1

Parallel Graph-Based Semi-Supervised Learning

Jeff Bilmes and Amarnag Subramanya

[a]Department of Electrical Engineering, University of Washington, Seattle, WA
[b]Google Research, Mountain View, CA

1.1 Introduction

Semi-supervised learning (SSL) is the process of training decision functions using small amounts of labeled and relatively large amounts of unlabeled data. In many applications, annotating training data is time-consuming and error prone. Speech recognition is the typical example, which requires large amounts of meticulously annotated speech data (Evermann et al., 2005) to produce an accurate system. In the case of document classification for Internet search, it is not even feasible to accurately annotate a relatively large number of web pages for all categories of potential interest. SSL lends itself as a useful technique in many machine learning applications as one needs only to annotate relatively small amounts of the available data. SSL is related to the problem of *transductive learning* (Vapnik, 1998). In general, a learner is transductive if it is designed for prediction on only a closed data set, where the test set is revealed at training time. In practice, however, transductive learners can be modified to handle unseen data (Sindhwani et al., 2005; Zhu, 2005a). Chapter 25 in (Chapelle et al., 2007) gives a full discussion on the relationship between SSL and transductive learning. In this chapter, SSL refers to the semi-supervised transductive classification problem.

Let $\mathbf{x} \in \mathbf{X}$ denote the input to the decision function (classifier), f, and $\mathrm{y} \in \mathrm{Y}$ denote its output label, i.e., $f : \mathbf{X} \rightarrow \mathrm{Y}$. In most cases $f(\mathbf{x}) = \mathrm{argmax}_{y \in \mathrm{Y}}\, p(y|\mathbf{x})$. In SSL, certain reasonable assumptions are made so that properties of the distribution $p(\mathbf{x})$ (which is available from the unlabeled data sampled from $p(\mathbf{x})$) can influence $p(y|\mathbf{x})$. These assumptions are as follows:

1. *Manifold Assumption* - the data items $\mathbf{x} \in \mathbf{X}$ lie on a low-dimensional manifold

embedded within a high dimensional space. There are two ways to interpret this. First, the data may lie, irrespective of class, on one global low-dimensional manifold embedded within a high dimensional space. Second, the data for each class might lie on its own specific manifold, and the manifolds for different classes might or might not intersect. While the first case is more commonly discussed, in either case the decision boundary between classes can be more accurately determined using not just the labeled but also the unlabeled data.

2. *Smoothness Assumption* - if two points $\mathbf{x}_1$ and $\mathbf{x}_2$ in a high-density region are close based on a given distance measure on $\mathbf{X}$ (which might depend on the manifold), then their corresponding output labels are also likely to be close or identical. Stated differently, a decision boundary between classes will lie in a low-density region. A third way to state this is, if between two points there exists a trajectory that always lies within the same connected high-density region, then the two points will likely have the same label. Here, a high-density region is a subset of $\mathbf{X}$ that has high probability according to $p(\mathbf{x})$. This is sometimes also called the *cluster assumption*.

These assumptions often being essentially true for real-world data is the reason why a large set of semi-supervised learning algorithms work quite well for many applications.

SSL has a long history of previous research. Early work includes methods such as *Self-training* (Scudder, 1965) which involves training the decision function using data annotated during a previous classification run over the unlabeled data. This so called "labeled" data needs to be carefully chosen and/or filtered or else it would amount to adding noise (i.e., incorrectly labeled samples) to the training set. In general, self-training offers no guarantees except under certain conditions (Haffari and Sarkar, 2007). *Co-training* is a related algorithm where one uses two sets of decision functions with one learning from the output of the other and vice versa. The co-training algorithm is one of the more well-studied SSL algorithms (Goldman and Zhou, 2000; Balcan and Blum, 2005). *Expectation-Maximization* (EM) (Dempster et al., 1977; Bilmes, 1998) can also be seen as an SSL algorithm. EM is a general procedure to maximize the likelihood of data given a model with hidden variables and is guaranteed to converge to a local optimum. EM lends itself naturally to SSL as the labels of the unlabeled samples can be treated as missing (or hidden). Examples of algorithms that use EM within a SSL setting include (Hosmer, 1973; Nigam, 2001; McLachlan and Ganesalingam, 1982). Self-training, co-training and EM all make use of the smoothness assumption in one way or another.

Transductive support vector machines (TSVM) (Vapnik, 1998) are based on the premise that the decision boundary must avoid high density regions in the input space (i.e., the low density separation assumption). They are related to support

vector machines (SVM) used for supervised learning. Computing the exact TSVM solution is in general NP hard and a number of approximation algorithms have been proposed (Zhu, 2005a). Gaussian processes with a "null category noise model" is yet another technique for SSL (Lawrence and Jordan, 2005) and are related to TSVMs.

Graph-based SSL algorithms are an important sub-class of SSL techniques that have received much attention in the recent past (Blum and Chawla, 2001; Zhu et al., 2003; Joachims, 2003; Belkin et al., 2005; Corduneanu and Jaakkola, 2003; Tsuda, 2005; Szummer and Jaakkola, 2001; Zhu and Ghahramani, 2002a; Zhu, 2005a; Chapelle et al., 2007; Subramanya and Bilmes, 2008, 2009b,a, 2011). Here one assumes that the data (both labeled and unlabeled) lies on a low-dimensional manifold that may be reasonably approximated by a graph (this constitutes the manifold assumption). Each data sample is represented by a vertex in an edge-weighted graph with the weights providing a measure of similarity between vertices. We discuss graph-based SSL in section 1.3. We refer the reader to Seeger (2000); Zhu (2005a); Chapelle et al. (2007); Blitzer and Zhu (2008); Zhu and Goldberg (2009). for additional discussion regarding SSL in general.

In the present chapter, we discuss the scalability to very large problems sizes (> 100 million nodes) of graph-based SSL algorithms on different types of parallel machines, either shared-memory symmetric multi-processors (SMPs) or distributed computers. A common feature of most graph-based algorithms that we exploit is that their optimization can be expressed as simple and efficient messages passed along edges of the graph (see Figure 1.1). This is also true of a recent graph-based SSL algorithm proposed by the authors, and that seems to perform better than most other graph-based SSL algorithms (Subramanya and Bilmes, 2009a, 2011). For shared-memory symmetric multi-processors (SMP), we propose a simple generic linear-time (in the number of graph nodes) cache-cognizant node ordering heuristic to improve the efficiency of message passing. On distributed computers, we propose a modification of this heuristic that is still linear time and generic, but that is more intelligent regarding the nonuniform memory access on such machines. We test these heuristics on a large semi-supervised learning task consisting of a graph with 120 million nodes, and show that for both a 16-node SMP and a 1000-node distributed computer, significant improvements in machine efficiency can be obtained.

1.2 Scaling SSL to Large Data Sets

As SSL is based on the premise that unlabeled data is easily obtained, and adding large quantities of unlabeled data leads to improved performance[1], it is thus impor-

[1] Note, there are recent exceptions (Nadler et al., 2010) where more data can in fact hurt certain algorithms.

tant that SSL algorithms scale easily to large amounts of (unlabeled) data. In recent times, the degree to which an algorithm scales has become practically synonymous with the ease and efficiency at which it can be parallelized.

In general, previous work has focused more on improving SSL algorithms and less on parallelization. For example, in the case of TSVMs which as stated above are NP hard, early work could only handle a few thousand samples (Bie and Cristianini, 2003). Therefore, Collobert et al. (2006) proposed a method based on the convex-concave procedure (CCCP) to scale TSVMs to larger data set sizes. More recently, Sindhwani and Selvaraj (2006) proposed an efficient implementation of TSVMs with linear kernels suitable for text applications. Current state-of-the-art TSVMs can only handle tens of thousands of samples when using an arbitrary kernel. For example, Karlen et al. (2008) report that for a problem with about 70,000 samples (both labeled and unlabeled included), a CCCP-TSVM took about 42 hours to train.

In the case of graph-based approaches, Delalleau et al. (2005) proposed to create a small graph with a subset of the unlabeled data thereby enabling fast computation. We are not aware, however, of a published principled algorithm to choose such a subset. Garcke and Griebel (2005) proposed the use of sparse grid for semi-supervised learning. The idea was to approximate the function space with a finite basis with sparse grids. While their approach scales linearly in the number of samples, in practice, it only works for relatively low-dimensional (< 20) data. Karlen et al. (2008) solved a graph transduction problem with 650,000 samples using a Neural Network. They make use of standard stochastic gradient techniques to scale the approach. However the lack of convexity of the neural network training objective means that there are no convergence guarantees. Gradient based approaches, moreover, pose other challenges such as the setting of learning rates and convergence criteria. Note that we are not arguing against the general use of stochastic gradient techniques to find optima of non-convex objectives (e.g., the discriminative training of hidden Markov models (HMM), or our own previous work on the semi-supervised training of parametric discriminative classifiers (Malkin et al., 2009)) but when other convex alternatives are available (such as the ones described below) and are suitable for a given application, and when they work well in practice, it may be more prudent to use the convex formulations. To the best of our knowledge, the largest graph-based SSL problem solved to date had about 900,000 samples (including both labeled and unlabeled data) (Tsang and Kwok, 2006). Clearly, this is a fraction of the amount of unlabeled data at our disposal. For example, on the Internet, society creates over 1.6 billion blog posts, 60 billion emails, 2 million photos and 200,000 videos every day (Tomkins, 2008). SSL holds promise to produce a proper and practical taxonomy of this enormous wealth of information.

1.3 Graph-based SSL

In general, graph-based SSL algorithms often have a global objective (see, for example, Section 1.3.2). This objective might even be convex and have an analytic solution that uses matrix inversion (Zhu et al., 2003; Belkin et al., 2005) or eigen-based matrix decomposition (Joachims, 2003), but because of the inherent $O(m^3)$ computation associated with these approaches (where m is the data set size), they are difficult to scale to large problems.

We are interested in graph-based SSL algorithms, however, for the following important reasons:

1. Time and again, and for many applications, they have performed better than most other SSL algorithms in comparative evaluations (see chapter 21 in Chapelle et al. (2007));
2. Most graph-based methods have a convex objective thereby providing convergence guarantees, making them attractive for solving large-scale problems;
3. For most graph-based SSL approaches, optimizing the objective can be achieved via message passing on graphs. Each iteration of the algorithm consists of a set of updates to each graph node. An updated node value is computed based on the node's current value as well as the neighbors' current set of values (see Figure 1.1). Unlike other algorithms such as Transductive SVMs that require a specialized design and implementation, a majority of the graph-based SSL algorithms may be represented within this common framework of message passing with respect to a given graph (further discussion of this framework is given in the next section);
4. The message passing approach to optimizing a graph-based SSL objective will often have its own convergence guarantees. For example, it can sometimes be shown that the simple message passing algorithm linearly converges to the true global optimum of the convex objective (Subramanya and Bilmes, 2011);
5. It is possible (as we show in the present chapter) to derive simple fast heuristics that enable such algorithms to scale to large parallel machines with good machine efficiency.

Graph-based SSL algorithms broadly fall under two categories – those that use the graph structure to spread labels from labeled to unlabeled samples (Szummer and Jaakkola, 2001; Zhu and Ghahramani, 2002a; Baluja et al., 2008) and those that optimize a loss function based on smoothness constraints derived from the graph (Blum and Chawla, 2001; Zhu et al., 2003; Joachims, 2003; Belkin et al., 2005; Corduneanu and Jaakkola, 2003; Tsuda, 2005). These categories, however, are often different only in form rather than in their underlying goal. For example, label propagation (Zhu and Ghahramani, 2002a) and the harmonic functions algo-

rithm (Zhu et al., 2003; Bengio et al., 2007) optimize a similar loss function (Zhu, 2005b; Bengio et al., 2007). Next we describe some of the previous work in graph-based SSL in more detail.

Spectral graph transduction (SGT) (Joachims, 2003) is an approximate solution to the NP-hard normalized cut problem. The use of norm-cut instead of a min-cut (as in (Blum and Chawla, 2001)) ensures that the number of unlabeled samples in each side of the cut is more balanced. SGT requires that one compute the eigen-decomposition of an $m \times m$ matrix which can be challenging for very large data sets (where m is the total number of samples in the data set). *Manifold regularization* (Belkin et al., 2005) proposes a general framework where a parametric loss function is defined over the labeled samples and is regularized by a graph smoothness term defined over both the labeled and unlabeled samples. When the loss function satisfies certain conditions, it can be shown that the representer theorem applies and so the solution is a weighted sum over kernel computations. The goal of the learning process is thus to discover these weights. When the parametric loss function is based on least squares, the approach is referred to as *Laplacian Regularized Least Squares* (LapRLS) (Belkin et al., 2005) and when the loss function is based on hinge loss, the approach is called *Laplacian Support Vector Machines* (LapSVM)) (Belkin et al., 2005). In the case of LapRLS, the weights have a closed form solution which involves inverting an $m \times m$ matrix while in the case of LapSVM, optimization techniques used for SVM training may be used to solve for the weights. In general, it has been observed that LapRLS and LapSVM give similar performance (see chapter 21 in Chapelle et al. (2007)). Note that while LapSVM minimizes hinge loss (over the labeled samples) which is considered more optimal than squared loss for classification, the graph regularizer is still based on squared error.

A majority of the graph-based SSL algorithms discussed above attempt to minimize squared-loss. While squared-loss is optimal under a Gaussian noise model, it is not optimal in the case of classification problems. We discuss more about the relative merits of using a squared-loss based objective in section 1.3.2. Another potential drawback in the case of some graph-based SSL algorithms (Blum and Chawla, 2001; Joachims, 2003; Belkin et al., 2005) is that they assume binary classification tasks and thus require the use of suboptimal (and often computationally expensive) approaches such as one vs. rest to solve multi-class problems. Yet another issue relates to the use of priors – most graph-based SSL algorithms are not capable of tightly integrating priors into their training objective. To address the above issues, in Subramanya and Bilmes (2008, 2009a, 2011), we have proposed a graph-based SSL algorithm based on minimizing *Kullback-Leibler divergence* (KLD) between probability distributions which we call *Measure Propagation*. We discuss this in more detail in Section 1.3.2. Next we describe the two stages of solving a semi-

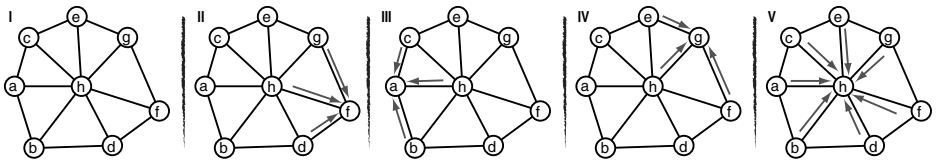

Figure 1.1 I: A graph $G = (V, E)$ with $V = \{a, b, c, d, e, f, g, h\}$ and edges as indicated. II-V: Various messages on the graphs are given. For example, in II, node f is being processed which requires reading information not only from f but also from all of f's neighbors $\Gamma(f) = \{d, g, h\}$ — this essentially constitutes a "message" being sent to node f from its neighbors.

supervised learning problem using graph-based methods – (a) graph construction, (b) the use of the graph as a regularizer to solve the graph-based SSL problem.

1.3.1 Graph Construction

Let $\mathcal{D}_l = \{(\mathbf{x}_i, r_i)\}_{i=1}^{l}$ be the set of labeled samples, $\mathcal{D}_u = \{\mathbf{x}_i\}_{i=l+1}^{l+u}$ the set of unlabeled samples and $\mathcal{D} \triangleq \{\mathcal{D}_l, \mathcal{D}_u\}$. Here r_i is an encoding of the labeled data and will be explained shortly. We are interested in solving the transductive learning problem, i.e., given $\mathcal{D}$, the task is to predict the labels of the samples in $\mathcal{D}_u$ (for inductive extensions, see Subramanya and Bilmes (2009a, 2011)). We are given an undirected weighted graph $\mathcal{G} = (V, E)$, where the vertices (equivalently nodes) $V = \{1, \ldots, m\}$ (with $m = l + u$) represent the data points in $\mathcal{D}$ and the edges $E \subseteq V \times V$ connect related nodes. Let $V_l \cup V_u = V$ be a partition of V where V_l is the set of labeled vertices and V_u the set of unlabeled vertices. $\mathcal{G}$ may be represented via a matrix $\mathbf{W} = \{w_{ij}\}_{i,j}$ with nonnegative values referred to as the weight or affinity matrix. If $w_{ij} > 0$ we say that vertices i and j are adjacent or are neighbors in G. Given a vertex $v \in V$, let $\Gamma(v) \subseteq V$ denote the set of neighbors of the vertex v, and given a set $S \subset V$ let $\Gamma(S) \subseteq V$ be the set of neighbors of nodes in S. Thus, by definition, $\Gamma(v) = \Gamma(\{v\})$ for a single vertex v. For example, in Figure 1.1-I, $\Gamma(a) = \{c, h, b\}$ and $\Gamma(\{a, f\}) = \{c, h, b, d, g\}$. Also $\Gamma(S)$ might include some or all of S when $|S| > 1$ — in the figure, $\Gamma(\{a, b\}) = \{c, h, d, a, b\}$ so $\Gamma(S) \setminus S$ is the set of neighbors of S not including S. In this work, no two vertices have more than one edge between them, and thus $|\Gamma(i)|$ represents vertex i's degree.

There are many ways of constructing a graph for a given data set. In some applications, it might be a natural result of a relationship between the samples in $\mathcal{D}$. For example, consider the case where each vertex represents a web page and the edges represent the links between web pages. In other cases, such as the work of (Wang and Zhang, 2006), the graph is generated by performing an operation

similar to local linear embedding (LLE) with the constraint that the LLE weights are nonnegative. In the majority of applications, including those considered in this chapter, we use k-nearest neighbor (k-NN) graphs. In fact, we make use of symmetric k-NN graphs with edge weights $w_{ij} = [\mathbf{W}]_{ij}$ given by

$$w_{ij} = \begin{cases} \mathrm{sim}(\mathbf{x}_i, \mathbf{x}_j) & \text{if } j \in \mathcal{K}(i) \text{ or } i \in \mathcal{K}(j) \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{K}(i)$ is the set of k-NNs of $\mathbf{x}_i$ ($|\mathcal{K}(i)| = \mathrm{k},\ \forall i$) according to sim, and $\mathrm{sim}(\mathbf{x}_i, \mathbf{x}_j)$ is a measure of similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$ (which are represented by nodes i and j). We note that sim is implicit in $\mathcal{K}(i)$, in that $\mathcal{K}(i)$ contains the k data points nearest to i based on sim. The neighbors function, $\Gamma(i)$, on the other hand, is based on a graph once it has already been constructed.

It is assumed that the similarity measure is symmetric, i.e., $\mathrm{sim}(x, y) = \mathrm{sim}(y, x)$. Further $\mathrm{sim}(x, y) \geq 0$. Choosing the correct similarity measure and k are crucial steps in the success of any graph-based SSL algorithm as it determines the graph. Some popular similarity measures include

$$\mathrm{sim}(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma}} \quad \text{or} \quad \mathrm{sim}(\mathbf{x}_i, \mathbf{x}_j) = \cos(\mathbf{x}_i, \mathbf{x}_j) = \frac{\langle \mathbf{x}_i, \mathbf{x}_j \rangle}{\| \mathbf{x}_i \|_2 \| \mathbf{x}_j \|_2}$$

where $\| \mathbf{x}_i \|_2$ is the L_2 norm, and $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ is the inner product of $\mathbf{x}_i$ and $\mathbf{x}_j$. The first similarity measure is a radial-basis function (RBF) kernel of width σ applied to the squared Euclidean distance while the second is cosine similarity. The choice of $\mathbf{W}$ depends on a number of factors such as whether $\mathbf{x}_i$ is continuous or discrete and characteristics of the problem at hand.

1.3.2 Graph Regularization

For each $i \in V$ and $j \in V_l$, we define multinomial distributions p_i and r_j respectively over the set of classifier outputs Y. That is, for each vertex in the graph, we define a measure p_i, and for each labeled vertex we also define r_i (we explain the reason for including r_i shortly). Here $|\mathrm{Y}| = 2$ yields binary classification while $|\mathrm{Y}| > 2$ yields multi-class classification. Note that $p_i(y)$ represents the probability that the sample represented by vertex i belongs to class y. We assume that there is at least one labeled sample for every class.

We have that $\sum_y p_i(y) = 1$, $p_i(y) \geq 0$, $\sum_y r_i(y) = 1$, and $r_i(y) \geq 0$. Thus p_i and r_i lie within a $|\mathrm{Y}|$-dimensional probability simplex which we depict using $\triangle_{|\mathrm{Y}|}$ and so $p_i, r_i \in \triangle_{|\mathrm{Y}|}$ (henceforth, we abbreviate $\triangle_{|\mathrm{Y}|}$ as $\triangle$). Also let $\mathrm{p} \triangleq (p_1, \ldots, p_m) \in \triangle^m$ denote the set of distributions to be learned, and $\mathrm{r} \triangleq (r_1, \ldots, r_l) \in \triangle^l$ are the set of measures (representing labels, more on this

below) which are given. Here, $\triangle^m \triangleq \triangle \times \ldots \times \triangle$ (the Cartesian product of $\triangle$ repeated m times). Finally let u be the uniform probability measure defined over Y, i.e, $u(\mathrm{y}) = \frac{1}{|\mathrm{Y}|} \; \forall \, \mathrm{y} \in \mathrm{Y}$.

The $\{r_i\}_i$'s represent the labels of the supervised portion of the training data and are derived in one of the following ways: (a) if $\hat{y}_i$ is the single supervised label for input $\mathbf{x}_i$ then $r_i(y) = \delta(y = \hat{y}_i)$, which means that r_i gives unity probability for y equaling the label $\hat{y}_i$; (b) if $\hat{y}_i = \{\hat{y}_i^{(1)}, \ldots, \hat{y}_i^{(t)}\}$, $t \leq |\mathrm{Y}|$ is a *set* of possible outputs for input $\mathbf{x}_i$, meaning an object validly falls into all of the corresponding categories, we set $r_i(y) = (1/t)\delta(y \in \hat{y}_i)$ meaning that r_i is uniform over only the possible categories and zero otherwise; (c) if the labels are given in the form of a set of nonnegative scores, or even a probability distribution itself, we just set r_i to be equal to those scores (possibly) normalized to become a valid probability distribution. Thus, the r_i's can represent various degrees of *label uncertainty*, ranging from completely certain (the label is a single integer) to fairly uncertain (r_i has relatively high entropy), and there can be differing degrees of uncertainty associated with different labels. It is important to distinguish between the classical multi-label problem and the use of uncertainty in r_j. In our case, if there are two nonzero outputs during training as in $r_j(\bar{y}_1), r_j(\bar{y}_2) > 0$, $\bar{y}_1, \bar{y}_2 \in \mathrm{Y}$, it does not imply that the input $\mathbf{x}_j$ is necessarily a member of both of the two corresponding classes. Rather, there is *uncertainty* regarding truth, and we utilize a discrete probability measure over the labels to represent this uncertainty. This can be useful in the document classification task where in the case of a majority of documents, there is an uncertainty associated with the appropriate topic (label) for the document (Subramanya and Bilmes, 2008). To express the alternate case, where an $\mathbf{x}_j$ can be a member of more than one class, we would need multiple binary distributions for each data point — we do not consider this case further in the present chapter.

We define two graph-based SSL objectives, the first uses a squared-error objective while the second makes use of KLD to measure distance between probability distributions at the vertices.

Algorithm Based on Squared Error: Consider the optimization problem $\mathcal{P}_1$: $\min_{\mathrm{p} \in \triangle^m} C_1(\mathrm{p})$ where

$$C_1(\mathrm{p}) = \sum_{i=1}^{l} \parallel r_i - p_i \parallel^2 + \mu \sum_{i=1}^{m} \sum_{j \in \Gamma(i)} w_{ij} \parallel p_i - p_j \parallel^2 + \nu \sum_{i=1}^{m} \parallel p_i - u \parallel^2$$

where $\parallel p \parallel^2 = \sum_y p^2(y)$. $\mathcal{P}_1$ can also be seen as a multi-class extension of the *quadratic cost criterion* (Bengio et al., 2007) or as a variant of one of the objectives in (Zhu and Ghahramani, 2002b; Talukdar and Crammer, 2009).

The goal of the above objective is to find the best set of measures p_i that attempt

to: 1) agree with the labeled data r_j wherever it is available (the first term in C_1); 2) agree with each other when they are close according to a graph (the second graph-regularizer term in C_1); and 3) not be overconfident (the last term in C_1). In essence, SSL on a graph consists of finding a labeling for $\mathcal{D}_u$ that is consistent with both the labels provided in $\mathcal{D}_l$ and the geometry of the data induced by the graph. In this case the error is measured using squared-loss.

$\mathcal{P}_1$ can be reformulated as the following equivalent optimization problem $\mathcal{P}_1$: $\min_{\mathrm{p}\in\triangle^m} C_1(\mathrm{p})$ where

$$C_1(\mathrm{p}) = \mathrm{Tr}\big((S\mathrm{p}-\mathrm{r}')(S\mathrm{p}-\mathrm{r}')^T\big) + 2\mu\mathrm{Tr}(\mathbf{L}\mathrm{p}\mathrm{p}^T) + \nu\mathrm{Tr}((\mathrm{p}-\mathrm{u})(\mathrm{p}-\mathrm{u})^T),$$
$$S \triangleq \begin{pmatrix}\mathbf{I}_l & 0\\ 0 & 0\end{pmatrix}, \quad \mathrm{r}' \triangleq \begin{pmatrix}\mathrm{r} & 0\\ 0 & 0\end{pmatrix}, \quad \mathrm{u} \triangleq (u,\ldots,u) \in\triangle^m,$$

$\mathbf{1}_m \in \mathbb{R}^m$ is a column vector of 1's, and $\mathbf{I}_l$ is the $l \times l$ identity matrix. Here $\mathbf{L} \triangleq \mathbf{D} - \mathbf{W}$ is the unnormalized graph Laplacian, $\mathbf{D}$ is a diagonal matrix given by $d_i = [\mathbf{D}]_{ii} = \sum_j w_{ij}$. C_1 is convex if $\mu, \nu \geq 0$ and, as the constraints that ensure $\mathrm{p} \in\triangle$ are linear, we can make use of the KKT conditions (Bertsekas, 1999) to show that the solution to $\mathcal{P}_1$ is given by

$$\hat{\mathrm{p}} = (S + 2\mu\mathbf{L} + \nu\mathbf{I}_m)^{-1}\left[S\mathrm{r} + \nu\mathrm{u} + \frac{2\mu}{|\mathrm{Y}|}\mathbf{L}\mathbf{1}_m\mathbf{1}_{|Y|}^T\right].$$

Henceforth we refer to the above approach to solving $\mathcal{P}_1$ as *SQ-Loss-C*. The above closed-form solution involves inverting a matrix of size $m \times m$. As a result it may not be amenable to very large problems. In such cases, one may solve $\mathcal{P}_1$ in an iterative fashion. It can be shown that the iterative update for each p_i is given by

$$p_i^{(n)}(y) = \frac{r_i(y)\delta(i \leq l) + \nu u(y) + \mu\sum_j w_{ij}p_j^{(n-1)}(y)}{\delta(i \leq l) + \nu + \mu\sum_j w_{ij}}.$$

Here n is the iteration index. More interestingly, it can be shown that $\mathrm{p}^{(n)} \rightarrow \hat{\mathrm{p}}$ (Bengio et al., 2007). We refer to the iterative approach to solving $\mathcal{P}_1$ as *SQ-Loss-I*.

Measure Propagation: Next we consider a graph regularization framework based on KLD (Subramanya and Bilmes, 2009a, 2011). Consider the optimization problem $\mathcal{P}_2 : \min_{\mathrm{p}\in\triangle^m} C_2(\mathrm{p})$ where

$$C_2(\mathrm{p}) = \sum_{i=1}^{l} D_{KL}\big(r_i||p_i\big) + \mu\sum_{i=1}^{m}\sum_{j\in\Gamma(i)} w_{ij}D_{KL}\big(p_i||p_j\big) - \nu\sum_{i=1}^{n} H(p_i).$$

Here $H(p) = -\sum_{\mathrm{y}} p(\mathrm{y})\log p(\mathrm{y})$ is the Shannon entropy of p and $D_{KL}(p_i||q_j)$ is the KLD between measures p_i and q_j and is given by $D_{KL}(p||q) = \sum_{\mathrm{y}} p(\mathrm{y})\log\frac{p(\mathrm{y})}{q(\mathrm{y})}$.

(μ, ν) are hyper-parameters that can be set via cross-validation. The three terms in $C_2(\mathrm{p})$ have the same purpose as the three terms in $C_1(\mathrm{p})$ but in this case, loss is measured in the KLD sense. We note that C_2 is still convex in p. We solve $C_2(\mathrm{p})$ using alternating minimization (AM) and the updates are given by

$$p_i^{(n)}(y) = \frac{\exp\{\frac{\mu}{\gamma_i}\sum_j w'_{ij}\log q_j^{(n-1)}(y)\}}{\sum_y \exp\{\frac{\mu}{\gamma_i}\sum_j w'_{ij}\log q_j^{(n-1)}(y)\}} \quad \text{and}$$

$$q_i^{(n)}(y) = \frac{r_i(y)\delta(i \le l) + \mu\sum_j w'_{ji} p_j^{(n)}(y)}{\delta(i \le l) + \mu\sum_j w'_{ji}}$$

where $\gamma_i = \nu + \mu\sum_j w'_{ij}$, and where $\mathrm{q}^{(\mathrm{n})} \triangleq (q_1^{(n)}, \ldots, q_m^{(n)}) \in \triangle^m$ is another set of m distributions that are learned simultaneously with p. In Subramanya and Bilmes (2009a, 2011) we show that $\lim_{n\to\infty} D_{KL}\big(q_i^{(n)} || p_i^{(n)}\big) = 0$ for each i, and moreover that they both converge to the minimum of C_1. We call this iterative procedure *measure propagation* (MP).

There are number of reasons to prefer the KLD-based objective to the one defined in terms of squared-error. Firstly, as shown in (Subramanya and Bilmes, 2008, 2009b,a), the KLD-based objective outperforms other squared-loss based approaches on a wide variety of datasets. Secondly, while squared-error has worked well in the case of regression problems (Bishop, 1995),[2] for classification, it is often argued that squared-loss is not the optimal criterion and alternative loss functions such as the cross-entropy (Bishop, 1995), logistic (Ng and Jordan, 2002), hinge-loss (Vapnik, 1998) have been proposed. Thirdly, for measuring the dissimilarity between measures, KLD is said to be asymptotically consistent w.r.t. the underlying probability distributions (Bishop, 1995). Finally, KLD based loss is based on relative error rather than absolute error as in the case of squared-error. Indeed, the results given in Figure 1.2 show that the KLD-based SSL objective significantly outperforms algorithms based on the squared-error objective, and also based on an multi-layered perceptron (MLP) trained using only on the labeled data.

Table 1.1 gives a summary of the update equations for different graph-based algorithms. Note that SQ-Loss-C and SQ-Loss-I are in-fact reformulations of the popular squared-loss based objectives in terms of multinomial distributions. MP, SQ-Loss-I and LP, however, are iterative and in fact correspond precisely to message passing on graphs as depicted in Table 1.1. SQ-Loss-C has a closed form solution that involves inverting an $m \times m$ matrix. In practice, such a matrix will be sparse, but implementing SQ-Loss-C is arguably not as straightforward as the iterative message-passing cases. Moreover, in the message-passing cases, the updates at every vertex are a function of the values of its neighbors and so are quite easy

[2] Assuming a Gaussian noise model in a regression problem leads to an objective based on squared-loss.

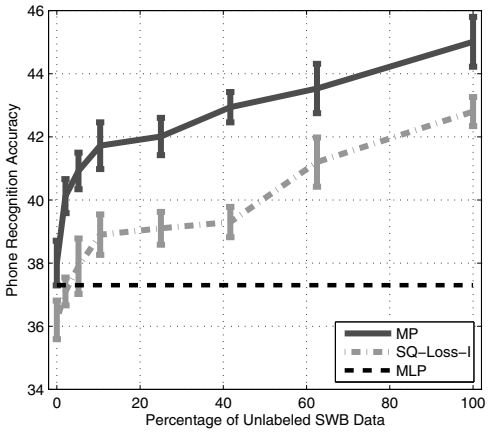

Figure 1.2 Phone Accuracy vs. Percentage of switchboard (SWB) I training data. The STP portion of SWB is fully described in Section 1.4. Phone Accuracy was measured on the STP data. Note that when all the Switchboard I data was added, the resulting graph had 120 million vertices! The dashed black line shows the performance of a multi-layered perceptron (MLP) in the no unlabeled data case, measured using the same training, development and test sets as MP and LP. That is, the MLP here is trained fully supervised, unlike in our other work (Malkin et al., 2009). More details may be found in Subramanya and Bilmes (2011).

to parallelize. We thus turn our attention to how best to parallelize such message passing algorithms.

1.4 Data set: A 120 Million Node Graph

Our interest in this chapter is on fast parallel implementations of message-passing based semi-supervised learning algorithms, and we therefore need a large real-world dataset that is deserving of our efforts.

We therefore utilized the popular speech dataset Switchboard I (SWB) which is a collection of about 2,400 two-sided telephone conversations among 543 speakers (302 male, 241 female) from all areas of the United States (Godfrey et al., 1992). A computer-driven system handled the calls, giving the caller appropriate recorded prompts, selecting and dialing another person (the callee) to take part in a conversation, introducing a topic for discussion and recording the speech from the two subjects into separate channels until the conversation was finished. SWB consists of about 300 hours of speech data and is very popular in the speech recognition community for the training of large vocabulary conversational speech recognition systems (Evermann et al., 2005; Subramanya et al., 2007).

SWB has been annotated in multiple ways. There are manually produced word-

Algorithm	Update Equation(s)
MP (Subramanya and Bilmes, 2009a)	$p_i^{(n)}(y) = \frac{\exp\{\frac{\mu}{\gamma_i}\sum_j w'_{ij}\log q_j^{(n-1)}(y)\}}{\sum_y \exp\{\frac{\mu}{\gamma_i}\sum_j w'_{ij}\log q_j^{(n-1)}(y)\}}$ $q_i^{(n)}(y) = \frac{r_i(y)\delta(i\leq l)+\mu\sum_j w'_{ji}p_j^{(n)}(y)}{\delta(i\leq l)+\mu\sum_j w'_{ji}}$ $\gamma_i = \nu + \mu\sum_j w_{ij}$
SQ-Loss-C	$\hat{\mathrm{p}} = (S + 2\mu\mathbf{L} + \nu\mathbf{I}_m)^{-1}\left[S\mathrm{r} + \nu\mathrm{u} + \frac{2\mu}{\lvert\mathrm{Y}\rvert}\mathbf{L}\mathbf{1}_m\mathbf{1}_c^T\right]$ $\mathbf{L} \triangleq \mathbf{D} - \mathbf{W}, [\mathbf{D}]_{ii} = \sum_j w_{ij}$
SQ-Loss-I	$p_i^{(n)}(y) = \frac{r_i(y)\delta(i\leq l)+\nu u(y)+\mu\sum_j w_{ij}p_j^{(n-1)}(y)}{\delta(i\leq l)+\nu+\mu\sum_j w_{ij}}$
LP (Zhu and Ghahramani, 2002a)	$p_i^{(n)}(y) = \frac{r_i(y)\delta(i\leq l)+\delta(i>l)\sum_j w_{ij}p_j^{(n-1)}(y)}{\delta(i\leq l)+\delta(i>l)\sum_j w_{ij}}$

Table 1.1 *A summary of update equations for various graph-based SSL algorithms. μ and ν are hyper-parameters.*

level transcriptions. In addition, one also has access to less reliable phone-level annotations generated in an automatic manner by a speech recognizer with a nonzero error rate (Deshmukh et al., 1998).

Most interesting from the perspective of SSL, the *Switchboard Transcription Project* (STP) (Greenberg, 1995; Greenberg et al., 1996) was undertaken to accurately annotate SWB at the phonetic and syllabic levels. Within the annotated portion of the STP data, every phone (or syllabic) segment is marked with a temporally high-resolution start- and end-time and a phone (or syllable) identity, and this is done painstakingly by a human annotator. These segments can be used to make a decision regarding the phone identity of every 25ms (millisecond) window of speech, a process known as *frame* annotation. One of the hopes for the STP data was that it could be used to improve the performance of conversational speech recognition systems — each frame label could be used to train an HMM by having a phonetic label determine each state variable at each time frame during training. As the transcription task was time-consuming, costly, and error prone, only 75 minutes of speech selected from different SWB conversations were annotated at the phone level. Completing this annotation task is thus the perfect job for transductive SSL: produce labels for the unlabeled data. Having access to such annotations for all of SWB could potentially be useful for large vocabulary speech recognition and speech research in general and this, in fact, was the process we undertook in (Subramanya and Bilmes, 2009b). This data is also an ideal real-world task for SSL, and accuracy results showed that measure propagation was significantly more accurate

than alternatives on this data (Subramanya and Bilmes, 2009a, 2011). The STP data is useful for a third purpose, as it corresponds to a very large real-world graph (approximately 120 million nodes, corresponding to 120 million speech frames), suitable for developing scalable SSL algorithms, the topic of this chapter (also see Figure 1.2).

The following process was used to construct this graph from the STP data. The speech wave files were first segmented and then windowed using a Hamming window of size 25ms at 100Hz (15ms window overlap). We then extracted 13 Perceptual Linear Prediction (PLP) (Huang et al., 2001) coefficients from these windowed features and appended both deltas and double deltas resulting in a 39-dimensional feature vector. To increase context information, we used a 7 frame context window (3 frames in the past and 3 in the future) yielding a 273-dimensional sample $\mathbf{x}_i$. We used

$$\text{sim}(\mathbf{x}_i, \mathbf{x}_j) = \exp\{-(\mathbf{x}_i - \mathbf{x}_j)^T \Sigma^{-1} (\mathbf{x}_i - \mathbf{x}_j)\}$$

as the similarity measure to generate the weights w_{ij}, where Σ is the "grand" covariance matrix computed using all the data. Our task is the phone classification problem, and so Y is the set of phones (an approximately 40-class multi-class classification problem). We constructed a symmetrized k-NN graph using the above data with each vertex having at least $k = 10$ neighbors using the procedure described in section 1.4.1. The graph, hereinafter referred to as the *SWB graph*, had about 120 million vertices. More details are given in (Subramanya and Bilmes, 2009b,a, 2011).

1.4.1 Graph Construction in large problems

Graph construction is an important step in the use of a graph-based algorithm for solving a SSL problem. At a very high level, the graph determines how information flows from one sample to another and thus an incorrect choice of a neighborhood can lead to poor results. Improper choice of a graph can also lead to degenerate solutions (where either all samples belong to a single class or all samples have a uniform distributions) (Blum and Chawla, 2001; Joachims, 2003). In general, graph construction "is more of an art, than science" (Zhu, 2005b) and is an active research area (Alexandrescu and Kirchhoff, 2007).

Constructing a graph over a large number of samples (> 1 million) itself poses an interesting computational challenge – conventional brute-force construction of k-NN graphs does not scale to large datasets as it is $O(m^2)$. Nearest neighbor search is a well researched problem, however, with many approximate solutions. A large number of solutions to this problem are based on variations of the classic

kd-tree data structure (Friedman et al., 1977). However kd-trees or their derivatives are not ideally suited to the case of high-dimensional data, as we have.

As we have continuous data in this work, we make use of the Approximate Nearest Neighbor (ANN) library (see `http://www.cs.umd.edu/~mount/ANN/`) (Arya and Mount, 1993; Arya et al., 1998). It constructs a modified version of the kd-tree data structure which is then used to query the NNs. The query process requires that one specify an error term, ϵ, and guarantees that $\text{sim}(\mathbf{x}_i, \text{N}(\mathbf{x}_i))/\text{sim}(\mathbf{x}_i, \text{N}_\epsilon(\mathbf{x}_i)) \leq 1+\epsilon$ where $\text{N}(\mathbf{x}_i)$ is a function that returns the exact nearest neighbor of $\mathbf{x}_i$ while $\text{N}_\epsilon(\mathbf{x}_i)$ returns the approximate NN. Larger values of ϵ improve the speed of the nearest neighbor search at the cost of accuracy. For more details, see (Arya and Mount, 1993; Arya et al., 1998). We note that one could also use b-matching (Jebara et al., 2009) to obtain a symmetrized k-NN graph but it is quadratic in the graph size so does not scale well to problem at hand.

1.5 Large-Scale Parallel Processing

In the next few sections, we discuss the scaling up of graph-based message passing algorithms for SSL learning on both a shared-memory computer (the next section), and a distributed memory computer (Section 1.5.3). .

1.5.1 Inference on a Shared-Memory Symmetric Multi-Processor

In an SMP, all processing units share the same address space and are often cache coherent. Such computing environments are important, as individual modern microprocessors are becoming ever more like SMPs, with multi-core (and soon "many core") units being available in a single package.

Recall from Table 1.1 that in the case of most graph-based algorithms the update at each node is a function of the current value of its neighbors. In the case of Measure Propagation (MP), for example, we see that one set of measures is held fixed while the other set is updated without any required communication amongst set members, so there is no write contention. In the case of SQ-Loss-I, the same holds if one considers $\text{p}^{(n-1)}$ as the set of measures that are held fixed while a new separate $\text{p}^{(n)}$ may change. This immediately yields a T-threaded implementation where the graph is evenly but otherwise arbitrarily T-partitioned and each thread operates over only a size $m/T = (l+u)/T$ subset of the graph nodes, the intent being of course that each of the T threads runs in parallel.

We implemented such a multi-threaded application and ran timing tests on the aforementioned graph with about 120 million nodes. We ran a timing test on a 16 core SMP with 128GB of RAM, each core operating at 1.6GHz. We varied the number T of threads from 1 (single threaded) up to 16, in each case running three

iterations of MP (i.e., three each of p and q updates). Each experiment was repeated 10 times, and we measured the minimum CPU time over these 10 runs. CPU time does not include the time taken to load data structures from disk. The speedup for T threads is typically defined as the ratio of time taken for single thread to time taken for T threads. The solid line in Figure 1.4 (a) represents the ideal case (a linear speedup), i.e., when using T threads results in a speedup of T. The pointed line shows the actual speedup of the above procedure, typically less than ideal due to inter-process communication and poor shared L1 and/or L2 microprocessor cache interaction. When $T \leq 4$, the speedup is close to ideal, but for increasing T the algorithm increasingly falls away from the ideal case.

1.5.2 Graph Reordering Algorithm for SMP

We assert that the sub-linear speedup is due to the poor cache cognizance of the algorithm. At a given point in time, suppose thread $t \in \{1, \ldots, T\}$ is operating on node i_t. The collective set of neighbors that are being used by these T threads are $\{\cup_{t=1}^{T} \Gamma(i_t)\}$ and this, along with nodes $\cup_{t=1}^{T}\{i_t\}$ (and all memory for the associated measures), constitute the current *working set*. The working set should be made as small as possible to increase the chance it will fit in any shared machine caches, but this becomes decreasingly likely as T increases since the working set is monotonically increasing with T. Our goal, therefore, is for the nodes that are being simultaneously operated on to have a large amount of neighbor overlap thus minimizing the working set size.

Recall that $\Gamma(v)$ is the set of v's neighbors. Also, $\Gamma(\Gamma(v))$ is the set of v's neighbors' neighbors or equivalently, the neighbors of v's neighbors. For example, in Figure 1.1-I, $\Gamma(a) = \{c, h, b\}$ and $\Gamma(\Gamma(a)) = V$. Note that $v \in \Gamma(\Gamma(v))$ but this will not affect any decisions made by our procedure. For notational simplicity, for a given set $S \subseteq V$, we define $\Gamma^2(S) \triangleq \Gamma(\Gamma(S))$.

Viewed as the optimization problem, our goal is to find a partition of V into a set of clusters $(V_1, V_2, \ldots, V_{m/T})$ that minimizes $\max_{j \in \{1,\ldots,m/T\}} | \cup_{v \in V_j} \Gamma(v)|$, where $|V_i| = T$ and the nodes in V_i are run in parallel. With such a partition, we may also produce an order $\pi = (\pi_1, \ldots, \pi_{m/T})$ of the clusters so that the neighbors of V_{π_i} would have maximal overlap with the neighbors of $V_{\pi_{i+1}}$. We then schedule the clusters according to this order, so that the nodes in V_{π_i} run simultaneously, and which would also act to prefetch many of the neighbors of the nodes in $V_{\pi_{i+1}}$.

The time to produce such a partition, of course, must not dominate the time to run the algorithm itself. Therefore, we propose a simple linear-time (i.e., $O(m)$) *node* ordering procedure (Algorithm 1) that can be run once before the parallelization begins. The algorithm produces a node ordering $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ such that successive nodes are likely to have a high amount of neighbor overlap with each

Algorithm 1: Graph Node Ordering Algorithm Pseudocode, SMP Case

Input: A Graph $G = (V, E)$

Result: A node ordering, by when they are marked.

Select an arbitrary node v ;

while *There are unselected nodes remaining* **do**

  Select an unselected $v' \in \Gamma^2(v)$ that maximizes $|\Gamma(v) \cap \Gamma(v')|$. If the intersection is empty, select an arbitrary unselected v'. ;

  Mark v' as selected.; `// v' is next node in the order`

  $v \leftarrow v'$. ;

other and, by transitivity, with nearby nodes in the ordering. It does this by, given a current node v, choosing as the next node v' in the order (from amongst v's neighbors' neighbors) the one that has the largest number of shared neighbors. We need not search all m nodes for this, since anything other than v's neighbors' neighbors has no overlap with the neighbors of v.

Assuming the nodes are ordered according to σ (something that in practice is done only implicitly), the t^{th} thread operates on nodes $\{t, t+m/T, t+2m/T, \dots\}$. If the threads proceed synchronously at the graph node level (which we do not enforce in our implementation) the set of nodes being processed by multiple processors at time instant j are $V_j = \{1 + jm/T, 2 + jm/T, \dots, T + jm/T\}$. This assignment is beneficial not only for maximizing the set of neighbors being simultaneously used, but also for successive chunks of T nodes since once a chunk of T nodes have been processed, it is likely that many of the neighbors of the next chunk of T nodes will already have been prefetched into the caches. With the graph represented as an adjacency list, and sets of neighbor indices sorted, our algorithm is $O(mk^3)$ in time and linear in memory since the intersection between two sorted lists may be computed in $O(k)$ time. Here k is the number of neighbors of a given vertex in the graph. This can be better than $O(m \log m)$ since $k^3 < \log m$ for very large m.

The utility of this heuristic is depicted in Figure 1.3. On the left, we see a partition that might at first look reasonable, since each cluster consists of neighboring nodes (e.g., V1 consists of nodes $\{c, e\}$) but this clustering is poor since each pair of nodes, in all clusters but one, shares only one neighbor (e.g., c and e in V1 have only one neighbor in common, namely h). On the right, we see a much better clustering, where each pair of nodes in all clusters shares two neighbors (e.g., e and a in V1 share neighbors c and h). In fact, this clustering can result from running Algorithm 1.

We ordered the SWB graph nodes, and ran timing tests using MP as explained

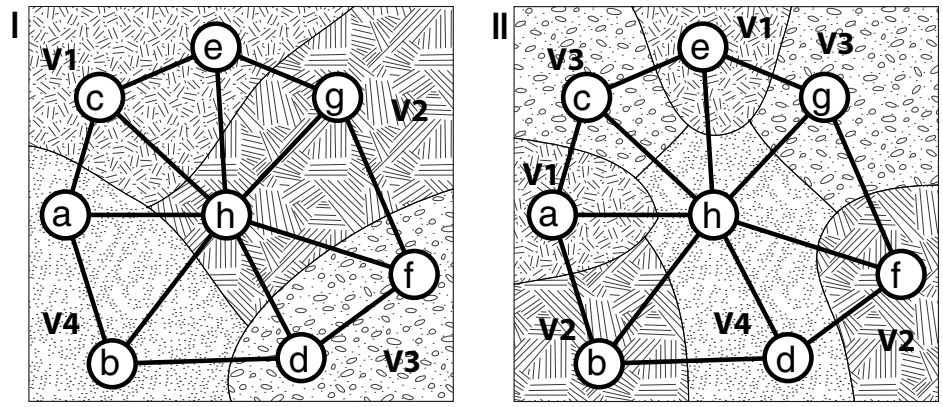

Figure 1.3 I-Left: The graph from Figure 1.1 that has been partitioned into four subsets of nodes (marked as regions V1, V2, V3, and V4), where each subset is to be processed on a separate processor (on a four-processor parallel machine). Within V1, V3, and V4, each node has only one common neighbor with the other node, while in region V2, the two nodes (g and h) have two neighbors in common (e and f). In regions V1, V3, and V4, therefore, reading one node's neighbors will not prefetch as much useful information as in region V2, thereby slowing down the entire parallel computation. II-Right: Here, every node in each region has a neighbor overlap of two with the other node in its region. For example, in region V1, nodes e and a both have as neighbors nodes $\{c, h\}$. Therefore, processing the neighbors of one node in each region will prefetch more of what the other node in a region needs. This region allocation is what Algorithm 1 can produce, when starting at node a — the ordering is (a, e, f, b, c, g, d, h).

above. To be fair, the CPU time required for ordering the nodes by the heuristic is included in every run along with the time for running MP. The results are shown in Figure 1.4(a) (pointed line) where the results are much closer to ideal, and there are no obvious diminishing returns like in the unordered case. Running times are given in Figure 1.4(b). Moreover, the ordered case showed better performance even for a single thread $T = 1$ (the CPU time is about 790 minutes for the ordered case, vs. about 815 minutes for the unordered case, on 2 iterations of MP). The reason for this difference is that the ordering heuristic exhibits better cache behavior even on a single node, since nearby nodes in the order tend to share neighbors (see Figure 1.5).

Finally, we note that since we made use of speech data to generate the graph, it is already naturally well-ordered by time. This is because human speech is a slowly changing signal, so the nodes corresponding to consecutive frames are similar, and can be expected to have similar neighbors. This is confirmed by Figure 1.5, which shows the average neighbor overlap as a function of distance, for a random order, the speech order, and for an ordering produced by our heuristic. Therefore, we ex-

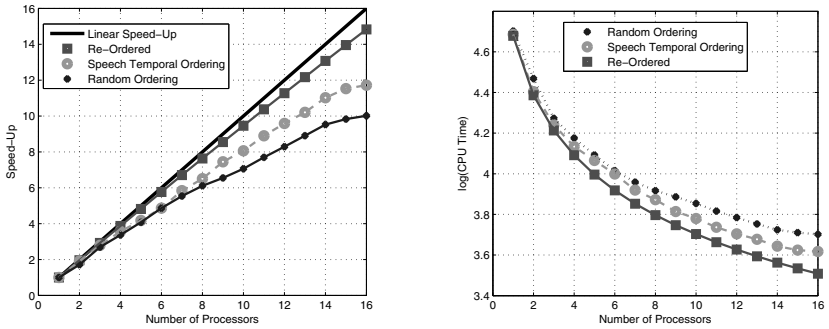

Figure 1.4 (a) speedup vs. number of threads for the SWB graph. The process was run on a 128GB, 16 core machine with each core at 1.6GHz. (b) The actual CPU times in seconds on a *log scale* vs. number of threads for with and without ordering cases. "Random" corresponds to the case where we choose a random unselected node rather than the one with maximum overlap (see Algorithm 1).

pect our "baseline" speech graph to be better than an arbitrary order, one that might be encountered in a different application domain. In order to measure performance for such arbitrarily ordered graphs, we took the original graph and reordered uniformly at random (a uniform node shuffle). As seen in Figure 1.5, the random order had the least degree of neighbor overlap at nearby distances. We ran timing experiments on the resulting graph and the results are shown in Figure 1.4 as "Random". As can be seen, there is indeed a benefit from the speech order, and relative to this random baseline, our node ordering heuristic improves machine efficiency quite significantly.

We conclude this section by noting that: 1) reordering may be considered a preprocessing (offline) step; 2) the SQ-Loss algorithm may also be implemented in a multi-threaded manner and this is supported by our implementation; and 3) our reordering algorithm is general and fast and can be used for any graph-based algorithm where the iterative updates for a given node are a function of its neighbors (i.e., the updates are *harmonic* w.r.t. the graph (Zhu et al., 2003)).

1.5.3 Inference in a Distributed Computing Environment

Our results in the previous section for the SMP show that it is possible to get good efficiency on an SMP using only a simple node ordering heuristic. Unfortunately, an SMP does not scale to tens of thousands (or even thousands) of processing units, as is typical in large distributed computing environments. Distributed computers do not have shared memory, and any communication between them is typically done via a form of messaging library. Such environments, moreover, are likely to be more realistic for the case when one wishes to utilize inexpensive massive

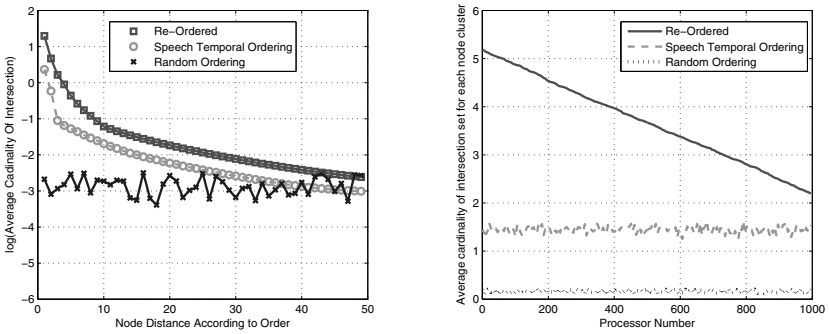

Figure 1.5 Left; Average cardinality of the intersection between neighbors of node i and node $i + k$ where k was varied between 1 and 50 (all using natural logarithm). "Speech Temporal Ordering" is the graph generated from the data, "Re-ordered" is the above graph ordered using Algorithm 1, and "Random Ordering" is a random shuffle of the nodes. Right: With the nodes ordered according to Algorithm 2, and with the clustering as specified in Section 1.5.3, this plot shows the average cardinality of neighbor intersection for successive nodes, as a function of the cluster index (equivalent to processor number). Note that the overall average in this figure is approximately equal to the distance-1 average on the left figure ($\ln(3.5) \approx 1.25$). Note also that, due to the greedy nature of Algorithm 2, the average intersection decreases as the heuristic proceeds.

parallelism. In this section, we see how it is possible to modify our SMP reordering algorithm for the distributed computer case, while retaining its linear time.

On a distributed computer, things are quite different than on an SMP. On each individual distributed computer node, there is still a benefit for successive nodes to have a large amount of neighbor overlap. The reason is that if node i is adjacent to a set of neighbors common with node $i + 1$, node i will essentially prefetch, into microprocessor caches, data useful for node $i + 1$. This is the reason for the single thread speedup exhibited in Section 1.5.2.

Assume we have a T processor distributed computer. We then we want to partition the graph $G = (V, E)$ into T clusters $(V_1 \cup V_2 \cup \cdots \cup V_T)$, where the nodes in V_i are to be run by processor t. Each of the nodes in V_i can thus be ordered locally on each processor using Algorithm 1 to maximize within-processor neighbor overlap to take advantage of the node's local caches. On the other hand, there should be as little communication cross processing elements as possible, to reduce potential waiting time for necessary data. This means that the neighbors of nodes assigned to different processors $|\Gamma(V_i) \cap \Gamma(V_j)|$, for $i \neq j$ should be as small as possible, reducing the chance that any processing element will need to wait for data.

One possible solution to this problem is to perform a minimum T-cut (or T-partition) of the graph, which is a known NP-complete optimization problem (Vazirani, 2001). This, if solved optimally, would produce a graph clustering that min-

Algorithm 2: Graph Node Ordering Algorithm for a Distributed Computer

Input: A Graph $G = (V, E)$ with $m = |V|$ nodes. Parameter T indicating the number of compute nodes. A positive integer threshold τ.

Result: A node ordering, by when they are marked.

Select an arbitrary node v ;
$i \leftarrow 0$;
while *There are unselected nodes remaining* **do**
  if $\min_\ell |i - \ell m/T| < \tau$ **then** `// near a transition`
    Select uniformly at random any unselected node v' ;
  else `// not near a transition`
    Select an unselected $v' \in \Gamma^2(v)$ that maximizes $|\Gamma(v) \cap \Gamma(v')|$. If the intersection is empty, select an arbitrary unselected v'. ;
  Mark v' as selected.; `// v' is next node in the order`
  $v \leftarrow v'$;
  $i \leftarrow i + 1$;
foreach ℓ **do** `// randomly scatter boundary nodes to internal locations`
  Define segment ℓ boundary node indices as $B_\ell = \{i : 0 \le i - \ell m/T < \tau \text{ or } 0 \le (\ell + 1)m/T - i < \tau\}$;
  foreach $i \in B_\ell$ **do**
    Insert node i uniformly at random between nodes $\ell m/T + \tau$ and $(\ell + 1)m/T - \tau$;

imizes the total number of edges that crosses between any two clusters. The good news is that this can be constant-factor ($(2 - 2/T)$) approximated. The bad news, however, is that even this approximation algorithm is too expensive for the large graphs we wish to use (the approximation algorithm requires a Gomory-Hu tree (Vazirani, 2001) which requires computing $|V| - 1$ (s, t)-cuts). Perhaps even more problematic, we have another constraint which is that each cluster should have about the same number of nodes to achieve a good load balance (and thereby, high computer efficiency). The approximation algorithm for T-cut is quite outlier sensitive, and some processors could easily end up with very little work to do. Therefore, we desire a procedure for normalized T-partition, and it needs to run very fast. Performing even normalized 2-partition is NP-complete (Shi and Malik, 2000), however, so we must resort to a heuristic.

Fortunately, our SMP heuristic (Algorithm 1) can be modified to suit the distributed case, and the clues on how to do so lie within Figure 1.5. We see that in the

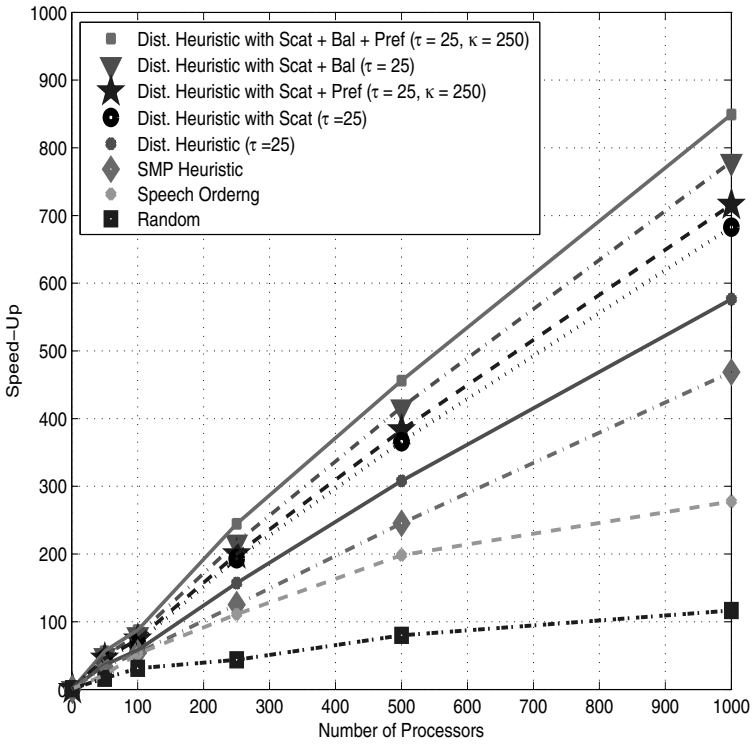

Figure 1.6 Speedup vs. number of processors for the SWB graph.

re-ordered case, successive nodes have many neighbors in common. Due to transitivity, nearby nodes also have neighbors in common. As we move away in distance, the number of neighbors in common decreases. Moreover, the degree of neighbor overlap is much higher than the speech ordering, and also much higher than the random order which, as expected, has very little neighbor overlap. While such a random order might seem to be pointless, we can exploit this property of random orders to ensure that V_i and V_j, for $i \neq j$, has a small neighbor intersection.

Thus, a very simple heuristic is to do the following: Produce a hybrid node ordering heuristic that switches between: A) choosing the next node from the set of neighbors' neighbors, based on maximizing the neighbor overlap, and B) choosing the next node uniformly at random. Under the assumption of uniform load balance (i.e., every processor gets m/T nodes allocated to it, but see below where this assumption no longer holds), the transitions in the order between processors are at locations $\ell m/T$ for $\ell = 0 \ldots T$. Let $R = \{\ell m/T : \ell = 0 \ldots T\}$ be the set of transitions. Let i refer to the current position in the ordering. When i is nowhere near any of the transitions, nearby nodes should have a large degree of neighbor

overlap, but as i approaches one of the transitions (i.e., i is on the "boundary" regions), the amount of neighbor overlap should decrease. We can choose a threshold τ such that if i is within τ nodes of a transition, it is a boundary node, and it should be chosen at random. This is described in Algorithm 2, lines 3 through 10.

We implemented this heuristic on a 1000 node distributed computer. We did not have available detailed information about this computer such as speed and/or topology of the communications network. Therefore, the heuristics we describe in this section are generic, rather than being specific to this (or any) distributed computer. As a result, given more information about the machine, we believe it would be possible to further improve machine efficiency over what our current results for these heuristics show.

Figure 1.6 shows the results. The first thing to note is that a purely random order (marked as "Random") does poorly, as it did for an SMP in Figure 1.4. The random order is what we might expect using an arbitrary order for a given application. When we use the speech temporal order, the situation improves. Further improvement can be acquired by running the SMP heuristic (Algorithm 1) on the distributed computer, marked as "SMP Heuristic" in the figure. As mentioned above, this heuristic makes no attempt to ensure that $|\Gamma(V_i) \cap \Gamma(V_j)|$ is small for $i \neq j$. When we use a heuristic consisting of *just* lines three through ten in Algorithm 2, we see a significant efficiency improvement over the SMP heuristic — this is marked as "Dist. Heuristic" in the figure. The results correspond to a $\tau = 25$ which was obtained empirically, but other values might be better still.

On the other hand, there is still a problem in that all communication on the machine is happening simultaneously. That is, when only lines three through ten are executed, the message passing algorithm has two phases with very different communications network behavior. Phase one is when all processors are operating on non-boundary nodes (and there is very little network communication). Phase two is when all processors are operating on boundary nodes, and the communications network is being flooded with requests for data. To mitigate this problem, we can randomly scatter the boundary nodes onto points internal to a segment, as done in lines 11-14 of Algorithm 2. Performing this scatter results in improved efficiency, as shown in Figure 1.6 "Dist. Heuristic with Scat".

A further improvement can be obtained by taking advantage of the fact that once the scatter has been performed, we still know exactly which nodes are likely to have most or all off their neighbors on processor (i.e., those that have not been scattered), and those that are likely to have most or all of their neighbors off processor (i.e., those that have been scattered). This information can be exploited by prefetching the cross-processor neighbors of the scattered nodes early. This is controlled by parameter κ, which states that scattered node i's off-processor neighbors should be asynchronously prefetched at the time that we are processing node $i - \kappa$. With $\kappa =$

250, this results in further improvements as shown in Figure 1.6 "Dist. Heuristic with Scat + Pref."

Both Algorithm 1 and Algorithm 2 are "greedy" in the sense that they select and then commit to the next node that looks best at the time, without regard to how this decision might adversely affect decisions made later. There are some cases where greedy algorithms are indeed optimal (White, 1986) but we do not expect this to be one of them. Indeed, Figure 1.5-right shows that as the distributed heuristic proceeds, the average cardinality of neighbor intersection (over adjacent nodes in the order, and within each cluster) decreases from about 5.2 at the beginning of the heuristic to about 2.3 at the end. Therefore, processor 0 has a set of nodes with significantly more neighbor overlap on average than processor 1000. While this was not a significant issue on the SMP (with only 16 processors), on a large distributed computer this decreases efficiency significantly. We note that any imbalance between processors is only due, in this case, to the fact that earlier processors will have better locality properties in their local caches. One solution to this problem is for multiple orderings to be deduced simultaneously (and in parallel) along with a collision avoidance scheme and a restart mechanism on reaching a dead end. On the other hand, a simpler solution is to simply give earlier processors more nodes than later processors, thus recovering balance. We have done this and show the results in Figure 1.6 "Dist. Heuristic with Scat + Bal." To achieve balance, we derived a simple re-balance mechanism based on a fraction of the slope of the plot in Figure 1.5-right. This fraction corresponds to how important local cache misses on each processor are relative to their computational workloads and network communication costs. As can be seen, further significant improvements are obtained.

Lastly, prefetching and load balancing can be utilized simultaneously to achieve further gains as shown in Figure 1.6 "Dist. Heuristic with Scat + Bal + Pref." As mentioned above, these heuristics are fairly generic and could be easily exploited on any distributed computer. With more knowledge about the latency and bandwidth of the underlying communications network, we expect that we could improve machine efficiency even further.

1.6 Discussion

In this chapter we have proposed graph node reordering heuristics that make it possible to scale graph-based SSL algorithms to large datasets. We have shown that it is possible to recover a near linear speedup relative to standard order on an SMP and have achieved an 85% efficiency on a 1000-node distributed computer. In (Subramanya and Bilmes, 2009b,a, 2011), we use the above SMP implementation of measure propagation on STP data, and show that it significantly outperforms the only other SSL algorithms that can scale to such large data sets (label propaga-

tion). More importantly, we found that the performance on the STP data improves with the addition of increasing amounts of unlabeled data, and MP seems to get a better benefit with this additional unlabeled data, although even SQ-Loss-I has not reached the point where unlabeled data starts becoming harmful (Nadler et al., 2010). This portends well for large scale semi-supervised learning approaches.

We also wish to point out that since the graph-based SSL algorithms listed in Table 1.1 are all instances of the more general procedure of message passing on graphs (which includes random walks (Woess, 2000), (loopy) belief propagation (Pearl, 1988), affinity propagation (Frey and Dueck, 2007), and others (Baluja et al., 2008)). All of these algorithms could stand to benefit from the simple node ordering algorithms presented in this work.

This work was supported by ONR MURI grant N000140510388, by NSF grant IIS-0093430, by the Companions project (IST programme under EC grant IST-FP6-034434), and by a Microsoft Research Fellowship.

References

Alexandrescu, A., and Kirchhoff, K. 2007. Graph-Based Learning for Statistical Machine Translation. In: *Proc. of the Human Language Technologies Conference (HLT-NAACL)*.

Arya, S., and Mount, D. M. 1993. Approximate nearest neighbor queries in fixed dimensions. In: *ACM-SIAM Symp. on Discrete Algorithms (SODA)*.

Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., and Wu, A. 1998. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*.

Balcan, M.-F., and Blum, A. 2005. A PAC-Style Model for Learning from Labeled and Unlabeled Data. Pages 111–126 of: *COLT*.

Baluja, S., Seth, R., Sivakumar, D., Jing, Y., Yagnik, J., Kumar, S., Ravichandran, D., and Aly, M. 2008. Video suggestion and discovery for YouTube: Taking Random Walks Through the view graph. Pages 895–904 of: *Proceeding of the 17th international conference on World Wide Web*. ACM.

Belkin, M., Niyogi, P., and Sindhwani, V. 2005. On manifold regularization. In: *Proc. of the Conference on Artificial Intelligence and Statistics (AISTATS)*.

Bengio, Y., Delalleau, O., and Roux, N. L. 2007. *Semi-Supervised Learning*. MIT Press. Chap. Label Propogation and Quadratic Criterion.

Bertsekas, D. 1999. *Nonlinear Programming*. Athena Scientific Publishing.

Bie, T.D., and Cristianini, N. 2003. Convex Methods for Transduction. Pages 73–80 of: *Advances in Neural Information Processing Systems 16*. MIT Press.

Bilmes, J.A. 1998. *A gentle tutorial on the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models*. Tech. rept. ICSI-TR-97-021. University of Berkeley.

Bishop, C. (ed). 1995. *Neural Networks for Pattern Recognition*. Oxford University Press.

Blitzer, J., and Zhu, J. 2008. *ACL 2008 tutorial on Semi-Supervised learning*. `http://ssl-acl08.wikidot.com/`.

Blum, A., and Chawla, S. 2001. Learning from Labeled and Unlabeled Data Using Graph Mincuts. Pages 19–26 of: *Proc. 18th International Conf. on Machine Learning*. Morgan Kaufmann, San Francisco, CA.

Chapelle, O., Scholkopf, B., and Zien, A. 2007. *Semi-Supervised Learning*. MIT Press.

Collobert, R., Sinz, F., Weston, J., Bottou, L., and Joachims, T. 2006. Large Scale Transductive SVMs. *Journal of Machine Learning Research*.

Corduneanu, A., and Jaakkola, T. 2003. On information regularization. In: *Uncertainty in Artificial Intelligence*.

Delalleau, O., Bengio, Y., and Roux, N. L. 2005. Efficient Non-parametric Function Induction in Semi-Supervised Learning. In: *Proc. of the Conference on Artificial Intelligence and Statistics (AISTATS)*.

Dempster, A.P., Laird, N.M., Rubin, D.B., et al. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, **39**(1), 1–38.

Deshmukh, N., Ganapathiraju, A., Gleeson, A., Hamaker, J., and Picone, J. 1998 (November). Resegmentation of Switchboard. Pages 1543–1546 of: *Proceedings of the International Conference on Spoken Language Processing*.

Evermann, G., Chan, H. Y., Gales, M. J. F., Jia, B., Mrva, D., Woodland, P. C., and Yu, K. 2005. Training LVCSR Systems on Thousands of Hours of Data. In: *Proc. of ICASSP*.

Frey, B.J., and Dueck, D. 2007. Clustering by passing messages between data points. *science*, **315**(5814), 972.

Friedman, J.H., Bentley, J.L., and Finkel, R.A. 1977. An algorithm for finding best matches in logarithmic expected time. *ACM Transaction on Mathematical Software*, **3**.

Garcke, J., and Griebel, M. 2005. Semi-supervised learning with sparse grids. In: *Proc. of the 22nd ICML Workshop on Learning with Partially Classified Training Data*.

Godfrey, J., Holliman, E., and McDaniel, J. 1992 (March). SWITCHBOARD: Telephone Speech Corpus for Research and Development. Pages 517–520 of: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1.

Goldman, S., and Zhou, Y. 2000. Enhancing Supervised Learning with Unlabeled Data. Pages 327–334 of: *Proc. 17th International Conf. on Machine Learning*. Morgan Kaufmann, San Francisco, CA.

Greenberg, S. 1995. *The Switchboard Transcription Project*. Tech. rept. The Johns Hopkins University (CLSP) Summer Research Workshop.

Greenberg, S., Hollenback, J., and Ellis, D. 1996. Insights Into Spoken Language Gleaned From Phonetic Transcription Of The Switchboard Corpus. Pages 24–27 of: *ICSLP*.

Haffari, G.R., and Sarkar, A. 2007. Analysis of Semi-Supervised Learning with the Yarowsky Algorithm. In: *UAI*.

Hosmer, D. W. 1973. A comparison of iterative maximum likelihood estimates of the parameters of a mixture of two normal distributions under three different types of sample. *Biometrics*.

Huang, X., Acero, A., and Hon, H. 2001. *Spoken Language Processing*. Prentice-Hall.

Jebara, T., Wang, J., and Chang, S.F. 2009. Graph Construction and b-Matching for Semi-Supervised Learning. In: *International Conference on Machine Learning*.

Joachims, T. 2003. Transductive Learning via Spectral Graph Partitioning. In: *Proc. of the International Conference on Machine Learning (ICML)*.

Karlen, M., Weston, J., Erkan, A., and Collobert, R. 2008. Large Scale Manifold Transduction. In: *International Conference on Machine Learning, ICML*.

Lawrence, N. D., and Jordan, M. I. 2005. Semi-supervised learning via Gaussian processes. In: *Neural Information Processing Systems*.

Malkin, J., Subramanya, A., and Bilmes, J.A. 2009 (September). On the Semi-Supervised Learning of Multi-Layered Perceptrons. In: *Proc. Annual Conference of the International Speech Communication Association (INTERSPEECH)*.

McLachlan, G. J., and Ganesalingam, S. 1982. Updating a Discriminant Function on the basis of Unclassified data. *Communication in Statistics: Simulation and Computation*.

Nadler, B., Srebro, N., and Zhou, X. 2010. Statistical Analysis of Semi-Supervised Learning: The Limit of Infinite Unlabelled Data. In: *Advances in Neural Information Processing Systems (NIPS)*.

Ng, A., and Jordan, M. 2002. On Discriminative vs. Generative Classifiers: A comparison of logistic regression and Naive Bayes. In: *Advances in Neural Information Processing Systems (NIPS)*.

Nigam, G. 2001. *Using unlabeled data to improve text classification*. Ph.D. thesis, CMU.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc.

Scudder, H. J. 1965. Probability of Error of some Adaptive Pattern-Recognition Machines. *IEEE Transactions on Information Theory*, **11**.

Seeger, M. 2000. *Learning with Labeled and Unlabeled Data*. Tech. rept. University of Edinburgh, U.K.

Shi, J., and Malik, J. 2000. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Sindhwani, V., and Selvaraj, S.K. 2006. Large scale semi-supervised linear SVMs. In: *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR*.

Sindhwani, V., Niyogi, P., and Belkin, M. 2005. Beyond the point cloud: from transductive to semi-supervised learning. In: *Proc. of the International Conference on Machine Learning (ICML)*.

Subramanya, A., and Bilmes, J. 2008. Soft-Supervised Text Classification. In: *EMNLP*.

Subramanya, A., and Bilmes, J. 2009a. Entropic Regularization in Non-parametric graph-based learning. In: *NIPS*.

Subramanya, A., and Bilmes, J. 2009b. The Semi-Supervised Switchboard Transcription Project. In: *Interspeech*.

Subramanya, A., and Bilmes, J. 2011. Semi-Supervised Learning with Measure Propagation. *Journal of Machine Learning Research (to appear)*.

Subramanya, A., Bartels, C., Bilmes, J., and Nguyen, P. 2007. Uncertainty in training Large Vocabulary Speech Recognizers. In: *Proc. of the IEEE Workshop on Speech Recognition and Understanding*.

Szummer, M., and Jaakkola, T. 2001. Partially labeled classification with Markov random walks. In: *Advances in Neural Information Processing Systems*, vol. 14.

Talukdar, P. Pratim, and Crammer, K. 2009. New Regularized Algorithms for Transductive Learning. In: *European Conference on Machine Learning (ECML-PKDD)*.

Tomkins, A. 2008. *Keynote Speech*. CIKM Workshop on Search and Social Media.

Tsang, I. W., and Kwok, J. T. 2006. Large-scale sparsified manifold regularization. In: *Advances in Neural Information Processing Systems (NIPS) 19*.

Tsuda, K. 2005. Propagating distributions on a hypergraph by dual information regularization. In: *Proceedings of the 22nd International Conference on Machine Learning*.

Vapnik, V. 1998. *Statistical Learning Theory*. Wiley Series.

Vazirani, V. V. 2001. *Approximation Algorithms*. Springer-Verlag.

Wang, F., and Zhang, C. 2006. Label propagation through linear neighborhoods. Pages 985–992 of: *Proceedings of the 23rd international conference on Machine learning*. New York, NY, USA: ACM.

White, N. 1986. *Theory of matroids*. Cambridge.

Woess, W. 2000. *Random walks on infinite graphs and groups*. Cambridge tracts in mathematics 138, Cambridge University Press.

Zhu, X. 2005a. *Semi-Supervised Learning Literature Survey*. Tech. rept. 1530. Computer Sciences, University of Wisconsin-Madison.

Zhu, X. 2005b. *Semi-Supervised Learning with Graphs*. Ph.D. thesis, Carnegie Mellon University.

Zhu, X., and Ghahramani, Z. 2002a. *Learning from labeled and unlabeled data with label propagation*. Tech. rept. Carnegie Mellon University.

Zhu, X., and Ghahramani, Z. 2002b. *Towards semi-supervised classification with Markov random fields*. Tech. rept. CMU-CALD-02-106. Carnegie Mellon University.

Zhu, X., and Goldberg, A.B. 2009. *Introduction to semi-supervised learning*. Morgan & Claypool Publishers.

Zhu, X., Ghahramani, Z., and Lafferty, J. 2003. Semi-supervised learning using Gaussian fields and harmonic functions. In: *Proc. of the International Conference on Machine Learning (ICML)*.