Applications and Advances in Similarity-based Machine Learning

by

Quico Pepijn Spaen

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Industrial Engineering and Operations Research

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Dorit S Hochbaum, Chair
Professor Alper Atamturk
Assistant Professor Paul Grigas
Professor Satish Rao

Fall 2019

Applications and Advances in Similarity-based Machine Learning

Abstract

Applications and Advances in Similarity-based Machine Learning

by

Quico Pepijn Spaen

Doctor of Philosophy in Engineering - Industrial Engineering and Operations Research

University of California, Berkeley

Professor Dorit S Hochbaum, Chair

Similarity-based machine learning methods differ from traditional machine learning methods in that they also use pairwise similarity relations between objects to infer the labels of unlabeled objects. A recent comparative study for classification problems by Baumann et al. [2019] demonstrated that similarity-based techniques have superior performance and robustness when compared to well-established machine learning techniques. Similarity-based machine learning methods benefit from two advantages that could explain superior their performance: They can make use of the pairwise relations between unlabeled objects, and they are robust due to the transitive property of pairwise similarities.

A challenge for similarity-based machine learning methods on large datasets is that the number of pairwise similarity grows quadratically in the size of the dataset. For large datasets, it thus becomes practically impossible to compute all possible pairwise similarities. In 2016, Hochbaum and Baumann proposed the technique of *sparse computation* to address this growth by computing only those pairwise similarities that are relevant. Their proposed implementation of sparse computation is still difficult to scale to millions objects.

This dissertation focuses on advancing the practical implementations of sparse computation to larger datasets and on two applications for which similarity-based machine learning was particularly effective. The applications that are studied here are cell identification in calcium-imaging movies and detecting aberrant linking behavior in directed networks.

For sparse computation we present faster, geometric algorithms and a technique, named *sparse-reduced computation*, that combines sparse computation with compression. The geometric algorithms compute the exact same output as the original implementation of sparse computation, but identify the relevant pairwise similarities faster by using the concept of data shifting for identifying objects in the same or neighboring blocks. Empirical results on datasets with up to 10 million objects show a significant reduction in running time. Sparse-reduced computation combines sparse computation with a technique for compressing highly-similar or identical objects, enabling the use of similarity-based machine learning on massively-large datasets. The computational results demonstrate that sparse-reduced computation provides a significant reduction in running time with a minute loss in accuracy.

A major problem facing neuroscientists today is cell identification in calcium-imaging movies. These movies are in-vivo recordings of thousands of neurons at cellular resolution. There is a great need for automated approaches to extract the activity of single neurons from these movies since manual post-processing takes tens of hours per dataset. We present the HNCcorr algorithm for cell identification in calcium-imaging movies. The name HNCcorr is derived from its use of the similarity-based Hochbaum's Normalized Cut (HNC) model with pairwise similarities derived from correlation. In HNCcorr, the task of cell detection is approached as a clustering problem. HNCcorr utilizes HNC to detect cells in these movies as coherent clusters of pixels that are highly distinct from the remaining pixels. HNCcorr guarantees, unlike existing methodologies for cell identification, a globally optimal solution to the underlying optimization problem. Of independent interest is a novel method, named *similarity-squared*, that we devised for measuring similarity between pixels. We provide an experimental study and demonstrate that HNCcorr is a top performer on the Neurofinder cell identification benchmark and that it improves over algorithms based on matrix factorization.

The second application is detecting aberrant agents, such as fake news sources or spam websites, based on their link behavior in networks. Across contexts, a distinguishing characteristic between normal and aberrant agents is that normal agents rarely link to aberrant ones. We refer to this phenomenon as *aberrant linking behavior*. We present an Markov Random Fields (MRF) formulation, with links as the pairwise similarities, that detects aberrant agents based on aberrant linking behavior and any prior information (if given). This MRF formulation is solved optimally and in polynomial time. We compare the optimal solution for the MRF formulation to well-known algorithms based on random walks. In our empirical experiment with twenty-three different datasets, the MRF method outperforms the other detection algorithms. This work represents the first use of optimization methods for detecting aberrant agents as well as the first time that MRF is applied to directed graphs.

Contents

List of Figures

List of Tables

Acknowledgments

This work and the completion of my PhD would not have been possible without the support of numerous people. In particular, I would like to thank:

My parents, Frank Spaen and Jolande IJsseldijk, for guiding me to find my own path in life, giving me the freedom to move to another continent, and for their unwavering support for me; My girlfriend Aisling Scott for spicing up my life with fun over the past two years, her advice on numerous topics, and for believing in me; My brother and sister Joep and Floor Spaen for being the best brother and sister ever and for annihilating me at board games.

My advisor Dorit S. Hochbaum for her guidance, wisdom, and for believing in me. I really appreciate the countless hours we spent to discuss, analyze, and strategize for our research and teaching followed by extensive conversations about the most diverse and unrelated topics. We sometimes disagreed but our discussions were productive and always resulted in a better outcome.

My main co-authors, Roberto Asín Achá, Philipp Baumann, Christopher Thraves Caro, and Mark Velednitsky, for our fruitful and enjoyable collaborations; Professors Alper Atamturk, Paul Grigas, Hillel Adesnik, and Satish Rao for their willingness to serve on my committee; The IEOR department staff, Keith McAleer, Diana Salazar, James Percy, Anayancy Paz, and Rebecca Pauling, for their support, helping me with questions I had, and for their positive energy; Professor Rhonda Righter for her care and interest in the well-being of students in the department.

My roommates Jian (Sam) Ju, Rob Clarke, and Valence Li for making me feel at home and for our many late-night conversations; My friend and fellow PhD student Amber Richter for our near-daily lunches and for introducing me to Aisling; My friends from the GPC Tuesday ride for sharing my joy in cycling and for keeping me healthy; Sheila Satin and friends for organizing so many fun social activities.

I would also like to thank all my fellow PhD students for our friendships, bar nights, Lunch & Learns, good advice, and many fun conversations. My thanks go to: Erik Bertelli, Clay Campaigne, Haoyang Cao, Junyu Cao, Ying Cao, Chen Chen, Carlos Deck, Shiman Ding, Yuhao Ding, Ilgin Dogan, Pelagie Elimbi Moudio, Salar Fattahi, Han Feng, Hao Fu, Andrés Gómez, Dean Grosbard, Tuğçe Gürek, Pedro Hespanhol, Anran Hu, Arman Jabbari, Titouan Jehl, Hansheng Jiang, Yusuke Kikuchi, Heejung Kim, Jiung Lee, Kevin Li, Tianyi Lin, Paula Lipka, Heyuan Liu, Sheng Liu, Stewart Liu, Alfonso Lobos, Cheng Lu, Yonatan Mintz, Igor Molybog, Julie Mulvaney-Kemp, Matt Olfat, Sangwoo Park, Georgios Patsakis, Meng Qi, Wei Qi, Xu Rao, Amber Richter, Rebecca Sarto Basso, Jiaying Shi, Auyon Siddiq, Birce Tezel, Mark Velednitsky, Renyuan Xu, Nan Yang, Min Zhao, Mo Zhou, Ruijie Zhou, and many others. Finally, I would like to thank all my other Berkeley friends and my fellow Graduate Assembly (GA) participants.

Chapter 1

Introduction

In supervised learning, the goal is to predict the label of an object based on the object's feature vector. Most traditional machine learning methods approach this task by learning a function from a labeled training dataset that maps a feature vector to a label. This function is then applied to decide labels for unlabeled objects. A less common approach is to infer the labels of unlabeled objects based on pairwise similarity relations between all objects in the dataset, including those that are unlabeled, in addition to feature vectors. We refer to this approach as *similarity-based* machine learning.

There is mounting evidence that adding pairwise similarities considerably enhances the quality of pattern recognition and data mining techniques. This was demonstrated in a recent comparative study of fourteen different supervised learning techniques across twenty different datasets [Baumann et al., 2019]. In this study, similarity-based machine learning methods provided superior performance, as measured by F1-score, relative to non-similarity-based methods for classification problems. Similar observations have been made previously for classification problems [Dembczyński et al., 2009], medical diagnosis [Ryu et al., 2004], and for semi-supervised learning [Zhu et al., 2003].

A major advantage for similarity-based algorithms is that they benefit from a transitivity property. Distantly-similar objects can be labeled or grouped together due to a transitive chain of similarities [Kawaji et al., 2004]. The transitivity property also provides similarity-based algorithms with robustness, since similarity between a pair of objects is not only measured by comparing the two objects directly but also via multiple other paths of pairwise similarities via intermediate objects.

Another advantage for certain similarity-based machine learning methods is that they make use of the pairwise relations between unlabeled objects such as those in the test dataset. There are two similarity-based machine learning algorithms, Hochbaum's Normalized Cut (HNC) [Hochbaum, 2010, 2013b] and Markov Random Fields (MRF) [Geman and Geman, 1984; Hochbaum, 2001], that have this unique feature. Other similarity-based machine learning algorithms, such as the k-Nearest Neighbors algorithm [Fix and Hodges, 1951] that classifies an object based on the labels of its k closest neighbors, are limited to pairwise similarities relations between labeled and unlabeled objects. Empirical evidence by Tresp

[2000] indicates that the performance of a method can improve by considering the relation between unlabeled objects. In his study, the performance of his ensemble method, the Bayesian Committee Machine (BCM), improved when considering multiple test objects simultaneously. This occurred because the BCM utilized the covariance between the test objects.

There are two primary reasons as to why similarity-based machine learning models are rarely used in practice. The first reason is the perceived lack of efficient machine learning techniques for dealing with pairwise similarities. However, efficient algorithms exist for e.g. both the HNC and MRF models [Ahuja et al., 2003, 2004; Hochbaum, 2001, 2010, 2013a]. The second concern is that it becomes practically impossible to compute all pairwise similarities for a large dataset, since the number of pairwise similarities grows quadratically in the size of the dataset. Hochbaum and Baumann [2016] introduced the method of *sparse computation* to mitigate this problem.

This dissertation focuses on advancing the practical implementations of sparse computation and on two applications for which similarity-based machine learning was particularly effective. These applications are: Cell detection in calcium-imaging movies and detecting aberrant agents in networks based on link behavior. We now provide a brief description of the similarity-based models HNC and MRF that were used in these applications. Subsequently, we discuss the method of sparse computation for mitigating the quadratic growth in the number of pairwise similarities and two extensions for larger datasets. We then introduce the two applications in more detail.

1.1 Models for Similarity-Based Machine Learning: Hochbaum's Normalized Cut and Markov Random Fields

Two models for similarity-based machine learning are Hochbaum's Normalized Cut (HNC) [Hochbaum, 2010, 2013b] and Markov Random Fields (MRF) [Geman and Geman, 1984; Hochbaum, 2001, 2013a]. Both the HNC model and the MRF model with convex penalties are solved to global optimality in polynomial time with combinatorial algorithms [Ahuja et al., 2003, 2004; Hochbaum, 2001, 2010, 2013a].

Hochbaum's Normalized Cut (HNC)

The clustering model Hochbaum's Normalized Cut (HNC) [Hochbaum, 2010, 2013b] is a variant of the Normalized Cut problem for image segmentation that was popularized by Shi and Malik [2000]. The HNC model is defined on graph where nodes represent objects and edges are pairwise similarity relations. The model provides a trade-off between two objectives: High similarity between the objects in the cluster (homogeneity), and low similarity between objects in the cluster and the remaining objects (distinctness). The trade-off between the two objectives is represented as either a ratio or a weighted linear combination. In either form, the model is polynomial time solvable [Hochbaum, 2010, 2013b] with combinatorial algorithms. In contrast, the Normalized Cut problem is known to be NP-Hard [Shi and Malik, 2000].

HNC is applicable to both supervised and unsupervised problems, and it has been successfully used across different contexts. These include image segmentation [Hochbaum et al., 2013], evaluating the effectiveness of drugs [Hochbaum et al., 2012], the detection of special nuclear materials [Yang et al., 2013], tracking moving objects in videos [Fishbain et al., 2013], and a comparative study on classification problems by Baumann et al. [2019]. In this study, it was demonstrated that the supervised variants of HNC (SNC & K-SNC) have the best overall performance and were the most robust among all algorithms considered, which included state-of-the-art machine learning techniques. Hochbaum et al. [2012]; Yang et al. [2013] also provide similar but less comprehensive comparisons between machine learning methods for drug ranking and the detection of special nuclear materials. These experiments also indicated that SNC was among the top performing machine learning methods for these contexts.

Markov Random Fields Problem (MRF)

The Markov Random Fields (MRF) [Geman and Geman, 1984; Hochbaum, 2001, 2013a; Kleinberg and Tardos, 2002] was originally considered in image segmentation, where pixels from a noisy image should be assigned to a set of colors. Similar to the HNC model, the objective in the MRF model also presents a trade-off. The trade-off for the MRF model is between two types of similarities: The similarity between an object and its prior and the pairwise similarity between pairs of objects. Each type has an associated penalty in the objective function: A *deviation* penalty for when the assigned value of an object deviates away from a prior value, and a *separation* penalty for when the assigned values of a pair of objects differ. In image segmentation, the deviation penalties penalizes deviations with respect to the original color of the pixel whereas the separation penalties smooth the colors assigned to adjacent pixels. When the separation penalty functions are convex, the MRF problem is solved optimally and efficiently in either continuous or integer variables [Ahuja et al., 2003, 2004; Hochbaum, 2001]. The problem is NP-hard otherwise [Hochbaum, 2001].

Aside from image segmentation [Hochbaum, 2001, 2013a; Qranfal et al., 2011], MRF has been applied successfully for group decision-making [Hochbaum and Levin, 2006], rating customers' propensity to buy new products [Hochbaum et al., 2011], ranking the credit risk of countries [Hochbaum and Moreno-Centeno, 2008], and for yield prediction in semiconductor manufacturing [Hochbaum and Liu, 2018]. In all of these applications, the separation penalties were defined on undirected graphs. The MRF application presented below is unique in that the separation penalties are directional and are defined on a directed graph. The algorithms listed above apply to both types of graphs.

1.2 Sparse Computation for Mitigating the Quadratic Growth of Similarities

A challenge for similarity-based machine learning methods on large datasets is that the number of pairwise similarity grows quadratically in the size of the dataset. For large datasets, it thus becomes practically impossible to compute *all* possible pairwise similarities.

Hochbaum and Baumann [2016] proposed the technique of *sparse computation* to address this growth by computing only those pairwise similarities that are relevant. Sparse computation enables the scaling of similarity-based algorithms to large datasets. The method first project the objects' feature vectors into a low-dimension space. This is commonly done with the dimension reduction method *approximate principle component analysis*, a variant of principle component analysis (PCA). Sparse computation uses closeness of two objects in the low-dimensional space as a proxy for the relevance of the associated pairwise similarity. To identify close pairs of objects in the low-dimensional space, the low-dimensional space is discretized into grid blocks. A pair of objects is considered relevant if the objects belong to the same or adjacent grid blocks. The pairwise similarities are then computed for all relevant pairs with the original feature vectors.

The Hochbaum and Baumann [2016] implementation of sparse computation is still difficult to scale to millions objects. We present in this work two extensions that scale the method of sparse computation to very-large datasets (see also [Baumann et al., 2016, 2017]).

The first extension consists of faster geometric algorithms for sparse computation. These geometric algorithms compute the exact same output as the original implementation of sparse computation, but identify the relevant pairwise similarities faster by using the concept of *data shifting* for identifying objects in the same or neighboring blocks. Empirical results on datasets with up to 10 million objects show a significant reduction in running time. The new algorithms also result in improved scaling for sparse computation with respect to the dimension of the low-dimensional space.

The second extension is the technique of *sparse-reduced computation*. Sparse-reduced computation combines sparse computation with a technique for compressing highly-similar or identical objects, enabling the use of similarity-based machine learning on massively-large datasets. Due to the compression, sparse-reduced computation provides a different output than sparse computation and replaces groups of objects by new representative objects. The computational results demonstrate that sparse-reduced computation provides very similar accuracy as sparse computation with a significant reduction in runtime. Sparse-reduced computation allows for highly-accurate classification of datasets with millions of objects in seconds.

1.3 Applications: Cell Identification in Neuroscience and Aberrant Agent Detection in Networks with Similarity-based Machine Learning

A major problem facing neuroscientists today is detecting cells in calcium-imaging movies. Calcium imaging is a modern technique used by neuroscientists for recording movies of in-vivo neuronal activity at cellular resolution. Using genetically encoded calcium indicators and fast laser-scanning microscopes, it is now possible to record thousands of neurons simultaneously. However, the manual post-processing needed to extract the activity of single neurons requires tens of hours per dataset. Consequently, there is a great need to develop automated approaches for the extraction of neuronal activity from imaging movies.

We present a similarity-based algorithm, named HNCcorr, for cell detection in calcium imaging movies (see also [Asín-Achá et al., 2019; Spaen et al., 2019]). The name HNCcorr is derived from the use of HNC and that of pairwise similarities based on correlation. In HNCcorr, the task of cell detection in calcium imaging movies is approached as a clustering problem. HNCcorr utilizes HNC to detect cells in these movies as coherent clusters of pixels that are highly distinct from the remaining pixels. HNCcorr guarantees, unlike existing methodologies for cell identification, a globally optimal solution to the underlying optimization problem. Of independent interest is a novel method, named *similarity-squared*, that we devised for measuring similarity between pixels. We provide an experimental study and demonstrate that HNCcorr is a top performer on the Neurofinder cell identification benchmark and that it improves over algorithms based on matrix factorization [Pachitariu et al., 2017; Pnevmatikakis et al., 2016]. This algorithm represents the first use of similarity-based machine learning methods in neuroscience.

The second application is detecting aberrant agents in networks based on their link behavior. Agents with aberrant behavior are commonplace in today's networks; there are fake profiles in social media, spam websites on the internet, and fake news sources that are prolific in spreading misinformation. The viral spread of digital misinformation has become so severe that the World Economic Forum considers it among the main threats to human society [World Economic Forum, 2013]. It is thus crucially important to be able to identify aberrant agents. Across contexts, a distinguishing characteristic between normal and aberrant agents is that normal agents rarely link to aberrant ones. We refer to this phenomenon as *aberrant linking behavior*.

We present an MRF formulation that detects aberrant agents based on the link behavior of agents in the network. The formulation balances two objectives: to satisfy aberrant linking behavior by having as few links as possible from normal to aberrant agents, as well as to deviate minimally from prior information (if given). The MRF formulation is solved optimally and efficiently. We compare the optimal solution for the MRF formulation to well-known algorithms based on random walks [Rajaraman and Ullman, 2011], including PageRank [Page et al., 1999], TrustRank [Gyöngyi et al., 2004], and AntiTrustRank [Krishnan and Raj, 2006]. To assess the performance of the algorithms, we present a variant of the modularity clustering metric that overcomes the known shortcomings of modularity in directed graphs. We show that this new metric has desirable properties and prove that optimizing it is NP-hard. In our empirical experiment with twenty-three different datasets, the MRF method outperforms the other detection algorithms. This work is the first use of optimization methods for detecting aberrant agents as well as the first time that MRF is applied with directed links.

1.4 Overview of this Dissertation

Chapter 2 presents the improved methodologies for sparse computation. The chapter starts with an overview of the sparse computation method. Then, the geometric algorithms for sparse computation are presented, and their performance is analyzed in a computational study. Subsequently, the sparse-reduced computation method is presented. Sparse-reduced computation extends sparse computation by adding compression of highly-similar objects. An experimental study of its performance relative to sparse computation is provided.

Chapter 3 introduces the HNCcorr algorithm for cell detection in calcium imaging. The chapter includes a discussion of the HNC model and algorithms for solving it. The HNCcorr algorithm is described, and a discussion of how it incorporates HNC and sparse computation is provided. Having described the algorithm, the chapter concludes with an experimental comparison between HNCcorr and other cell identification algorithms.

Chapter 4 lays out our approach for detecting aberrant agents in directed networks based on their linking behavior with MRF. The chapter provides the general MRF model, algorithms for solving it, and our MRF formulation for the detection of aberrant agents. The experimental performance of MRF is compared with other well-known algorithms.

Chapter 2

Improved Methodologies for Sparse Computation: Geometric Algorithms and Sparse-Reduced Computation

Computing pairwise similarities poses a challenge in terms of scalability as the number of pairwise similarities between objects grows quadratically in the number of objects in the dataset. For large datasets, it is prohibitive to compute and store all pairwise similarities.

Various methods have been proposed that sparsify a complete similarity matrix, which contains all pairwise similarities, while preserving specific matrix properties [Arora et al., 2006; Jhurani, 2013; Spielman and Teng, 2011]. A sparse similarity matrix requires less memory and allows faster classification as the running time of the algorithms depends on the number of non-zero entries in the similarity matrix. These approaches are not suitable for large-scale datasets because they require as input the complete similarity matrix.

Recently, Hochbaum and Baumann [2016] introduced a methodology called *sparse computation* that generates a sparse similarity matrix without having to compute the complete similarity matrix first. Sparse computation enables the scaling of similarity-based machine learning methods to large dataset by significantly reducing the running time of the classifiers without affecting their accuracy (see Figure 2.1). In sparse computation the data is efficiently projected onto a low-dimensional space using a probabilistic variant of principal component analysis. The low-dimensional space is then subdivided into grid blocks and pairwise similarities are only computed between objects in the same or in neighboring grid blocks. The density of the similarity matrix can be controlled by varying the grid resolution. A higher grid resolution leads to a sparser similarity matrix. In section 2.1, we review the sparse computation technique in more detail.

For large-scale datasets, the computational bottleneck of sparse computation is the identification of pairs of adjacent blocks in the grid structure. For each non-empty block, it identifies adjacent blocks and checks whether these blocks are non-empty. This process is referred to as *block enumeration.* For large-scale datasets to which sparse computation is applied with a high grid resolution, the vast majority of adjacent blocks are empty, but are

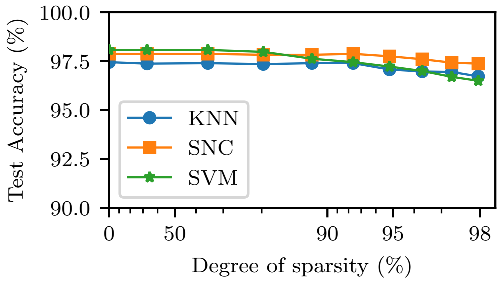


Figure 2.1: Accuracy at selected sparsity levels for the letter recognition (LER) dataset obtained with three similarity-based classifiers - k-nearest neighbors, SNC [Hochbaum, 2010], and SVM [Cortes and Vapnik, 1995] - on a sparse similarity matrix generated with sparse computation. Sparsity, reported on a log scale, is measured as the percentage of matrix entries that are not computed. The runtime of the algorithms is inversely proportional to the sparsity of the similarity matrix.

still checked. Hence, a large fraction of the computational workload is unnecessary.

In section 2.2, we present two new algorithms for sparse computation that address this computational bottleneck. The algorithms are based on a computational geometry concept called data shifting, which is used to identify pairs of similar objects in a low-dimensional space much faster than with state-of-the-art techniques.

Computational experiments in section 2.3 demonstrate that the new algorithms for sparse computation are up to five times faster. The improved algorithms enable the scaling of sparse computation to datasets with millions of objects. Furthermore, the new algorithms improve the scaling of sparse computation with respect to the dimension of the low-dimensional space.

Regardless of which sparse computation algorithm is applied, it may occur that large groups of highly-similar or identical objects project to the same grid block even when the grid resolution is high. As a result, a large number of pairwise similarities between nearly-identical objects is identified by sparse computation. The computation of similarities between these objects is unnecessary as they often belong to the same class. In massively-large datasets, large numbers of highly-similar objects are particularly common. The grid block structure created in sparse computation reveals the existence of such highly-similar objects in the dataset.

In section 2.4, we propose an extension of sparse computation called *sparse-reduced computation* that avoids the computation of similarities between highly-similar and identical objects through compression. The method builds on sparse computation by using the grid block structure to identify highly-similar and identical objects efficiently. In each grid block, the objects are replaced by a small number of representatives. The similarities are then computed only between representatives in the same and in neighboring blocks. The resulting similarity matrix is not only sparse but also smaller in size due to the consolidation of objects.

We evaluate sparse-reduced computation on four real-world and one artificial benchmark

Table 2.1: Notation

Symbol	Description
n	Number of objects
d	Number of features
$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$	Objects - represented by their feature vectors
p	Number of dimensions in low-dimensional space
k and k'	Grid resolution
$\underline{\kappa}$	Sub-block grid resolution in sparse-reduced computation
ω	Pre-specified L_∞ distance

datasets containing up to 10 million objects in section 2.5. Sparse-reduced computation delivers highly-accurate classification at very low computational cost for most of the studied datasets.

Throughout this chapter, we use the notation given in Table 2.1.

2.1 Overview of Sparse Computation

Sparse computation [Hochbaum and Baumann, 2016] takes as input a dataset with n objects $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$ and d features. The method consists of the steps: dimension reduction, grid construction and selection of pairs, and similarity computation.

In the dimension reduction step, the input data is projected from a d-dimensional space onto a p-dimensional space, where $p \ll d$. The projection is done with a probabilistic variant of PCA called approximate PCA [Hochbaum and Baumann, 2016]. Let the data in the p-dimensional space be normalized, i.e., the values of each dimension are scaled to the range $[0, 1]$.

In the grid construction and selection of pairs step, the goal is to select all pairs of objects that have an L_∞ distance smaller or equal to ω in the p-dimensional space. This is achieved as follows. First, the range of values along each dimension is subdivided into $k = \frac{1}{\omega}$ equally long intervals. This partitions the p-dimensional space into k^p grid blocks. Parameter k denotes the grid resolution. Each object is then assigned to a single block based on its p coordinates. Objects which lie exactly on a grid line (horizontally and/or vertically) are assigned to the upper and/or right grid block. If the upper and/or right grid block is outside the grid, then the object is assigned to the lower and/or left block.

Since the largest L_∞ distance within a block is equal to $\omega = \frac{1}{k}$, all pairs of objects that belong to the same block are selected. In addition, some pairs of objects are within a distance of ω but fall in different blocks. To select those pairs as well, horizontally, vertically, and diagonally adjacent blocks need to be considered. Each block has up to $3^p - 1$ neighbors. We refer to this process as *block enumeration.* By selecting all pairs of objects that are assigned to adjacent blocks, it can be guaranteed that all pairs of objects whose distance is less than or equal to ω are selected. It is possible that pairs of objects whose L_∞ distance is more than ω but less than 2ω are selected as well, whereas objects whose distance is more than 2ω will not be selected.

The total number of selected pairs depends on the grid resolution k and the dimension of the low-dimensional space p. A higher grid resolution results in smaller blocks and thus reduces the set of pairs that fall in a block or its adjacent blocks. Similarly, when the number of dimensions of the low-dimensional space is increasing, the blocks contain fewer objects and this reduces the number of pairs selected.

In the similarity computation step, a similarity function is used to quantify the similarity for each of the pairs selected in the previous step. The similarity value is computed with respect to the original d-dimensional space.

The selection of pairs in sparse computation relies on enumerating $(3^p - 1)/2$ adjacent blocks for each non-empty block to determine pairs of adjacent non-empty blocks[1]. This computation becomes the bottleneck of the method when a) the number of non-empty blocks is large, and b) most of the adjacent blocks are empty. Checking empty blocks is unnecessary and only contributes to the runtime. Conditions a) and b) are often met when sparse computation is applied with high grid resolution to a large-scale dataset with millions of objects. To illustrate this issue, Figure 2.2 shows a two-dimensional projection of a dataset that contains 12 objects. With a grid resolution of $k = 4$, the two-dimensional space was partitioned into 16 blocks, six of which are non-empty. To find the adjacent blocks of the six non-empty blocks, 24 other blocks (visualized by arrows) are considered. Only 6 out of these 24 blocks are non-empty (blue arrows). The plot on the right hand side of Figure 2.2 highlights all selected pairs by blue lines that connect the corresponding objects.

2.2 Geometric Algorithms for Sparse Computation

To address the bottleneck of identifying pairs in adjacent blocks, we introduce a computational geometry concept called data shifting. We show how data shifting can be used to devise two geometric algorithms for sparse computation: *object shifting* and *block shifting*. These algorithms replace the block enumeration process in the second step of sparse computation.

The object shifting algorithm shifts the objects multiple times along different directions within the grid structure. For each shift, the pairs of objects that fall in the same block are deemed to be similar. Object shifting avoids having to explicitly compute adjacent blocks, but the same pair of objects may be selected for multiple shifts. We refer to such pairs as *duplicate pairs*. The block shifting algorithm partially addresses the drawback of duplicate pairs by identifying all pairs of non-empty adjacent blocks by shifting representatives for non-empty blocks instead of the individual objects. Note that these two algorithms generate the exact same set of pairs as sparse computation with block enumeration.

[1]For each non-empty block, only half of the adjacent blocks need to be checked since the adjacency relation is symmetric.

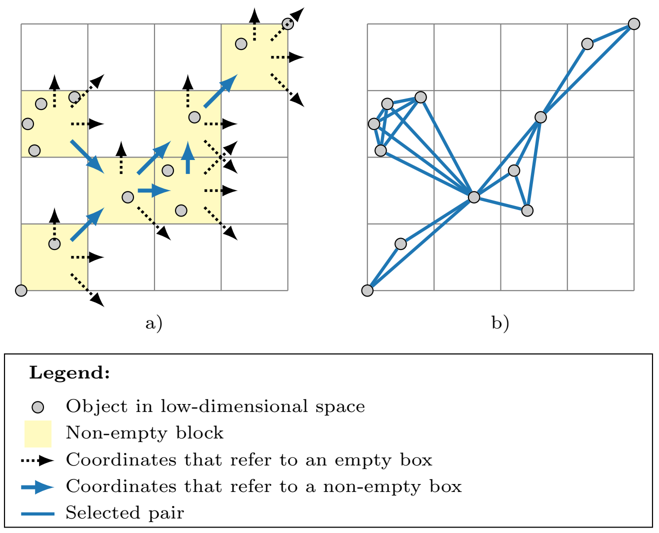

a)
b)
Legend:
Object in low-dimensional space
Non-empty block
Coordinates that refer to an empty box
Coordinates that refer to a non-empty box
Selected pair

Figure 2.2: Visualization of the block enumeration strategy with $k = 4$ and $p = 2$: a) for each non-empty block, four adjacent blocks must be considered to identify all pairs of adjacent non-empty blocks. The majority of considered blocks are empty (dotted arrows). b) all pairs of objects that fall into the same or in neighboring blocks are selected (blue lines)

The concept of data shifting

Sparse computation relies on a grid to identify close pairs of objects in the low-dimensional space. The identified pairs are all within an L_∞ distance of ω and potentially some within an L_∞ distance of 2ω. In contrast to sparse computation with block enumeration, the low-dimensional space is partitioned into k'^p grid blocks for $k' = \frac{1}{2\omega}$. Each grid block is thus twice as large in each dimension. All objects within a grid block are now within an L_∞ distance of 2ω. The grid, however, might still arbitrarily separate objects that are close by a grid line. Two objects that are within an L_∞ distance of ω, but separated by a grid line, are denoted as border pair. In a two-dimensional grid, there are three types of border pairs: horizontal border pairs, vertical border pairs, and diagonal border pairs. In a p-dimensional grid, there are $2^p - 1$ types of border pairs. Data shifting addresses the issue of identifying border pairs by shifting the data from its initial position along a single or multiple axes such that all border pairs of one type will no longer be separated by a grid line after the shift. To capture all types of border pairs, $2^p - 1$ shifts are required. In each shift, the data is shifted by ω along the respective axes. This is sufficient to identify all close neighbors of an object, since for each neighbor that is within a distance of ω from the object there exist at least one grid such that they belong to the same grid block. By efficiently identifying all close neighbors, data shifting provides a 2-approximation for the problem of identifying, for each object, the set of neighbors that are within an L_∞ distance of ω.

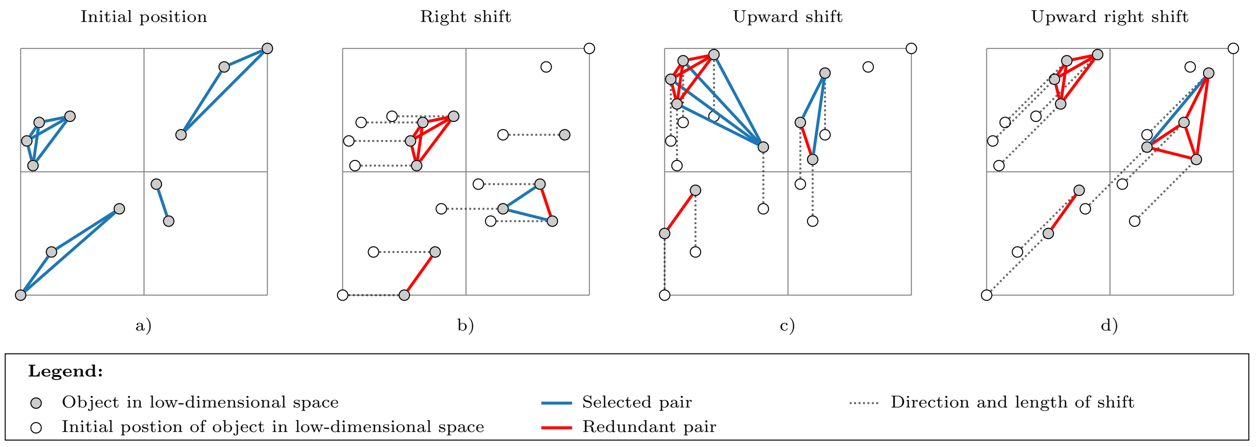

Initial position
Right shift
Upward shift
Upward right shift
a)
b)
c)
d)
Legend:
● Object in low-dimensional space
○ Initial postion of object in low-dimensional space
Selected pair
Redundant pair
Direction and length of shift

Figure 2.3: Visualization of the object shifting algorithm with grid resolution $k' = 2$ for a dataset projected to a $p = 2$ dimensional space: a) selection of pairs of objects that fall in the same grid block, b) objects are horizontally shifted by $1/k$ to capture horizontal border pairs. c) objects are vertically shifted by $1/k$ to capture vertical border pairs, d) objects are diagonally shifted to capture diagonal border pairs. Redundant pairs are highlighted in red.

Object shifting algorithm

To get exactly the same selection of similar pairs as the block enumeration strategy with a grid resolution of k, the object shifting algorithm first partitions the low-dimensional space into k'^p grid blocks for $k' = \frac{1}{2}k$. Based on the concept of data shifting, $2^p - 1$ directions are determined. Given p, the directions can be determined by generating binary representations of width p of the integers $[1, \ldots, 2^p - 1]$. Each bit corresponds to a dimension and a one means that the data is shifted along this dimension. For example, if $p = 2$, the binary representations are 01, 10, and 11. The algorithm is called object shifting because all objects are shifted by $\frac{1}{k} = \frac{1}{2k'}$ in that direction. Based on the new coordinates, each object is assigned to a single grid block and all pairs of objects that are assigned to the same block are selected. Since the same pairs of objects might be assigned to the same block in different shifts, one needs to identify and remove duplicate pairs. If the grid resolution is high and hence the total number of selected pairs is low, the runtime for identifying and removing redundant pairs is negligible. However, if the grid resolution is low and the total number of selected pairs is large, then identifying and removing redundant pairs can become computationally expensive. Figure 2.3 illustrates object shifting algorithm for our two-dimensional example. The original position of the data is shown in the top left plot.

Block shifting

The disadvantage of object shifting is that certain pairs are selected for multiple shifts. In particular, objects that fall in the same sub-block, defined by splitting the blocks in half in each dimension, will fall in the same block for each shift and are thus repeated $2^p - 1$ times. Block shifting addresses this by replacing them with a single object.

To generate the same pairs as sparse computation with block enumeration with grid resolution k, the p-dimensional space is first partitioned with a grid resolution of k into k^p grid blocks and each object is assigned to the corresponding block. The corresponding pairs for each block are selected. Instead of identifying the border pairs directly with data shifting each non-empty block is first replaced with a representative object at its center. Note that this new dataset consists of objects that are located at multiples of $\frac{1}{2k}$ along the dimensions. All coordinates are within the range $[\frac{1}{2k}, 1 - \frac{1}{2k}]$ and the L_∞ distance between pairs of objects that represent adjacent blocks is exactly $\frac{1}{k}$. Hence, we can apply the object shifting algorithm to the new dataset with a grid resolution of $k' = \frac{1}{2}k$ to find all pairs of adjacent blocks. Finally, all pairs of objects that consists of objects in adjacent blocks are selected. Figure 2.4 illustrates block shifting for our two-dimensional example.

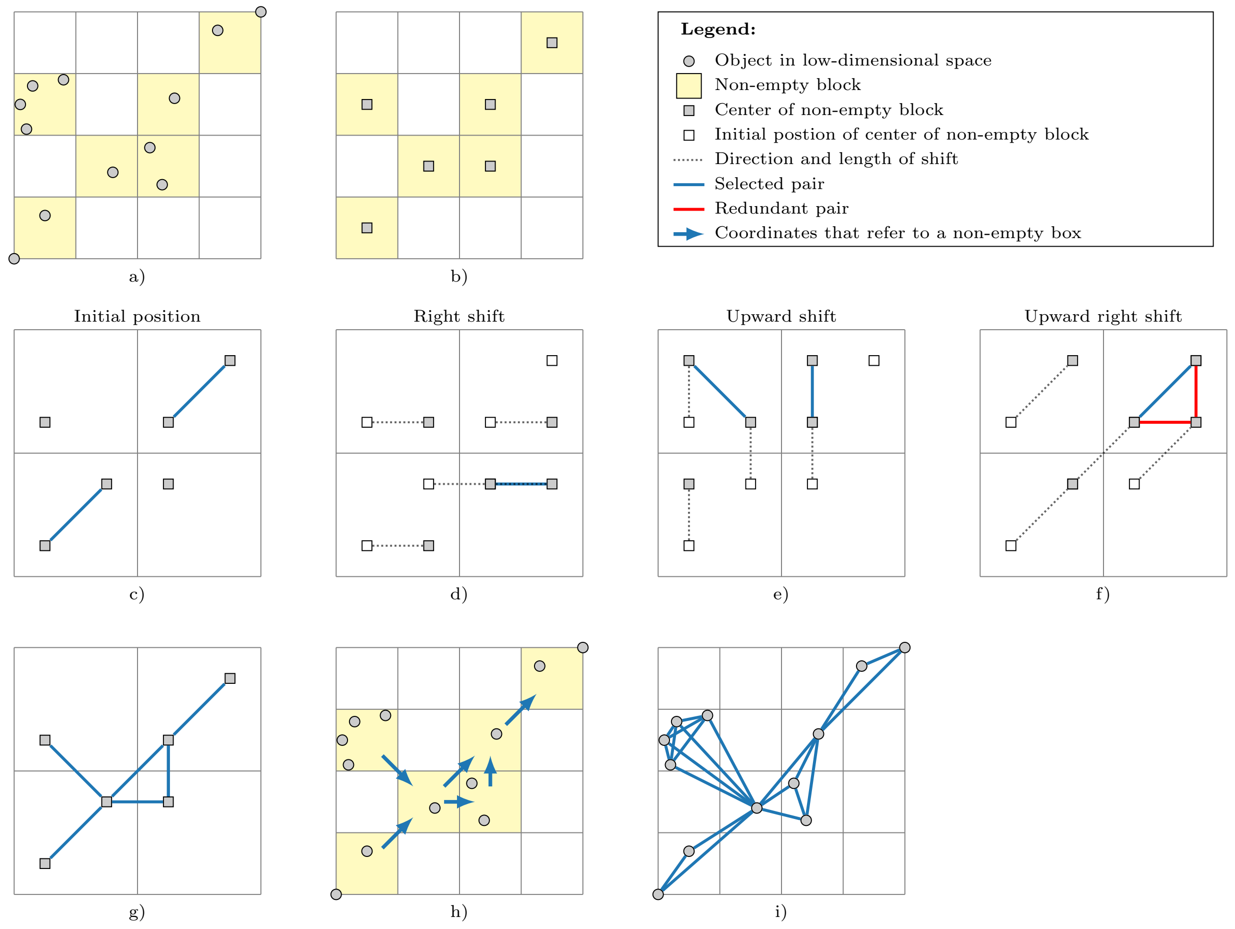
Runtime analysis

The runtime of block enumeration, object shifting, and block shifting depends on the parameters k and p and on the distribution of the objects within the low-dimensional space. Since all algorithms have the same output, they only differ with respect to the overhead (unnecessary work performed): The object/block shifting algorithms may identify pairs of objects/representatives more than once, and the block enumeration algorithm may enumerate empty adjacent blocks.

The overhead of the block enumeration is large when objects fall into many isolated blocks. This typically occurs for large k and p. The overhead of object shifting is large when many objects fall into few adjacent blocks. This typically occurs for small k and p. The overhead of block shifting is only large when the number of adjacent representatives is large. This does not occur for small or for large values of k and p. To obtain a large number of adjacent representatives, few objects must fall in a large number of adjacent blocks. Hence, the number of adjacent representatives tends to initially increase with increasing p and k, but decreases again once the blocks become isolated. In practice, we observe that block shifting identifies a pairwise similarity no more than twice. For more details, see the experimental results in section 2.3 and Figure 2.6.

2.3 Experimental Analysis of Geometric Algorithms for Sparse Computation

We compared the runtime of block enumeration, object shifting, and block shifting for different grid resolutions on nine datasets. We evaluated only runtime because all algorithms generate the same output when the grid resolutions are chosen accordingly. Extensive empirical results on how sparse computation and thus the new algorithms affect the accuracy of (un)supervised learning algorithms are presented in [Baumann, 2016; Hochbaum and Baumann, 2016].



Legend:
Object in low-dimensional space
Non-empty block
Center of non-empty block
Initial postion of center of non-empty block
Direction and length of shift
Selected pair
Redundant pair
Coordinates that refer to a non-empty box
a)
b)
Initial position
Right shift
Upward shift
Upward right shift
c)
d)
e)
f)
g)
h)
i)

Figure 2.4: Visualization of the block shifting algorithm with grid resolution $k = 4$ for a dataset projected to a $p = 2$ dimensional space: a) low-dimensional space is partitioned with grid resolution k, b) each non-empty block is replaced by a representative, c)–f) the object shifting algorithm is applied with grid resolution $k' = 2$ to find pairs of representatives that correspond to neighboring blocks (redundant pairs of representatives are highlighted in red), g)–i) the identified pairs of representatives are used to select pairs of objects that fall within the same or within neighboring blocks.

We implemented the algorithms and the computational analysis in Python 3.5. The source code is available at `https://github.com/hochbaumGroup/sparsecomputation`. The source code of the computational analysis is available at `https://github.com/quic0/sparse-experiments`.

datasets

The datasets represent various domains including life sciences, engineering, social sciences and business and have sizes ranging from hundreds to millions of objects. Six real-world datasets were taken from the UC Irvine Machine Learning Repository [Lichman, 2013], two datasets were taken from previous studies [Breiman, 1996; Dong, Jian-xiong et al., 2005; Lee and Mangasarian, 2001; Tsang et al., 2005], and one real-world dataset was taken from the Neurofinder public benchmark for calcium imaging [CodeNeuro, 2017]. In all datasets, we substituted categorical features by a binary feature for each category and removed any duplicate objects. We briefly describe each dataset and mention further modifications that we made. Statistics about each of the datasets are provided in Table 2.2.

The dataset *Adult* (ADU) [Kohavi, 1996] stems from the census bureau database and each object represents a person. In the original dataset there is a categorical and a continuous feature to capture the educational level of the persons. To avoid double use, we removed the categorical feature. In addition, we removed all features that contain missing values.

In the *Bag of Words* dataset (BOW), the objects are documents from five different sources. The sources are Enron emails, KOS blog entries, New York Times articles, NIPS full papers, and PubMed abstracts. A document in this dataset is represented as a bag of words, i.e., a set of vocabulary words. This representation disregards word order but keeps word multiplicity. We generated two datasets from the bag of words dataset: BOW1 contains the NIPS full papers and the Enron emails. For each document in BOW1, we divide the number of occurrences of each word by the total number of words in the document to obtain the

Table 2.2: Statistics of the datasets (after modification)

Abbreviation	# of objects	# of unique objects	# of features
NEU	961	961	800
LER	20,000	18,668	16
BOW1	41,361	37,993	33,781
ADU	45,222	45,170	88
COV	581,012	581,012	54
CKR	1,000,000	999,910	2
RLC	5,749,132	5,749,133	16
BOW2	8,544,543	3,087,117	100
RNG	10,000,000	10,000,000	20

relative frequencies. BOW2 contains the New York Times articles and the PubMed abstracts. For BOW2, we limit the feature space to the 100 words with the largest absolute difference in mean relative frequency between the New York Times articles and the PubMed abstracts. A binary vector indicates which of those words occur in the respective document.

The dataset *Checkerboard* (CKR) is an artificial dataset that has been used for evaluating large-scale SVM implementations [Lee and Mangasarian, 2001; Tsang et al., 2005]. Each object represents a point on a 4×4 checkerboard that is characterized by two features which represent the x- and the y-coordinates. Following [Tsang et al., 2005], we create a checkerboard dataset with 1 million objects by randomly choosing the x- and y-coordinates of objects between -2 and 2 according to a uniform distribution.

The dataset *Covertype* (COV) [Blackard and Dean, 1999] contains cartographic characteristics of forest cells in northern Colorado. There are seven different cover types which are labeled 1 to 7.

The dataset *Letter Recognition* (LER) [Frey and Slate, 1991] comprises 20,000 objects. Each object corresponds to an image of a capital letter from the English alphabet.

The dataset *Neuron* (NEU) [CodeNeuro, 2017] is a calcium imaging recording of a neuron, a brain cell. The objects are the pixels in the recording and the features are the intensity of a pixel for each of the frames. Sparse computation is used as a subroutine in a leading algorithm for cell identification in calcium imaging recordings [Spaen et al., 2019].

In the dataset *Record Linkage Comparison Patterns* (RLC) [Schmidtmann et al., 2009], the objects are comparison patterns of pairs of patient records. We substitute the missing values with value zero and introduce a binary feature to indicate missing values. The objects can be classified as match or no-match.

The dataset *Ringnorm* (RNG) is an artificial dataset that has been used in [Breiman, 1996] and [Dong, Jian-xiong et al., 2005]. The objects are points in a 20-dimensional space and belong to one of two Gaussian distributions. Following the procedure of [Dong, Jian-xiong et al., 2005], we generate a Ringnorm dataset instance with 10 million objects.

Experimental design

We compared the runtime of sparse computation with block enumeration, object shifting, and block shifting on the nine datasets for different grid resolutions k. For large datasets, it was impossible to construct the complete similarity matrix due to memory limitations of our machine. The grid resolutions used for each of the datasets are described in Table 2.3. We chose the number of dimensions p of the low-dimensional space to be 3 for all datasets except for CKR for which we chose 2 because it only has two features. For the dimension reduction step, we used approximate PCA [Hochbaum and Baumann, 2016] which is much faster than exact PCA as it reduces the size of the original dataset by sampling a small fraction of objects and features. Here we set the sampling fraction for objects to one percent and the sampling fraction for features to five percent unless the number of features is less than 150 in which case all features were used. Only the NEU and BOW1 datasets have more than 150 features.

Table 2.3: Grid resolutions reported for each dataset

Abbreviation	Grid resolution k
NEU	4, 10, 20, 50, 100, 200
LER	4, 10, 20, 50, 100, 200
BOW1	4, 10, 20, 50, 100, 200, 500, 1000, 2000
ADU	4, 10, 20, 50, 100, 200, 500, 1000, 2000
COV	50, 100, 200, 500, 1000, 2000
CKR	100, 200, 500, 1000, 2000
RLC	200, 500, 1000, 2000, 4000, 10000
BOW2	200, 500, 1000, 2000, 4000, 10000
RNG	200, 500, 1000, 2000, 4000, 10000

We ran each combination of algorithm, dataset, and grid resolution five times and recorded the mean runtime and the standard deviation across all five runs. In addition, we recorded various statistics that affect the runtime of the algorithms. The computational analysis was performed on a workstation with Intel Xeon CPUs (model E5-2667 v2) with clock speed 3.30 GHz and 256 GB of RAM.

Experimental results

The mean and standard deviations of the runtimes of the different algorithms are reported in Figure 2.5 and for the largest three datasets in Table 2.4. Sparse computation with block enumeration performs well at low grid resolutions, but gets slow when applied with high grid resolutions. In contrast, sparse computation with object shifting performs poorly at low grid resolutions because there is a large number of pairs consisting of objects that fall into the same sub-block and are selected for each shift. Sparse computation with object shifting performs well when the grid resolution is high. At these grid resolutions, few objects fall in the same sub-block and fewer duplicate pairs are selected.

Sparse computation with block shifting combines the properties of sparse computation with block enumeration and object shifting and provides (near-) best runtime across datasets and grid resolutions. Although outperformed by object shifting at extremely high grid resolutions, it does not suffer from the pitfalls that affect block enumeration and object shifting. By replacing each sub-block by a single representative, it significantly reduces the number of duplicate pairs. This results in faster runtimes for low grid resolutions when each block typically contains multiple objects. Figure 2.6 shows that block shifting identifies only few duplicate pairs for low grid resolutions. For all datasets block shifting identifies, on average, a pair no more than twice independent of the grid resolution.

An interesting insight can be gained from the CKR dataset: although object shifting generates more duplicate pairs than block shifting, it is still slightly faster for very high grid resolutions. This is most likely due to the overhead resulting from generating the

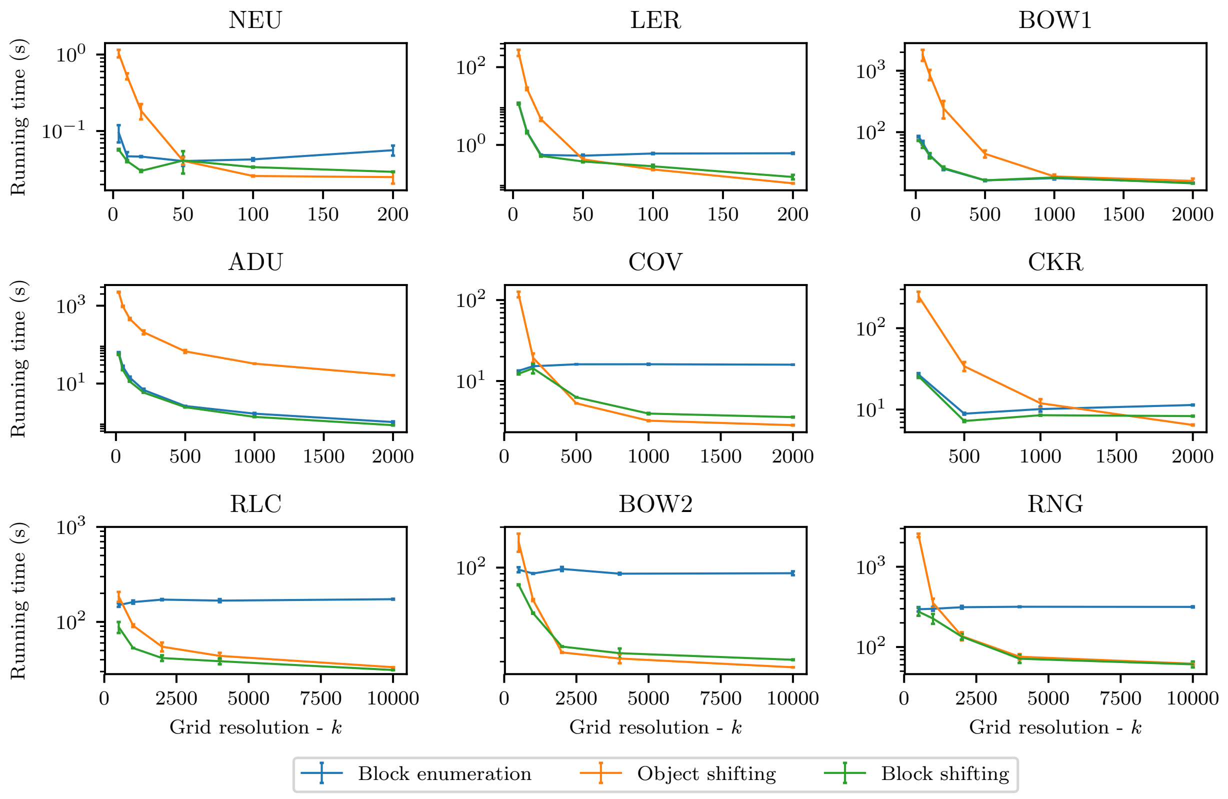


Figure 2.5: Mean runtimes, measured in log scale, across datasets for sparse computation with block enumeration, object shifting, and block shifting. The error bars indicate the standard deviation of the runtimes. Grid resolution is measured with respect to sparse computation with block enumeration.

Table 2.4: Mean runtime for RLC, BOW2, and RNG for selected grid resolutions

Algorithm	**Dataset**								
	RLC			BOW2			RNG		
	$k =$ 500	$k =$ 2000	$k =$ 10000	$k =$ 500	$k =$ 2000	$k =$ 10000	$k =$ 500	$k =$ 2000	$k =$ 10000
Block enumeration	150 ± 6	172 ± 2	173 ± 1	96 ± 4	98 ± 3	91 ± 2	296 ± 18	313 ± 12	316 ± 5
Object shifting	185 ± 22	55 ± 5	33 ± 0.07	155 ± 23	**23** ± 0.2	**18** ± 0.04	2466 ± 122	137 ± 15	62 ± 2
Block shifting	**88** ± 11	**42** ± 2	**31** ± 0.2	**74** ± 0.6	26 ± 0.2	21 ± 0.1	**276** ± 30	**135** ± 10	**61** ± 5

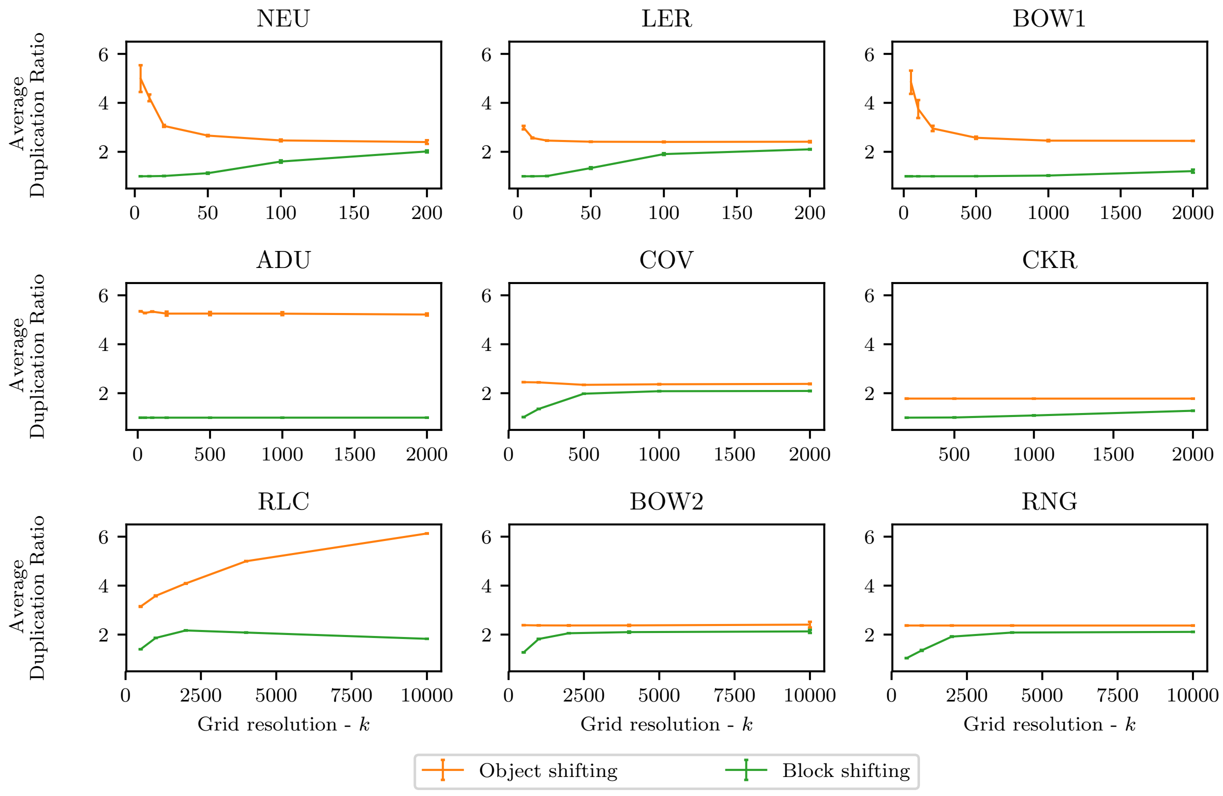


Figure 2.6: Average duplication ratio across datasets for sparse computation with block and object shifting. The duplication ratio is computed as `1 + # duplicate objects or representative pairs / # unique object pairs`. The error bars indicate the standard deviation in the number of duplicate pairs. Grid resolution is measured with respect to sparse computation with block enumeration.

representatives and mapping representatives to objects. In the CKR dataset, the objects are uniformly spread, which means that at a grid resolution of $k = 1,000$, there are 1 million blocks ($p = 2$), that are mostly non-empty. Since there are only 1 million objects, each block contains a single object in expectation. Therefore, block shifting gains little by replacing the objects in each block by a representative but still incurs the overhead.

The computational effort required by sparse computation also depends on the number of dimensions p of the low-dimensional space. For example, the number of (adjacent) blocks, and the number of required shifts for object shifting and block shifting grow exponentially in p. To explore the impact of the dimension of the low-dimensional space on runtime, we applied the different algorithms to the COV dataset with p = 2, 3, and 4. As shown in Figure 2.7, the runtime of sparse computation with block enumeration increases steadily as p is increased even though the number of selected pairs decreases with increasing grid resolution. Sparse computation with object shifting and block shifting scales much better

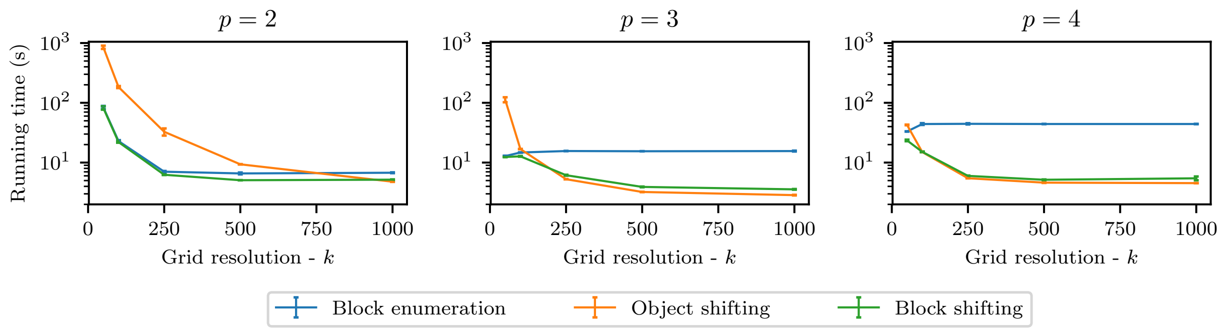


Figure 2.7: Effect of the number of dimensions in the low-dimensional space, p, on the mean runtimes of sparse computation with block enumeration, object shifting, and block shifting for the COV dataset. The error bars indicate the standard deviation of the runtimes. Grid resolution is measured with respect to sparse computation with block enumeration.

with increasing values of p.

2.4 Compression for Massively-Large Datasets: Sparse-Reduced Computation

Sparse-reduced computation is an extension of sparse computation. The idea of sparse computation is to avoid the computation of very small similarities as they are unlikely to affect the classification result. Sparse-reduced computation not only avoids the computation of very small similarities, but also avoids the computation of similarities between highly-similar and identical objects. Intuitively, sparse-reduced computation not only "rounds" very small similarities to zero but it also "rounds" very large similarities to one. Sparse-reduced computation consists of the four steps of data projection, space partitioning, data reduction, and similarity computation.

Data projection: The first step in sparse-reduced computation coincides with what is done in sparse computation. The d-dimensional dataset is projected onto a p-dimensional space, where $p \ll d$. This is implemented using a sampling variant of principal component analysis called Approximate PCA [Hochbaum and Baumann, 2014]. Approximate PCA computes principal components that are very similar to those computed by exact PCA as shown in [Drineas et al., 2006], yet requires drastically reduced running time. Approximate PCA is based on a technique devised by [Drineas et al., 2006]. The idea is to compute exact PCA on a submatrix W of the original matrix A. The submatrix W is generated by random selection of columns and rows of A with probabilities proportional to the L_2 norms of the respective column or row vectors.

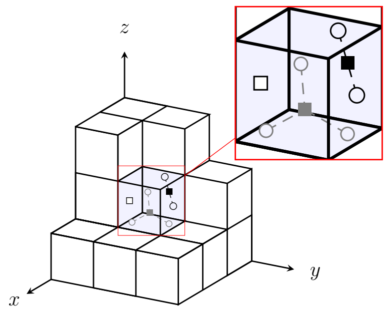


Figure 2.8: Data reduction with resolution $k = 3$ and $\underline{\kappa} = 1$. As shown in the magnified block, the negative training objects (gray), the positive training objects (white fill), and the testing objects (black) are each replaced by a single representative.

Space partitioning: Once all objects are mapped into the p-dimensional space, the next step is to subdivide, in each dimension, the subspace occupied by the objects into k intervals of equal length. This partitions the p-dimensional space into k^p grid blocks. Using a uniform value of k for all dimensions allows us to control the total number of grid blocks with a single parameter. In the following, parameter k is referred to as the grid resolution. Each object is assigned to a single block based on the respective intervals in which its p coordinates fall.

Data reduction: The grid is then used to reduce the size of the dataset by replacing so-called δ-identical objects by a single representative. Let $\delta = \frac{1}{\underline{\kappa}}$ for $\underline{\kappa} \in \mathbb{N}$. In order to identify δ-identical objects we subdivide each block into $\underline{\kappa}^p$ sub-blocks. The sub-blocks are obtained by partitioning the grid block along each dimension into $\underline{\kappa}$ intervals of equal length. For each sub-block, we replace objects of the same type (negative training objects, positive training objects and testing objects) by a single representative. For example, if $\underline{\kappa} = 1$, then all objects of the same type that fall in the same grid block are considered δ-identical and are thus grouped together. If $\underline{\kappa} = 2$ for a three-dimensional space, then the grid block is split into $2^3 = 8$ sub-blocks and the replacement of objects by representatives is done for each sub-block separately. Different values for $\underline{\kappa}$ can be selected for different blocks to account for an unequal distribution of the data. The representatives are computed as the center of gravity of the corresponding objects and have a multiplicity weight equivalent to the number of objects they represent. Note that a representative may represent a single object. Figure 2.8 illustrates the data reduction step for a three-dimensional grid and $\underline{\kappa} = 1$.

Similarity computation: The sparse similarity matrix on the representatives is computed based on the concept of grid neighborhoods. This concept is borrowed from image segmentation where similarities are computed only between adjacent pixels [Hochbaum et al., 2013]. Here, we use the space partitioning and first consider all representatives in the same block to be adjacent and thus have their similarities computed. Then, adjacent blocks are identified and the similarities between representatives in those blocks are computed. Two blocks are adjacent if they are within a one-interval distance from each other in each dimension (the L_∞ metric). Hence, for each block, there are up to $3^p - 1$ adjacent blocks. The similarities are computed in the original d-dimensional space. For very high-dimensional datasets, the similarities could also be computed in the low p-dimensional space. A finer grid resolution (higher value of k) generally leads to a lower density of the similarity matrix. Notice that for $k = 2$ all representatives are neighbors of each other, and thus we get the complete similarity matrix. The set of representatives and the generated similarity matrix constitutes the input to the classification algorithms. The class labels that are assigned to representatives will be passed on to each of the objects that they represent.

2.5 Experimental Analysis of Sparse-Reduced Computation

We apply both sparse computation and sparse-reduced computation with two similarity-based classifiers to five datasets. A comparison with existing sparsification approaches is not possible because these approaches require to first compute the full similarity matrix, which would exceed the memory capacity of our machine.

Similarity-based classifiers

We present two similarity-based machine learning techniques as binary classifiers that assign a set of testing objects to either the positive or the negative class based on a set of training objects. The two techniques are the K-nearest neighbor algorithm and the supervised normalized cut algorithm [Hochbaum, 2010].

K-nearest neighbor algorithm (KNN): The KNN algorithm [Fix and Hodges, 1951] finds the K nearest training objects to a testing object and then assigns the predominant class among those K neighbors. We use the euclidean metric to compute distances between objects and the nearest training object is used to break ties. When KNN is applied with sparse-reduced data, we consider the multiplicity weight of the representatives to determine the K nearest training representatives. For example, if $K = 3$ and the nearest training representative is positive and has a multiplicity weight of 2 and the second nearest training representative is negative and has a multiplicity weight of 5, the testing object is assigned to the positive class. A testing object is assigned to the negative class if all similarities to training objects are zero. In the experimental analysis we treat K as a tuning parameter.

Supervised normalized cut (SNC): SNC is a supervised version of HNC (Hochbaum's Normalized Cut), which is a variant of normalized cut Hochbaum [2010]. We provide a detailed discussion of HNC in sectionsection 3.1, but we will give a short summary here for the purpose of these experiments.

HNC is defined on an undirected graph $G = (V, E)$, where V denotes the set of nodes and E the set of edges. A weight w_{ij} is associated with each edge $[i, j] \in E$. A bi-partition of a graph is called a *cut*, $(S, \bar{S}) = \{[i, j] \mid i \in S, j \in \bar{S}\}$, where $\bar{S} = V \setminus S$. The *capacity of a cut* $(S, \bar{S})$ is the sum of weights of edges with one endpoint in S and the other in $\bar{S}$:

$$C(S, \bar{S}) = \sum_{i \in S, j \in \bar{S}, [i,j] \in E} w_{ij}.$$

In particular, the *capacity of a set*, $S \subset V$, is the sum of edge weights within the set S,

$$C(S, S) = \sum_{i,j \in S, [i,j] \in E} w_{ij}.$$

In the context The nodes of the graph correspond to objects in the dataset and the edge weights w_{ij} quantify the similarity between the respective feature vectors associated with nodes i and j. Higher similarity is associated with higher weights.

The goal of one variant of HNC is to find a cluster that minimizes a linear combination of two criteria. One criterion is to maximize the total similarity of the objects within the cluster (the intra-similarity). The second criterion is to minimize the similarity between the cluster and its complement (the inter-similarity). A linear combination of the two criteria is minimized:

$$\min_{\emptyset \subset S \subset V} C(S, \bar{S}) - \lambda C(S, S). \tag{2.1}$$

The relative weighting parameter λ is one of the tuning parameters.

The input graph for SHNC contains labeled nodes (training data) whose class label (either positive or negative) is known and unlabeled nodes. SNC is derived from HNC by assigning all labeled nodes with a positive label to the set S and all labeled nodes with a negative label to the set $\bar{S}$. The goal is then to assign the unlabeled nodes to either the set S or the set $\bar{S}$. For a given value of λ, this optimization problem is solved with a minimum cut procedure in polynomial time [Hochbaum, 2013b].

We implement SNC with Gaussian similarity weights. The Gaussian similarity between object i and j is defined as:

$$w_{ij} = e^{-\alpha ||x_i - x_j||_2},$$

where parameter α represents a scaling factor, and x_i and x_j are the feature vector of objects i and j. The Gaussian similarity function is commonly used in image segmentation, spectral clustering, and classification. When SNC is applied with sparse-reduced computation, we multiply each similarity value by the product of the weights of the two corresponding representatives. There are two tuning parameters: the relative weighting parameter of the two objectives, λ, and the scaling factor of the exponential weights, α. The minimum cut problems are solved with the MatLab implementation of the HPF pseudoflow algorithm version 3.23 of [Chandran and Hochbaum, 2009] that was presented in [Hochbaum, 2008].

Table 2.5: Datasets statistics (after modifications)

Abbr	# Objects	# Features	$\frac{\text{\# Positives}}{\text{\# Negatives}}$
COV	581,012	54	0.574
KDD	4,898,431	122	4.035
RLC	5,749,132	16	0.004
BOW2	8,499,752	234,151	0.037
RNG	10,000,000	20	1.000

datasets

We select four real-world datasets from the UCI Machine Learning Repository [Lichman, 2013] and one artificial datasets from previous studies [Breiman, 1996; Dong, Jian-xiong et al., 2005]. We substituted categorical features by a binary feature for each category. In the following, we briefly describe each dataset and mention further modifications that are made. The characteristics of the adjusted datasets are summarized in Table 2.5.

In the *Bag of Words* (BOW2) dataset, the objects are text documents from two different sources (New York Times articles and PubMed abstracts). A document is represented as a so-called bag of words, i.e. a set of vocabulary words. For each document, an vector indicates the number of occurrences of each word in the document. We treated New York Times articles as positives.

The dataset *Covertype* (COV) contains cartographic characteristics of forest cells in northern Colorado. There are seven different cover types which are labeled 1 to 7. Following [Caruana and Niculescu-Mizil, 2006], we treat type 1 as the positive class and types 2 to 7 as the negative class.

The dataset *KDDCup99* (KDD) is the full dataset from the KDD Cup 'twork. Each connection is labeled as either normal, or as an attack. We treat attacks as the positive class.

In the dataset *Record Linkage Comparison Patterns* (RLC), the objects are comparison patterns of pairs of patient records. We substitute the missing values with value 0 and introduce an additional binary attribute to indicate missing values. The goal is to classify the comparison patterns as matches (the corresponding records refer to same patient) or non-matches. Matches are treated as positive class.

The dataset *Ringnorm* (RNG) is an artificial dataset that has been used in [Breiman, 1996] and [Dong, Jian-xiong et al., 2005]. The objects are points in a 20-dimensional space and belong to one of two classes. Following the procedure of [Dong, Jian-xiong et al., 2005], we generate a Ringnorm dataset instance with 10 million objects where each object is equally likely to be in either the positive or the negative class.

For each of the five datasets, we generate five subsets by randomly sampling 5,000, 10,000, 25,000, 50,000, and 100,000 objects from the full dataset.

Experimental Design

Sparse computation and sparse-reduced computation are tested with the two similarity-based machine learning algorithms KNN, and SNC. In the following, we use the term classifier to refer to a combination of a machine learning algorithm and sparse/sparse-reduced computation. The performance of the classifiers is evaluated in terms of accuracy (ACC) and F1-scores (F1). The experimental analysis is implemented in MatLab R2014a and the computations are performed on a standard workstation with two Intel Xeon CPUs (model E5-2687W) with 3.10 GHz and 128 GB RAM.

Tuning & Testing We randomly partition each dataset into a training set (60%), a validation set (20%) and a testing set (20%). The union of the training and the validation sets forms the tuning set which is used as input for a given classifier. We tune each classifier with and without first normalizing the tuning set. Normalization prevents that features with large values dominate distance or similarity computations [Witten and Frank, 2005]. For each of the two resulting similarity matrices the classifier is applied multiple times with different combinations of tuning parameter values. For KNN, we selected tuning parameter K from the set $\{1, 2, \ldots, 25\}$. For SNC, we selected tuning parameter α from the set $\{1, 3, \ldots, 15\}$ and the tuning parameter λ from the set $\{0, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$.

The testing is performed on the union of the training and the testing sets, i.e., the same training objects are used for tuning and testing. Unlike most machine learning algorithms that first train a model based on the training data and then apply the trained model to the testing data, KNN and SNC require the training data for classifying the testing data. For a given classifier, and performance measure, we select the combination of preprocessing option and tuning parameter values which achieve the best performance with respect to the given performance measure for the validation set. The performance measure is then reported for the testing set.

Approximate PCA The fraction of rows selected for approximate PCA is one percent for all subsets of the datasets. With this fraction, approximate PCA requires less than a second of CPU time for all datasets, and the produced principle components are close to the principle components returned by exact PCA as determined by manual inspection. For all subsets of datasets other than BOW2, all features (columns) are retained in the submatrix W. For BOW2, we select the 100 columns that correspond to the words with the highest absolute difference between the average relative frequencies of the word in the positive and the negative training documents. In that, we deviate from the probabilistic column selection described in section 2.4. For all subsets of BOW2, we compute the similarities in the low-dimensional space, i.e., with respect to these 100 most discriminating words. Thereby, we first discretize the feature vectors by replacing all positive frequencies with value 1.

Numerical Results

We compare sparse computation and sparse-reduced computation in terms of scalability by applying both methods with the same grid resolution of $k = 40$ and the same grid dimensionality of $p = 3$ to all five datasets. With these values, the low-dimensional space is partitioned into 64,000 blocks which is a good starting point for both methods across datasets. For sparse-reduced computation, parameter $\delta = \frac{1}{\kappa}$ is set to 1.

Tables 2.6–2.10 show the results for each of the tested datasets and both classifiers. The entry "lim" indicates that MatLab ran out of memory. This happens when we applied sparse computation to the complete datasets. For the dataset KDD, MatLab already runs out of memory for a sample size of 100,000.

The main conclusion from these tables is that sparse-reduced computation scales orders of magnitude better than sparse computation while achieving almost equally high accuracy for all datasets except COV (to be discussed below). The running time reduction is most impressive for KDD and RLC. For these two datasets both sparse and sparse-reduced computation perform equally well but the tuning times obtained with sparse-reduced computation are up to 1,000 times smaller. Similar results have been obtained for datasets BOW2 and

Table 2.6: Comparison of sparse and sparse-reduced computation for dataset COV.

	Sparse computation						Sparse-reduced computation							**Speed-up**	
	Accuracy [%]		F$_1$-score [%]		Tuning time [s]		Accuracy [%]		F$_1$-score [%]		Tuning time [s]		# Reps		
Sample size	KNN	SNC	KNN	SNC	KNN	SNC	KNN	SNC	KNN	SNC	KNN	SNC		KNN	SNC
5,000	78.50	80.10	70.26	72.63	2.1	2.6	72.20	72.90	61.13	59.42	2.5	2.6	1,981	**0.8**	**1.0**
10,000	81.80	82.15	73.74	72.89	3.6	5.5	73.35	73.55	58.64	60.37	3.7	3.9	3,060	**1.0**	**1.4**
25,000	88.04	87.94	83.51	83.23	8.3	19.6	69.78	72.46	57.85	58.74	4.5	4.8	3,504	**1.8**	**4.1**
50,000	91.42	91.90	88.22	88.81	24.5	75.7	72.24	73.31	59.12	63.06	6.9	7.3	5,097	**3.6**	**10.4**
100,000	94.04	94.20	91.74	91.98	60.1	195.9	72.37	74.06	59.88	61.28	12.3	13.1	5,427	**4.9**	**15.0**
581,012	lim	lim	lim	lim	lim	lim	73.19	73.80	58.93	61.11	17.1	17.6	4,058	-	-

Table 2.7: Comparison of sparse and sparse-reduced computation for dataset KDD.

	Sparse computation						Sparse-reduced computation							**Speed-up**	
	Accuracy [%]		F$_1$-score [%]		Tuning time [s]		Accuracy [%]		F$_1$-score [%]		Tuning time [s]		# Reps		
Sample size	KNN	SNC	KNN	SNC	KNN	SNC	KNN	SNC	KNN	SNC	KNN	SNC		KNN	SNC
5,000	99.60	99.50	99.75	99.69	3.0	11.8	99.60	99.50	99.75	99.69	0.4	0.4	290	**7.5**	**29.5**
10,000	99.80	99.85	99.94	99.91	11.8	53.8	99.85	99.85	99.91	99.91	0.5	0.5	369	**23.6**	**107.6**
25,000	99.82	99.80	99.89	99.87	76.1	373.4	99.78	99.78	99.86	99.86	1.0	1.0	634	**76.1**	**373.4**
50,000	99.83	99.85	99.89	99.91	303.9	1,520.2	99.71	99.74	99.82	99.84	1.6	1.6	876	**189.9**	**950.1**
100,000	lim	lim	lim	lim	lim	lim	99.82	99.84	99.89	99.90	2.7	2.8	1,085	-	-
4,898,431	lim	lim	lim	lim	lim	lim	99.91	99.91	99.94	99.94	165.0	167.0	4,365	-	-

Table 2.8: Comparison of sparse and sparse-reduced computation for dataset RLC.

	Sparse computation						Sparse-reduced computation							**Speed-up**	
	Accuracy [%]		F_1-score [%]		Tuning time [s]		Accuracy [%]		F_1-score [%]		Tuning time [s]		# Reps		
Sample size	KNN	SNC	KNN	SNC	KNN	SNC	KNN	SNC	KNN	SNC	KNN	SNC		KNN	SNC
5,000	100.00	100.00	100.00	100.00	.9	3.2	100.00	100.00	100.00	100.00	0.3	0.4	421	**3.0**	**8.0**
10,000	100.00	100.00	100.00	100.00	3.4	14.8	100.00	100.00	100.00	100.00	0.5	0.5	864	**6.8**	**29.6**
25,000	100.00	99.98	100.00	96.97	21.6	97.2	100.00	99.98	100.00	96.97	0.8	0.9	903	**27.0**	**108.0**
50,000	100.00	100.00	100.00	100.00	94.8	457.1	100.00	100.00	100.00	100.00	1.7	1.8	1,021	**55.8**	**253.9**
100,000	100.00	100.00	99.30	100.00	382.0	1,714.7	100.00	100.00	99.30	100.00	2.0	2.1	1,159	**191.0**	**816.5**
5,749,132	lim	lim	lim	lim	lim	lim	99.99	99.99	98.76	98.82	43.8	51.0	1,309	-	-

Table 2.9: Comparison of sparse and sparse-reduced computation for dataset BOW2.

	Sparse computation						Sparse-reduced computation							**Speed-up**	
	Accuracy [%]		F_1-score [%]		Tuning time [s]		Accuracy [%]		F_1-score [%]		Tuning time [s]		# Reps		
Sample size	KNN	SNC	KNN	SNC	KNN	SNC	KNN	SNC	KNN	SNC	KNN	SNC		KNN	SNC
5,000	98.70	98.80	81.16	82.86	3.2	3.6	98.70	98.80	81.16	82.86	3.4	3.4	2,418	**0.9**	**1.1**
10,000	99.05	98.95	83.76	81.74	6.4	9.0	99.05	98.95	83.76	81.74	5.6	5.7	2,749	**1.1**	**1.6**
25,000	98.38	98.22	70.55	67.16	17.8	30.8	98.56	98.46	74.47	72.20	13.6	14.1	8,115	**1.3**	**2.2**
50,000	99.46	99.51	91.84	92.61	45.0	147.5	99.54	99.56	93.09	93.43	11.6	12.1	5,137	**3.9**	**12.2**
100,000	99.68	99.56	95.13	93.21	126.1	475.7	99.59	99.63	93.72	94.38	20.4	21.1	8,171	**6.2**	**22.5**
8,544,543	lim	lim	lim	lim	lim	lim	99.68	99.67	95.36	95.19	1,181.2	1,196.7	22,287	-	-

Table 2.10: Comparison of sparse and sparse-reduced computation for dataset RNG.

	Sparse computation						Sparse-reduced computation							**Speed-up**	
	Accuracy [%]		F_1-score [%]		Tuning time [s]		Accuracy [%]		F_1-score [%]		Tuning time [s]		# Reps		
Sample size	KNN	SNC	KNN	SNC	KNN	SNC	KNN	SNC	KNN	SNC	KNN	SNC		KNN	SNC
5,000	79.40	81.00	78.18	82.36	3.3	3.6	79.40	81.00	81.33	80.78	4.2	4.3	3,206	**0.8**	**0.8**
10,000	81.25	81.30	82.29	82.67	6.2	7.1	81.55	81.30	82.52	82.75	7.7	8.0	5,467	**0.8**	**0.9**
25,000	79.76	81.46	82.95	83.81	14.3	23.2	81.32	81.42	83.81	83.69	15.1	15.9	8,728	**0.9**	**1.5**
50,000	80.34	82.13	83.65	84.11	32.8	87.5	81.43	82.15	83.87	84.19	22.5	23.2	12,925	**1.5**	**3.8**
100,000	79.81	82.24	82.89	83.61	88.2	335.0	80.76	82.23	82.63	83.60	33.3	35.3	16,518	**2.6**	**9.5**
10,000,000	lim	lim	lim	lim	lim	lim	82.48	83.76	84.12	85.31	1,960.1	1,990.9	37,511	-	-

RNG as reported in Tables 2.9, and 2.10, respectively. For the dataset COV, we observe a reduction in accuracy and F1-scores with sparse-reduced computation. One way to improve accuracy and F1-scores is to select a value greater than 1 for $\underline{\kappa}$. Increasing $\underline{\kappa}$ results in additional representatives per grid block and thus improves the representation of the dataset. Indeed preliminary work has shown improvements in COV results with an adjustment of the sparse-reduced approach.

KNN and SNC achieve very similar accuracies and F1-scores for all datasets. KNN has smaller tuning times than SNC because a smaller number tuning parameter combinations is tested. Apart from the number of tuning parameter combinations, the tuning time is driven by the size and the density of the similarity matrix. This can be seen from the relation between the tuning times and the number of representatives in Tables 2.6–2.10. In sparse computation the size of the similarity matrix is proportional to the sample size. On the contrary, in sparse-reduced computation, the size of the similarity matrix is determined by the number of representatives. When there are large groups of highly-similar objects as is the case in the datasets KDD and RLC, the number of representatives is significantly lower than the number of objects and an increase in sample size tends to increase the multiplicity weight of the representatives but not the number of representatives.

2.6 Conclusions

We introduced two new algorithms for sparse computation that utilize a new computational geometry concept called data shifting. These new algorithms enable the use of sparse computation on very-large datasets. The use of data shifting allows the algorithms to find pairs of close objects in a low-dimensional space faster than in the original implementation. Based on real-world datasets with up to 10 million objects, we demonstrate that sparse computation with block shifting provides overall the best runtime performance across grid resolutions for datasets of up to 10 million objects.

Furthermore, we proposed a novel method named sparse-reduced computation that enables similarity-based machine learning on massive datasets by compressing highly-similar objects with only a minor loss of information. This is achieved by efficiently projecting the original dataset onto a low-dimensional space. In the low dimensional space, a grid is used to identify highly-similar and identical objects, which are replaced by representatives to reduce the size of the dataset. A set of computational experiments demonstrates that sparse-reduced computation achieves significant reductions in running time with minimal loss in accuracy.

Promising directions for future research include the application of data shifting in other contexts, such as approximate nearest neighbor search and grid-based clustering, as well as a variant of sparse-reduced computation where the degree of consolidation into representatives is dependent on local properties of the objects in the low-dimensional space.

Chapter 3

Cell Identification in Calcium-Imaging Movies with HNCcorr

In this chapter we study how the clustering model HNC is applied to an important application in neuroscience: The identification of cells in calcium imaging movies. To our knowledge, it is the first time that similarity-based machine learning methods and combinatorial optimization are used for this problem. We demonstrate here how HNC is adapted to an algorithm, named HNCcorr, for cell identification in two-photon calcium imaging movies. The name HNCcorr is derived from HNC and the use of a similarity measure based on correlation.

Calcium imaging is an imaging technique used by neuroscientists for measuring calcium concentrations at a cellular resolution in live and behaving animals [Stosiek et al., 2003]. It is commonly used to simultaneously record the neuronal activity of thousands of neurons, the primary component of the central nervous system. When a neuron activates (fires), calcium ions are released into the cell. The cell then becomes fluorescent due to calcium sensitive, genetically modified proteins. After the spike, the calcium concentration rapidly decays to normal levels. These changes in fluorescence are recorded with fast-scanning laser microscopes, producing movies of the fluorescence of cells in a slice of the brain. In these movies, a pixel is characterized by a fluorescence signal value which changes over the length of the movie. From these movies, researchers extract the activity signals from individual neurons by estimating the time-varying fluorescence of these cells. The extracted activity signals are then used as the starting point for all subsequent analyses to test hypotheses about the function of neurons and neural circuits in relation to the behavior of the animal.

A crucial step in the analysis of calcium imaging datasets is to identify the locations of the individual cells in the movie in order to extract their signal. This *cell identification* problem consists of outlining the spatial footprint, represented by a set of pixels, of each cell in the imaging plane. The identification is complicated by high noise levels in the recorded movie, the presence of out-of-focus fluorescence sources above or below the imaging plane, and the fact that most cells are only visible when they activate. Our interest here is in active cells with measurable changes in fluorescence. The task of cell identification is cast here as a clustering problem of constructing a cohesive cluster of pixels that are distinct from their environment. The spatial footprint of a cell is such a cluster since the pixels in the footprint share the unique time-varying signal of the cell.

If done manually, the process of identifying cells requires hours of manual user input for each dataset, which, accumulated over a single project, adds up to hundreds of hours of time committed by researchers. Hence, a number of automated methods have been developed for cell identification in calcium imaging movies. These existing techniques can be classified into three classes: Semi-manual region of interest (ROI) detection [Driscoll et al., 2017; Kaifosh et al., 2014], shape-based detection algorithms [Apthorpe et al., 2016; Gao, 2016; Klibisz et al., 2017; Pachitariu et al., 2013], and matrix factorization algorithms [Diego-Andilla and Hamprecht, 2014; Levin-Schwartz et al., 2017; Maruyama et al., 2014; Mukamel et al., 2009; Pachitariu et al., 2017; Pnevmatikaki and Paninski, 2013; Pnevmatikakis et al., 2016]. Semi-manual ROI detection techniques rely on user's input for detecting and segmenting cells. This process has been reported to be highly labor-intensive [Resendez et al., 2016] and may miss cells with a low signal to noise ratio or a low activation frequency. Shape-based identification methods locate the characteristic shape(s) of cells using deep learning [Apthorpe et al., 2016; Gao, 2016; Klibisz et al., 2017] or dictionary learning [Pachitariu et al., 2013]. Shape-based techniques are typically applied to a summary image of the movie obtained by removing the time dimension and compressing the movie to a single mean intensity image. The third class of techniques uses a matrix factorization model to decompose a movie into the spatial and temporal properties of the individual neuronal signals. The use of the matrix factorization algorithm CNMF [Pnevmatikakis et al., 2016] is currently prevalent for the task of cell identification.

A major distinction of HNCcorr as compared to most alternative algorithms is that HNC is solved efficiently and to global optimality [Hochbaum, 2010, 2013b]. The optimality guarantee of HNC makes the output of the optimization model transparent, since the effect of the model input and parameters on the resulting optimal solution is well understood. In contrast, most other approaches for cell identification, such as matrix factorization algorithms, rely on non-convex optimization models that are intractable. This means that the algorithms cannot find a global optimal solution to their optimization model, but instead they typically find a locally optimal solution dependent on the initial solution. As a result, the algorithms provide no guarantee on the quality of the delivered solutions and cells may remain undetected.

For HNCcorr, we devised a novel method for computing similarities between pairs of pixels named $(\textsc{sim})^2$, *similarity-squared.* The idea of $(\textsc{sim})^2$ is to associate, with each pixel, a feature vector of correlations with respect to a subset of pixels. Similarities between pairs of pixels are then computed as the similarity of the respective two feature vectors. An important feature of $(\textsc{sim})^2$ over regular pairwise correlation is that it considers any two background pixels, pixels not belonging to a cell, as highly similar whereas correlation deems them dissimilar. This improves the clustering since it incentivizes that background pixels are grouped together.

We present here the experimental performance of the HNCcorr on the Neurofinder benchmark [CodeNeuro, 2017] for cell identification in two-photon calcium imaging datasets. This benchmark is currently the only available benchmark that objectively evaluates cell identification algorithms. The datasets in this benchmark are annotated two-photon calcium imaging movies recorded under varying experimental conditions. On this benchmark, HNCcorr achieves a higher average score than two frequently used matrix factorization algorithms CNMF [Pnevmatikakis et al., 2016] and Suite2P [Pachitariu et al., 2017].

We further provide a running time comparison between the MATLAB implementations of HNCcorr, CNMF, and Suite2P. HNCcorr has similar running time performance as Suite2P and is about 1.5 times faster than CNMF. Python and MATLAB implementations of HNCcorr are available at `https://github.com/quic0/HNCcorr`.

The remainder of this chapter is organized as follows: In section 3.1, we provide a detailed discussion of the HNC problem. In section 3.2, we describe the HNCcorr algorithm and how it relates to HNC. We describe the novel similarity measure $(\textsc{sim})^2$ in section 3.2 since the motivation for $(\textsc{sim})^2$ is tied to the problem of cell identification. In section 3.3, we evaluate the experimental performance of HNCcorr on the Neurofinder benchmark datasets. Finally, we present concluding remarks in section 3.4.

3.1 Hochbaum's Normalized Cut (HNC) Model

HNC [Hochbaum, 2010, 2013b] is a intuitive model for similarity-based data mining and clustering. It is applicable for both supervised and unsupervised learning problems. We describe here how HNC is defined and how it relates to Normalized Cut [Shi and Malik, 2000], a well-known problem from image segmentation.

Notation

Let $G = (V, E)$ denote an undirected graph. Let n denote the number of nodes in V, and let m be the number of edges in E. Every edge $[i, j] \in E$ has an associated weight $w_{ij} \geq 0$. We denote the weighted degree of node $i \in V$ by $d_i = \sum_{[i,j]\in E} w_{ij}$. For $A, B \subseteq V$, let $C(A, B) = \sum_{\substack{[i,j]\in E,\\ i\in A,\, j\in B}} w_{ij}$ be the sum of weights of the edges between nodes in the set A and the set B. Note the following relation for undirected graphs: $\sum_{i\in A} d_i = 2C(A, A) + C(A, V \setminus A)$.

Let the set $S \subset V$ be a non-empty set of nodes. Its complement is $\bar{S} = V \setminus S$. The sets $(S, \bar{S})$ form a bi-partition referred to here as a *cut*. The *capacity* of a cut $(S, \bar{S})$ is $C(S, \bar{S})$. Given a source node $s \in V$ and sink node $t \in V$, an s, t-cut is a cut $(S, \bar{S})$ where $s \in S$ and $t \in \bar{S}$. The set S of an s, t-cut $(S, \bar{S})$ is known as the *source set* of the cut, and the set $\bar{S}$ is known as the *sink set* of the cut. The minimum s, t-cut problem on G is to find an s, t-cut that minimizes $C(S, \bar{S})$.

For a directed graph $G_D = (V_D, A)$, let $w_{ij} \geq 0$ denote the arc weight of arc $(i, j) \in A$. We use the same notation as for undirected graphs to denote a cut $(S, \bar{S})$ and its capacity $C(S, \bar{S})$. Note that the capacity $C(S, \bar{S})$ of a cut $(S, \bar{S})$ in a directed graph includes only the sum of weights on arcs from S to $\bar{S}$ and not on arcs in the reverse direction. Similar to undirected graphs, the minimum s, t-cut problem on a directed graph G_D is to find an s, t-cut that minimizes $C(S, \bar{S})$.

The HNC Model

HNC [Hochbaum, 2010, 2013b] is a graph-based clustering model that trades-off two objectives: The homogeneity of the cluster and the distinctness of the cluster to the remaining objects. HNC is defined here on an undirected graph $G = (V, E)$, but it applies to directed graphs as well. The node set V is the set of objects, and the edges in E are pairwise similarity relations between objects. The pairwise similarity along edge $[i, j] \in E$ is measured with a similarity weight $w_{ij} \geq 0$. The objects i and j are considered more similar as w_{ij} increases.

The cluster of interest in HNC is a non-empty set $S \subset V$. We represent the homogeneity of the cluster S with $C(S, S)$, the total sum of similarities between objects in the cluster. We measure distinctness with the capacity of the cut $C(S, \bar{S})$. This is the capacity of the cut $(S, \bar{S})$. The HNC model balances the maximization of $C(S, S)$ and the minimization of $C(S, \bar{S})$.

HNC requires that S contains one or more seed objects, otherwise the optimal solution is to select $S = V$ with $C(S, S) = \sum_{[i,j]\in E} w_{ij}$ and $C(S, \bar{S}) = 0$. Let $S_P \subset V$ denote a non-empty set of *positive seeds* that must be contained in the cluster S. Similarly, $S_N \subset V$ denotes the non-empty set of *negative seeds* that are in $\bar{S}$. These seed sets guarantee that both S and $\bar{S}$ are non-empty. In a supervised context, these seed sets contain the labeled objects in the training data.

The HNC model is to optimize the ratio of distinctness and homogeneity:

$$\min_{\substack{\emptyset \subset S \subset V \\ S_P \in S,\, S_N \in \bar{S}}} \frac{C\left(S, \bar{S}\right)}{C\left(S, S\right)}. \tag{3.1}$$

This objective is equivalent to minimizing the cut capacity $C(S, \bar{S})$ divided by the weighted degree of the nodes in S:

$$\min_{\substack{\emptyset \subset S \subset V \\ S_P \in S,\, S_N \in \bar{S}}} \frac{C\left(S, \bar{S}\right)}{\sum_{i \in S} d_i}, \tag{3.2}$$

as shown next:

Lemma 1 (Hochbaum 2010)**.** *The set of optimal solutions to* (3.1) *and* (3.2) *are identical.*

Proof.

$$\frac{C(S, \bar{S})}{C(S, S)} = \frac{2C(S, \bar{S})}{\sum_{i \in S} d_i - C(S, S')} = \frac{2}{\frac{\sum_{i \in S} d_i}{C(S, \bar{S})} - 1}.$$

The problems are thus equivalent, since they have equivalent objectives and the same set of feasible solutions. □ □

Relation to Normalized Cut

HNC is a close relative of the Normalized Cut (NC) problem. Its use in the context of image segmentation was popularized by Shi and Malik [2000]. The NC problem is defined as:

$$\min_{\emptyset \subset S \subset V} \frac{C\left(S, \bar{S}\right)}{\sum_{i \in S} d_i} + \frac{C\left(S, \bar{S}\right)}{\sum_{i \in \bar{S}} d_i} \tag{3.3}$$

Compared to (3.2), the NC problem has an additional term in the objective. Although this change may seem minor, it has profound impact on the complexity of the problem. In fact, Shi and Malik showed that the Normalized Cut problem is NP-hard [Shi and Malik, 2000] whereas there is a practical, polynomial-time algorithm for the HNC problem [Hochbaum, 2010]. In previous work [Sharon et al., 2006], it was mistakenly assumed that the HNC problem is identical to Normalized Cut and therefore NP-hard.

The Normalized Cut problem is often used as the theoretical motivation for the *spectral clustering* method. Spectral clustering was popularized by Shi and Malik [2000] as a heuristic for the Normalized Cut problem, since spectral clustering solves a continuous relaxation of the Normalized Cut problem. HNC was shown to be a form of discrete relaxation of normalized cut [Hochbaum, 2013b]. Yet, HNC is solved more efficiently than the spectral clustering method. Furthermore, an extensive empirical comparison of HNC to spectral clustering for the purpose of image segmentation was provided by Hochbaum et al. [2013]. The outcomes of that comparison indicate that the HNC solutions are superior, both visually and in terms of approximating the objective of the Normalized Cut problem, to the spectral clustering algorithm.

Linearizing Ratio Problems

A common method to solve ratio problems such as (3.1) and (3.2) is to instead consider an optimization problem over the epigraph of the ratio function. When you minimize a ratio objective over a feasible region X, $\min_{x \in X} \frac{f(x)}{g(x)}$, this optimization problem is equivalent to:

$$\begin{array}{ll} \min\limits_{x, \lambda \geq 0} & \lambda \\ \text{s.t} & \dfrac{f(x)}{g(x)} \leq \lambda \\ & x \in X \end{array} \quad \Leftrightarrow \quad \begin{array}{ll} \min\limits_{x, \lambda \geq 0} & \lambda \\ \text{s.t} & f(x) - \lambda g(x) \leq 0 \\ & x \in X \end{array}$$

This is equivalent to finding the smallest λ such that there is a yes answer to the following λ-question: “Is there a solution $x \in X$ such that $f(x) - \lambda g(x) \leq 0$?” This reduces the optimization problem to a sequence of oracle calls to the λ-question.

For the HNC formulation in (3.1), we can answer the associated λ-question by solving the linear optimization problem in (3.4) and checking whether the objective value is non-positive.

$$\min_{\substack{\emptyset \subset S \subset V \\ S_P \in S,\, S_N \in \bar{S}}} C\left(S, \bar{S}\right) - \lambda C(S, S) \tag{3.4}$$

Similarly, we answer the λ-question for the HNC formulation in (3.2) with the linear optimization problem in (3.5).

$$\min_{\substack{\emptyset \subset S \subset V \\ S_P \in S,\, S_N \in \bar{S}}} C\left(S, \bar{S}\right) - \lambda \sum_{i \in S} d_i \tag{3.5}$$

We note that the linear formulations in (3.4) and (3.5) are alternative formulations of HNC irrespective of their associated ratio problems, since they capture the trade-off between homogeneity and distinctness.

Algorithms for Solving HNC

In section 3.1 we provide integer programming formulations for both (3.4) and (3.5). We show how these formulations imply efficient polynomial time algorithms based on minimum cut for solving linearized HNC with a fixed value of λ [Hochbaum, 2010, 2013b]. In section 3.1, we extend this analysis and demonstrate how (3.5) is solved for all λ values simultaneously in the same complexity, that of a single minimum cut, by using a parametric minimum cut procedure [Hochbaum, 2010, 2013b]. This directly provides an algorithm to solve the ratio version of HNC.

Solving for Fixed λ Values

In this section, we demonstrate how to solve the linearized HNC problems in (3.4) and (3.5) for a fixed value of $\lambda \geq 0$. We first present an integer programming formulation for (3.4). The integer program uses the decision variable x_i to denote whether $i \in V$ is in the cluster S:

$$x_i = \begin{cases} 1 & \text{if } i \in S, \\ 0 & \text{if } i \in \bar{S}. \end{cases}$$

We also define the decision variable z_{ij} to denote whether edge $[i, j] \in E$ is in the cut, whereas the decision variable y_{ij} denotes for edge $[i, j]$ whether i and j are both in the cluster S:

$$z_{ij} = \begin{cases} 1 & \text{if } i \in S, j \in \bar{S} \text{ or } i \in \bar{S}, j \in S, \\ 0 & \text{otherwise,} \end{cases} \qquad y_{ij} = \begin{cases} 1 & \text{if } i, j \in S, \\ 0 & \text{otherwise.} \end{cases}$$

With these decision variables, the following integer programming problem is a formulation of HNC [Hochbaum, 2010]:

$$\begin{array}{lllll} \min & \displaystyle\sum_{[i,j] \in E]} w_{ij} z_{ij} - \lambda \sum_{[i,j] \in E]} w_{ij} y_{ij}, & & & \\ \text{s.t.} & x_i - x_j \leq z_{ij} & \forall [i,j] \in E, & x_j - x_i \leq z_{ji} & \forall [i,j] \in E, \\ & y_{ij} \leq x_i & \forall [i,j] \in E, & y_{ij} \leq x_j & \forall [i,j] \in E, \\ & x_{s_P} = 1 & \forall s_P \in S_P, & x_{s_N} = 0 & \forall s_N \in S_N, \\ & x_i \in \{0,1\} & \forall i \in V, & y_{ij}, z_{ij} \in \{0,1\} & \forall [i,j] \in E. \end{array} \tag{3.6}$$

This integer program is recognized as a monotone integer programing with up to three variables per inequality (IP3) [Hochbaum, 2002]. Monotone IP3 are integer programs with up to three variables per inequality, where in each constraint two of the variables appear with opposite sign coefficients and a third variable, if present, appears only in one constraint. A typical monotone constraint on three variables is $a_{ij}x_i - b_{ij}x_j \leq c_{ij} + z_{ij}$ where a_{ij} and b_{ij} are non-negative. Any monotone IP3 is solvable with a minimum cut on an associated directed graph [Hochbaum, 2002]. The associated graph for the formulation in (3.6) has $m+n$ nodes and up to $4m+2n$ arcs, where n is the number of nodes and m is the number of edges in G. The complexity of solving the integer program in (3.6) is therefore $T(m+n+2, 4m+2n)$. The notation $T(n,m) = O(nm \log \frac{n^2}{m})$ denotes the complexity of solving the minimum s,t-cut problem on a graph with n nodes and m arcs with the HPF (Hochbaum's PseudoFlow) algorithm [Hochbaum, 2008] or the Push-relabel algorithm [Goldberg and Tarjan, 1988].

The linearized version of (3.2) as given in (3.5) translates to another monotone IP3 for which the associated graph is smaller [Hochbaum, 2013b]:

$$\begin{aligned}
\min \quad & \sum_{[i,j]\in E} w_{ij}z_{ij} - \lambda \sum_{i\in V} d_i x_i, & & & & (3.7)\\
\text{s.t.} \quad & x_i - x_j \leq z_{ij} & \forall [i,j] \in E, \quad & x_j - x_i \leq z_{ji} & \forall [i,j] \in E,\\
& x_{s_P} = 1 & \forall s_P \in S_P, \quad & x_{s_N} = 0 & \forall s_N \in S_N,\\
& x_i \in \{0,1\} & \forall i \in V, \quad & z_{ij} \in \{0,1\} & \forall [i,j] \in E.
\end{aligned}$$

The graph associated with the formulation in (3.7) is smaller and contains only $O(n)$ nodes. We now show the construction of the associated graph of this integer program, which we name the *auxiliary* graph [Hochbaum, 2013b].

Let the auxiliary graph be a directed graph $G_A = (V \cup \{s,t\}, A)$. The arc set A consists of the following sets of arcs: For every edge $[i,j] \in E$, add the two arcs (i,j) and (j,i) to A. Both arcs have a capacity of $w_{ij} \geq 0$. We also add arcs (s,i) with capacity λd_i for every node $i \in V$. Finally, for every positive seed $i \in S_P$ we add an infinite capacity arc (s,i), and for every negative seed $j \in S_N$ we add an infinite capacity arc (j,t). A schematic of the auxiliary graph is provided in Figure 3.1.

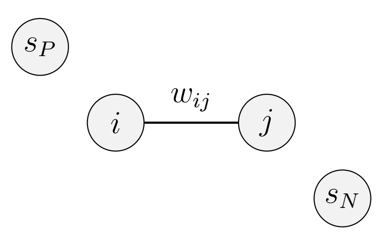
The source set of a minimum s,t-cut in the auxiliary graph G_A is an optimal solution to the linearized HNC problem in (3.5):

Theorem 1 (Hochbaum 2013b)**.** *The source set S^* of a minimum s,t-cut $(S^* \cup \{s\}, \bar{S}^* \cup \{t\})$ in G_A is an optimal solution to the linearized HNC problem in* (3.5) *for a fixed value of λ.*

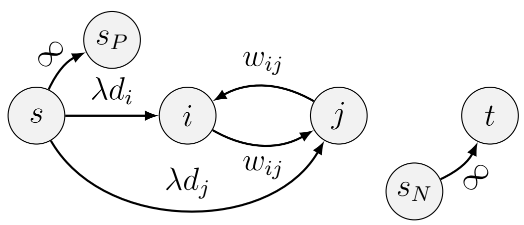
Proof. Let $(S \cup \{s\}, \bar{S} \cup \{t\})$ be any finite s,t-cut in G_A. The infinite capacity arcs (s,i) for $i \in S_P$ guarantee that $S_P \subseteq S$. Similarly, $S_N \subset \bar{S}$. Thus, the set S is a feasible solution for 3.5. G_A always contains a finite cut when the seed sets are disjoint, $S_P \cap S_N = \emptyset$.

The capacity of cut $(S \cup \{s\}, \bar{S} \cup \{t\})$ is equal to:

$$C(S \cup \{s\}, \bar{S} \cup \{t\}) = C(S,\bar{S}) + \lambda \sum_{i\in\bar{S}} d_i = C(S,\bar{S}) + \lambda \left(D - \sum_{i\in S} d_i\right) = \lambda D + C(S,\bar{S}) - \lambda \sum_{i\in S} d_i,$$



(a) Similarity graph G with one edge.



(b) Auxiliary graph G_A.

Figure 3.1: Schematics of a similarity graph G and the associated auxiliary graph G_A for the linearized HNC in (3.5). The node $s_P \in S_P$ is a positive seed, whereas the node $s_N \in S_N$ is a negative seed.

where $D = \sum_{i \in V} d_i$ denotes the sum of weighted degrees. Thus, minimizing the capacity of the cut is equivalent to minimizing $C(S, \bar{S}) - \lambda \sum_{i \in S} d_i$. □ □

The auxiliary graph has $n + 2$ nodes and up to $2m + 2n$, since every edge $[i, j] \in E$ has two associated arcs and every node has up to two associated arcs. One arc connecting it to the source node s with capacity λd_i, and potentially another arc that connects a seed node to either the source node s or to the sink node t.

Corollary 1. *The linearized HNC problem in* (3.5) *for a fixed value of* λ *is solvable in* $O\left(T(n + 2, 2m + 2n)\right)$.

Since the linearized HNC problem in (3.5) provides a more efficient algorithm than the problem in (3.4), we will consider only this formulation and its associated ratio problem in (3.2) moving forward.

Solving Simultaneously for All λ Values

While the previous section describes an algorithm to solve the problem in (3.5) for a fixed value of λ, we now show that we can solve (3.5) simultaneously for all values of λ in the complexity of a single minimum cut, $T(n + 2, 2m + 2n)$, with a parametric minimum cut problem [Gallo et al., 1989; Hochbaum, 2008]. This directly implies that the ratio version of HNC in (3.2) is solved in the same complexity, since these problems are equivalent to finding the smallest λ such that the objective of the linear problem in (3.5) is non-positive.

Parametric minimum cut solves the minimum cut problem as a function of λ on a special type of graph, the *parametric flow graph*. A directed s,t-graph $G(\lambda)$ is a parametric flow graph if all arcs capacities adjacent to the source node s are monotone non-decreasing and all arc capacities adjacent to the sink node t are monotone non-increasing as a function of λ.

The only arcs in G_A whose capacity depends on λ are the arcs adjacent to the source node s. The capacity of arc $(s, i) \in A$ is λd_i and is monotone non-decreasing in λ since d_i, the weighted degree of node i, is non-negative for all nodes $i \in V$. The graph $G_A(\lambda)$ is therefore a parametric flow graph.

It was shown that the source set of a minimum cut in a parametric flow graph as a function of λ are nested:

Lemma 2 (Gallo et al. 1989; Hochbaum 2008)**.** *Let $S^*(\lambda)$ denote the source set of a minimum s,t-cut in the parametric flow graph $G(\lambda)$. Then, for $\lambda_1 < \lambda_2$:*

$$S^*(\lambda_1) \subseteq S^*(\lambda_2).$$

Although the capacity of the cut increases with increased λ, the source set of the minimum cut is strictly incremented at most n times. This implies that there are at most n values of $0 \leq \lambda_1 < \lambda_2 < \cdots < \lambda_\ell$ for which the source set is augmented by at least one node. We refer to such λ-values as *breakpoints*. Let the sets $S_1^* \subset S_2^* \subset \cdots \subset S_\ell^*$ denote the associate minimal source sets of the minimum cut, where S_1^* is the source set of the minimum cut for $\lambda \in [0, \lambda_1]$.

Gallo et al. [1989] showed that the breakpoints and their associated source sets are found in the complexity of a single minimum cut with the push-relabel algorithm [Goldberg and Tarjan, 1988]. Hochbaum [2008] showed that the HPF algorithm can also be used to identify all breakpoints in the complexity of a single minimum cut. These are the only two known algorithms that solve the parametric minimum cut in the complexity of a single minimum cut. Since the auxiliary graph is a parametric flow graph with $n + 2$ nodes and $2m + 2n$ arcs, it implies that solutions to the linearized HNC in (3.5) for all values of λ are found simultaneously in $T(n + 2, 2n + 2m)$.

With the breakpoints λ_i and associated source sets S_i^* for $i = 1, \ldots, \ell$ we are able to solve the ratio version of HNC as well. Recall that these problems are equivalent to finding the smallest λ such that the objective value of (3.5) is non-positive. Since the source sets only change at the breakpoints, it is sufficient to find the smallest λ-breakpoint where the objective value of (3.5) is non-positive. The source set for this breakpoint is the optimal cluster for the ratio version of HNC.

Theorem 2 (Hochbaum 2013b)**.** *The HNC ratio problem in* (3.2) *is solved in* $T(n+2, 2n+2m)$ *with a parametric minimum cut procedure.*

3.2 A Description of the HNCcorr Algorithm

HNCcorr is an adaptation of HNC for cell identification in calcium imaging. It takes as input a motion-corrected calcium imaging movie and outputs the spatial footprints of the cells that were identified in the movie. The resulting spatial footprints can then be used to identify the activity signal of each cell with specialized algorithms (cf. Grewe et al. 2010; Jewell and Witten 2018; Pnevmatikakis et al. 2013; Theis et al. 2016; Vogelstein et al. 2010).

We now give a brief description of HNCcorr. Additional details on each aspect of the algorithm is provided in the appropriate subsections. The HNCcorr algorithm initializes with a nearly-exhaustive ordered list of candidate locations for cells. A candidate location is a single pixel in the imaging plane that is indicative of a potential cell. A description of how the list is generates is provided below. At each iteration, HNCcorr selects the next candidate location in the list for evaluation. This repeats until all candidate locations are processed. Let us refer to the selected pixel as pixel i. First, HNCcorr checks if pixel i was part of a spatial footprint of a previously identified cell. If it is, then pixel i is discarded. This prevents repeated identification of the same cell. If it was not, it constructs a graph on a subset of the pixels and computes the pairwise similarities with $(\text{SIM})^2$ as explained below. For this graph, we solve the HNC problem in (3.5) simultaneously for all λ with the seeds selected as described below. We refer to the clusters for each λ-breakpoint as *candidate clusters*. Based on these candidate clusters, a simple post-processor decided whether a cell was detected and returns its spatial footprint if any. Potential extensions for HNCcorr are discussed at the end of this section.

The pseudocode of the HNCcorr algorithm is provided in algorithm 1. The core elements of the HNCcorr algorithm are the construction of the graph, the computation of similarity weights with $(\text{SIM})^2$, and the associated HNC instance. The other components of the algorithm, e.g. the procedure for selecting candidate locations and the post-processing method for candidate clusters, can be replaced with alternative methods. Python and MATLAB implementations of HNCcorr are available on GitHub at `https://github.com/quic0/HNCcorr`.

Notation

A calcium imaging movie consists of T frames each with $N = n_1 \times n_2$ pixels . We assume that the pixels are numbered in row-major order. Each pixel i has an associated fluorescence vector $\mathbf{x}_i \in \mathbb{R}^T_+$. Furthermore, the pairwise correlation $\text{corr}(u, v)$ between pixel i and j is:

$$\text{corr}(u, v) = \frac{1}{T-1} \frac{(\mathbf{x}_u - \hat{\mu}(\mathbf{x}_u))^\mathsf{T} (\mathbf{x}_v - \hat{\mu}(\mathbf{x}_v))}{\hat{\sigma}(\mathbf{x}_u)\, \hat{\sigma}(\mathbf{x}_v)},$$

where $\hat{\mu}(\mathbf{x})$ is the sample mean of the vector $\mathbf{x}$ and $\hat{\sigma}(\mathbf{x})$ is the sample standard deviation.

Generating Candidate Locations for Cells

The candidate locations are an ordered list of pixels that indicate a potential location of a cell in the dataset. With each candidate location, the goal is to identify a new cell. The candidate locations are generated at the start of the algorithm and processed one at a time. As mentioned, candidate locations are discarded if they are in previously identified cell's footprint. This prevents duplicate segmentations of the same cell.

The HNCcorr algorithm is indifferent to the method used to generate the list of candidate locations. This feature allows for any, possibly parallelized, procedure. Our implementation of HNCcorr uses a nearly-exhaustive enumeration method, so it is unlikely to miss any prospective cell.

Algorithm 1 Overview of the HNCcorr algorithm.

Input: M: Movie with fluorescence vector $\mathbf{x}_i$ for pixels $i = 1, \ldots, n$
Output: Set of spatial footprints of cells
Parameters:

- $n_{\text{patch}} = 31$: Patch size in pixels
- $\gamma = 32\%$: Percentage of pixels selected from the patch for the reference set
- $k = 3$: Size of super pixel for positive seeds
- $n_{\text{neg}} = 10$: Number of negative seeds
- $\rho = 13$ - Radius of the negative seeds circle (depends on m),
- $\alpha = 1$: Scaling factor
- $p = 3$: Dimension of the low-dimensional space for sparse computation
- $\kappa = 35$: Grid resolution for sparse computation

```
function HNCcorr($M$)
    $L \leftarrow$ CandidateLocations($M$)
    $S \leftarrow \emptyset$ // Set of spatial footprints
    $SP \leftarrow \emptyset$ // Set of pixels in spatial footprints
    for $cl \in L$
        if $cl \in SP$ // Skip to prevent duplication
            continue
        Construct $n_{\text{patch}} \times n_{\text{patch}}$ patch of pixels centered at $cl$. $V$ is the set of pixels in the patch. $n = |V|$.
        Let $S_P$ be the $k \times k$ super pixel centered at $cl$.
        Select $S_N$, set of $n_{\text{neg}}$ negative seeds, uniformly from a circle of radius $\rho$ centered at $lc$.

        Sample without replacement pixels $r_1, \ldots, r_{n_R}$ from $V$ for $n_R = \gamma \times n$. // Sample reference set for (sim)$^2$
        for $i \in V$
            $(R_i)_h = corr(i, r_h)$ for $h = 1, \ldots, n_R$.

        Use sparse computation with feature vectors $R_i$, dimension $p$, and resolution $\kappa$ to determine edge set $E$.
        for $[i, j] \in E$
            $w_{ij} = \exp(-\alpha \|R_i - R_j\|_2^2)$.

        Solve linearized HNC in (3.5) on graph $G = (V, E)$ with weights $w_{ij}$ and seed sets $S_P$ and $S_N$.
        Let $S_1, \ldots, S_\ell$ denote the candidate clusters.

        Clean each candidate cluster $S_1, \ldots, S_\ell$ to obtain the cleaned clusters $S_1^C, \ldots, S_\ell^C$.
        $C \leftarrow$ PostProcessor($S_1^C, \ldots, S_\ell^C$) // Select best cluster (empty set if not a cell)
        if $C \neq \emptyset$
            $S \leftarrow S \cup \{C\}$
            $SP \leftarrow SP \cup C$
    return $S$
```

The algorithm first computes, for each pixel, the average pairwise correlation to its 8 neighboring pixels. We refer to this value as the *local correlation* of pixel i:

$$lc_i = \frac{1}{|N(i)|} \sum_{j \in N(i)} \text{corr}(i, j),$$

where $N(i)$ denotes the 8-neighborhood of pixel i and $|N(i)| = 8$. Subsequently, the pixels are partitioned into $n_{grid} \times n_{grid}$ grid blocks starting from the north-west corner. By default, $n_{grid} = 5$. These grid blocks are used for spatial aggregation. For each grid block, it selects the pixel with the highest local correlation. All other pixels are discarded. We sort the remaining pixels from high to low in terms of local correlation. The top p_{seed} percent of the these pixels are selected as candidate locations. A summary of this procedure is provided in algorithm 2. The complexity of this step is $O(N \log N)$, where N is the number of pixels in the dataset.

Algorithm 2 Method for generating candidate locations for cells.

Input: M: Movie with fluorescence vector $\mathbf{x}_i$ for pixels $i = 1, \ldots, n$
Output: List of candidate locations
Parameters:

- $n_{\text{grid}} = 5$: Size of each square grid block
- $p_{\text{seed}} = 40\%$: Percentage of candidate locations kept for processing

```
function GenerateLocations($M$)
    // Compute local correlation
    for $i = 1, \ldots, n$
        $lc_i = \frac{1}{|N(i)|} \sum_{j \in N(i)} \text{corr}(i, j)$

    // Select pixel from each grid block with highest local correlation
    Initialize $L$ as empty list
    Split the pixels into a blocks $B_1, \ldots, B_k$ of $n_{\text{grid}} \times n_{\text{grid}}$ pixels starting at north-west pixel
    for $b = 1, \ldots, k$
        $i^* \leftarrow \arg\max_{i \in GB_b} lc_i$
        Append $i^*$ to $L$

    Sort $L$ in decreasing order of local correlation: $lc_1 \geq lc_2 \geq lc_3 \ldots$
    return first $\lceil p_{\text{seed}} \times |L| \rceil$ pixels in $L$.
```

Typically, p_{seed} is set to 40 percent. This value was decided based on an empirical study. This study indicates that the benefit of increasing it above 40 percent is small for most datasets, whereas it increases running time.

This procedure for selecting candidate location is an enumeration of all possible pixels with two types of speedups: Spatial aggregation and discarding of pixels with low local correlation. These speedups can be deactivate by constructing grid blocks of 1×1 pixel ($n_{\text{grid}} = 1$) and selecting all remaining pixels ($p_{\text{seed}} = 100\%$).

Construction of the Graph

Given a pixel as a candidate location, the algorithm constructs a graph on an $n_{\text{patch}} \times n_{\text{patch}}$ square subset of pixels, which we call a *patch*. The purpose of the patch is to limit the number of cells and the size of the similarity graph. The patch is centered on the input pixel, the candidate location. The patch size n_{patch} should be chosen such that it contains a typical cell. By default, $n_{\text{patch}} = 31$, which is significantly less than the size of a movie, typically $n_1 \times n_2 = 512 \times 512$ pixels.

The algorithm then constructs the graph $G = (V, E)$ consisting of the set of nodes, denoted by V, and the set of edges that represent the pairwise similarity relations. Each edge has an associated similarity weight $w_{ij} \geq 0$. The similarity weights are described in section 3.2. The nodes represent the pixels in the patch. The edge set is determined with sparse computation [Hochbaum and Baumann, 2016], see section 2.1. We use sparse computation to reduce the number of edges. Sparse computation also helps to identify structure in the data as shown in Figure 3.2.

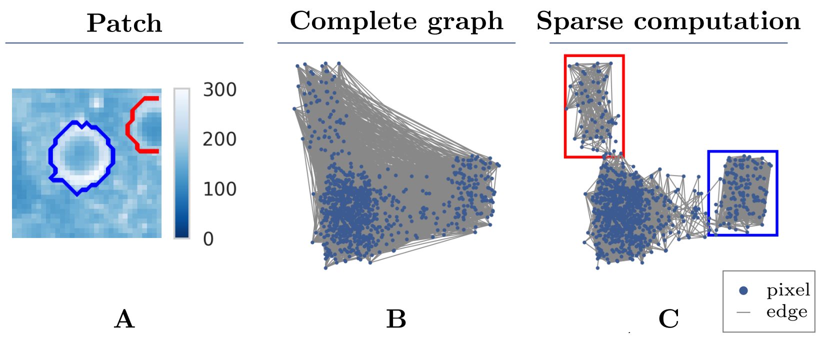


Figure 3.2: Sparse computation constructs a sparse similarity graph. Comparison of a complete similarity graph and the similarity graph constructed by sparse computation for an example patch. For the purpose of illustration, the nodes are positioned based on the 2-dimensional PCA projection of the pixels' feature vectors offset by a small uniformly sampled perturbation. **A)** Mean intensity image of the patch with the outline of two cells marked in red and blue. **B)** Complete similarity graph with an edge between every pair of pixels. For the purpose of illustration, only 10,000 randomly sampled edges are shown. **C)** Sparse similarity graph constructed by sparse computation with a three-dimensional PCA projection and a grid resolution of $\kappa = 25$. Two clusters of pixels (marked with red and blue rectangles) are identified by Sparse Computation. These two clusters match the spatial footprints of the two cells shown in A).

Similarity-Squared: A Novel Similarity Measure

In HNCcorr, the similarity weight w_{ij} associated with edge $[i, j] \in E$ measures the similarity between pixels i and j. We describe here the novel similarity measure $(\textsc{sim})^2$. We devised $(\textsc{sim})^2$ to measure similarity between pixels in calcium imaging movies, but the concept itself is applicable in other contexts as well. $(\textsc{sim})^2$ measures similarities between the similarity patterns. The name $(\textsc{sim})^2$ explains that it measures second-order similarity.

Pairwise Correlation and Background Pixels

A common approach to assess the similarity between two pixels i and j is to measure the correlation $\text{corr}(i, j)$ between their fluorescence signals across all frames of the movie. Correlation is effective for measuring the similarity between two pixels from the same cell, since the pixels have a correlated fluorescence pattern due to the shared signal of the cell. Another important group of pixels is the background pixels. These pixels do not belong to a cell, and they do not typically share a common pattern. Background pixels are only weakly correlated to each other. As such, the similarity between these background pixels is low if the similarity is measured by correlation.

However, the lack of strong correlation with other pixels is effectively what characterizes background pixels. We use this observation to strengthen the similarity measure and aid the clustering. In other words, the background pixels are similar to each other in the pattern of being weakly correlated to every other pixel, whereas pixels that belong to a cell are highly correlated to other pixels in the same cell.

Instead of computing the similarity between pixels directly based on their correlation, we should evaluate the similarity between any two pixels based on how these two pixels correlate with all pixels. Figure 3.3 illustrates this. For a small patch, six pixels are marked in red and are shown with their pairwise correlation to all pixels in the patch. We refer to these images as correlation images. Two pixels in each of the two visible cells are marked as well as two background pixels. The correlation image for each pixel, e.g. pixel 1, shows the pairwise correlation between pixel 1 and every pixel in the patch, represented by the color of the pixel, with a lighter color corresponding to higher correlation. The following observations are drawn from the figure: Pixels belonging to the same cell have nearly identical correlation images (see pixels 1 & 2 and pixels 3 & 4). Furthermore, background pixels also have nearly identical correlation images even though the pixels themselves are not correlated (see pixels 5 & 6).

Similarity between background pixels can thus be captured by comparing their correlation images, whereas correlation or other standard similarity measures would not recognize this. Comparing correlation images instead of computing signal correlation also boosts the similarity between pixels in cells with a low signal-to-noise ratio, such as the cell containing pixel 3 & 4. Whereas pixels 3 and 4 are only weakly correlated, their correlation images are nearly identical. This approach is an application of a novel technique for computing pairwise similarities that we call $(\textsc{sim})^2$.

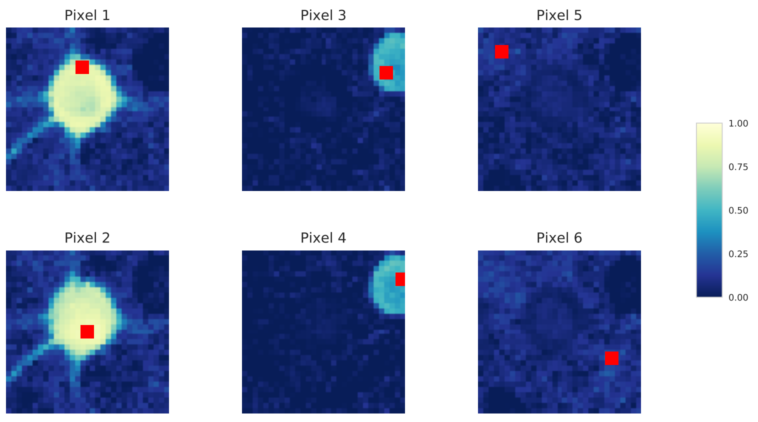


Figure 3.3: Visualization of the correlation images of six pixels. The color (value) of each pixel shown in each correlation image is the pixel-to-pixel correlation between that pixel and the pixel marked in red. Lighter colors represent higher correlations, with the correlation scale truncated at zero. The correlation image is a visualization of the feature vector of a pixel. The patch is taken from the Neurofinder 02.00 training dataset and contains two cells. Pixels 1 and 2 belong to the same cell, pixels 3 and 4 belong to the same cell, and pixels 5 and 6 are background pixels. Even though the pairs of pixels 1 & 2, pixels 3 & 4, and pixels 5 & 6 are not always highly correlated, their correlation images are nearly identical.

Similarity-Squared

The idea of $(\text{SIM})^2$ is to determine the similarity between a pair of objects (here pixels), not by directly comparing their features, but rather by comparing the objects' similarities to a set of reference objects, called the *reference set*. The resulting pairwise similarities between objects can be interpreted as a similarity derived from other similarities, hence the name $(\text{SIM})^2$.

Our use of $(\text{SIM})^2$ in HNCcorr consists of a three-step procedure. In the first step, HNCcorr samples a random subset of pixels. This subset of pixels forms the reference set. Let γ denote the percentage of pixels that are sampled. If n denotes the number of pixels in the patch, then the reference set RS consists of pixels $r_1, \dots, r_{n_R}$ for $n_R = \gamma n$. We use sampling to reduce the running time of the feature vector computation. It has a negligible effect on the cell detection performance of HNCcorr as long as γ is at least 25 percent.

In the second step, we compute the feature vector R_i of pixel i. R_i is a vector of dimension n_R. Mathematically, the k^{th} element of the feature vector R_i of pixel $i \in V$ is defined as $R_i[k] = \text{corr}(i, r_k)$. Assuming that $\gamma = 100\%$, the feature vector can be interpreted as a vector representation of the correlation image of pixel i or as the row of pixel i in the pixel-to-pixel correlation matrix defined over the pixels in the patch.

In the third step, the similarity weights w_{ij} are computed for every edge $[i, j] \in E$. For general $(\text{SIM})^2$, we can use any function of the feature vectors to compute the similarity. We choose here the Gaussian similarity:

$$w_{ij} = \exp\left(-\alpha \|R_i - R_j\|_2^2\right), \tag{3.8}$$

where α is a scaling parameter, which is typically set to 1. Note that $w_{ij} \neq \text{corr}(i, j)$. The complexity of computing all similarity weights is $O\left(\gamma n_{\text{patch}}^2 \left(m + n_{\text{patch}}^2 T\right)\right)$, where m is the number of edges in the similarity graph and T is the number of frames in the movie.

The default value of $\gamma = 32\%$ for sampling was determined based on an experimental evaluation of the cell detection performance and the running time performance. The value of 32% provides a substantial improvement in running time compared to $\gamma = 100\%$ and almost no loss in cell detection accuracy.

The Selection of Seeds for HNC

Recall that HNC requires a set of positive seeds S_P for the cell S and a set of negative seeds S_N for $\bar{S}$. For a given candidate pixel and the associated patch, the algorithm selects a super pixel, a square of $k \times k$ pixels, as positive seed set S_P. The super pixel is centered on the input pixel and is thus located in the center of the patch. k is typically set to 3 or selected by validation on a dataset annotated with cell locations. The super pixel forms the basis of the cell cluster S. It also ensures that HNC attempts to delineate the footprint of the current cell of interest.

In addition to the positive seed, HNC requires also a set of negative seeds S_N. These negative seeds are pixels that cannot be selected as part of the cluster in HNC. The negative seeds ensure that not all pixels are selected. The negative seeds are selected uniformly from a circle of sufficiently large radius centered at the positive seed. The radius ρ of the circle is chosen such that any cell that contains the positive seed is inside the circle. It is typically chosen to be slightly less than $\frac{m}{2}$.

The seeds are selected in $O(1)$ per patch.

Post-Processing

As output of the HNC problem, we obtain all nested optimal clusters $S_1^* \subset S_2^* \subset \ldots S_\ell^*$. These clusters are candidates for the footprint of a cell. A cleaning process is applied to each of the clusters. Pixels that are not contiguous to the positive seed are removed, and pixels that are fully enclosed by pixels in the candidate cluster are added to the cluster. The resulting clusters are ${S_1^*}^C \subset {S_2^*}^C \subset \ldots {S_\ell^*}^C$

The post-processing algorithm selects one of the cleaned candidate clusters as the cell footprint or decides that no cell was detected. First, each cleaned candidate cluster is compared against a user-provided size range, typically 40 to 200 pixels. A cleaned candidate cluster ${S_i^*}^C$ is discarded if the number of pixels in the cluster, $\left|{S_i^*}^C\right|$, falls outside the allowed range. In case all clusters are discarded, then the algorithm returns an empty set, indicating that no cell was detected. Otherwise, each remaining cluster's size is compared to a preferred

Algorithm 3 Size-based post-processor of candidate clusters.

Input: $S^C = \{S_1^{*C} \subset \dots S_\ell^{*C}\}$: Set of cleaned candidate clusters, where each cluster is a set of pixels
Output: Spatial footprint of the cell (empty set if no cell detected)
Parameters:

- n_{min}: Minimum number of pixels in a spatial footprint (dataset dependent)
- n_{avg}: Expected number of pixels in a spatial footprint (dataset dependent)
- n_{max}: Maximum number of pixels in a spatial footprint (dataset dependent)

```
function POSTPROCESSOR($C$)
    for $s \in S^C$
        if $|s| < n_{\text{min}}$ or $|s| > n_{\text{max}}$   // Discard small/large clusters
            $S^C \leftarrow S^C \setminus \{s\}$.
    if $S^C = \emptyset$
        return $\emptyset$
    else
        return $\arg\min_{s \in S^C} \left(\sqrt{|s|} - \sqrt{n_{\text{avg}}}\right)^2$
```

cell size that is set by the user. The candidate cluster which is closest in size to the expected cell size is selected as the spatial footprint of the cell. We use $\sqrt{\left|S_i^{*C}\right|}$ for this comparison since this best reflects the scaling of the area of a circle. This algorithm is summarized in algorithm 3. More complex post-processing techniques based on convolutional neural networks have been explored as well. However, preliminary experiments showed no substantial improvements.

The complexity of this post-processing procedure is $O(n_{\text{patch}}^2 \ell)$.

Extensions for HNCcorr

We discuss here how HNCcorr can be extended for real-time data collection and to counter signal contamination due to overlapping cells.

Online, Real-time Data Collection

Most cell identification algorithms are designed for fully recorded movies, allowing the algorithms to identify cells based on all available data. There is also an interest in online algorithms that detect cells in real-time. Such algorithms enable closed-loop experiments, where the environment is adapted according to previously recorded activity. Real-time detection requires that cell identification algorithms works with streaming data, observing only a single frame at a time.

We sketch here an adaptation of HNCcorr for online, real-time cell detection. Similar to the offline implementation, we use local correlation to identify candidate locations for cells.

The main idea for enabling real-time processing is to implement HNCcorr in parallel, where one process is responsible for maintaining the local correlation estimates, and one or more other processes independently select and evaluate candidate locations to identify cells based on the recorded data so far. This parallel setup requires that only the process for updating the local correlation estimates runs in real-time, whereas the processes for evaluating candidate locations can be run asynchronously.

To see how local correlation can be updated in real-time, we note that a straightforward arithmetic manipulation shows that it is possible to update a pairwise correlation estimate in constant time when a new frame arrives. This makes it possible to update the local correlation estimate for a pixel in constant time. The work per new frame is thus $O(N)$, where N is the number of pixels in the movie. This approach enables the processing of movies with 512×512 pixels, at 100 frames per second or more, on a modern laptop or desktop computer.

Overlapping Cells

As it stands, HNCcorr does not have the capability to demix cell signals of overlapping. However, it tends to identify the part of the spatial footprint that does not overlap with another cell, unless the positive seed happens to be located on the overlap. This makes HNCcorr robust to contamination. In order to further counter signal contamination for overlapping cells, one could locally apply a matrix factorization method, initialized with the footprints identified by HNCcorr, as post-processing.

3.3 Experimental Comparison of Cell Identification Algorithms

To evaluate the experimental performance of HNCcorr we compare against other leading algorithms on the Neurofinder benchmark. Specifically, we compare HNCcorr with the matrix factorization algorithms CNMF [Pnevmatikakis et al., 2016] and Suite2P [Pachitariu et al., 2017] as well as 3dCNN, a 3-dimensional convolutional neuronal network.

3dCNN is an algorithm that only recently appeared on the Neurofinder benchmark. As of the submission of this manuscript, there is no corresponding publication nor is there publicly available code for the algorithm. We requested the code for 3dCNN but have not received a response so far. We therefore excluded the 3dCNN algorithm from the runtime analysis.

Experimental Setup

Datasets

The Neurofinder community benchmark [CodeNeuro, 2017] is an initiative of the CodeNeuro collective of neuroscientists that encourages software tool development for neuroscience research. The collective also hosts the Spikefinder benchmark, which has led to improved algorithms for spike-inference in calcium imaging data [Berens et al., 2018]. The Neurofinder benchmark aims to provide a collection of datasets with ground truth labels for benchmarking the performance of cell detection algorithms.

The benchmark consists of 28 motion-corrected calcium imaging movies provided by four different labs. Datasets are annotated manually or based on anatomical markers. They differ in recording frequency, length of the movie, magnification, signal-to-noise ratio, and in the brain region that was recorded. The datasets are split into two groups, *training* datasets and *test* datasets. The eighteen training datasets are provided together with reference annotations for the cell locations, whereas the reference annotations for the nine test datasets are not disclosed.The test datasets and their undisclosed annotations are used by the Neurofinder benchmark to provide an unbiased evaluation of the performance of the algorithms. The characteristics of the test datasets are listed in Table 3.1. We note that some cells are not marked in the reference annotation. This does not invalidate the experimental analysis since the task of annotating cells remains equally difficult for each algorithm.

All datasets can be downloaded directly from the Neurofinder benchmark for cell identification [CodeNeuro, 2017], `https://neurofinder.codeneuro.org`.

Active versus inactive cells

We group the datasets into two sub-groups: Datasets in which most annotated cells are active, i.e. have a detectable fluorescence signal other than noise, and datasets in which most annotated cells are inactive. We make this distinction because inactive cells cannot be found by CNMF, Suite2P, and HNCcorr due to the lack of measurable changes in fluorescence other than noise. These cells are detectable with shape-based detection algorithms - e.g. [Gao, 2016], when the cell's fluorescence differs from the background.

The datasets with inactive cells are: 00.00, 00.01, and 03.00. The presence of inactive cells in these datasets has been noted in previous work [Pachitariu et al., 2017] as well as

Table 3.1: Characteristics of the test datasets of the Neurofinder benchmark and their corresponding training datasets. Data reproduced from Neurofinder [CodeNeuro, 2017].

Name	Source	Resolution	Length	Frequency	Brain region	Annotation method
00.00	Svoboda Lab	512 × 512	438 s	7.00Hz	vS1	Anatomical markers
00.01	Svoboda Lab	512 × 512	458 s	7.00Hz	vS1	Anatomical markers
01.00	Hausser Lab	512 × 512	300 s	7.50Hz	v1	Human labeling
01.01	Hausser Lab	512 × 512	667 s	7.50Hz	v1	Human labeling
02.00	Svoboda Lab	512 × 512	1000 s	8.00Hz	vS1	Human labeling
02.01	Svoboda Lab	512 × 512	1000 s	8.00Hz	vS1	Human labeling
03.00	Losonczy Lab	498 × 467	300 s	7.50Hz	dHPC CA1	Human labeling
04.00	Harvey Lab	512 × 512	444 s	6.75Hz	PPC	Human labeling
04.01	Harvey Lab	512 × 512	1000 s	3.00Hz	PPC	Human labeling

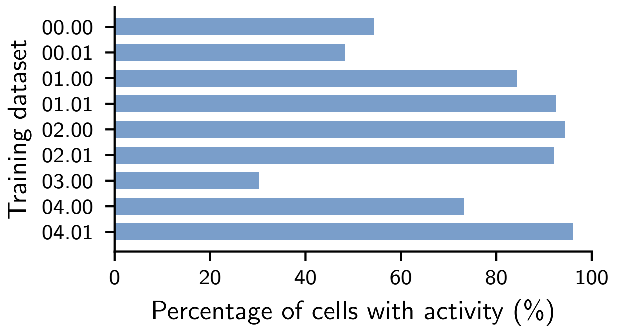


Figure 3.4: Approximate percentage of active cells among annotated cells in the training datasets. To determine the activity of the cells in a movie we used the following approximate analysis. First, we downsample the movie by averaging 10 frames. For every annotated cell, we compute the average intensity over time of the pixels in the spatial footprint. This timeseries is used an estimate of the cell's signal. A cell is then considered active if the $\Delta f/f$ [Jia et al., 2011] of this timeseries is at least 3.5 standard deviations above the median of $\Delta f/f$ for a total of at least 3 time steps in the movie. Due to the approximate nature of this analysis, its interpretation should be limited to understanding the general ratio between active and inactive cells in the datasets.

in a discussion of the benchmark's reference annotation on GitHub[1]. Experimentally, we also observed that the percentage of annotated cells that are active is substantially lower for training datasets 00.00, 00.01, and 03.00 as shown in Figure 3.4. Results for the test datasets could not be evaluated due to the lack of a reference annotation but similar results are expected.

Evaluation metrics

The list of cells identified by each of the algorithms is compared with the reference annotation. For this comparison, we use the submissions on the Neurofinder website. Each algorithm's submission was submitted by the algorithm's authors. This ensures that the results are representative of the algorithm's performance. Furthermore, the evaluation is fair since none of the authors had access to the reference annotation.

The algorithms are scored based on their ability to reproduce the reference annotation using standard metrics from machine learning. Each cell in the reference annotation is counted as a true positive (TP) if it also appears in the algorithm's annotation. The cells in the reference annotation that are not identified by the algorithm are counted as false negatives (FN), and the cells identified by the algorithm but that do not appear in the reference annotation are counted as false positives (FP). Each algorithm's performance is scored on a dataset based on $recall = \frac{TP}{TP+FN}$ and $precision = \frac{TP}{TP+FP}$. The overall performance of the

[1]See e.g. issues 16 and 24 on `https://github.com/codeneuro/neurofinder`.

algorithm on a dataset is summarized by the *F1-score*, which is the harmonic mean of recall and precision. For all metrics, higher scores are better.

A cell in the ground truth annotation is identified if the center of mass of the closest cell in the algorithm's annotation is within 5 pixels. These two cells are then matched and can not be matched with another cell. Each cell in either annotation is thus matched at most once.

Code Accessibility

The software used to generate the results in this work is available for non-commercial use at `https://github.com/quic0/HNCcorr`. The code is also available as supplementary material.

Computational Hardware

The experiments were run with MATLAB 2016a on a single core of a Linux machine running Linux Mint 18.1. The machine is equipped with an Intel i7-6700HQ CPU running at 2.60 GHz and 16GB RAM.

Algorithm Implementations

The results in Figures 3.5 and 3.7 are taken directly from Neurofinder [CodeNeuro, 2017] and were produced by the authors of the respective algorithms. The results in 3.8 were generated by us. In the remainder, we describe the HNCcorr implementation used for all experiments including those reported in Figures 3.5 and 3.7. We also describe the CNMF and Suite2P implementation used for the experiment reported in Figure 3.8.

HNCcorr All datasets for HNCcorr were preprocessed by averaging every ten frames into a single frame to reduce the noise. All parameters were kept at their default settings with the exception of dataset specific parameters listed in Table 3.2. These parameters are dataset dependent due to varying cell sizes across datasets. For the experiments reported in Figures 3.5 and 3.7, we used non-default values for the reference set sampling rate ($\gamma = 100\%$) and the grid resolution used for sparse computation ($\kappa = 25$). These changes have a marginal effect on the cell detection quality of the algorithm, but they result in increased running times.

CNMF The CNMF implementation was obtained from `https://github.com/epnev/ca_source_extraction`. The base configuration was taken from the file `run_pipeline.m` in the CNMF repository. We turned off the patch-based processing, and set the number of cells, as denoted by K, equal to 600. We used the same values for the maximum and minimum cell size, `max_size_thr` and `min_size_thr`, as in HNCcorr. We also set the temporal down-sampling `tsub` to 10 to match the down-sampling used with HNCcorr.

Table 3.2: Dataset dependent parameter values used for the HNCcorr implementation.

Dataset	Patch size (m)	Radius circle negative seeds (ρ)	Size superpixel (k)	Post-processor lower bound ($n_{\min}$)	Post-processor upper bound ($n_{\max}$)	Post-processor expected size (n_{avg})
00.00	31×31	10 pixels	5×5	40 pixels	150 pixels	60 pixels
00.01	31×31	10 pixels	5×5	40 pixels	150 pixels	65 pixels
01.00	41×41	14 pixels	5×5	40 pixels	380 pixels	170 pixels
01.01	41×41	14 pixels	5×5	40 pixels	380 pixels	170 pixels
02.00	31×31	10 pixels	1×1	40 pixels	200 pixels	80 pixels
02.01	31×31	10 pixels	1×1	40 pixels	200 pixels	80 pixels
03.00	41×41	14 pixels	5×5	40 pixels	300 pixels	120 pixels
04.00	31×31	10 pixels	3×3	50 pixels	190 pixels	90 pixels
04.01	41×41	14 pixels	3×3	50 pixels	370 pixels	140 pixels

Suite2P The Suite2P implementation was obtained from `https://github.com/cortex-lab/Suite2P`. The base configuration was taken from the file `master_file_example.m` in the Suite2P repository. The `diameter` parameter was tweaked per dataset to maximize the F1-score on the Neurofinder training datasets. The selected values per (dataset) are: 8 (00.00), 10 (00.01), 13 (01.00), 13 (01.01), 11 (02.00), 11 (02.01), 12 (03.00), 11 (04.00), and 12 (04.01).

Cell Detection Performance

The experimental performance of the algorithms HNCcorr, 3dCNN, CNMF, and Suite2P on the six test datasets containing active cells is shown in figure 3.5. The cells identified by HNCcorr are shown for two datasets in Figure 3.6. Overall, the HNCcorr algorithm has superior performance across datasets compared to the matrix factorization algorithms. The HNCcorr algorithm achieves a 15 percent relative improvement compared to CNMF in terms of average F1-score across datasets. It also attains a minor improvement of 4 percent compared to Suite2P. However, it performs about 3 percent worse than 3dCNN, which detects both active and inactive cells unlike HNCcorr, Suite2P, and CMNF.

HNCcorr performs particularly well on datasets 02.00 and 02.01 where it attains substantially higher F1-scores than the other algorithms, due to higher detection capability as measured by recall. Although 3dCNN is able to match the near-perfect recall of HNCcorr, it attains lower precision for datasets 02.00, 02.01. This indicates that 3dCNN detects either cells that are not in the reference annotation or incorrectly marks non-cell regions as cells.

Leading Neurofinder Submissions

HNCcorr and matrix factorization algorithms identify cells based on their unique fluorescence signal. As such, they are able to detect cells that activate, i.e. have one or more spikes

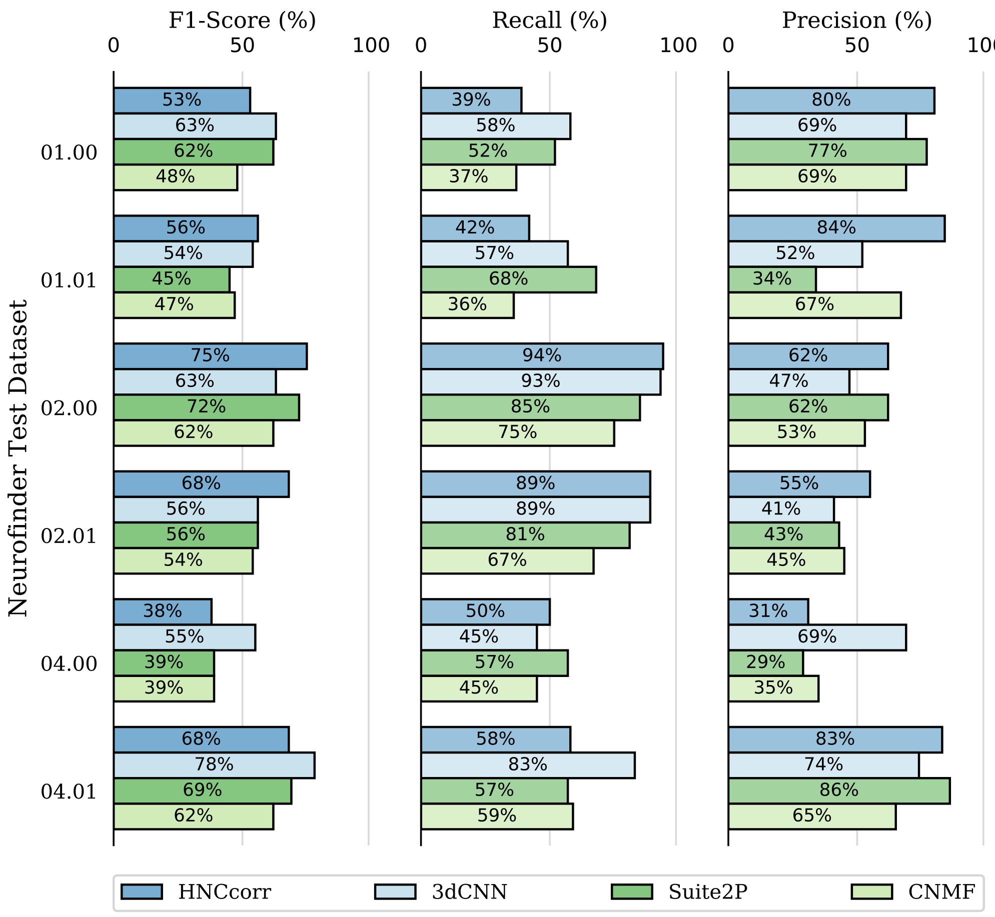
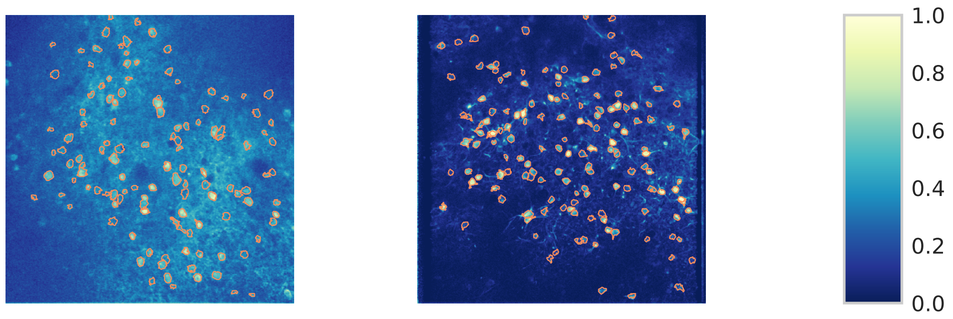


Figure 3.5: Cell identification scores for the HNCcorr, CNMF, and Suite2P algorithms on the Neurofinder test datasets with active cells. For each of the listed metrics, higher scores are better. The data is taken from Neurofinder submissions `Sourcery` by `Suite2P`, `CNMF_PYTHON` by `CNMF`, `3dCNN` by `ssz`, and submission `HNCcorr` by `HNCcorr`.



(a) Neurofinder 01.01 test. (b) Neurofinder 02.00 test.

Figure 3.6: Contours of the cells identified by HNCcorr for two Neurofinder datasets overlaid on the respective local correlation image.

in calcium concentration, but they cannot detect cells without any signal. As discussed, Neurofinder datasets 00.00, 00.01, and 03.00 have a large number of inactive cells. Therefore, matrix factorization algorithms and HNCcorr only detect a small percentage of cells in these datasets.

The leading Neurofinder submission for both Suite2P and HNCcorr therefore rely on a shape-based detection algorithm for these datasets. The HNCcorr + Conv2d submission uses the Conv2d [Gao, 2016] neural network for datasets 00.00, 00.01, and 03.00 and HNCcorr for the remaining datasets. Similarly, Suite2P + Donuts submission uses Donuts [Pachitariu et al., 2013] for the datasets 00.00, 00.01, and 03.00 and Suite2P for the remaining datasets. Together with the 3dCNN, these submissions are the top three submissions for the Neurofinder benchmark. Figure 3.7 shows how the three submissions compare. In particular, the 3dCNN algorithm outperforms the other shape-based detection algorithms on datasets 00.00, 00.01, and 03.00. As discussed before, HNCcorr attains higher F1-scores for the 02.00 and 02.01 datasets.

Runtime Analysis

We compared the running time performance of HNCcorr, CNMF, and Suite2P on nine training datasets of the Neurofinder benchmark. 3dCNN was excluded since the underlying code is not available. Running time results are given in Figure 3.8. The measured time for CNMF and Suite2P also includes the time to provide the associated cell signals, since they are determined simultaneously, whereas for HNCcorr it does not. On average, HNCcorr is 1.5 times faster than CNMF. Compared to Suite2P, HNCcorr performs equally well on average. We observed similar performance for a large experimental dataset consisting of 50,000 frames not reported here.

3.4 Conclusions

We present here the HNCcorr algorithm for cell detection in two-photon calcium imaging datasets. The HNCcorr algorithm builds upon the combinatorial clustering problem HNC and makes use of the newly devised similarity measure $(\textsc{sim})^2$. Experimentally, we demonstrate that HNCcorr is a top performer for the Neurofinder benchmark both in terms of cell detection quality and running time. HNCcorr achieves a higher average score than CNMF and Suite2P with similar or faster running time. This work demonstrates that similarity-based machine learning and combinatorial optimization are valuable tools for analysis in neuroscience or other disciplines.

In future work, HNCcorr could be extended to support real-time data, which enables direct feedback experimentation. Another extension of interest is adapting HNCcorr for one-photon and light-field calcium imaging [Prevedel et al., 2014], where the main challenge is dealing with overlapping cells.

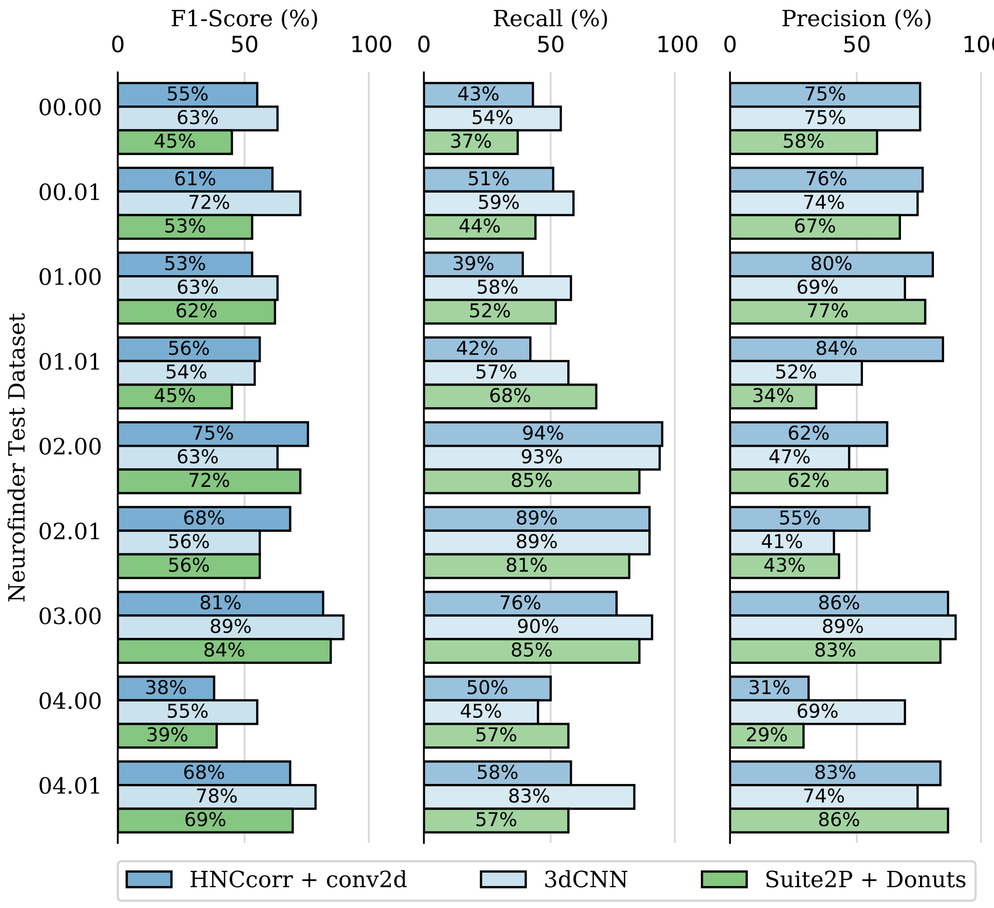


Figure 3.7: Cell identification scores on all test datasets for the three leading submissions of the Neurofinder benchmark in July 2018. For each of the listed metrics, higher scores are better. The 3dCNN entry is based on the Neurofinder submission `3dCNN` by `ssz`. The Suite2P + Donuts [Pachitariu et al., 2013] entry is taken from the submission `Sourcery` by `Suite2P`. It uses the Donuts algorithm for datasets 00.00, 00.01, and 03.00 and the Suite2P algorithm for the remaining datasets. The HNCcorr + Conv2d entry is taken from the submission `HNCcorr + conv2d` by `HNCcorr`. It uses the Conv2d algorithm [Gao, 2016] for datasets 00.00, 00.01, and 03.00 and the HNCcorr algorithm for the remaining datasets. The results obtained with the Conv2d algorithm reported here differ slightly from the `Conv2d` submission on the Neurofinder benchmark since the Conv2d model was retrained by the authors of this manuscript.

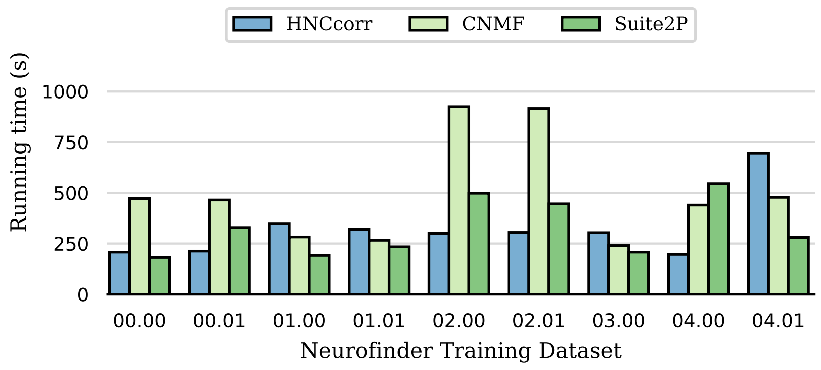


Figure 3.8: Running time results for nine training datasets of the Neurofinder benchmark. The measured time for CNMF and Suite2P also includes the time to provide the associated cell signals, since they are determined simultaneously, whereas for HNCcorr it does not. Running times are based on a single evaluation.

Chapter 4

Detecting Aberrant Linking Behavior in Directed Networks with Markov Random Fields

Agents with aberrant behavior are commonplace in today's networks. There are malicious websites on the internet, fake profiles in social media, and fake news sources prolific in spreading misinformation. There is considerable interest in detecting such aberrant agents in networks. Across contexts, the unifying theme is that normal agents rarely link to aberrant ones. We call this property *aberrant linking behavior*. This behavior occurs in a number of different contexts.

Aberrant linking behavior was observed in web graphs in the early days of search engines. In these web graphs, the goal is to separate informative websites (normal) from spam or malicious websites (aberrant). Informative websites typically link to other relevant and informative websites, whereas spam websites link to either informative websites or other spam websites. Spam websites linking to each other creates "link farms" [Castillo et al., 2007; Wu and Davison, 2005]. From the linking behavior of the websites, the expected structure of normal and aberrant agents, with aberrant linking behavior, arises. In one empirical study, this structure of normal and spam sites is verified in a Japanese web graph with 5.8 million sites and 283 million links [Saito et al., 2007].

In social networks, the goal is to separate real profiles (normal) from fake profiles (aberrant). On Facebook, it is estimated that nearly 10% of accounts are either fake or otherwise "undesirable" (intentionally spreading misinformation) [Fire et al., 2014]. In an empirical study of $40,000$ fake Twitter accounts, it was observed that most users of authentic social media accounts avoid following fake accounts [Ghosh et al., 2012]. A similar study of $250,000$ fake accounts on LinkedIn corroborated this result [Xiao et al., 2015]. It is evident that the expected structure of normal and aberrant agents, with aberrant linking behavior, arises in these settings.

In the context of fake news, the goal is to separate credible sources (normal) from dubious ones (aberrant) [Shu et al., 2017, 2019; Törnberg, 2018]. Credible sources typically link to other credible sources and will not link to dubious ones. Thus, the expected structure of normal and aberrant agents, with aberrant linking behavior, arises. In [Shu et al., 2017], the authors observe that the credibility of a news event is highly related to the credibility of the sources it references. In [Shu et al., 2019] and [Törnberg, 2018], the authors use the colloquial term "echo chamber" to describe small networks of agents that amplify the spread of false information by repeating it.

The problem of aberrant agent detection is formalized here as a classification problem on a directed graph, where each vertex represents an agent and each arc represents a link from one agent to another. In contrast to an agent-based approach that classifies an agent based on its individual features [Erdélyi et al., 2011; Ntoulas et al., 2006; Webb et al., 2008], we use information about the links in the graph to perform the classification [Becchetti et al., 2006; Gan and Suel, 2007]. These agent-based and link-based approaches are synergistic [Becchetti et al., 2008; Roul et al., 2016]. For example, the output of an agent-based machine learning algorithm can be incorporated in a link-based approach as prior scores for the agents.

Multiple link-based techniques have been proposed in the literature, including spectral methods and random walks.

Spectral techniques for classifying aberrant agents typically optimize a symmetric objective function, which is the sum of a measure of disagreement between adjacent agents [Von Luxburg, 2007; Wu and Chellapilla, 2007; Zhou et al., 2007]. Such methods are sometimes called "graph regularization methods", since they interpolate missing labels from known ones [Abernethy et al., 2010]. In other work, the objective functions are NP-hard cut problems, such as normalized cut [Shi and Malik, 2000] or generalized weighted cut [Meilă and Pentney, 2007]. Since these cut problems are NP-Hard, they are approximated with spectral methods on an appropriately chosen symmetric matrix. After the spectral algorithm returns a partition, further refinements may be made with pairwise swaps until a local optimum is reached [Malliaros and Vazirgiannis, 2013].

Random-walk based approaches include PageRank [Page et al., 1999], TrustRank [Gyöngyi et al., 2004], AntiTrustRank [Krishnan and Raj, 2006], and several domain-specific variants which have been proposed over the years [Gori and Pucci, 2006; Liu, 2009; Rosvall and Bergstrom, 2008; Sayyadi and Getoor, 2009; Shu et al., 2019; Wu and Chellapilla, 2007]. If a set of agents is highly internally connected and minimally externally connected, a random walk starting in that set is expected to spend a lot of time inside it before exiting. Thus, in a graph with aberrant linking behavior, a random walk is expected to visit the normal agents more frequently than the aberrant agents [Rosvall and Bergstrom, 2008]. In some cases, the behavior of random walks is equivalent to optimizing an explicit objective function. In other cases, no objective function is specified. We describe the mechanics of these algorithms in detail in section 4.1.

we propose a new approach that formulates the problem for the first time as a Markov Random Fields (MRF) problem [Geman and Geman, 1984]. This formulation balances obeying any given prior information with minimizing the number of links from normal to aberrant agents. The optimal solution to the formulation is obtained efficiently with known algorithms [Hochbaum, 2001]. One advantage of the proposed formulation is its ability to be given prior labels for the agents with various degrees of confidence.

In an extensive empirical study, we compare MRF with three well-known algorithms from the literature: PageRank [Page et al., 1999], TrustRank [Gyöngyi et al., 2004], and AntiTrustRank [Krishnan and Raj, 2006]. We also use a random classifier as a baseline algorithm. We find that MRF outperforms its competitors. The average score of MRF is approximately 25% higher than the next-best algorithm.

The main contributions in this chapter are:

1. We formulate the problem of detecting aberrant linking behavior as a directed Markov Random Fields (MRF) problem. The formulation is solved optimally and efficiently. This is the first time that MRF is used to model the detection of aberrant agents and the first use of MRF for directed graphs.

2. We develop a new variant of the modularity metric [Newman and Girvan, 2004] that addresses its known shortcomings on directed graphs. We show that our metric has desirable properties and prove that optimizing it is NP-hard.

3. We present an extensive empirical study in which we compare MRF with three well-known algorithms from the literature.

The rest of this chapter is organized as follows: in section 4.1, we present preliminaries, including the details of the existing algorithms. In section 4.2, we present the MRF model. In section 4.3, we provide an MRF formulation for aberrant agent detection. In section 4.4, we describe the methods we used to evaluate the quality of the results of the algorithms. In section 4.5 and section 4.6 we describe the experimental setup and results. Finally, we conclude the manuscript in section 4.7.

4.1 Preliminaries

Notation We represent the network as a directed graph $G = (V, A)$, where V is the set of vertices and A is the set of arcs. Each node in the graph G represents an agent, and an arc represents a link from one agent to another. Each arc $(i, j) \in A$ has an associated $w_{ij} \in \mathbb{R}^+$, which represents the number of links from agent i to agent j. We use d_i^{out} and d_i^{in} to denote the weighted out-degree and in-degree of vertex i, respectively.

We will ultimately partition V into two sets: C_0, the set of "normal" vertices and C_1, the set of "aberrant" vertices. The notation W_{pq} denotes the total weight of arcs from C_p to C_q. That is,

$$W_{pq} = \sum_{i \in C_p, j \in C_q} w_{ij}.$$

We also define $W = \sum_{(i,j)\in A} w_{ij}$ to denote to the total sum of weights.

Priors All the algorithms described here will take as input a graph in which some or all of the vertices have prior values associated with them. We use $V_{\text{prior}} \subseteq V$ to denote the set of vertices which have priors, and we use $c_i \in [0,1]$ to denote the prior value of vertex i. These priors could represent information about known vertices, the ratings of human judges or the output of another algorithm. The prior values do *not* represent "ground truth". The priors may have various degrees of confidence and may occasionally be imprecise or unreliable. Besides the structure of the graph itself, the priors are the only information we have on which to base our final classifications.

Output All algorithms output a continuous score, $x_i \in [0,1]$, for every vertex $i \in V$, where closer to 0 means more likely normal and closer to 1 means more likely aberrant. From these continuous scores, we decide how to partition V into C_0 and C_1.

Existing Algorithms We consider the three existing algorithms that are most prevalent in the literature: PageRank, TrustRank, and AntiTrustRank.

In PageRank [Page et al., 1999], a Markov chain is defined over the vertices in the graph. The trust score of vertex $i \in V$ is equal to the stationary probability that the Markov chain is in state i. The state transitions from vertex i to vertex j with probability

$$P_{ij} = \alpha \frac{w_{ij}}{d_i^{out}} + (1-\alpha) r_j \quad \forall (i,j) \in V \times V.$$

Here, $\alpha \in [0,1]$ is a hyperparameter known as the *attenuation factor* and $r_j \in [0,1]$ is the probability of *restarting* the Markov chain from vertex j. The values of r_j must sum to 1: $\sum_{j \in V} r_j = 1$.

In PageRank, $r_j = \frac{1}{n} \forall j \in V$. The unique eigenvector with eigenvalue 1, π, is computed with the Power method. The score, x_i, returned by PageRank, is equal to $1 - \pi_i$.

TrustRank [Gyöngyi et al., 2004] is a modification of PageRank where the probability for (re)starting at vertex j is proportional to $1 - c_j$ (a measure of how "trusted" the vertex is):

$$r_j^{trust} = \frac{1 - c_j}{\sum_{i \in V_{\text{prior}}} (1 - c_i)} \quad \forall j \in V_{\text{prior}}.$$

The intuition behind TrustRank is that from trusted nodes you should only reach other trusted nodes. Like PageRank, the returned score is $1 - \pi_i$, where π_i is the probability that the Markov chain is in state i.

AntiTrustRank [Krishnan and Raj, 2006] is similar to TrustRank, but differs in two aspects: The Markov chain traverses the graph in the reverse direction, and the (re)start distribution is proportional to c_j (a measure of how "distrusted" the vertex is):

$$r_j^{anti} = \frac{c_j}{\sum_{i \in V_{\text{prior}}} c_i} \quad \forall j \in V_{\text{prior}}.$$

The underlying idea is that from normal vertices you should rarely reach aberrant vertices, and thus if you follow arcs in the reverse direction, then from aberrant vertices you should reach mostly aberrant vertices.

In contrast to PageRank and TrustRank, in AntiTrustRank the eigenvector with eigenvalue 1 represents the *distrust* of the vertices. The score, x_i, returned by AntiTrustRank for vertex $i \in V$ is equal to π_i.

4.2 Markov Random Fields (MRF) Model

The MRF model [Geman and Geman, 1984] is defined on a directed graph $G = (V, A)$, where each node $i \in V$ has an associated decision variable x_i. For given *deviation* functions G_i and *separation* functions F_{ij}, the MRF model is defined as:

$$\begin{aligned} \min \quad & \lambda \sum_{i \in V} G_i(x_i, c_i) + \sum_{(i,j) \in A} F_{ij}(x_i - x_j) \\ \text{s.t.} \quad & l_i \leq x_i \leq u_i \quad \forall i \in V \end{aligned}$$

The deviation function $G_i(\cdot, \cdot)$ penalizes a deviation of the variable x_i away from the prior value c_i, whereas the separation function $F_{ij}(\cdot)$ penalizes the difference between the values assigned to neighboring nodes in the graph. The *trade-off* parameter $\lambda \geq 0$ determines the trade-off between the deviation and separation penalties. When the separation functions $F_{ij}(\cdot)$ are convex, this problem is solved optimally and efficiently in either continuous or integer variables. The problem is NP-hard otherwise. The algorithms and their complexity are listed in Table 4.1.

Table 4.1: Complexity of Markov Random Fields (MRF) problems. For complexity results in this table, let $U = \max_i |x_i|$ be the number of labels, $n = |V|$ be the number of nodes in the graph, and $m = |A|$ be the number of arcs in the graph. A bi-linear function refers here to a function of the form $f(z) = \max\{u^+ z, -u^- z\}$ for $u^+, u^- \geq 0$ and $z \in \mathbb{R}$.

Deviation function $G_i(x_i)$	**Separation function** $F_{ij}(x_i - x_j)$	**Complexity**	**Reference**
Convex	Convex	$O(mn \log \frac{n^2}{m} \log nU)$	Ahuja et al. [2003]
Convex	Bi-linear	$O(mn \log \frac{n^2}{m} + n \log U)$	Hochbaum [2001]
Quadratic	Bi-linear	$O(mn \log \frac{n^2}{m})$	Hochbaum [2001]
Non-linear	Convex	$O(mnU^2 \log \frac{n^2 U}{m})$	Ahuja et al. [2004]
Linear	Non-linear	NP-Hard	Hochbaum [2001]

4.3 Markov Random Fields Model for Detecting Aberrant Agents

For the problem of detecting aberrant agents, a “good” solution is characterized by two properties. First, there should be few links from normal to aberrant agents. Second, the difference between the score assigned to an agent and its prior value, if any, should be small. These goals naturally map to an MRF model.

For each vertex $i \in V_{prior}$, with associated prior c_i, we choose a quadratic penalty function $G_i = (x_i - c_i)^2$ to measure the deviation between the assigned x_i and the prior c_i. The remaining vertices without priors do not have an associated deviation penalty. That is, $G_i(x_i, c_i) = 0$ for $i \notin V_{prior}$. For each arc $(i, j) \in A$ with weight w_{ij}, we have a separation function $F_{ij} = w_{ij}\,(x_j - x_i)^+$, where $(x_j - x_i)^+ = \max\{x_j - x_i, 0\}$. This separation function results in a penalty of $w_{ij}(x_j - x_i)$ for arc $(i, j) \in A$ if the score x_i of vertex i is lower than the score x_j of vertex j, since the (more) normal vertex i links to a (more) aberrant vertex j.

The resulting optimization problem is:

$$\begin{aligned} \min \quad & \lambda \sum_{i \in V_{\text{prior}}} (x_i - c_i)^2 + \sum_{(i,j) \in A} w_{ij}\,(x_j - x_i)^+ \\ \text{s.t.} \quad & 0 \le x_i \le 1 \quad \forall i \in V. \end{aligned} \tag{MRF-Detection}$$

In the optimization problem, each agent $i \in V$ is assigned a score $x_i \in [0, 1]$. The behavior of an agent with a score of 1 is considered aberrant, whereas a score of 0 corresponds to normal behavior.

(MRF-Detection) is a special case of the MRF problem with convex deviations and bi-linear separation functions[1], which was shown by Hochbaum [2001] to be solvable with a parametric minimum cut problem in the complexity of a single minimum cut problem plus the complexity required to find the minima of the convex deviation functions. Two parametric cut algorithms, based on the pseudoflow algorithm [Hochbaum, 2008; Hochbaum and Orlin, 2013] or the push-relabel algorithm [Gallo et al., 1989; Goldberg and Tarjan, 1988], achieve this complexity. Since the deviation functions are quadratic here, the complexity of finding the minima of the deviation functions is $O(|V|)$, which is dominated by the complexity of a minimum cut problem. As a result, the complexity of solving this parametric minimum cut problem with either of these two algorithms is expressible as $O\left(mn \log \frac{n^2}{m}\right)$ where n is the number of nodes in the graph and m is the number of arcs. These results imply that we can solve the MRF formulation for aberrant agent detection efficiently and optimally.

[1] A bi-linear function refers here to a function of the form $f(z) = \max\{u^+ z, -u^- z\}$ for $u^+, u^- \ge 0$ and $z \in \mathbb{R}$.

4.4 Performance Evaluation Metrics

In the datasets available to us, there is no "ground truth" to which we can compare our results. Instead we assess the classification performance in terms of how well the resulting partition obeys the property of having few links from normal vertices to aberrant vertices.

For this purpose, we will lay out several metrics. The first are a series of ad-hoc metrics, such as the average out-degree from normal vertices to aberrant ones. We also present a directed variant of the established modularity clustering metric [Newman and Girvan, 2004].

Ad-Hoc Metrics

Aberrant linking behavior implies that there should be few arcs from normal to aberrant. For that reason, a relevant metric is $\frac{W_{01}}{N_0}$, the average degree of a normal vertex to the set of aberrant vertices.

As a baseline for comparison, we also calculate $\frac{W_{11}}{N_1}$, the average degree of an aberrant vertex to the set of aberrant vertices. If our labeling has the desired property, then we expect that $\frac{W_{01}}{N_0}$ will be significantly smaller than $\frac{W_{11}}{N_1}$. F

In order to normalize these values of across graphs, we divide the degree measures by the average degree of the graph, $d_{\text{avg}} = \frac{W_{00}+W_{01}+W_{10}+W_{11}}{N_0+N_1}$. This leads to the following two metrics:

$$\frac{W_{01}/N_0}{d_{\text{avg}}} \text{ and } \frac{W_{11}/N_1}{d_{\text{avg}}}.$$

$$\frac{W_{01}/N_0}{d_{\text{avg}}}, \ \frac{W_{11}/N_1}{d_{\text{avg}}}, \text{ and } \frac{W_{01}}{W_{01}+W_{11}}.$$

Modularity

A metric commonly used in the graph partitioning literature is *modularity* [Newman and Girvan, 2004]. It measures how many edges are within clusters versus edges between clusters and compares that to a random graph with the same degree distribution [Kim et al., 2010; Newman and Girvan, 2004].

Given a weighted, undirected graph and a partition of the set of vertices into clusters $C_0, \dots, C_k$, modularity [Newman and Girvan, 2004] is defined as

$$\frac{1}{2W} \sum_{i,j \in V} \left[w_{ij} - \frac{d_i d_j}{2W} \right] \delta(i,j). \qquad \text{(UndirMod)}$$

Here, d_i is the weighted degree of vertex i, and

$$\delta(i,j) = \begin{cases} 1 & \exists p \mid i,j \in C_p, \\ 0 & \text{otherwise.} \end{cases}$$

A larger modularity value indicates that the cluster assignment is superior since there are more edges within the clusters than in a random graph. It is NP-hard to find the set of clusters that maximize modularity in a graph [Brandes et al., 2006].

When modularity is applied to directed graphs, there is no agreed-upon generalization [Kim et al., 2010; Newman, 2006; Rosvall and Bergstrom, 2008]. One straightforward adaptation of modularity to directed graphs would be to calculate [Kim et al., 2010; Newman, 2006; Rosvall and Bergstrom, 2008]

$$\frac{1}{W}\sum_{i,j\in V}\left[w_{ij}-\frac{d_i^{\text{out}}d_j^{\text{in}}}{W}\right]\delta(i,j). \tag{DirMod}$$

Several issues with this generalization have been observed. In [Malliaros and Vazirgiannis, 2013], small example graphs are shown in which certain arcs can be reversed without affecting the modularity. This is problematic given our interest in asymmetry between clusters.

In fact, we show in claim 1 that this definition of directed modularity, with 2 clusters, is proportional to the determinant of a matrix with entries W_{ij} for $i, j \in \{0, 1\}$. We defer the proof to the appendix. This result implies that this definition is symmetric with respect to the cluster assignment and that the cluster labels are exchangeable without affecting the modularity.

Claim 1. *Let $G = (V, A)$ be a directed graph with vertex labels in $\{0, 1\}$, defining two clusters C_0 and C_1. The modularity of the clustering assignment on G, as defined in equation* (DirMod)*, is proportional to*

$$W_{00}W_{11} - W_{01}W_{10}.$$

Proof.

The expression for modularity is

$$\frac{1}{W}\sum_{i,j\in V}\left[w_{ij}-\frac{d_i^{\text{out}}d_j^{\text{in}}}{W}\right]\delta(i,j).$$

Since W is constant, we can multiply by W^2:

$$\begin{aligned}&\propto \sum_{i,j\in V}\left[Ww_{ij}-d_i^{\text{out}}d_j^{\text{in}}\right]\delta(i,j)\\ &= W\sum_{i,j\in V}w_{ij}\delta(i,j)-\sum_{i,j\in V}d_i^{\text{out}}d_j^{\text{in}}\delta(i,j).\end{aligned}$$

Now, since $\delta(i, j) = 1$ if and only if i and j are in the same cluster:

$$\begin{aligned}
&= W(W_{00} + W_{11}) - \sum_{i \in C_0} \sum_{j \in C_0} d_i^{\text{out}} d_j^{\text{in}} - \sum_{i \in C_1} \sum_{j \in C_1} d_i^{\text{out}} d_j^{\text{in}} \\
&= (W_{00} + W_{01} + W_{10} + W_{11})(W_{00} + W_{11}) \\
&\quad - (W_{00} + W_{01})(W_{00} + W_{10}) \\
&\quad - (W_{11} + W_{10})(W_{11} + W_{01}) \\
&= 2(W_{00}W_{11} - W_{10}W_{01})
\end{aligned}$$

□

Corollary 2. *Let $G = (V, A)$ be a directed graph with vertex labels in $\{0, 1\}$, defining a cluster assignment $\mathcal{C}$. Let $\mathcal{C}'$ be the assignment with opposite labels. Then, the modularity of cluster assignment $\mathcal{C}$ on G, as defined in equation* (DirMod)*, is the same as the modularity of assignment $\mathcal{C}'$.*

The symmetry with respect to the clustering labels is undesirable for the problem of agent detection, since only links from a normal vertex to an aberrant one should be penalized. That is, we would like to penalize arcs from C_0 to C_1 without penalizing arcs from C_1 to C_0. If the labels can be interchanged, then these penalties are necessarily symmetric. We will propose a change to the modularity metric which overcomes this deficiency.

One option might be to change the definition δ so that $\delta(1, 0) = \delta(0, 0) = \delta(1, 1) = 1$. In other words, "rewarding" arcs from C_1 to C_0 in addition to arcs within clusters. However, repeating the calculation in Claim 1 gives a surprising result: even with the new definition of δ, the modularity metric remains proportional to $W_{00}W_{11} - W_{01}W_{10}$. For that reason, a different change is needed.

We suggest a new variant of the directed modularity metric, which captures the asymmetric nature of the relation between normal and aberrant vertices. Our metric only penalizes arcs in one direction between the two clusters. We keep the $W_{00}W_{11}$ term, but instead of subtracting off $W_{01}W_{10}$, we subtract off $\frac{3}{4}W_{01}^2$. We add the $\frac{3}{4}$ coefficient to account for the larger expected value of the term W_{01}^2 as compared to $W_{01}W_{10}$[2]. Our new definition of directed modularity in the two-cluster case is:

$$\frac{4\left(W_{00}W_{11} - \frac{3}{4}W_{01}^2\right)}{W^2}. \tag{AsymMod}$$

Similarly to the result for undirected modularity, we establish that maximizing AsymMod is NP-hard. Our reduction is from the minimum bisection problem, which is different from the undirected case. Again, we defer the proof to the appendix.

Claim 2. *Maximizing* (AsymMod) *is NP-Hard.*

[2]The coefficient $\frac{3}{4}$ is chosen such that the terms have equal expectation when the W_{pq}/W values are drawn independently and uniformly from the unit interval.

Proof. The proof is a reduction from the minimum bisection problem on an undirected and unweighted graph $G = (V, E)$. A bisection (C_0, C_1) is a partition of the set of vertices V such that $|C_0| = |C_1| = n/2$, where n is the number of nodes in the original graph. The minimum bisection problem is to find a bisection (C_0, C_1) that minimizes $W_{01} = W_{10}$. This problem is known to be NP-Hard [Garey et al., 1974].

Consider an undirected and unweighted graph G on which we would like to solve the minimum bisection problem. We create the graph G' where we copy G by turning each edge into two directed arcs and add two vertices, s and t, with an arc from s to every vertex in G and an arc from every vertex in G to t. Let the weight of all these arcs be M, for sufficiently large M (e.g. $M \geq m^2$ where m is the number of arcs in the original graph).

Consider a partition of this new graph into C_0 and C_1. We still use W_{pq} to denote the total weight *in the original graph* between clusters p and q. We break the modularity [3] calculation into four cases, depending on which clusters s and t are in:

$s \in C_0, t \in C_0$:

$$(2|C_0|M + W_{00})(W_{11}) - \frac{3}{4}(|C_1|M + W_{01})^2$$

$s \in C_0, t \in C_1$:

$$(|C_0|M + W_{00})(|C_1|M + W_{11}) - \frac{3}{4}(nM + W_{01})^2$$

$s \in C_1, t \in C_0$:

$$(|C_0|M + W_{00})(|C_1|M + W_{11}) - \frac{3}{4}(W_{01})^2$$

$s \in C_1, t \in C_1$:

$$(W_{00})(2|C_1|M + W_{11}) - \frac{3}{4}(|C_0|M + W_{01})^2$$

Expanding these expressions, we see that the coefficient of M^2 is largest when $s \in C_1$, $t \in C_0$, and $|C_0| = |C_1| = n/2$. Thus, for sufficiently large M, these are necessary conditions to maximize the modularity of the clustering in G'. The expression becomes:

$$\frac{n^2}{4}M^2 + \frac{n}{2}(W_{00} + W_{11})M + W_{00}W_{11} - \frac{3}{4}W_{01}^2.$$

Assuming M is sufficiently large, an optimal solution maximizes $\frac{n}{2}M(W_{00} + W_{11})$ and thus $W_{00} + W_{11}$. However, maximizing this quantity is equivalent to minimizing $W_{01} + W_{10} = 2W_{01}$. Thus, the partition which maximizes modularity in G' gives us the minimum bisection in G. □

[3]We ignore the normalization term $\frac{4}{W^2}$.

From now on, when we refer to modularity, it is assumed that we are referring to (AsymMod).

4.5 Experimental Setup

We compare MRF against the algorithms PageRank [Page et al., 1999], TrustRank [Gyöngyi et al., 2004], AntiTrustRank [Krishnan and Raj, 2006], and a randomized baseline algorithm, which we name Random. We measure the performance of the algorithms in terms of modularity score and the ad-hoc metrics described in section 4.4.

Datasets We evaluate the experimental performance of MRF on twenty-one different datasets from the KONECT project [Kunegis, 2013], as well as two Web Spam datasets [Castillo et al., 2006]. The datasets in KONECT are categorized into twenty-three categories. We chose twenty-one datasets in categories in which we plausibly expected to find high-modularity clusters. The two Web Spam datasets are based on web crawls of the .uk top-level domain. The properties of the datasets are summarized in Table 4.2. The KONECT datasets are available at `http://konect.uni-koblenz.de/`, and the Web Spam datasets are available at `http://chato.cl/webspam/datasets/`. We excluded from our analysis four Konect datasets for which no algorithm was able to attain a modularity score above 0.2.

Priors Since we do not have access to datasets with prior labels, we generate priors with a simple heuristic. For each dataset, we assign priors such that $p_{prior} \leq 50\%$ percent of the vertices are marked aberrant and the same number of vertices are marked normal. The vertices are chosen according to the difference between out-degree and in-degree, since we expect that nodes with high out-degree and low in-degree are potentially aberrant. We define $\Delta_i = d_i^{out} - d_i^{in}$ for every vertex $i \in V$. The vertices are then sorted such that $\Delta_{\sigma(1)} \geq \Delta_{\sigma(2)} \geq \cdots \geq \Delta_{\sigma(n)}$. The first $n_{\text{prior}} = \lfloor p_{prior}\ n \rfloor$ vertices with the highest Δ value are considered aberrant and given the prior values $c_i = 1$. The last n_{prior} vertices with the lowest Δ value are considered normal and given the prior values $c_i = 0$.

Labeling Each of the algorithms returns a score $x_i \in [0, 1]$ for every vertex $i \in V$. We convert these scores into a binary assignment to the clusters C_0 and C_1 by means of a threshold τ. The clusters are given by:

$$C_1(\tau) = \{i \in V : x_i \geq \tau\}$$
$$C_0(\tau) = \{i \in V : x_i < \tau\}$$

We report the highest modularity obtained over a set of thresholds T:

$$\max_{\tau \in T} \quad \text{Modularity}(C_0(\tau), C_1(\tau))$$

Table 4.2: Basic properties of twenty-one datasets from the KONECT project and the two datasets from the Web Spam project.

Dataset	**Vertices**	**Arcs**	**Link Significance**	**Weights**
Animal - Bison	26	314	Dominance behavior between bisons.	yes
Animal - Cattle	28	217	Dominance behavior between cattle.	yes
Animal - Hens	32	496	Pecking order among hens.	no
Citation - Cora	23166	91500	Scientific citations.	no
Citation - DBLP	12591	49728	Scientific citations in computer science.	no
Communication - Company	167	5783	Emails within manufacturing company.	yes
Communication - DNC Emails	1891	5517	Emails within committee.	yes
Communication - Slashdot	51083	130370	Responses to messages.	yes
Communication - University	1899	20296	Messages within university.	yes
Hyperlink - Blogs	1224	19022	Links between political blogs.	no
Hyperlink - Google	15763	170335	Links between internal Google pages.	no
Hyperlink - Spam 2006	11402	730774	Links between UK sites.	yes
Hyperlink - Spam 2007	114529	1771291	Links between UK sites.	yes
Infrastructure - Airports 1	3425	37594	Flights between airports.	yes
Infrastructure - Airports 2	2939	30501	Flights between airports.	no
Metabolic - Proteins (Small)	1706	6171	Interactions between proteins.	no
Social - Advogato	6539	47135	Trust relationships between users.	yes
Social - Dorm	217	2672	Friendship ratings between students.	yes
Social - High School	70	366	Friendship ratings in a high school.	yes
Social - Physicians	241	1098	Trust relationships between physicians.	no
Social - Twitter	23370	33101	Following relationships between users.	no
Trophic - FL Dry Season	128	2137	Carbon exchanges between organisms.	yes
Trophic - FL Wet Season	128	2106	Carbon exchanges between organisms.	yes

For the MRF algorithm, the set of thresholds T is given by the set of unique values of the scores $\{x_i : i \in V\}$. For the other algorithms, the set of threshold values T consists of the $0^{\text{th}}, 5^{\text{th}}, \ldots, 100^{\text{th}}$ percentiles of the scores $\{x_i : i \in V\}$. We apply a different method for MRF since its solution typically contains only a few distinct values. In the results reported here, the median number of unique scores in the MRF solution is 13. In contrast, the node scores tend to be unique for each of the other algorithms.

Algorithm Implementations For MRF, we use the formulation as described in equation (MRF-Detection) with one modification: we normalize the trade-off parameter λ. We call this new hyperparameter the normalized trade-off parameter, λ_{norm}. It is defined by the following equation:

$$\lambda = \lambda_{norm} \frac{\sum_{(i,j)\in A} w_{ij}}{|V_{prior}|}.$$

We expect that the normalized parameter λ_{norm} is more consistent across datasets, since it corrects for the size of the graph.

We use a parametric implementation of the pseudoflow algorithm [Hochbaum, 2008] to solve the parametric minimum cut problem.

For PageRank, we use the implementation of PageRank in the NetworkX package for Python [Hagberg et al., 2008]. For TrustRank and AntiTrustRank, we wrote a wrapper around the NetworkX implementation of PageRank. We use the default settings of the PageRank solver, except for the attenuation factor α, which we treat as a hyperparameter. The maximum number of iterations is set to 1000.

The Random algorithm provides baseline comparison. Each vertex is assigned a score, x_i, uniformly drawn from the unit interval. We report the average of the best modularity obtained in each of 10 trials.

Our implementations of these algorithms, as well as the code used to run the experiments, are available open-source at `removedfordouble-blindreview`.

Hyperparameters All algorithms, except for Random, have hyperparameters. The hyperparameters for MRF are the normalized trade-off parameter λ_{norm} and the percentage of priors p_{prior}. The hyperparameters for TrustRank and AntiTrustRank are the attenuation factor α and the percentage of priors p_{prior}. PageRank has only the attenuation factor α as its hyperparameter, since the algorithm does not utilize the prior values.

For each algorithm and dataset, we tested 200 combinations of hyperparameter values to maximize modularity. Each combination of hyperparameter values was selected with the Tree-structured Parzen estimator (TPE) algorithm [Bergstra et al., 2011] based on previous evaluations and a prior distribution for each of the parameters. The TPE algorithm uses approximate Bayesian Optimization to select a combination of hyperparameter values that has the highest expected increase in modularity score. Bayesian optimization methods, such as TPE, have been shown to outperform grid search and random search [Bergstra and Bengio, 2012] and to rival domain experts in finding good hyperparameter settings [Bergstra et al., 2011].

The prior distributions for the hyperparameters were selected as follows: For the normalized trade-off parameter λ_{norm}, we used a lognormal distribution with zero mean and a standard deviation of two. For the attenuation factor α, we used a uniform distribution on the unit interval, and for the percentage of priors p_{prior} we used a uniform distribution over the interval from 1 to 50 percent.

4.6 Results

For each of the twenty-three datasets and five algorithms, we report the best modularity found after the hyperparameter search. In Table 4.3, in the final column we report the highest modularity score found by any algorithm. In the remaining columns, we show the percentage of the top modularity score obtained by each algorithm. Thus, one algorithm always scores 100 percent.

Table 4.3: Normalized asymmetric modularity score by algorithm for twenty-three datasets. The scores are normalized as a percentage of the highest asymmetric modularity score obtained for the dataset.

Dataset	**MRF**	**Trust Rank**	**AntiTrust Rank**	**Page Rank**	**Random**	**Best Modularity Score (100%)**
Animal - Bison	82.4	**100.0**	94.3	84.4	36.2	0.279
Animal - Cattle	84.5	**100.0**	80.5	97.9	31.3	0.294
Animal - Hens	99.3	**100.0**	99.8	**100.0**	29.5	0.233
Citation - Cora	55.7	57.3	**100.0**	32.1	12.2	0.517
Citation - DBLP	**100.0**	89.3	41.2	58.1	16.4	0.409
Communication - Company	66.1	**100.0**	89.8	65.5	22.1	0.405
Communication - DNC Emails	**100.0**	47.4	37.7	22.3	21.2	0.402
Communication - Slashdot	93.9	69.2	**100.0**	49.9	20.7	0.321
Communication - University	**100.0**	63.1	62.1	29.7	18.1	0.342
Hyperlink - Blogs	**100.0**	70.8	81.2	57.8	23.3	0.302
Hyperlink - Google	**100.0**	71.2	94.4	53.1	15.7	0.448
Hyperlink - Spam 2006	**100.0**	53.4	91.5	37.2	12.8	0.743
Hyperlink - Spam 2007	**100.0**	47.6	63.7	45.5	19.9	0.505
Infrastructure - Airports 1	**100.0**	45.9	51.6	17.8	10.6	0.577
Infrastructure - Airports 2	**100.0**	71.2	47.9	19.9	10.7	0.591
Metabolic - Proteins (Small)	**100.0**	68.5	68.9	1.2	13.6	0.517
Social - Advogato	**100.0**	72.7	76.8	53.8	17.4	0.392
Social - High School	89.2	**100.0**	92.2	94.9	14.5	0.739
Social - Physicians	94.1	82.8	**100.0**	43.8	8.7	0.931
Social - Twitter	92.7	7.7	15.1	**100.0**	31.9	0.244
Social - University	57.3	**100.0**	66.4	63.4	11.1	0.615
Trophic - FL Dry Season	**100.0**	85.3	51.1	90.4	22.1	0.581
Trophic - FL Wet Season	**100.0**	93.4	78.2	93.4	28.3	0.588
Average	**92.0**	**73.8**	**73.2**	**57.0**	**19.5**	—

On average, our MRF algorithm achieves the highest percentage of the maximum modularity, at 92%. The next best algorithms, TrustRank and AntiTrustRank, achieve 74% and 73% respectively. MRF achieves the highest modularity on thirteen of the twenty-three datasets.

In Table 4.4, we examine the properties of these partitions with maximum modularity, using the ad-hoc metrics described in section 4.4. Across all datasets, the median value of $\frac{W_{01}/N_0}{d_{\text{avg}}}$ for MRF is 0.04. For AntiTrustRank, it is 0.13. This suggests that MRF is indeed finding partition with fewer arcs from normal to aberrant. It attains the smallest value on twenty of the twenty-three datasets. Looking at the median value of $\frac{W_{11}/N_1}{d_{\text{avg}}}$, we see another side of the story. For AntiTrustRank, the median value is 1.18. For MRF, it is 0.60. This suggests that the vertices classified as aberrant by AntiTrustRank are more interlinked than those classified by MRF. These results make sense: whereas MRF is minimizing the links from normal to aberrant, without any stipulation about connections between aberrant vertices,

Table 4.4: Ad-hoc metrics of the partition returned by each algorithm. The results are reported for the partition that maximizes asymmetric modularity, as in Table 4.3. For brevity, we exclude PageRank and Random. For a partition that satisfies aberrant linking behavior we expect the first and third metric to be small.

	$\frac{W_{01}/N_0}{d_{\text{avg}}}$			$\frac{W_{11}/N_1}{d_{\text{avg}}}$		
	MRF	Trust	AntiTrust	MRF	Trust	AntiTrust
Animal - Bison	0.16	0.20	0.16	0.46	0.51	0.59
Animal - Cattle	0.02	0.08	0.03	0.58	0.63	0.69
Animal - Hens	0.03	0.04	0.04	0.45	0.48	0.48
Citation - Cora	0.00	0.02	0.05	0.36	0.41	0.98
Citation - DBLP	0.00	0.09	0.01	0.43	0.48	1.18
Communication - Company	0.33	0.94	0.19	0.83	0.55	1.91
Communication - DNC Emails	0.06	2.35	0.11	0.66	0.20	0.86
Communication - Slashdot	0.06	0.35	0.17	0.80	0.31	1.27
Communication - University	0.23	1.11	0.19	0.60	0.36	2.46
Hyperlink - Blogs	0.08	0.49	0.15	0.33	0.31	1.13
Hyperlink - Google	0.00	0.07	0.13	0.60	0.32	0.96
Hyperlink - Spam 2006	0.00	0.28	0.01	1.92	0.61	10.10
Hyperlink - Spam 2007	0.00	0.41	0.24	0.71	0.12	7.37
Infrastructure - Airports 1	0.24	1.84	0.19	0.87	0.25	4.15
Infrastructure - Airports 2	0.07	0.64	0.33	0.47	0.28	0.82
Metabolic - Proteins (Small)	0.19	0.69	0.22	0.52	0.37	1.25
Social - Advogato	0.11	0.75	0.13	0.39	0.29	1.86
Social - Dorm	0.19	0.21	0.25	0.55	0.65	0.86
Social - High School	0.04	0.08	0.09	0.62	0.85	0.76
Social - Physicians	0.00	0.00	0.00	0.88	0.79	1.03
Social - Twitter	0.00	0.03	0.00	0.80	0.74	1.57
Trophic - FL Dry Season	0.00	0.00	0.09	1.96	1.42	7.09
Trophic - FL Wet Season	0.00	0.35	0.05	1.70	0.34	5.80
Median	**0.04**	**0.28**	**0.13**	**0.60**	**0.41**	**1.18**

AntiTrustRank is working backwards from known aberrant vertices to find aberrant clusters.

The running times of all algorithms, with the exception of Random, were in the same order of magnitude.

4.7 Conclusions

We studied here the problem of identifying agents with aberrant behavior in networks. Such agents frequently appear in today's networks, including malicious websites on the internet, fake profiles in social media, and fake news sources prolific in spreading misinformation. The unifying property of these networks is that normal agents rarely link to aberrant ones. We call this *aberrant linking behavior*.

We formulated the detection problem in a novel way: as a directed Markov Random Fields (MRF) problem. This formulation balances obeying any given prior information with minimizing the links from normal to aberrant agents. We discussed how the formulation is solved optimally and efficiently.

To compare the performance of the algorithms, we developed a new, asymmetric variant of the modularity metric for directed graphs, addressing a known shortcoming of the existing metric. We showed that our metric has desirable properties and proved that maximizing it is NP-hard. We also used several ad-hoc metrics to better understand properties of the solutions.

In an empirical experiment, we found that the MRF method outperforms competitors such as PageRank, TrustRank, AntiTrustRank, and Random. The solutions returned by MRF had the largest modularity score on thirteen of the twenty-three datasets tested. The modularity for MRF was, on average, 25 percent better than the modularity returned by TrustRank or AntiTrustRank.

Chapter 5

Concluding Remarks

This dissertation focuses on the advantages and challenges of similarity-based machine learning. One such challenge is the quadratic growth in the number pairwise similarities. The technique of *sparse computation* was introduced to mitigate the quadratic growth [Hochbaum and Baumann, 2016]. We devise in this work two algorithmic extensions for sparse computation. We demonstrate the effect of similarity-based machine learning via two important applications where similarity-based machine learning yields strong results: Cell identification in calcium-imaging movies and detecting aberrant agents in networks based on link behavior.

The first extension for sparse computation consists of faster, geometric algorithms for sparse computation. These algorithms use the novel computational geometry concept of data shifting to identify pairs of close objects faster than in the original implementation. A computational study demonstrates that the geometric algorithms provide a significant reduction in running time.

The second extension is the *sparse-reduced computation* technique that combine sparse computation with compression technique for highly-similar objects. The use of compression enables similarity-based machine learning on datasets with up to 10 million objects. An experimental comparison between sparse-reduced computation and sparse computation shows that sparse-reduced computation attains similar accuracy with a significant reduction in running time.

For the problem of cell identification in calcium imaging we demonstrate how HNCcorr detects cells based on pairwise similarities between pixels. The algorithm is a top performer for cell identification according to the Neurofinder benchmark and outperforms matrix factorization methods. For the detection of aberrant agents in network based on link behavior, we discuss an MRF formulation with links as pairwise similarities and an associated algorithm. The MRF approach outperforms competing algorithms based on random walks. Both applications provide evidence that similarity-based machine learning is an effective tool.

Bibliography

J. Abernethy, O. Chapelle, and C. Castillo. Graph regularization methods for web spam detection. *Machine Learning*, 81(2):207–225, 2010.

R. K. Ahuja, D. S. Hochbaum, and J. B. Orlin. Solving the Convex Cost Integer Dual Network Flow Problem. *Management Science*, 49(7):950–964, 2003. doi: 10.1287/mnsc.49.7.950.16384.

R. K. Ahuja, D. S. Hochbaum, and J. B. Orlin. A Cut-Based Algorithm for the Nonlinear Dual of the Minimum Cost Network FlowProblem. *Algorithmica*, 39(3):189–208, 2004. doi: 10.1007/s00453-004-1085-2.

N. Apthorpe, A. Riordan, R. Aguilar, J. Homann, Y. Gu, D. Tank, and H. S. Seung. Automatic neuron detection in calcium imaging data using convolutional networks. In *Advances in Neural Information Processing Systems*, pages 3270–3278, 2016.

S. Arora, E. Hazan, and S. Kale. A Fast Random Sampling Algorithm for Sparsifying Matrices. In J. Díaz, K. Jansen, J. D. P. Rolim, and U. Zwick, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, Lecture Notes in Computer Science, pages 272–279. Springer Berlin Heidelberg, 2006.

R. Asín-Achá, D. S. Hochbaum, and Q. Spaen. HNCcorr: Combinatorial Optimization for Neuron Identification. *Annals of Operations Research*, 2019. Under consideration (minor revision).

P. Baumann. Sparse-reduced computation for large-scale spectral clustering. In *2016 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 1284–1288, 2016. doi: 10.1109/IEEM.2016.7798085.

P. Baumann, D. S. Hochbaum, and Q. Spaen. Sparse-Reduced Computation - Enabling Mining of Massively-large Data Sets:. In *Proceedings of the 5th International Conference on Pattern Recognition Applications and Methods*, pages 224–231, Rome, Italy, 2016. SCITEPRESS. doi: 10.5220/0005690402240231.

P. Baumann, D. S. Hochbaum, and Q. Spaen. High-performance geometric algorithms for sparse computation in big data analytics. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 546–555, 2017. doi: 10.1109/BigData.2017.8257970.

P. Baumann, D. Hochbaum, and Y. Yang. A comparative study of the leading machine learning techniques and two new optimization algorithms. *European Journal of Operational Research*, 272(3):1041–1057, 2019. doi: 10.1016/j.ejor.2018.07.009.

L. Becchetti, C. Castillo, D. Donato, S. Leonardi, and R. Baezayates. Linkbased characterization and detection of web spam. In *2nd International Workshop on Adversarial Information Retrieval on the Web, AIRWeb 2006-29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2006*, 2006.

L. Becchetti, C. Castillo, D. Donato, S. Leonardi, and R. Baeza-Yates. Web spam detection: Link-based and content-based techniques. In *The European Integrated Project Dynamically Evolving, Large Scale Information Systems (DELIS): Proceedings of the Final Workshop*, volume 222, pages 99–113, 2008.

P. Berens, J. Freeman, T. Deneux, N. Chenkov, T. McColgan, A. Speiser, J. H. Macke, S. Turaga, P. Mineault, P. Rupprecht, S. Gerhard, R. W. Friedrich, J. Friedrich, L. Paninski, M. Pachitariu, K. D. Harris, B. Bolte, T. A. Machado, D. Ringach, J. Stone, L. E. Rogerson, N. J. Sofroniew, J. Reimer, E. Froudarakis, T. Euler, M. Roman Roson, L. Theis, A. S. Tolias, and M. Bethge. Community-based benchmarking improves spike rate inference from two-photon calcium imaging data. *PLoS Computational Biology*, 14(5):e1006157, 2018.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for Hyper-Parameter Optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.

J. A. Blackard and D. J. Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24(3):131–151, 1999. doi: 10.1016/S0168-1699(99)00046-0.

U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner. Maximizing modularity is hard. *arXiv preprint physics/0608255*, 2006.

L. Breiman. Bias, variance, and arcing classifiers. Technical Report 460, Statistics Department, University of California, Berkeley, 1996.

R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pages 161–168, Pittsburgh, PA, 2006.

C. Castillo, D. Donato, L. Becchetti, P. Boldi, S. Leonardi, M. Santini, and S. Vigna. A Reference Collection for Web Spam. *SIGIR Forum*, 40(2):11–24, 2006. doi: 10.1145/1189702.1189703.

C. Castillo, D. Donato, A. Gionis, V. Murdock, and F. Silvestri. Know your neighbors: Web spam detection using the web topology. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 423–430. ACM, 2007.

B. G. Chandran and D. S. Hochbaum. A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem. *Operations Research*, 57(2):358–376, 2009.

CodeNeuro. The Neurofinder Challenge. `http://neurofinder.codeneuro.org/`, 2017.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. doi: 10.1007/BF00994018.

K. Dembczyński, W. Kotłowski, and R. Słowiński. Learning rule ensembles for ordinal classification with monotonicity constraints. *Fundamenta Informaticae*, 94(2):163–178, 2009. doi: 10.3233/FI-2009-124.

F. Diego-Andilla and F. A. Hamprecht. Sparse space-time deconvolution for calcium image analysis. In *Advances in Neural Information Processing Systems*, pages 64–72, 2014.

Dong, Jian-xiong, A. Krzyzak, and C. Y. Suen. Fast SVM training algorithm with decomposition on very large data sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(4):603–618, 2005. doi: 10.1109/TPAMI.2005.77.

P. Drineas, R. Kannan, and . Mahoney. Fast monte carlo algorithms for matrices III: Computing a compressed approximate matrix decomposition. *SIAM J. Computing*, 36: 184–206, 2006.

L. N. Driscoll, N. L. Pettit, M. Minderer, S. N. Chettih, and C. D. Harvey. Dynamic reorganization of neuronal activity patterns in parietal cortex. *Cell*, 170(5):986–999, 2017.

M. Erdélyi, A. Garzó, and A. A. Benczúr. Web spam classification: A few features worth more. In *Proceedings of the 2011 Joint WICOW/AIRWeb Workshop on Web Quality*, pages 27–34. ACM, 2011.

M. Fire, D. Kagan, A. Elyashar, and Y. Elovici. Friend or foe? Fake profile identification in online social networks. *Social Network Analysis and Mining*, 4(1):194, 2014.

B. Fishbain, D. S. Hochbaum, and Y. T. Yang. Real-time robust target tracking in videos via graph-cuts. In *Real-Time Image and Video Processing 2013*, volume 8656, page 865602. International Society for Optics and Photonics, 2013. doi: 10.1117/12.2002947.

E. Fix and . Hodges, Jr. Discriminatory analysis, nonparametric discrimination, consistency properties. *Randolph Field, Texas, Project 21-49-004, Report No. 4*, 1951.

P. W. Frey and D. J. Slate. Letter recognition using Holland-style adaptive classifiers. *Machine Learning*, 6(2):161–182, 1991. doi: 10.1007/BF00114162.

G. Gallo, M. Grigoriadis, and R. Tarjan. A Fast Parametric Maximum Flow Algorithm and Applications. *SIAM Journal on Computing*, 18(1):30–55, 1989. doi: 10.1137/0218003.

Q. Gan and T. Suel. Improving web spam classifiers using link structure. In *Proceedings of the 3rd International Workshop on Adversarial Information Retrieval on the Web*, pages 17–20. ACM, 2007.

S. Gao. Conv2d: Convolutional neural network. `https://github.com/iamshang1/Projects/tree/master/Advanced_ML/Neuron_Detection`, 2016.

M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete problems. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, pages 47–63. ACM, 1974.

S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6 (6):721–741, 1984.

S. Ghosh, B. Viswanath, F. Kooti, N. K. Sharma, G. Korlam, F. Benevenuto, N. Ganguly, and K. P. Gummadi. Understanding and combating link farming in the twitter social network. In *Proceedings of the 21st International Conference on World Wide Web*, pages 61–70. ACM, 2012.

A. V. Goldberg and R. E. Tarjan. A New Approach to the Maximum-flow Problem. *J. ACM*, 35(4):921–940, 1988. doi: 10.1145/48014.61051.

M. Gori and A. Pucci. Research paper recommender systems: A random-walk based approach. In *Web Intelligence, 2006. WI 2006. IEEE/WIC/ACM International Conference On*, pages 778–781. IEEE, 2006.

B. F. Grewe, D. Langer, H. Kasper, B. M. Kampa, and F. Helmchen. High-speed in vivo calcium imaging reveals neuronal network activity with near-millisecond precision. *Nature Methods*, 7(5):399–405, 2010.

Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating Web Spam with Trustrank. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 576–587, Toronto, Canada, 2004. VLDB Endowment.

A. Hagberg, P. Swart, and D. S Chult. Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

D. Hochbaum and P. Baumann. Sparse computation for large-scale data mining. In J. Lin, J. Pei, X. Hu, W. Chang, R. Nambiar, C. Aggarwal, N. Cercone, V. Honavar, J. Huan, B. Mobasher, and S. Pyne, editors, *Proceedings of the 2014 IEEE International Conference on Big Data*, pages 354–363, Washington DC, 2014.

D. S. Hochbaum. An Efficient Algorithm for Image Segmentation, Markov Random Fields and Related Problems. *J. ACM*, 48(4):686–701, 2001. doi: 10.1145/502090.502093.

D. S. Hochbaum. Solving integer programs over monotone inequalities in three variables: A framework for half integrality and good approximations. *European Journal of Operational Research*, 140(2):291–321, 2002. doi: 10.1016/S0377-2217(02)00071-1.

D. S. Hochbaum. The Pseudoflow Algorithm: A New Algorithm for the Maximum-Flow Problem. *Operations Research*, 56(4):992–1009, 2008. doi: 10.1287/opre.1080.0524.

D. S. Hochbaum. Polynomial Time Algorithms for Ratio Regions and a Variant of Normalized Cut. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(5):889–898, 2010. doi: 10.1109/TPAMI.2009.80.

D. S. Hochbaum. Multi-Label Markov Random Fields as an Efficient and Effective Tool for Image Segmentation, Total Variations and Regularization. *Numerical Mathematics: Theory, Methods and Applications*, 6(1):169–198, 2013a. doi: 10.1017/S1004897900001185.

D. S. Hochbaum. A Polynomial Time Algorithm for Rayleigh Ratio on Discrete Variables: Replacing Spectral Techniques for Expander Ratio, Normalized Cut, and Cheeger Constant. *Operations Research*, 61(1):184–198, 2013b. doi: 10.1287/opre.1120.1126.

D. S. Hochbaum and P. Baumann. Sparse Computation for Large-Scale Data Mining. *IEEE Transactions on Big Data*, 2(2):151–174, 2016. doi: 10.1109/TBDATA.2016.2576470.

D. S. Hochbaum and A. Levin. Methodologies and Algorithms for Group-Rankings Decision. *Management Science*, 52(9):1394–1408, 2006. doi: 10.1287/mnsc.1060.0540.

D. S. Hochbaum and S. Liu. Adjacency-Clustering and Its Application for Yield Prediction in Integrated Circuit Manufacturing. *Operations Research*, 66(6):1571–1585, 2018. doi: 10.1287/opre.2018.1741.

D. S. Hochbaum and E. Moreno-Centeno. Country credit-risk rating aggregation via the separation-deviation model. *Optimization Methods and Software*, 23(5):741–762, 2008. doi: 10.1080/10556780802402432.

D. S. Hochbaum and J. B. Orlin. Simplifications and speedups of the pseudoflow algorithm. *Networks*, 61(1):40–57, 2013. doi: 10.1002/net.21467.

D. S. Hochbaum, E. Moreno-Centeno, P. Yelland, and R. A. Catena. Rating Customers According to Their Promptness to Adopt New Products. *Operations Research*, 59(5): 1171–1183, 2011. doi: 10.1287/opre.1110.0963.

D. S. Hochbaum, C.-N. Hsu, and Y. T. Yang. Ranking of multidimensional drug profiling data by fractional-adjusted bi-partitional scores. *Bioinformatics*, 28(12):i106–i114, 2012. doi: 10.1093/bioinformatics/bts232.

D. S. Hochbaum, C. Lyu, and E. Bertelli. Evaluating performance of image segmentation criteria and techniques. *EURO Journal on Computational Optimization*, 1(1):155–180, 2013. doi: 10.1007/s13675-012-0002-8.

S. Jewell and D. Witten. Exact spike train inference via ℓ_{0} optimization. *The Annals of Applied Statistics*, 12(4):2457–2482, 2018. doi: 10.1214/18-AOAS1162.

C. Jhurani. Subspace-preserving sparsification of matrices with minimal perturbation to the near null-space. Part I: Basics. *arXiv:1304.7049 [cs, math]*, 2013.

H. Jia, N. L. Rochefort, X. Chen, and A. Konnerth. In vivo two-photon imaging of sensory-evoked dendritic calcium signals in cortical neurons. *Nature Protocols*, 6(1):28, 2011.

P. Kaifosh, J. D. Zaremba, N. B. Danielson, and A. Losonczy. SIMA: Python software for analysis of dynamic fluorescence imaging data. *Frontiers in Neuroinformatics*, 8, 2014.

H. Kawaji, Y. Takenaka, and H. Matsuda. Graph-based clustering for finding distant relationships in a large set of protein sequences. *Bioinformatics*, 20(2):243–252, 2004. doi: 10.1093/bioinformatics/btg397.

Y. Kim, S.-W. Son, and H. Jeong. Finding communities in directed networks. *Physical Review E*, 81(1):016103, 2010.

J. Kleinberg and É. Tardos. Approximation Algorithms for Classification Problems with Pairwise Relationships: Metric Labeling and Markov Random Fields. *J. ACM*, 49(5): 616–639, 2002. doi: 10.1145/585265.585268.

A. Klibisz, D. Rose, M. Eicholtz, J. Blundon, and S. Zakharenko. Fast, Simple Calcium Imaging Segmentation with Fully Convolutional Networks. In M. J. Cardoso, T. Arbel, G. Carneiro, T. Syeda-Mahmood, J. M. R. Tavares, M. Moradi, A. Bradley, H. Greenspan, J. P. Papa, A. Madabhushi, J. C. Nascimento, J. S. Cardoso, V. Belagiannis, and Z. Lu, editors, *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, Lecture Notes in Computer Science, pages 285–293. Springer International Publishing, 2017.

R. Kohavi. Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-tree Hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 202–207. AAAI Press, 1996.

V. Krishnan and R. Raj. Web Spam Detection with Anti-Trust Rank. In *Proceedings of the Second International Workshop on Adversarial Information Retrieval on the Web*, pages 45–48, 2006.

J. Kunegis. Konect: The koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350. ACM, 2013.

Y. Lee and O. Mangasarian. RSVM: Reduced Support Vector Machines. In *Proceedings of the 2001 SIAM International Conference on Data Mining*, Proceedings, pages 1–17. Society for Industrial and Applied Mathematics, 2001. doi: 10.1137/1.9781611972719.13.

Y. Levin-Schwartz, D. R. Sparta, J. F. Cheer, and T. Adalı. Parameter-free automated extraction of neuronal signals from calcium imaging data. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1033–1037. IEEE, 2017.

M. Lichman. UCI Machine Learning Repository. `http://archive.ics.uci.edu/ml`, 2013.

T.-Y. Liu. Learning to Rank for Information Retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009. doi: 10.1561/1500000016.

F. D. Malliaros and M. Vazirgiannis. Clustering and community detection in directed networks: A survey. *Physics Reports*, 533(4):95–142, 2013.

R. Maruyama, K. Maeda, H. Moroda, I. Kato, M. Inoue, H. Miyakawa, and T. Aonishi. Detecting cells using non-negative matrix factorization on calcium imaging data. *Neural Networks*, 55:11–19, 2014.

M. Meilă and W. Pentney. Clustering by weighted cuts in directed graphs. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 135–144. SIAM, 2007.

E. A. Mukamel, A. Nimmerjahn, and M. J. Schnitzer. Automated analysis of cellular signals from large-scale calcium imaging data. *Neuron*, 63(6):747–760, 2009.

M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006. doi: 10.1073/pnas.0601602103.

M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113, 2004. doi: 10.1103/PhysRevE.69.026113.

A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly. Detecting spam web pages through content analysis. In *Proceedings of the 15th International Conference on World Wide Web*, pages 83–92. ACM, 2006.

M. Pachitariu, A. M. Packer, N. Pettit, H. Dalgleish, M. Hausser, and M. Sahani. Extracting regions of interest from biological images with convolutional sparse block coding. In *Advances in Neural Information Processing Systems*, pages 1745–1753, 2013.

M. Pachitariu, C. Stringer, M. Dipoppa, S. Schröder, L. F. Rossi, H. Dalgleish, M. Carandini, and K. D. Harris. Suite2p: Beyond 10,000 neurons with standard two-photon microscopy. *bioRxiv*, page 061507, 2017. doi: 10.1101/061507.

L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Techreport SIDL-WP-1999-0120, Stanford InfoLab, Palo Alto, CA, USA, 1999.

E. A. Pnevmatikaki and L. Paninski. Sparse nonnegative deconvolution for compressive calcium imaging: Algorithms and phase transitions. In *Advances in Neural Information Processing Systems*, pages 1250–1258, 2013.

E. A. Pnevmatikakis, J. Merel, A. Pakman, and L. Paninski. Bayesian spike inference from calcium imaging data. In *Asilomar Conference on Signals, Systems and Computers*, pages 349–353, 2013.

E. A. Pnevmatikakis, D. Soudry, Y. Gao, T. A. Machado, J. Merel, D. Pfau, T. Reardon, Y. Mu, C. Lacefield, W. Yang, M. Ahrens, R. Bruno, T. M. Jessell, D. S. Peterka, R. Yuste, and L. Paninski. Simultaneous denoising, deconvolution, and demixing of calcium imaging data. *Neuron*, 89(2):285–299, 2016.

R. Prevedel, Y.-G. Yoon, M. Hoffmann, N. Pak, G. Wetzstein, S. Kato, T. Schrödel, R. Raskar, M. Zimmer, E. S. Boyden, and A. Vaziri. Simultaneous whole-animal 3D imaging of neuronal activity using light-field microscopy. *Nature Methods*, 11(7):727–730, 2014.

J. Qranfal, D. S. Hochbaum, and G. Tanoh. Experimental Analysis of the MRF Algorithm for Segmentation of Noisy Medical Images. *Algorithmic Operations Research*, 6(2):79–90, 2011.

A. Rajaraman and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011.

S. L. Resendez, J. H. Jennings, R. L. Ung, V. M. K. Namboodiri, Z. C. Zhou, J. M. Otis, H. Nomura, J. A. McHenry, O. Kosyk, and G. D. Stuber. Visualization of cortical, subcortical, and deep brain neural circuit dynamics during naturalistic mammalian behavior with head-mounted microscopes and chronically implanted lenses. *Nature Protocols*, 11(3): 566, 2016.

M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.

R. K. Roul, S. R. Asthana, M. Shah, and D. Parikh. Spam web page detection using combined content and link features. *International Journal of Data Mining, Modelling and Management*, 8(3):209–222, 2016.

Y. U. Ryu, R. Chandrasekaran, and V. Jacob. Prognosis Using an Isotonic Prediction Technique. *Management Science*, 50(6):777–785, 2004. doi: 10.1287/mnsc.1030.0137.

H. Saito, M. Toyoda, M. Kitsuregawa, and K. Aihara. A large-scale study of link spam detection by graph algorithms. In *Proceedings of the 3rd International Workshop on Adversarial Information Retrieval on the Web*, pages 45–48. ACM, 2007.

H. Sayyadi and L. Getoor. Futurerank: Ranking scientific articles by predicting their future pagerank. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 533–544. SIAM, 2009.

I. Schmidtmann, G. Hammer, M. Sariyar, and A. Gerhold-Ay. Evaluation des krebsregisters nrw-schwerpunkt record linkage-abschlussbericht. Technical report, Institut für medizinische Biometrie, Epidemiologie und Informatik, Universitätsmedizin Mainz, Mainz, Germany, 2009.

E. Sharon, M. Galun, D. Sharon, R. Basri, and A. Brandt. Hierarchy and adaptivity in segmenting visual scenes. *Nature*, 442(7104):810–813, 2006.

J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. doi: 10.1109/34.868688.

K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu. Fake news detection on social media: A data mining perspective. *ACM SIGKDD Explorations Newsletter*, 19(1):22–36, 2017.

K. Shu, H. R. Bernard, and H. Liu. Studying fake news via network analysis: Detection and mitigation. In *Emerging Research Challenges and Opportunities in Computational Social Network Analysis and Mining*, pages 43–65. Springer, 2019.

Q. Spaen, R. Asín-Achá, S. N. Chettih, M. Minderer, C. Harvey, and D. S. Hochbaum. HNCcorr: A Novel Combinatorial Approach for Cell Identification in Calcium-Imaging Movies. *eNeuro*, 6(2):ENEURO.0304–18.2019, 2019. doi: 10.1523/ENEURO.0304-18.2019.

D. Spielman and S. Teng. Spectral Sparsification of Graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011. doi: 10.1137/08074489X.

C. Stosiek, O. Garaschuk, K. Holthoff, and A. Konnerth. In vivo two-photon calcium imaging of neuronal networks. *Proceedings of the National Academy of Sciences*, 100(12):7319–7324, 2003.

L. Theis, P. Berens, E. Froudarakis, J. Reimer, M. R. Rosón, T. Baden, T. Euler, A. S. Tolias, and M. Bethge. Benchmarking spike rate inference in population calcium imaging. *Neuron*, 90(3):471–482, 2016.

P. Törnberg. Echo chambers and viral misinformation: Modeling fake news as complex contagion. *PloS one*, 13(9):e0203958, 2018.

V. Tresp. A Bayesian Committee Machine. *Neural Computation*, 12(11):2719–2741, 2000. doi: 10.1162/089976600300014908.

I. W. Tsang, J. T. Kwok, and P.-M. Cheung. Core Vector Machines: Fast SVM Training on Very Large Data Sets. *Journal of Machine Learning Research*, 6(Apr):363–392, 2005.

J. T. Vogelstein, A. M. Packer, T. A. Machado, T. Sippy, B. Babadi, R. Yuste, and L. Paninski. Fast nonnegative deconvolution for spike train inference from population calcium imaging. *Journal of Neurophysiology*, 104(6):3691–3704, 2010.

U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

S. Webb, J. Caverlee, and C. Pu. Predicting web spam with HTTP session information. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 339–348. ACM, 2008.

. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, San Francisco, 2nd ed. edition, 2005.

World Economic Forum. Global Risks Report 2013. Technical report, World Economic Forum, Cologny/Geneva, Switzerland, 2013. OCLC: 936624248.

B. Wu and K. Chellapilla. Extracting link spam using biased random walks from spam seed sets. In *Proceedings of the 3rd International Workshop on Adversarial Information Retrieval on the Web*, pages 37–44. ACM, 2007.

B. Wu and B. D. Davison. Identifying link farm spam pages. In *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web*, pages 820–829. ACM, 2005.

C. Xiao, D. M. Freeman, and T. Hwa. Detecting clusters of fake accounts in online social networks. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pages 91–101. ACM, 2015.

Y. T. Yang, B. Fishbain, D. S. Hochbaum, E. B. Norman, and E. Swanberg. The Supervised Normalized Cut Method for Detecting, Classifying, and Identifying Special Nuclear Materials. *INFORMS Journal on Computing*, 26(1):45–58, 2013. doi: 10.1287/ijoc.1120.0546.

D. Zhou, C. J. Burges, and T. Tao. Transductive link spam detection. In *Proceedings of the 3rd International Workshop on Adversarial Information Retrieval on the Web*, pages 21–28. ACM, 2007.

X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. In *Proceedings of the 20th International Conference on Machine Learning*, pages 912–919, Washington, DC, USA, 2003. AAAI.