experiments were executed using the default parameters as described in Table 7.2.

![Graphs showing accuracy vs. time and passes for BARE vs. SLIC and SUC vs. HSUC](image)

**Figure 7.2: Comparing Accuracy of HSLIC**

The accuracy results are presented in Figure 7.2. The graph presents the global accuracy over time and passes. The accuracy vs. time graph presents insight on how quick the algorithm learns.

We noticed that the BARE model obtains 75% training accuracy in 26.03 ± 1.3 seconds, compared to 55.08 ± 1.57 seconds using the traditional approach. A similar trend can also be noticed in the validation accuracy. However, the returns of this speed-up diminishes in the latter part of the training where the BARE model learns slowly. We realized this happens due to the algorithm re-optimizing on the visited corners of the search domain. In other words, the algorithm re-optimizes on the current level, obtains a small satisfactory step-size and hence a minor descent (when compared to what could be achieved on the next level).
In terms of the accuracy compared to the number of passes of the dataset, BARE is suboptimal per round. This is only natural, because the SLIC model spends more time decoding per training example. At this expense, BARE completes almost 4x the number of passes as SLIC in the same amount of time as seen in Figure 7.3.

Looking at the performance warrants the question if the hierarchy is used for decoding. On a partial dataset (30%) of the training, we obtained the level schedule implicitly used by HADE by default (step-size is higher than $\epsilon$ at the current level). This is shown in Figure 7.4. Notice that even with such a simple scheme, the algorithm utilizes the hierarchy. Moreover, there is a clear increase in the levels (or the finesse) or the CRF as the algorithm gets closer to the optimal estimate of the parameters.

There is also a clear correlation between the levels, step-sizes and the energy of an assignment with respect to time. Consider Figure 7.6a to analyze this.
7.4. Key Results

![Graphs showing step-size and energy over time for SLIC and HSLIC models.](image)

**Figure 7.6: Step-size and Energy**

in more detail. Let us focus on two observations for training example 1.4.\_s for HSLIC. Firstly, notice an inverted sawtooth pattern in the step step sizes – the step sizes decrease over time and resume at a higher step-size. Secondly, these patterns are correlated in the same way with the energy as shown in Figure 7.6b. The reason for the jumps can be clearly seen in relation to the levels described earlier in Figure 7.4. They occur simply because the procedure optimizes on candidates from the next level.

**Performance of various models**

Previously, we discussed the performance of the HSLIC BARE model. It was seen that it optimizes the objective twice as fast as the traditional SLIC model using coarse decodings to reach a 75% accuracy. This can be mainly attributed to the coarse-to-fine approximate decodings contributing to a satisfactory quality, but unable to get the best decodings using fine-grained decodings of SLIC in the latter part.

However, notice in Figure 7.2 the BARE model only marginally improves the training accuracy beyond the 100 seconds mark, albeit with many additional passes over the dataset. In contrast, the SLIC model achieves a higher increase in accuracy in fewer passes beyond this mark, but with more ex-
7.4. Key Results

Pensive decodings. We hypothesize this is because sometimes fewer “finer” decodings results in minimizing the objective more than numerous “coarse” decodings. As a result, five additional models were created using some insights obtained during the course of experiments. These models were previously described in Section 7.2.3. The training performance (accuracy) of these models is shown in Figure 7.7. The level schedules are shown in Figure 7.7.

$\epsilon$-threshold The key idea behind this is to parametrize the notion of quality for each level. Recall the quality is determined by the optimal step-size calculated using the approximate candidate. A candidate from a current level is used for an update only if it was slightly positive, i.e., greater than $\epsilon$. The $\epsilon$-threshold parametrizes the minimum quality $\epsilon$ required by a decoding on a level. The level schedule obtained using this rule for three parameters is shown in Figure 7.7a. It can be clearly seen that increasing $\epsilon$ forces finer decodings earlier during training. In terms of performance, this model failed to display an improvement over the BARE model, as can be seen in Figure 7.8f.

Fixed Schedule Given that approximate decodings do contribute towards learning, another avenue to explore is by introducing a fixed schedule. In this case, the $\epsilon$ criteria seen previously is dropped in favor of a fixed level schedule. The level schedule is then calculated as $l = \lfloor k^m / \nu \rfloor$ and clipped to $[0, l_{\text{max}}]$. Here, $k^m$ refers to the completed number of decodings for the $m$-th training example and $\nu$ is the parameter. Alternately, one can think of this as obtaining a decoding at a finer level after every $\nu$ passes through the dataset. The level schedule can be seen in Figure 7.7b. The training accuracy is shown in Figure 7.8b. It can be noticed that even a relaxed schedule (like $\nu = 75$) enabled reaching the same accuracy as SLIC in the same time.

Stub Repetitions The maximization oracle can often return the same decoding consecutively, contributing to an $\epsilon$ step-size. This is more pronounced as the parameter $w$ reaches close to the optimal. As a result, we introduced a cache to store the previous 10 decodings. A cache hit for a level then triggers decoding at the next level. We found that this helped in the later part of the training, where BARE tended to slow-down by marginally re-optimizing on the same decodings. This is observed in Figure 7.8c after $10^2$ seconds.

Resume Previous Level This model is based on two observations. Firstly, HADES-BCFW decodes on higher levels when the parameter $w$ reaches close to the optimal solution. As a result, decoding on level $l + 1$ entails unnecessary decodings between levels $0 \rightarrow l$. Secondly, we failed to notice a non-monotonic property with respect to the levels. In other words, fine decodings were sometimes followed by coarser ones (for the same training example), albeit with a smaller step-size. To implement this model, we associate and store each training example with the level of the latest decoding. In the following iterations, the decoding is resumed from this level and hence sidestepping low step-sizes and repeated decodings at some level. The level schedule followed by this model is presented in Figure 7.7d.
7.4. Key Results

Figure 7.7: Various models designed to improve the BARE model
Figure 7.8: Performance of various models
7.4. Key Results

**STUBR + RPL** The most suitable schedule found was a combination of stubbing the repetitions (STUBR) and resuming from the previous level (RPL). We found that on the MSRC-21 data, this yielded as fast a learning rate as the BARE model in the beginning and yet reached the same accuracy as SLIC in the given time. Figure 7.8e shows the training accuracy and 7.7e the level schedule.

**To Pairwise or not to Pairwise?**

In all the previous discussions and models, we specifically looked at the Potts model. This simply refers to a CRF which incorporates (a) the influence of the data $x_i^m$ on a label $y_i^m$; and (b) the interactions between two labels $y_i^m$. The complexity of the decoding arises mainly due to the latter. The interactions introduce many loops in the factor graph, making the MAP problem intractable (although this can be solved using approximate methods).

Now, we will shift our focus on a Unaries-only approach ‘UNA’ and compare it to the Potts pairwise model ‘UPW’ by first looking at the non-hierarchical SLIC approach. From our experiments, we found the UNA model outperform the UPW model in terms of performance as well as accuracy on the training set. This can be seen in Figure 7.9 where UNA obtained a 95% accuracy and UPW 88% on training. However, in the validation set, UPW surpassed UNA by 3% (73% and 76%) because UNA was over-fitting on the training dataset.

![Figure 7.9: Comparison of the Potts model with a Unaries-only model](image)

In comparison, we did not find a contrasting difference between the HSLIC UNA and UPW models. We speculate this being because pairwise features were not strong enough to greatly influence the accuracy, but the approximate decodings were relatively quick to provide the same learning speed. We will soon show that the model did however learn the pairwise weights.

Given this comparable performance between UNA and UPW warrants the question whether learning interactions justifies the complexity of decoding. It is well-known that leveraging context as a part of unary features introduces a side effect – it reduces the significance of the pairwise term. In our case,
7.4. Key Results

(a) SLIC : $37.58 \leq \|e_i - e_j\| < 75.12$

(b) SLIC : $75.12 \leq \|e_i - e_j\| < 112.76$

(c) HSLIC : $37.58 \leq \|e_i - e_j\| < 75.12$

(d) HSLIC : $75.12 \leq \|e_i - e_j\| < 112.76$

Figure 7.10

the unary features did include features from a larger context around the region and as a result, we did not significantly benefit by learning transitions between the labels. Furthermore, the MSRC-21 dataset contains imperfect ground-truth wherein the annotated objects are often separated by ‘void’ regions like in Figure 7.1. This causes the pairwise features to underperform. However, we now proceed to show despite this, the interactions learned by the model exhibits patterns observed in the data.

Recall the CRF parameters $w$ are just a concatenation of the unaries $w^U \in \mathbb{R}^{K \times d}$ and pairwise parameters $w^P \in \mathbb{R}^{K \times K}$. The pairwise parameters $w^P_{ij}$ contains the score for transitioning from $y_i^m$ to $y_j^m$, given their data terms $x_{ii}^m$ and $x_{jj}^m$. This score further depends on the difference in intensity between the data terms $0 \leq \|e_i - e_j\| \leq 375.863$ and the orientation between their respective centroids $c_i$ and $c_j$. In both cases, we discretize the range with 10 bins in case of the edge intensity difference and 3 bins in case of orientation.

A visualization of the pairwise scores is presented in Figure 7.10. We specif-
7.5. Additional results

**7.5.1 Influence of parameters**

Earlier in Table 7.2 we had seen the numerous parameters associated with our setup. In this section, we will discuss the influence of a few important parameters, with the rest available in the appendix.

**Features**

Feature engineering played a crucial role during this thesis. This can considered to be all together a different research topic. However, we explored various options and in the end obtained a satisfactory score comparable to many recent works.

Figure 7.11 presents the accuracy of various unary and pairwise features used during the course of this work. A brief description of the features were described earlier in Section 7.1.3 and keywords in Table 7.3.

Using solely RGB histogram features $x_i^m \in \mathbb{R}^{256}$ we were able to achieve a 40% accuracy on both the training and validation dataset. This is still considerably better than labeling the pixels at random, which yields an accuracy of $1/21 \approx 5\%$. Additionally, notice that concatenating this along with the histograms of its neighbors increases the accuracy by 5\%. Next, Let us observe the effect of global HOG features and OverFeat features extracted on layers 4 and 12 (which are interpolated and assigned to the nearest superpixel).
Table 7.3: Description of Features

<table>
<thead>
<tr>
<th>Feature type</th>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary</td>
<td>RGB</td>
<td>RGB Histogram</td>
</tr>
<tr>
<td></td>
<td>CTX</td>
<td>Context of adjacent neighbors</td>
</tr>
<tr>
<td></td>
<td>DL4</td>
<td>Overfeat Layer 4</td>
</tr>
<tr>
<td></td>
<td>DL12</td>
<td>Overfeat Layer 12</td>
</tr>
<tr>
<td></td>
<td>GLOB</td>
<td>Per-image HOG features</td>
</tr>
<tr>
<td></td>
<td>CNN</td>
<td>Overfeat combination of layers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4, 12 and 19 interpolated to closest superpixel</td>
</tr>
<tr>
<td>Pairwise</td>
<td>OCCUR</td>
<td>Occurrence</td>
</tr>
<tr>
<td></td>
<td>ORNT</td>
<td>Orientation</td>
</tr>
<tr>
<td></td>
<td>EID</td>
<td>Edge Intensity</td>
</tr>
</tbody>
</table>

Figure 7.11: Influence of features on SLIC and HSLIC
7.5. Additional results

Figure 7.12: Studying influence of costly oracles, by increasing the number of iterations performed as a part of Loopy Belief Propagation

(a) 20 iterations

(b) 40 iterations

(c) 60 iterations

This further increased the accuracy by another 6%. We noticed a slight 1-2% increase when used in conjunction with pairwise features.

We achieved the biggest jump by introducing the new CNN features as described in sec 7.1.3. Although the improvement achieved by including the pairwise features is marginal, we earlier saw that the pairwise features do indeed learn the interactions. As mentioned previously, this is a side effect of the unary term representing features from a larger context.

BP iterations

As we saw previously, the time spent computing the MAP assignment often dominates the cost compared to all other operations in max-oracle based solvers. Therefore, it is interesting to evaluate the performance of HSLIC when compared to a range of costly max-oracles.

In our case, we can achieve this by configuring additional iterations for Loopy Belief Propagation on the coarse-to-fine CRFs. As a result, we obtained the graphs as shown in Figure 7.12. We observed no change in the best accuracy obtained on the validation set beyond 20 iterations in SLIC and HSLIC. To reach a 75% accuracy, HSLIC required 24, 28 and 28 seconds respectively for 20, 40 and 80 iterations. In contrast, SLIC required 56, 106 and 129 seconds. This is not surprising since HSLIC spends this time performing these costly
7.5. Additional results

Figure 7.13: Effect of using a Cache

iterations on coarser levels (smaller CRFs) at the start as we saw earlier in
the level vs. time graph. This further testifies our approach in cases where
max-oracles are expensive.

Oracle cache

It is legitimate to ask whether such a coarse-to-fine scheme is required to
perform cheaper decodings, or if instead we can work with a simpler scheme.
In order to do this, we compare and evaluate the HSLIC algorithm with the
cache implementation as described in Section 4.3.1. Recall the oracle cache
stores the previous $n$ (10 in our case) max-oracle decodings for each training
example. In each iteration, if the best candidate in the cache is considered
satisfactory (once again determined if the step-size is greater than $\epsilon$), it is
reused instead of requesting the max-oracle for a new candidate.

The results obtained as a part of this experiment is presented in Figure 7.13.
We found that using the cache in both cases of SLIC and HSLIC resulted in
completing more passes over the data. In SLIC, we found that the solver was
able to complete as much as 5x the number of passes over the dataset (400
in SLIC-CACHE compared to 80) in the same time. We observed a similar
result in BARE, with the cache implementation completing 1.8x the number
of passes (480 in BARE-CACHE compared to 260).

In terms of performance, we however failed to notice any benefits of using
a cache with SLIC. We believe the criteria for selecting the cache candidate
using this strategy is too naive to yield a good step-size for the current iterate.
Moreover with BARE-CACHE, we observed a deterioration in performance
when the cache was introduced. This goes to show that the BARE model is
more suitable for the task of approximate decoding in the current context.
Chapter 8

Conclusion

In the thesis, we were motivated to learn from approximate oracles in the context of large margin structured output prediction. Max-oracle based solvers require an expensive “decoding” step – solving a maximum a posteriori (MAP) assignment for a factor graph with many loops – in each iteration. Given that this is often the bottleneck during training, we proposed a Block-Coordinate Frank-Wolfe based algorithm HADES-Bcfw. This algorithm searches for suitable candidates to the original decoding problem from a series of increasingly complex subspaces of the original output domain. We present this algorithm and the notion of coarse-to-fine candidates for the task of natural scene segmentation. The sequence of candidates is presented as a series of surrogate Conditional Random Fields (CRF) to the original problem. Solving each such CRF yields a label configuration (or segmentation) with the minimum energy in some subspace or “level”. We further show that the energy monotonically decreases with levels and further analyze the additive approximation quality between them.

We evaluate our proposed approach using the MSRC-21 dataset within a scalable framework dissolvestruct. In our experiments, we show that the technique is capable of learning through approximate decodings. As a side-effect, the algorithm is able to make 4x the number of passes and learn 2x as fast to reach an accuracy of 75%. Furthermore, we presented five other extensions to this approach by modeling the level schedule. As a result, we found a model which was able to reach the same accuracy as SLIC in the same amount of time, but reaching a satisfactory accuracy of 75% twice as fast. We believe this will be further beneficial on larger datasets, which is frustrating for practitioners due time it takes to train.
### Structured SVM

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{D} = {(x^m, y^m)}_{m=1}^{M}$</td>
<td>Labeled training examples</td>
</tr>
<tr>
<td>$x^m, y^m$</td>
<td>Input/output structured objects</td>
</tr>
<tr>
<td>$\mathcal{X}^m, \mathcal{Y}^m = \mathcal{Y}(x^m)$</td>
<td>Structured input/output domain for $(x^m, y^m)$</td>
</tr>
<tr>
<td>$\phi(x^m, y^m)$</td>
<td>Joint feature map</td>
</tr>
<tr>
<td>$\psi(y; x^m, y^m) = \phi(x^m, y^m) - \phi(x^m, y)$</td>
<td>Difference in Joint feature map</td>
</tr>
<tr>
<td>$\Delta(y^m, y)$</td>
<td>Structured error function</td>
</tr>
<tr>
<td>$F_w(x, y) = F(x, y; w) = \langle w, \phi(x, y) \rangle$</td>
<td>Discriminant/Utility function</td>
</tr>
<tr>
<td>$h_w(x) = \arg\max_{y \in \mathcal{Y}} F_w(x, y)$</td>
<td>Hypothesis/Prediction function</td>
</tr>
<tr>
<td>$H^m(w) = H(w; x^m, y^m)$</td>
<td>Maximization Oracle</td>
</tr>
<tr>
<td>$\hat{y}$</td>
<td>Optimal prediction</td>
</tr>
</tbody>
</table>
# Image Segmentation

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{G} = (\mathcal{V}, \mathcal{E}) )</td>
<td>Graph representation for an image</td>
</tr>
<tr>
<td>( x_i \in \mathbb{R}^d )</td>
<td>Feature vector for superpixel indexed by ( i \in \mathcal{V} )</td>
</tr>
<tr>
<td>( y_i \in {1, \ldots, K} )</td>
<td>Label for superpixel indexed by ( i \in \mathcal{V} )</td>
</tr>
<tr>
<td>( \delta(y_i, y'_i) )</td>
<td>Loss per label s.t. ( \Delta(y, y') = \sum_{i \in \mathcal{V}} \delta(y_i, y'_i) )</td>
</tr>
<tr>
<td>( Y_i )</td>
<td>Random Variable for superpixel label indexed by ( i \in \mathcal{V} )</td>
</tr>
<tr>
<td>( w^U, w^P )</td>
<td>Parameters for unary and pairwise components</td>
</tr>
<tr>
<td>( E(Y = y</td>
<td>X = x; w) \in \mathbb{R} )</td>
</tr>
<tr>
<td>( E^U, E^P )</td>
<td>Individual energies for unary and pairwise components</td>
</tr>
<tr>
<td>( \phi^U(x, y) = \sum_{i \in \mathcal{V}} \phi^U_i(x, y) )</td>
<td>Unary-based joint feature maps for a configuration ( y )</td>
</tr>
<tr>
<td>( \phi^P(x, y) = \sum_{(i,j) \in \mathcal{E}} \phi^P_{ij}(x, y) )</td>
<td>Joint feature maps for a configuration ( y )</td>
</tr>
<tr>
<td>( \phi^U_i(x, y), \phi^P_{ij}(x, y) )</td>
<td>Joint feature map per superpixel</td>
</tr>
<tr>
<td>( \mathcal{Y}_l \in \mathcal{Y} )</td>
<td>“l-th level” subspace of ( \mathcal{Y} )</td>
</tr>
<tr>
<td>( H^m_l(w) )</td>
<td>Max-oracle computed over domain ( \mathcal{Y}_l )</td>
</tr>
<tr>
<td>( \mathcal{G}_l = (\mathcal{V}_l, \mathcal{E}_l) )</td>
<td>Grouping representation for ( \mathcal{G} ) on level ( l )</td>
</tr>
<tr>
<td>( \tilde{\mathcal{Y}}_l )</td>
<td>Compact representation for ( \mathcal{Y}_l \in \mathcal{Y} ) using ( \tilde{\mathcal{G}}_l )</td>
</tr>
<tr>
<td>( E_l(Y = y</td>
<td>X = x; w) )</td>
</tr>
<tr>
<td>( E^U_l(Y_u = y_u</td>
<td>x^m; w) )</td>
</tr>
<tr>
<td>( E^S_l(Y_u = y_u</td>
<td>x^m; w) )</td>
</tr>
<tr>
<td>( E^P_l(Y_u = y_u, Y_v = y_v</td>
<td>x^m; w) )</td>
</tr>
</tbody>
</table>
Appendix B

Interfaces

B.1 Hades-Bcfw

```scala
trait DissolveFunctions[X, Y] extends Serializable {
  def featureFn(x: X, y: Y): Vector[Double]
  def lossFn(yPredicted: Y, yTruth: Y): Double
  def oracleFn(model: StructSVMModel[X, Y], x: X, y: Y): Y =
    oracleCandidateStream(model, x, y).head
  def oracleCandidateStream(model: StructSVMModel[X, Y], x: X, y: Y, initLevel : Int = 0): Stream[Y] =
    oracleFn(model, x, y) #:: Stream.empty
  def predictFn(model: StructSVMModel[X, Y], x: X): Y
}
```

Listing B.1: Extended DissolveFunctions interface

```scala
trait Quantize {
  def hasNext(level: Level, x: QImage, y: QLabel): Boolean
  def getQLevel(level: Level, x: QImage, y: QLabel): QLevel
}
```

Listing B.2: Quantize interface
Appendix C

Reproducing results

In this section, we intend to illustrate how to use the framework and reproduce the results discussed in Chapter 7. We will specifically look at using it for the task of Semantic Image Segmentation. This is provided as a series of steps.

Step 1 - Clone repository

HADES is an open source framework publicly hosted on the internet\(^1\). This can be cloned locally on a desktop as:

```bash
git clone -b deus-ex-machina \\ https://github.com/dalab/msc_tribhuvanesh.git
```

Step 2 - Obtain data

Central to the framework is the MSRC-21 dataset, which is freely available on their website\(^2\).

Besides this, our framework relies on features and superpixel mapping generated using SLIC\(^3\). We provide this using an auxiliary dataset, which contains the following:

1. The superpixels generated using SLIC.
2. A 5-level hierarchy of superpixels, along with their centroid information in the same format as the original superpixels.
3. Global 2x2 HOG features extracted using [13].
4. The deep learning features extracted using OverFeat [34] and indexed to each superpixel in point 1.

Alternately, the data generated in points 1 and 2 can also be obtained through the `helpers/slic_agglomerative/create_b_slic.py` script.

---

\(^1\) [https://github.com/dalab/msc_tribhuvanesh](https://github.com/dalab/msc_tribhuvanesh)


\(^3\) [http://ivrl.epfl.ch/research/superpixels](http://ivrl.epfl.ch/research/superpixels)
Since this auxiliary dataset is extremely large in size (5 Gb) and privately hosted, the link is available on request.

An RDD for the training and validation dataset is created using a combination of the MSRC-21 data, features and additional metadata. The training examples are cast into an RDD of LabeledObject[QImage, QLabel].

**Step 3 - The Quantize interface**

We provide two interfaces to create the superpixel hierarchy – QuadTreeQuantize and SLICQuantize. The latter uses a ‘static’ hierarchy as generated from the previous step. One could also implement the trait Quantize to generate a customized hierarchy.

**Step 4 - Prepare Executable**

The Hierarchical Semantic Image Segmentation code is developed to run using only an implementation of Quantize trait and the dataset as described in Step C. A fat binary executable can be created using the sbt assembly command within the cloned directory.

**Step 5 - Configure Parameters**

The parameters for execution is either provided through – command-line arguments for Spark and a configuration file for HADES. The default configuration parameters is shown in Listing C.

```bash
EXPT_NAME=HADES-DEFAULT
# Quantization strategy
VAR_PHI=hslic
# \Delta loss type : {"hamming","inv_freq"}
PER_LABEL_LOSS=inv_freq
PENALIZE_VOID=false
# Unary features
RGB_CONTEXT=false
OVERFEAT_FEATURES=false
OVERFEAT_CONTEXT_FEATURES=false
GLOBAL_FEATURES=false
# Pairwise features
BP_ITERS=20
EDGE_DIFF_BINS=10
ORIENTATION_BINS=3
DISABLE_PAIRWISE=false
NORMALIZE_PAIRWISE_FEATURES=false
# Additionally log the step-size that would have been taken
# if decoding took
# place on the final level
GAMMA_FINE=false
# Dump each decoding
DEBUG_DECODING=false
# Add bias variable
USE_BIAS=false
# Gamma threshold to step into next level
EPSILON=2.2204E-16
```
Step 6 - Execute and Analyze

Equipped with the data, executable jar and the parameter file ‘config.properties’ visible in the classpath, a job can be submitted using the snippet shown in Listing C.

```
/root/spark/bin/spark-submit \
  --class "ch.ethz.dalab.dem.appv2.HSegRunner" \
  --driver-memory 13G \
  --executor-memory 6G \
  hseg.jar \
  --samplefrac 0.4 \
  --stopcrit time \
  --timelimit 7200 \
  --lambda 0.0001 \
  --debug --debugmul 2 \
  --kwargs input_path=msrc,train=Train.txt,validation=Validation.txt,num_parts=16
```

Running this command produces a CSV file containing the execution statistics and accuracy. Additionally, the decode and level details can be found in the log file. These can be used in conjunction with the iPython notebook workbooks/2015-09-22-report-final to plot the required graphs.
Appendix D

Additional Experiments

**Pairwise weights** In Section 7.4.1, we previously discussed the interactions learned in the Potts Model. However, the focus was on relations between labels whose data-term differed in intensity. Figure D.1 displays the relations learned when the data-terms, i.e., the superpixels which are visually similar. Notice that the diagonals exhibit the strongest relations.

**Regularization** Regularization played a crucial parameter during tuning. We used the parameter which yielded the best results on the validation set. The effect of these parameters on the global accuracy is presented in Figure D.2. Because both SLIC and HSLIC minimize the same objective, notice that the parameters have a similar effect.

**Sample fraction**

Recall the CoDBcfw algorithm from Section 7.4.2. The local procedure LOCAL-Bcfw is responsible for solving the optimization problem on a fraction $f$ of the dataset in each round. Here we explore the effect of this parameter dur-

(a) SLIC : $0 \leq \|e_i - e_j\| < 37.58$

(b) HSLIC : $0 \leq \|e_i - e_j\| < 37.58$

Figure D.1: Spatial relations learned by the Potts model.
ing training. Because the local optimization procedure covers more passes over rounds, it can be seen in Figure D.3 that increasing f achieves higher accuracy in fewer passes.

Figure D.3: Effect of sampling fraction f, i.e., the fraction of training dataset sampled per round for the dual optimization procedure.
Bibliography


Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor’s thesis, Master’s thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

HADES: Hierarchical Approximate Decoding for Structured Prediction

Authored by (in block letters):
Orekondy
Tribhuvanesh

Name(s): First name(s):
OREKONDY TRIBHUVA

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the ‘Citation etiquette’ information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date
28-Sep-2015

Signature(s)
Tribhuvanesh

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.