# Learning and Reasoning with Fast Semidefinite Programming and Mixing Methods

Po-Wei Wang

July 2021
**CMU-ML-21-107**

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

**Thesis Committee**

J. Zico Kolter, Chair
Gary Miller
Ryan Tibshirani
Fatma Kılınç-Karzan
Max Welling

*Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.*



This research was supported by: the Air Force Research Laboratory award FA87501720027; the Defense Advanced Research Projects Agency award N660011714036; the National Science Foundation award CCF1522054; and grants from Duquesne Light Company, Google, Robert Bosch GmbH, and the Siebel Energy Institute.

**Keywords:** semidefinite programming, approximation algorithm, machine learning

*Dedicated to my loving family, who constantly stand behind me while I am pursuing my dream.*

# Abstract

Semidefinite programming has long been a theoretically powerful tool for solving relaxations of challenging, often NP-hard optimization problems. However, it has typically not been practical for most large-scale tasks, owing to the high memory and computational cost of typical solvers for solving SDPs. In this thesis, we aim to break the barrier and bring SDP's power back to large-scale machine learning problems. To achieve this, we introduce a series of optimization solvers, operating on the low-rank or low-cardinality manifolds of the semidefinite variables. We find that in many domains, these methods allow SDP relaxations to exceed the state of the art in terms of both computational cost and the relevant performance metrics.

First, we proposed the Mixing method, a low-rank SDP solver aimed at the classical MAXCUT SDP relaxation. We also show that the Mixing method can accurately estimate the mode and partition function of the pairwise Markov Random Fields, and scales to millions of variables. Further, we show how to learn the parameters inside SDPs by analytically differentiating through the optimization problem with implicit differentiation and the mixing methods, which leads to a differentiable SAT solver that can be integrated within the loop of larger deep learning systems. For nonnegative constraints, we propose a separate variant aimed at low cardinality SDPs, and demonstrate how to apply the method to community detection on finding clusters within large-scale networks. Finally, we show that the technique can also be applied to more generic problems, such as a *generic* linear programming problems (with arbitrarily structured constraints), and we use this approach to develop a scalable sparse linear programming solver that improves solution speed over existing state-of-the-art commercial solvers.

# Acknowledgments

First and foremost, I want to express my gratitude to my advisor, Zico Kolter, the person without whom the thesis would not have been possible. Zico is the kindest advisor a PhD student could have asked for. He is always inspiring, having magical instincts on do's and don'ts, while giving me abundant freedom to grow and explore as a researcher; From him, I learned not only vision and tastes about machine learning and optimization, but also how to be kind to others while keeping the progress. Many thanks are also due to Gary Miller, Ryan Tibshirani, Fatma Kılınç-Karzan, and Max Welling for serving on my thesis committee. Specifically, I'd like to thank Gary for those afternoon discussions, which allows me to see my topics through a different lens. I want to thank Ryan for our collaborations on the convex optimization courses, and I'd like to thank Fatma for encouraging me to continue working on SDPs during my bottlenecks. Finally, I'd like to thank Max for his vision of my works and for the opportunities during the job search.

During my PhD journey, I am fortunate to collaborate with many brilliant researchers, including Priya Donti, Shaojie Bai, Matt Wytock, Bryan Wilder, Wei-Cheng Chang, Chirag Pabbaraju, Chun Kai Ling, Chun-Liang Li, and Ching-Pei Lee. Many of our collaborations contribute to this thesis, and I learned a lot from them in research, mentorship, and being nice to others. Especially, I am grateful to Priya for always providing me feedback and encouragement. I appreciate Eric Wong for introducing me to Zico and for our foodie trips. And I thank Cho-Jui Hsieh and Yen-Chi Chen for being the role models in my early stage of research.

Special thanks to my officemates for their discussions and conversations, and for creating a comfortable working environment. Shout-out to Maruan Al-Shedivat, Yifei Ma, Avi Dubey, Xun Zheng, Anthony Platanios, Otilia Stretcu, and Ivan Stelmakh for our chats. I'd also like to thank my friends in the machine learning department and computer science department, including Alnur Ali, Brandon Amos, Vaishnavh Nagarajan, Gaurav Manek, En-Hsu Yen, Derek Liu, Simon Shaolei Du, Wei Dai, Han Zhao, Akash Umakantha, Jeremy Cohen, and Ezra Winston. I will definitely miss those academic coffee talks. Further, I'd like to give my gratitude to my board game mates, Shao-Wen Chiu, Yen-Chia Hsu, and Wei-Yu Chen. From Pandemic to Gloomhaven, we have so many wonderful memories. And I have to thank my pals Ting-Yao Hu, Carol Cheng, Hsiao-Yu Tung, Chih-Kuan Yeh, Yao-Hung Tsai, Ting Hsieh, Bingbin Liu, Huan Zhang, Mu-Chu Lee, Tzu-Ming Kuo, and Sz-Rung Shiang for the numerous lunches and dinners together.

Looking back to my journey of research, I need to express my gratitude to my advisor at National Taiwan University, Chih-Jen Lin, for opening the door to machine learning and optimization for me. I learned from him not only about how to do research but also about being a decent researcher. I need to thank Chieh-Chih Wang for letting me participate in his robotics seminar early in my undergrad, which had a lasting influence on my taste in research. I want to thank Diane Stidle for her warm greetings and care to the entire machine learning department. And I appreciate the opportunities and mentorship during my internships. I am grateful to Inderjit Dhillon

and Vijai Mohan from A9 for the great experience in my first-ever internship, and I thank Daria Stepanova and Csaba Domokos from Bosch for teaching me answer set programming and our time in Europe.

Last but not least, I owe my deepest gratitude to my family. I am forever grateful to my father and mother, Kuo-Liang Wang and Mun-Ying Lin, for their unconditional love and dedication. I am grateful to my sibling, Po-Ya Wang, for her company and encouragement. Finally, I want to thank my best friend, life partner, and beloved wife, Chieh Lin, who has been supporting me in the ebb and flow of my PhD, while providing unswerving love and happiness to my life.

# Contents


**1 Introduction** **1**
- 1.1 Thesis outline . . . 1
- 1.2 Itemized summary of contributions . . . 3

**2 Background** **5**
- 2.1 Semidefinite programming . . . 5
  - 2.1.1 SDP solvers . . . 5
- 2.2 Learning by implicit differentiation . . . 7
- 2.3 Reasoning with semidefinite programs . . . 8
  - 2.3.1 Approximation for NP-hard problems . . . 8
  - 2.3.2 Inference in Markov Random Field . . . 8
  - 2.3.3 Learning to reason with a differentiable satisfiability layer . . . 9
  - 2.3.4 Community detection with modularity maximization . . . 10
- 2.4 Linear programming via efficient piecewise quadratic optimization . . . 11

**3 The Mixing method for low-rank SDPs** **13**
- 3.1 The Mixing method . . . 15
  - 3.1.1 Convergence properties of the Mixing methods . . . 16
  - 3.1.2 Proof of Theorem 3.4: The instability of non-optimal criticals points . . . 18
- 3.2 Applications . . . 22
  - 3.2.1 Maximum cut problem . . . 22
  - 3.2.2 Maximum satisfiability problem . . . 23
- 3.3 Experimental results . . . 24
- 3.4 Discussion . . . 27

**4 Inference in Markov Random Field with fast low-rank SDPs** **29**
- 4.1 Inference in multi-class MRFs . . . 31
- 4.2 Partition function estimation . . . 34
- 4.3 Experimental results . . . 35
  - 4.3.1 Mode estimation . . . 36
  - 4.3.2 Partition function estimation . . . 37
  - 4.3.3 Image segmentation . . . 38
- 4.4 Discussion . . . 39

**5 Learning to reason with a differentiable satisfiability layer** **41**
5.1 A differentiable satisfiability solver . . . 42
5.1.1 Solving an SDP formulation of satisfiability . . . 42
5.1.2 SATNet: Satisfiability solving as a layer . . . 43
5.1.3 Computing the backward pass . . . 46
5.1.4 An efficient GPU implementation . . . 47
5.2 Experimental results . . . 48
5.2.1 Learning parity (chained XOR) . . . 48
5.2.2 Sudoku (original and permuted) . . . 50
5.2.3 Visual Sudoku . . . 51
5.3 Discussion . . . 52

**6 Community detection using fast low-cardinality SDPs** **53**
6.1 The Locale algorithm and application to community detection . . . 54
6.1.1 Generalizing the local move procedure by low-cardinality embeddings . . . 54
6.1.2 Rounding by changing the cardinality constraint . . . 58
6.1.3 The Leiden-Locale algorithm for community detection . . . 58
6.2 Experimental results . . . 59
6.3 Discussion . . . 62

**7 Linear programming via efficient piecewise quadratic optimization** **63**
7.1 The unconstrained piecewise quadratic formulation . . . 64
7.2 The QULP algorithm . . . 66
7.2.1 The coordinate descent method with linear convergence . . . 66
7.2.2 The semismooth Newton method with superlinear convergence . . . 68
7.2.3 A hybrid method . . . 69
7.3 Experimental results . . . 70
7.4 Conclusion . . . 71

**8 Conclusion** **73**

**A Proofs for the Mixing methods** **75**
A.1 Proof of Theorem 3.1: Convergence to critical points . . . 75
A.2 Proof of Theorem 3.3: Local Linear convergence . . . 76
A.3 Proof of Lemma 3.10: Divergence of Gauss-Seidel methods . . . 79
A.4 Proof of Theorem 3.5: Global convergence with a step size . . . 82
A.5 Proof of Lemma 3.11: Rank Deficiency in Critical Points . . . 84

**B Proofs and additional experimental results for MRF** **87**
B.1 Proof of Equivalence of (4.8) and (4.9) . . . 87
B.2 Derivation of (4.11) . . . 89
B.3 Proof of Theorem 4.1 . . . 93
B.4 Pseudocode for AIS . . . 94
B.5 Mode estimation comparisons . . . 95

B.6 Performance of AIS with varying parameters . . . 96
B.7 Image Segmentation - more results . . . 97

**C Derivations and additional experimental results for SATNet** **101**
C.1 Derivation of the forward pass coordinate descent update . . . 101
C.2 Details on backpropagation through the MAXSAT SDP . . . 102
C.3 Proof of pseudoinverse computations . . . 103
C.4 Derivation of the backward pass coordinate descent algorithm . . . 104
C.5 Results for the $4 \times 4$ Sudoku problem . . . 105
C.6 Convergence plots for $9 \times 9$ Sudoku experiments . . . 105

**D Proofs for the Locale algorithm** **109**
D.1 Proof of Proposition 6.1 . . . 109
D.2 Proof of Theorem 6.2 . . . 111
D.3 Proof for Lemma D.1 . . . 112
D.4 Experiments on networks with ground truth . . . 112
D.5 Pseudo-code for the Leiden-Locale algorithm . . . 113

**E Proofs for the linear programming algorithm** **115**
E.1 Proof of Theorem 7.3 . . . 115
E.1.1 Global Convergence . . . 115
E.1.2 Local Superlinear Convergence With Unit Step Size . . . 117
E.1.3 Unit Step Size Acceptance . . . 118
E.2 Other Proofs . . . 120
E.2.1 Proof of Theorem 7.2 . . . 120
E.3 Additional Experiments . . . 121
E.3.1 Benchmark Problems . . . 121

**Bibliography** **125**

# Chapter 1

# Introduction

Semidefinite programs (SDP) has long been a theoretical powerful tool for encoding a large range of practical problems, including relaxations of many combinatorial optimization tasks [34], approximate probabilistic inference [78], control theory [35], matrix completion [41], and many others. Also, it provides a systematic way of approximating the underlying NP-hard optimization problems [88, 121] in many machine learning applications. However, it is typically not practical for most large-scale machine learning problems, owing to the high memory and computational cost of typical solvers for solving SDPs. So despite the many progress and theoretical usages, SDP has not been widely adopted in solving real-world tasks.

In this thesis, we aim to break the barrier and unleash SDP's power back to large-scale machine learning problems. To achieve the goal, we introduce a series of optimization solvers, operating on the low-rank or low-cardinality manifolds of the semidefinite variables to leverage the problem structures. We find that in domains including probabilistic inference, parameter learning, and clustering, the proposed methods allow SDP relaxations to exceed the state of the art in terms of both computational cost and the relevant performance metrics. Further, we show that the technique can be generalized to scale up linear programming.

## 1.1 Thesis outline

In this initial chapter, we give an overview of the thesis and summarize the existing contributions. Chapter 2 gives more detailed presentation on the background of SDPs and their applications in machine learning. In Chapter 3, we present the Mixing method, the core tool we use to scale up SDPs, together with its theoretical properties. The following three chapters present the works on applying the Mixing methods or its variants on the three important problems in machine learning: probabilistic inference, parameter learning, and clustering (Chapter 4, 5, 6 respectively). Then, in Chapter 7, we demonstrate how to apply the solution technique to general linear constraints in a linear programming problem.

**Chapter 3: The Mixing method for low-rank SDPs.** We presents a low-rank block coordinate descent approach to structured semidefinite programming with diagonal constraints. The approach, which we call the Mixing method, is extremely simple to implement, has no free parameters,

and typically attains an order of magnitude or better improvement in optimization performance over the current state of the art. We show that the method is strictly decreasing, will converge to a critical point, and further that for sufficient rank all non-optimal critical points are unstable. Moreover, we prove that with a step size, the Mixing method converges to the global optimum of the semidefinite program almost surely in a locally linear rate under random initialization. This is one of the first low-rank semidefinite programming method that has been shown to achieve a global optimum without assumption. We demonstrate the algorithm to two related domains: solving the maximum cut semidefinite relaxation, and solving a maximum satisfiability relaxation. In all settings, we demonstrate substantial improvement over the existing state of the art along various dimensions, and in total, this work expands the scope and scale of problems that can be solved using semidefinite programming methods and inspires all the subsequent chapters.

**Chapter 4: Inference in Markov Random Field with fast low-rank SDPs.** Probabilistic inference in pairwise Markov Random Fields (MRFs), i.e. computing the partition function or computing a MAP estimate of the variables, is a foundational problem in probabilistic graphical models. Semidefinite programming relaxations have long been a theoretically powerful tool for analyzing properties of probabilistic inference, but have not been practical owing to the high computational cost of typical solvers for solving the resulting SDPs. In this chapter, we propose an efficient method for computing the partition function or MAP estimate in a pairwise MRF by instead exploiting a recently proposed coordinate-descent-based fast semidefinite solver. We also extend semidefinite relaxations from the typical binary MRF to the full multi-class setting, and develop a compact semidefinite relaxation that can again be solved efficiently using the solver. We show that the method substantially outperforms (both in terms of solution quality and speed) the existing state of the art in approximate inference, on benchmark problems drawn from previous work. We also show that our approach can scale to large MRF domains such as fully-connected pairwise CRF models used in computer vision.

**Chapter 5: Learning to reason with a differentiable satisfiability layer.** Integrating logical reasoning within deep learning architectures has been a major goal of modern AI systems. In this chapter, we propose a new direction toward this goal by introducing a differentiable (smoothed) maximum satisfiability (MAXSAT) solver that can be integrated into the loop of larger deep learning systems. Our approximate solver is based upon a fast coordinate descent approach to solving the SDP associated with the MAXSAT problem. We show how to analytically differentiate through the solution to this SDP and efficiently solve the associated backward pass, which enables parameters learning for the clause matrix. We demonstrate that by integrating this solver into end-to-end learning systems, we can learn the logical structure of challenging problems in a minimally supervised fashion. In particular, we show that we can learn the parity function using single-bit supervision (a traditionally hard task for deep networks) and learn how to play $9 \times 9$ Sudoku solely from examples. We also solve a "visual Sudoku" problem that maps images of Sudoku puzzles to their associated logical solutions by combining our MAXSAT solver with a traditional convolutional architecture.

**Chapter 6: Community detection using fast low-cardinality SDPs.** Modularity maximization has been a fundamental tool for understanding the community structure of a network, but the underlying optimization problem is nonconvex and NP-hard to solve. State-of-the-art algorithms like the Louvain or Leiden methods focus on different heuristics to help escape local optima, but they still depend on a greedy step that moves node assignment locally and is prone to getting trapped. In this chapter, we propose a new class of low-cardinality algorithm that generalizes the local update to maximize a semidefinite relaxation derived from max-k-cut. This proposed algorithm is scalable, empirically achieves the global semidefinite optimality for small cases, and outperforms the state-of-the-art algorithms in real-world datasets with little additional time cost. From the algorithmic perspective, it also opens a new avenue for scaling-up semidefinite programming when the solutions are sparse instead of low-rank.

**Chapter 7:Linear programming via efficient piecewise quadratic optimization.** We present a new approach to solving large-scale, structured linear programs (LPs). Specifically, we show that the primal-dual solution pair of any feasible LP can be formulated as an unconstrained piecewise convex quadratic objective, which requires only a few passes over the constraint matrix to compute and differentiate (and without the need for any matrix inversion). The formulation additionally requires only a single objective, without the need for scheduling any relaxation parameters (e.g., a central path parameter or augmented Lagrangian relaxation variable), as is common in most comparable linear programming solvers. We then propose an efficient algorithm for solving such problems, based upon a combination of coordinate descent and conjugate gradient steps. We show that this algorithm achieves global Q-linear convergence as well as local superlinear convergence. We evaluate the approach empirically on several large-scale machine learning and linear programming applications and show that the method solves problems substantially more quickly than state-of-the-art commercial interior-point and simplex methods as well as a recent augmented Lagrangian approach.

## 1.2 Itemized summary of contributions

- Chapter 3 presents the Mixing method, an low-rank algorithm that enables the SDP to scale to millions of variables. It has a linear dependency to the non-zeros in input and is one of the first solver that provably converges to the global optimum without assumption.
- Chapter 4 demonstrates how to apply the low-rank method to Markov Random Fields for inferencing the mode and partition function, which outperforms the state-of-the-art approaches in both speed and accuracy.
- Chapter 5 presents a differentiable MAXSAT solver, demonstrating how to learn optimization parameters with implicit differentiation and the Mixing method. It enables the end-to-end integration of logical reasoning and deep learning.
- Chapter 6 presents a variant SDP solver aimed at the low-cardinality solutions. It is applied to the maximum modularity problem in community detection, and outperforms the state of the art in both performance and speed.
- Chapter 7 demonstrates how to convert the linear constraints in a linear program to an

unconstrained piecewise quadratic problem, which is solvable using coordinate descent and second-order methods in linear time.

# Chapter 2

# Background

## 2.1 Semidefinite programming

Semidefinite programming (SDP) is a class of optimization problem involving symmetric matrix variables that are constrained to be positive semidefinite. It can be written as

$$\underset{X \succeq 0}{\text{minimize}} \ \langle C, X \rangle, \ \ \text{subject to} \ \ \langle A_i, X \rangle = b_i, \ \ i = 1 \dots p, \tag{2.1}$$

where $X \in \mathbb{S}^n$ is the optimization variable (a symmetric $n \times n$ matrix), $C \in \mathbb{S}^n$ is the cost matrix, and $A_i \in \mathbb{R}^{n \times n}, b_i \in \mathbb{R}$, $i = 1, \dots, p$ are the constraint coefficients. Semidefinite programs can encode a large range of practical problems, including relaxations of many combinatorial optimization tasks [34], approximate probabilistic inference [78], control theory [35], matrix completion [41], and many others. It has long been a theoretical powerful tool for approximating challenging problems. However, it is typically not practical for most large-scale tasks, owing to the high memory and computational cost of typical solvers for solving SDPs. In this thesis, we aim to break the barrier and bring SDP's power back to large-scale machine learning problems.

### 2.1.1 SDP solvers

**Interior point methods.** The primal-dual interior point methods [104, 160], including Sedumi[136] and DSDP[18], are the canonical solvers for SDPs. They take full Newton steps on the perturbed KKT system and control the tightness of perturbation on a primal-dual central path. Because they need to solve Newton steps for the $n^2$ scalars inside the variable matrix $X \in \mathbb{S}^n$, a naive implementation requires $O(n^2)$ memory for storing $X$ and $O((n^2)^3) = O(n^6)$ computational complexity per iteration. By utilizing the primal-dual relation [142], the cost per iteration can be reduced to $O(pn^4)$. If $p = O(n)$ and the SDP satisfies a certain KYP structure, the cost can be further reduced to $O(n^3)$. However, the overall complexity is still impractical for most large-scale machine learning problems.

**Multiplicative update algorithms.** Starting from the multiplicative weight update algorithms [10, 79] from online learning, there has been a series of breakthroughs on using the multiplicative update to solve traditional optimization problems, including the multiplicative update (MW)

algorithms for SDPs [8, 135] and the matrix multiplicative weight (MMW) algorithm [7] that combines fast matrix exponential with randomized sketching. The core idea of MW for SDP is to convert the optimization problem into a series of membership problems, solving it using oracles and adjust the weight with multiplicative update algorithms. Their complexities are sometimes optimal in certain parameters, but are not really practical because of their bad sublinear dependency on the error. For example, for solving a MAXCUT SDP for a $d$-regular graph with $m$ edges and $n$ nodes to a relative error of $\epsilon$, the MMW algorithm requires a $O(m \cdot 8d^2 \log(n)/\epsilon^2)$ computational complexity, which is optimal for $m$ edges but not practical for a moderate $\epsilon = 0.1$.

**Low-rank methods for SDP.** Given an SDP problem with $p$ constraints, it was proven by Barvinok [16], Pataki [123] that, if the problem is solvable, it admits solutions of rank $k = \lceil \sqrt{2p} \rceil$. That is, we have solutions satisfying $X = V^T V$, such that $V \in \mathbb{R}^{k \times n}$. Thus, if we can solve the problem in the space of $V$, we can ignore the semidefinite constraint and have many fewer variables. The idea of using this low-rank structure during optimization was first proposed by Burer and Monteiro [39] in their solver SDPLR, in which they solve the low-rank problem with L-BFGS on the extended Lagrangian problem. Since then, many low-rank optimization algorithms have been developed. One of the most notable is the Riemannian trust-region method introduced by Absil et al. [1]. They considered the Riemannian manifold of low-rank structures and extended the non-linear conjugate gradient method to work on the manifold. Later, Boumal and Absil [29] improved the method by including preconditioned CG; these methods are implemented in the popular Manopt package [30]. Later, a series of works on Riemannian optimization [1, 28, 30] further established the theoretical framework and extended the domain of applications for the low-rank optimization algorithms. Despite the fact that the low-rank problem is of course nonconvex, in practice this would virtually always find the *optimal* solution to the original SDPs. Although this remained an empirical rather than theoretical property for many years, recently Boumal et al. [32] has shown that there are *no spurious local optima* in the augmented Lagrangian form for these problems. Following the result, Boumal et al. [31, Theorem 12 and Proposition 19] proved that the Riemannian trust-region method converges to a suboptimal solution with Riemannian Hessian (the PSD constraint in the KKT condition) larger than $-\gamma I$ in $O(1/\gamma^3)$, and Bhojanapalli et al. [20] proved the connection between $\gamma$, the norm of gradients, and $f_\mu - f_\mu^*$ for the unconstrained penalty form $f_\mu$.

**The Mixing methods.** To summarize, the Mixing methods proposed in the thesis are a family of first-order feasible optimization algorithms, usually a variant of block coordinate descent methods. It exploits the problem structures and the low-rank [151, 153] or low-cardinality [147] properties of the solutions in the factorized space $V$ of the matrix variable $X = V^T V$. The Mixing methods and variants easily scale to millions of variables and have a complexity linear to the number of non-zeros in the inputs. Empirically, it converges linearly to global optimal solutions of the corresponding SDPs, and for certain cases like in chapter 3, it admits almost surely optimal convergence guarantees without any assumptions.

## 2.2 Learning by implicit differentiation

**Learning from KKT equations.** Once we have an efficient solver for the SDPs, we might want to learn the coefficients or cost matrix inside the optimization problem. This is closely related to the bi-level optimization problem [21, 131]. In this thesis, we focus on integrating the optimization problem as a layer inside a deep learning system and learn the parameters through stochastic gradient methods. That is, we only require the optimization layer to be differentiable to its parameters. Previous work has introduced differentiable modules for quadratic programs [5, 52], submodular optimization problems [51, 141, 158], equilibrium computation in zero-sum games [97], and general conic program [4]. Most of them rely on applying the implicit function theorem on KKT system of the optimization problem, where all the coefficients required to construct the gradient can be obtained from the corresponding primal-dual interior-point method [4]. However, the primal-dual interior-point method is not scalable, so learning from the KKT equations is also not scalable.

**Learning from solvers or fixed-point equations.** Another way to learn the coefficient is by differentiating the iterations inside a solver. We can do this by simply unrolling the optimization steps or by applying implicit function theorem through the fixed-point equation induced by the solver. The latter doesn't require storing the intermediate results, thus needs only constant memory regardless of the depth of optimization steps. A famous example of differentiating through the solver is Neural ODE [44], which models a residual block with "infinitesimal layers" by solving for the endpoint of an ODE flow. SATNet [152] (chapter 5) introduces a differentiable MAXSAT solver that differentiates through the fixed-point equations induced by the coordinate descent methods. Winston and Kolter [159] generalize the domain of differentiable optimization problem to variational equalities and implicitly differentiate through fixed-point of the forward-backward and Peaceman-Rachford splittings. El Ghaoui et al. [54] studies the well-posedness of these implicit-depth models. The deep equilibrium model (DEQ) [13] formulates the output of an infinite-depth sequence model (e.g., a Transformer block) as an equilibrium state, and uses quasi-Newton root solvers to directly solve for this point (and thus scalable). More broadly, the very notion of equilibrium state has also been explored in the original recurrent backpropagation literature [3, 125], which studies a constrained version of RNNs that take the same input at every time step.

Although learning from solvers is already much faster than learning from KKT equations, it is still less efficient and less stable compared to typical neural networks, which has turned into a major bottleneck for limiting these models for practical deployment. For instance, Neural ODE [44] methods so far mostly work on relatively low dimensional tasks (e.g., MNIST) due to the instability in ODE flow solving, and can be $3 - 4\times$ slower than conventional ResNets. Both Neural ODEs [44] (which rely on ODE solvers) and DEQ [13] (which rely on quasi-Newton root solvers) report increasing instability and slowdown of the implicit solvers as training progresses. Moreover, while Bai et al. [13] tries to force the convergence to (potentially unstable) roots, it cannot guarantee the existence of them and thus adds a huge computational burden to the model (and is about $2.5 - 3\times$ slower than stacked Transformers), despite its $O(1)$ memory cost.

## 2.3 Reasoning with semidefinite programs

### 2.3.1 Approximation for NP-hard problems

The SDPs has been widely applied to approximate of NP-hard problems since Goemans and Williamson [67], which introduced the MAXCUT SDP and its corresponding rounding process to turn the continuous SDP solution back to discrete variables. Under the unique game conjecture [82], the MAXCUT SDP gives the optimal approximation ratio among all polynomial-time algorithms. Other examples includes the MAX-k-CUT SDP by Frieze and Jerrum [59] and the ARV algorithm for the sparsest-cut [9]. Also, SDPs provide a systematic way of approximating NP-hard problem by the sum-of-square hierarchy [23, 121] and its dual the Lasserre hierarchy [86, 87, 88]. Other than combinatorial optimization problems, the SDP can also be applied to approximate the (non-convex) quadratic optimization problem [100], general polynomial optimization problem [23], and estimation of the mean of high-dimensional heavy-tail distributions [74].

### 2.3.2 Inference in Markov Random Field

**Variational methods and Continuous relaxations** One popular class of approaches in approximate inference consists of framing a relevant optimization problem whose solution can be treated as a reasonable approximation to the true mode/partition function. This includes techniques employing the Gibbs variational principle [22] that solve an optimization problem over the set of all possible distributions on the random variables (generally intractable). Amongst these, the mean-field approximation [118], which makes the simplifying (and possibly inaccurate) assumption of a product distribution amongst the variables, is extremely popular. Belief propagation [166] is another popular algorithm used for inference that has connections to variational inference, but has strong theoretical guarantees only when the underlying MRFs are loop-free. In addition, several LP-based and SDP-based continuous relaxations [80, 133] to the discrete optimization problem have been proposed. In particular, Frieze et al. [59] model the problem of estimating the mode in a multi-class MRF as an instance of the MAX $k$-CUT problem and propose an SDP relaxation as well as rounding mechanism for the same. Generally, such SDP-based approaches are theoretically attractive to analyze [39, 60, 109, 123], but practically infeasible for large MRFs with many constraints due to their high computational cost.

**MCMC and sampling-based methods** Another class of approaches involves running MCMC chains whose stationary distribution is the one specified by (4.1). These methods run a particular number of MCMC steps and then do some averaging over the samples at the end of the chain. Popular methods include Gibbs sampling [64] and Metropolis-Hastings [73]. A significant development in these methods was the introduction of annealing over a range of temperatures by Neal [108], which computes an unbiased estimate of the true partition function. Thereafter, several other methods in this line that employ some form of annealing and importance sampling have emerged [37, 42, 98]. However, it is typically difficult to determine the number of steps that the MCMC chain requires to converge to the stationary distribution (denoted mixing time). Further, as mentioned above, both variational methods (due to their propensity to converge to suboptimal

solutions) and MCMC methods (due to large mixing times) are known to underperform in the low-temperature setting.

**Other methods** Some other popular techniques for inference include variable elimination methods like bucket elimination [49, 50] that typically use a form of dynamic programming (DP) to approximately marginalize the variables in the model one-by-one. A significant recent development in this line, which is also based on DP, is the spectral approach by Park et al. [120]. By viewing all possible configurations of the random variables in the function space, Park et al. [120] build a bottom-up approximate DP tree which yields a fully polynomial-time approximation scheme for estimating $Z$, and markedly outperforms other standard techniques. However, as mentioned above, it is a priori unclear how their method could be extended to the multi-class Potts model, since their bottom-up dynamic programming chain depends on the variables being binary-valued. Other approximate algorithms for estimating the partition function with theoretical guarantees include those that employ discrete integration by hashing [56] as well as quadrature-based methods [124]. While our method does not purely belong to either of the two categories mentioned above (since it involves *both* an SDP relaxation and importance sampling), it successfully generalizes to multi-class MRFs.

### 2.3.3 Learning to reason with a differentiable satisfiability layer

**Semidefinite solvers for MAXSAT problems.** It is known that SDP relaxations produce strong approximation guarantees for MAXCUT and MAX-2SAT [67], and are empirically tighter than standard linear programming relaxations [69]. However, they have not generally been viewed as practical strategies for solving MAXSAT problems, owing to the high computational cost of solving an SDP. More recent work, e.g. Wang et al. [151] has developed low-rank SDP solvers for general MAXSAT problems, and Wang and Kolter [146] showed that the SDP-based solver is able to outperform state-of-the-art discrete solvers in MAX-2SAT problems using branch-and-bound and pruning strategies based on SDPs.

**Logical reasoning in deep networks.** Recently, the deep learning community has given increasing attention to the concept of embedding complex, "non-traditional" layers within deep networks in order to train systems end-to-end [62]. Most previous systems have focused on creating differentiable modules from an existing set of *known relationships*, so that a deep network can learn the parameters of these relationships [46, 75, 101, 128, 134, 161, 162]. For example, Palm et al. [116] introduce a network that carries out relational reasoning using hand-coded information about which variables are allowed to interact, and test this network on $9 \times 9$ Sudoku. Similarly, Evans and Grefenstette [57] integrate inductive logic programming into neural networks by constructing differentiable SAT-based representations for specific "rule templates." While these networks are seeded with prior information about the relationships between variables, our approach learns these relationships *and* their associated parameters end-to-end. While other recent work has also tried to jointly learn rules and parameters, the problem classes captured by these architectures have been limited. For instance, Cingillioglu and Russo [45] train a neural network

to apply a specific class of logic programs, namely the binary classification problem of whether a given set of propositions entails a specific conclusion.

### 2.3.4 Community detection with modularity maximization

Modularity was proposed in [111] to measure the quality of densely connected clusters. For an undirected graph with a community assignment, its modularity is given by

$$Q(c) := \frac{1}{2m} \sum_{ij} \left[ a_{ij} - \frac{d_i d_j}{2m} \right] \delta(c_i = c_j), \tag{2.2}$$

where $a_{ij}$ is the edge weight connecting nodes $i$ and $j$, $d_i = \sum_j a_{ij}$ is the degree for node $i$, $m = \sum_{ij} a_{ij}/2$ is the sum of edge weights, and $c_i \in [r]$ is the community assignment for node $i$ among the $r$ possible communities. The notation $\delta(c_i = c_j)$ is the Kronecker delta, which is one when $c_i = c_j$ and zero otherwise. The higher the modularity, the better the community assignment. However, as shown in [36], optimizing modularity is NP-hard, so researchers instead focus on different heuristics, including spectral methods [110], simulated annealing [127], linear programming and semidefinite programming [2, 76]. The most popular heuristic, the Louvain method [24], initializes each node with a unique community and updates the modularity $Q(c)$ cyclically by moving $c_i$ to the best neighboring communities [81, 110]. When no local improvement can be made, it aggregates nodes with the same community and repeats itself until no new communities are created. Experiments show that it is fast and performant [165] and can be further accelerated by choosing update orders wisely [12, 115]. However, the local steps can easily get stuck at local optima, and it may even create disconnected communities [140] containing disconnected components. In follow-up work, the Leiden method [140] fixes the issue of disconnected communities by adding a refinement step that splits disconnected communities before aggregation. However, it still depends on greedy local steps and still suffers from local optima. Beyond modularity maximization, there are many other metrics to optimize for community detection, including asymptotic surprise [139], motif-aware [93] or higher-order objectives [168].

**Semidefinite programming and clustering.** Semidefinite programming (SDP) has been a powerful tool in approximating NP-complete problems [88, 122]. Specifically, the max-$k$-cut SDP [59, 67] deeply relates to community detection, where max-$k$-cut maximizes the sum of edge weights *between* partitions, while community detection maximizes the sum *inside* partitions. The max-$k$-cut SDP is given by the optimization problem

$$\underset{X}{\text{maximize}} \quad -\sum_{ij} a_{ij} x_{ij}, \text{ s.t. } X \succeq 0, \; X \geq -1/(k-1), \; \text{diag}(X) = 1. \tag{2.3}$$

When limiting the rank of $X$ to be $k-1$, values of $x_{ij}$ become discrete and are either $1$ or $-1/(k-1)$, which works similarly to a Kronecker delta $\delta(c_i = c_j)$ [59]. If $k$ goes to infinity, the constraint set reduces to $\{X \geq 0, X \succeq 0, X_{ii} = 1\}$, and Swamy [137] provides a $0.75$ approximation ratio to correlation clustering on the relaxation (the bound doesn't apply to modularity maximization).

However, these SDP relaxations are less practical because known semidefinite solvers don't scale with the numerous constraints, and the runtime of the rounding procedure converting the continuous variables to discrete assignments grows with $O(n^2k)$. By considering max-2-cut iteratively at every hierarchical level, Agarwal and Kempe [2] is able to perform well on small datasets by combining SDPs with the greedy local move, but the method is still unscalable due to the SDP solver. DasGupta and Desai [48] discussed the theoretical property of SDPs when there are only 2 clusters. Other works [76, 106, 144] also apply SDPs to solve clustering problems, but they don't optimize modularity.

**Relation to copositive programming.** The constraint $DNN = \{X \mid X \succeq 0,\ X \geq 0\}$ in our SDP relaxation is connected to an area called "copositive programming" [38, 53, 102], which focuses on the sets

$$CP = \{X \mid v^T X v \geq 0,\ \forall v \geq 0\} \ \text{ and } \ CP^* = \{V^T V \mid V \geq 0,\ V \in \mathbb{R}^{n\times r},\ \forall r \in \mathbb{N}\}. \quad (2.4)$$

Interestingly, both $CP$ and $CP^*$ are convex, but the set membership query is NP-hard. The copositive sets relate to semidefinite cone $S = \{X \mid v^T X v \geq 0,\ \forall v\}$ by the hierarchy

$$CP \supseteq S \supseteq DNN \supseteq CP^*. \quad (2.5)$$

For low dimensions $n \leq 4$, the set $DNN = CP^*$, but $DNN \supsetneq CP^*$ for $n \geq 5$ [70]. Optimization over the copositive set is hard in the worst case because it contains the class of binary quadratic programming [38]. Approximation through optimizing the $V$ space has been proposed in [26, 71], but it is still time-consuming because it requires a large copositive rank $r = O(n^2)$ [27].

## 2.4 Linear programming via efficient piecewise quadratic optimization

In linear programming (LP), we consider the following constrained optimization problem

$$\underset{x\geq 0}{\text{minimize}}\ c^T x, \qquad \text{s.t. } Ax = b, \quad (2.6)$$

and its dual problem

$$\underset{y}{\text{maximize}}\ b^T y, \qquad \text{s.t. } A^T y \leq c, \quad (2.7)$$

where $A \in \mathbb{R}^{m\times n}$ is the coefficients matrix, $b \in \mathbb{R}^m$ is the bias, $c \in \mathbb{R}^n$ is the cost vector, and $x \in \mathbb{R}^n_+$ is the variable. Nowadays, linear programs are usually solved by the interior point method and augmented Lagrangian methods. The former has been introduced in the previous section and is well-documented in standard textbooks such as [112, Chapter 14], so we will only discuss the latter here.

**Augmented Lagrangian methods for LP.** Poljak and Tretjakov [126] first proposed to solve the dual LP through an augmented Lagrangian method, which iteratively solves the following subproblem at the $t$-th iteration.

$$x^{t+1} := \arg\min_{x \geq 0} \; c^T x + \sigma^t \|b - Ax^t - \sigma_t^{-1} y^T\|^2, \quad y^{t+1} := y^t + \sigma_t (b - Ax^{t+1}), \tag{2.8}$$

where $\{\sigma_t\}$ is a nonnegative and nondecreasing scaling sequence. The core problem to the augmented Lagrangian method is always about how to solve a sequence of the inner subproblems efficiently to a desired precision, since the subproblem may not be easier than the original problem and now we need to deal with multiple them.

Yen et al. [167] applied a coordinate newton solver to solve the inner problem and proved global linear convergence to the subproblem optimum for the expected objective value. They also showed that for a specific $\sigma_t$ fixed over the outer iterations, the augmented Lagrangian approach attains global linear convergence to an optimal solution $y$ of the dual when the inner problem is solved exactly. Their approach can also be applied to solve the primal instead with little modifications. Later, Li et al. [95] considered solving the primal LP by augmented Lagrangian, but they formulated the bound constraints in both the original problem and the subproblem as an indicator function and wisely utilized the Moreau decomposition to obtain a differentiable, strongly convex, and unconstrained subproblem whose gradient is semismooth. They then applied a semismooth Newton method to solve the subproblem and proved that their semismooth Newton method achieves fast local superlinear convergence while the outer part possesses a local linear convergence rate to a primal-dual solution pair.

The major target of [167] is large-scale machine learning applications, the same as what we consider in our work. These problems tend to have very large $m$ and $n$ while the matrix $A$ is either highly sparse or has a specific structure such that $Ax$ and $A^\top y$ can be computed efficiently without explicitly forming $A$. Their coordinate newton solver is able to utilize such properties of $A$ to execute each iteration swiftly, and they suggested a way to totally abandon parameter tuning for augmented Lagrangian while still achieving good practical speed over commercial solvers in their experiment. On the other hand, [95] mainly considered the situation that $A$ or $AA^\top$ is not sparse so that commercial solvers utilizing sparse matrix factorizations become inefficient. Their semismooth Newton approach is able to utilize the sparsity in the solution as well as the sparsity in the generalized Hessian in the subproblem. Similar to that of [167], the convergence speed of the outer loop in [95] is also linear.

# Chapter 3

# The Mixing method for low-rank SDPs

This chapter considers the solution of large-scale, structured semidefinite programming problems (SDPs). A generic SDP can be written as the optimization problem

$$\underset{X \succeq 0}{\text{minimize}} \ \langle C, X \rangle, \ \text{ subject to } \ \langle A_i, X \rangle = b_i, \ \ i = 1 \dots p \tag{3.1}$$

where $X \in \mathbb{S}^n$ is the optimization variable (a symmetric $n \times n$ matrix), and $A_i \in \mathbb{R}^{n \times n}, b_i \in \mathbb{R}$, $i = 1, \dots, p$ are problem data. Semidefinite programs can encode a huge range of practical problems, including relaxations of many combinatorial optimization tasks [34], approximate probabilistic inference [78], metric learning [164], matrix completion [41], and many others. Unfortunately, generic semidefinite programs involve optimizing a *matrix-valued* variable $X$, where the number of variables grows quadratically so that it quickly becomes unaffordable for solvers employing exact methods such as primal-dual interior point algorithms.

Fortunately, a property of these problems, which has been recognized for some time now, is that the solution to such problems is often *low-rank*; specifically, the problem always admits an optimal solution with at most rank $\lceil \sqrt{2p} \rceil$ [16, 123], and many SDPs are set up to have even lower rank solutions in practice. This has motivated the development of non-convex low-rank solvers for these systems: that is, we can attempt to solve the equivalent (but now non-convex) formulation of the problem

$$\underset{V \in \mathbb{R}^{k \times n}}{\text{minimize}} \ \langle C, V^T V \rangle, \ \text{ subject to } \ \langle A_i, V^T V \rangle = b_i, \ \ i = 1 \dots p$$

with the optimization variable $V \in \mathbb{R}^{k \times n}$. Here we are explicitly representing $X$ by the matrix $V$ of rank $k$ (typically with $k \ll n$), $X = V^T V$. Note that because we are representing $X$ in this way, we no longer need to explicitly enforce semidefiniteness, as it is implied by the change of variables. In a long series of work dating back several years, it has been shown that, somewhat surprisingly, this change to a non-convex problem does not cause as many difficulties as might be thought: in practice, local solutions to the problem tend to recover the optimal solution [39]; assuming sufficient rank $k$, all second order local optima of the problem are also global optima [32]; and it even holds that approximated local optima also have good approximation properties [105] for convex relaxations of some combinatorial problems. Despite these advances, solving for $V$ remains a practical challenge. Traditional methods for handling non-linear equality constraints,

such as augmented Lagrangian methods and Riemannian manifold methods, suffer from slow convergence, difficulties in selecting step size, or other deficiencies.

In this chapter, we present a low-rank coordinate descent approach to solving SDPs that have the additional specific structure of constraints (only) on the *diagonal* entries of the matrix (we consider the case of unit diagonal constraints, though we can easily extend to arbitrary positive numbers)

$$\underset{X \succeq 0}{\text{minimize}} \ \langle C, X \rangle, \ \ \text{subject to} \ \ X_{ii} = 1, \ \ i = 1 \dots n. \tag{3.2}$$

This is clearly a very special case of the full semidefinite program, but it also captures some fundamental problems, such as the famous semidefinite relaxation of the maximum cut (MAXCUT) combinatorial optimization problem; indeed, the MAXCUT relaxation will be one of the primary applications in this chapter. In this setting, if we consider the above low-rank form of the problem, we show that we can derive the coordinate descent updates in a very simple closed form, resulting in an algorithm several times faster than the existing state of the art. We call our approach the Mixing method, since the updates have a natural interpretation in terms of giving each $v_i$ as a mixture of the remaining $v_j$ terms. We will also show, however, that the method can be applied to other problems as well, such as a (novel, to the best of our knowledge) relaxation of the MAXSAT problem.

On the theoretical side, we prove several strong properties about the Mixing method. Most notably, despite the fact that the method is solving a non-convex formulation of the original MAXCUT SDP, it nonetheless will converge to the true *global* optimum of the original SDP problem, provided the rank is sufficient, $k > \sqrt{2n}$ (which is of course much smaller than optimizing over the entire $n^2$ variables of the original SDP). We prove this by first showing that the method is strictly decreasing, and always converges to a first-order critical point. We then show that all non-optimal critical points (that is, all points which are critical point in the non-convex formulation in $V$, but not optimal in $X$ in the original SDP), are *unstable*, i.e., they are saddle points in $V$ and will be unstable under updates of the Mixing method; this means that, in practice, the Mixing method is extremely unlikely to converge to any non-optimal solution. However, to formally prove that the method *will* converge to the global optimum, we consider a slightly modified "step size" version of the algorithm, for which we can prove formally that the method indeed converges to the global optimum in all cases (although in practice such a step size is not needed). Finally, for both the traditional and step size versions of the algorithm, we show that the Mixing method attains *locally linear* convergence around the global optimum. The primary tools we use for these proofs require analyzing the spectrum of the Jacobian of the Mixing method at non-optimal critical points, showing that it is always unstable around those points; we combine these with a geometric analysis of critical points due to [32] and a convergence proof for coordinate gradient descent due to [91] to reach our main result (both results require slight specialization for our case, so are included for completeness, but the overall thrust of those supporting points are due to these past papers).

**Contributions of this chapter.** In summary, the main contributions of this chapter are: 1) We propose a low-rank coordinate descent method, the Mixing method, for the diagonally constrained SDP problem, which is extremely fast and simple to implement. 2) We prove that despite its non-convex formulation, the method is guaranteed to converge to global optimum of the

original SDP with local linear convergence, provided that we use a small rank $k > \sqrt{2n}$. 3) We evaluate the proposed method on the MAXCUT SDP relaxation, showing that it is 10-100x times faster than the other state-of-the-art solvers and scales to millions of variables. 4) We extend the MAX-2SAT relaxation of Goemans and Williamson [67] to general MAXSAT problems, showing that the proposed formulation can be solved by the Mixing method in linear time to the number of literals. Further, experiments show that the proposed method has much better approximation ratios than other approximation algorithms, and is even comparable to the best partial MAXSAT solvers in some instances.

## 3.1 The Mixing method

As mentioned above, the goal of the Mixing method is to solve the semidefinite program (3.2) with a unit diagonal constraint. As discussed, we can replace the $X \succeq 0$ constraint with $X = V^T V$ for some $V \in \mathbb{R}^{k \times n}$; when we do this, the constraint $X_{ii} = 1$ translates to the constraint $\|v_i\| = 1$, where $v_i$ is the $i$th column of $V$. This leads to the equivalent (non-convex) problem on the spherical manifold

$$\underset{V \in \mathbb{R}^{k \times n}}{\text{minimize}} \ \langle C, V^T V \rangle \quad \text{subject to } \|v_i\| = 1, \ i = 1 \dots n. \tag{3.3}$$

Although the problem is non-convex, it is known [16, 123] that when $k > \sqrt{2n}$, the optimal solution for $V \in \mathbb{R}^{k \times n}$ can recover the optimal solution for $X$.

We consider solving the problem (3.3) via a coordinate descent method. The resulting algorithm is extremely simple to implement but, as we will show, it performs substantially better than existing approaches for the semidefinite problems of interest. Specifically, the objective terms that depend on $v_i$ are given by $v_i^T(\sum_{j=1}^n c_{ij} v_j)$. However, because $\|v_i\| = 1$ we can assume that $c_{ii} = 0$ without affecting the solution of the optimization problem. Thus, the problem is equivalent to simply minimizing the inner product $v_i^T g_i$ (where $g_i = \sum_{j=1}^n c_{ij} v_j$), subject to the constraint that $\|v_i\| = 1$; this problem has a closed form solution, given by $v_i = -g_i / \|g_i\|$ when $\|g_i\| \neq 0$, and no update otherwise. Put in terms of the original $v_j$ variable, the update is simply

$$v_i := \text{normalize}\left(-\sum_{j=1}^{n} c_{ij} v_j\right) \text{ if the norm is non-zero.}$$

This way, we can initialize $v_i$ on the unit sphere and perform cyclic updates over all the $i = 1 \dots n$ in closed-form. We call this the Mixing method, because for each $v_i$ our update mixes and normalizes the remaining vectors $v_j$ according to weight $c_{ij}$. In the case of sparse $C$ (which is common for any large data problem), the time complexity for updating all variables once is $O(k \cdot m)$, where $k$ is the rank of $V$ and $m$ is the number of nonzeros in $C$. This is significantly cheaper than the interior point method, which typically admits complexities cubic in $n$. However, the details for efficient computation differ depending on the precise nature of the SDP, so we will describe these in more detail in the subsequent application sections. A complete description of the generic algorithm is shown in Algorithm 3.1.

**Algorithm 3.1** The Mixing method for MAXCUT problem

Initialize $v_i$ randomly on a unit sphere
**while** not yet converged **do**
  **for** $i = 1, \dots, n$ **do**
    **if** $\| \sum_{j=1}^{n} c_{ij} v_j \| \neq 0$ **then** $v_i := \text{normalize}(-\sum_{j=1}^{n} c_{ij} v_j)$
  **end for**
**end while**

### 3.1.1 Convergence properties of the Mixing methods

This section presents the theoretical analysis of the Mixing methods, which constitutes four main properties:

- We prove that the Mixing methods are strictly decreasing in objective value and always converge to first-order critical points in the limit.
- Next, we establish the linear convergence for the Mixing methods when the iterate is close enough to a critical point, *regardless of the rank*. Together with the above limit property, it means that the Mixing methods converge to a critical point in a asymptotic linear rate.
- Further, we prove that for a rank[1] $k > \sqrt{2n}$, all non-optimal critical points $V \in \mathbb{R}^{k \times n}$ are unstable for the Mixing method. That is, when $V^T V$ is non-optimal for the convex problem (3.2), the critical point $V$ will sit on a saddle of the non-convex problem (3.3) and the Mixing method tends to *diverge* from the point *locally*.
- Finally, to rigorously prove the *global convergence*, we show that a variant of the Mixing method with a proper step size converges to a global optimum almost surely under random initialization for almost every cost matrix $C$, *without any assumptions*.

In total, our method represents the *first low-rank semidefinite programming method which will provably converge to a global optimum under constraints, and further in a locally linear rate.*

**Convergence to critical points.** Our first property shows that the Mixing method strictly decreases, and always converges to a first-order critical point for any $k$. This is a relatively weak statement, but useful in the context of the further proofs.

**Theorem 3.1.** *The Mixing method on the SDP problem* (3.2) *is strictly decreasing and converges to first-order critical points, with step size (no assumption) and without step size (with Assumption 3.2).*

For the proof on the mixing method without step size, we introduce an additional assumption on the non-degeneracy for its normalization.

**Assumption 3.2.** *Assume that for all $i = 1 \dots n$, $\| \sum_{j=1}^{n} c_{ij} v_j \|$ do not degenerate in the procedure. That is, all norms are always greater than or equal to a constant $\delta > 0$.*

In practice, the degeneracy is never observed. The proof of Theorem 3.1 is in Appendix A.1, which mainly involves setting up the Mixing iteration in a matrix form, and showing that the difference in objective value between two iterations is given by a particular positive term based upon this form.

[1]The tightness of the $\sqrt{2n}$ rank (actually, rank satisfying $k(k+1)/2 \geq n$) is proved in [15].

**Locally linear convergence.** Next, we show that the convergence of the Mixing methods exhibits linear convergence to the global optimum whenever the solution is close enough, regardless of the rank and the existence of nearby non-optimal critical points, for both versions with or without the step size. This also echoes practical experience, where the Mixing method does exhibit this rate of convergence.

**Theorem 3.3.** *The Mixing methods converge linearly to the global optimum when close enough to the solution, with step size (no assumption) or without step size (under Assumption 3.2).*

The full proof is provided in Appendix A.2. We prove it from the Lipschitz smoothness (Lipschitz continuous gradient) of the Mixing mappings. The main difficulty here is that the corresponding linear system in the Gauss-Seidel method, $S^* \in \mathbb{R}^{n\times n}$, is semidefinite so that the corresponding Jacobian $J_{GS}$ contains eigenvectors of magnitude $1$ in the null space of $S^*$. We overcome the difficulty by proving the result directly in the function value space like [149] so that the eigenvectors in null$(S^*)$ can be ignored. This is the first local linear convergence on the spherical manifold without assumption.

**Instability of non-optimal critical points.** Next we prove the main result of our approach, that not only is the function decreasing, but that every non-optimal critical point is unstable; that is, although the problem (3.3) is non-convex, the algorithm tends to diverge (locally) from any solution that is not globally optimal. Further, the local divergence from non-optimal critical points and the global convergence to critical points hint that the Mixing method can only converges to global optimal solutions.

**Theorem 3.4.** *Pick $k > \sqrt{2n}$. For almost all $C$, all non-optimal first-order critical points are unstable fixed points for the Mixing method.*

The full proof is provided in Section 3.1.2. The main idea is to show that the maximum eigenvalue of the dynamics Jacobian, evaluated at a critical point but when $V$ is not optimal, is guaranteed to have a spectral radius (magnitude of the largest eigenvalue) greater than one. We do this by showing that the Jacobian of the Mixing method has the same structure as the Jacobian of a standard Gauss-Seidel update plus an additional projection matrix. By a property from [32], plus an analysis of the eigenvector of Kronecker products, we can then guarantee that the eigenvalues of the Mixing method Jacobian contains those of the Gauss-Seidel update. We then use an argument based upon [91] to show that the Gauss-Seidel Jacobian is similarly unstable around non-optimal critical points, proving our main theorem.

**Globally optimal convergence of Mixing method with a step size.** Though the above two theorems in practice ensure convergence of the Mixing method to a global optimum, because the method makes discrete updates, there is the theoretical possibility that the method will "jump" directly to a non-optimal critical point (yet again, this has never been observed in practice). For completeness, however, in the next theorem we highlight the fact that a version of the Mixing method that is modified with a step size *will* always converge to a global optimum.

**Theorem 3.5.** *Consider the Mixing method with a step size $\theta > 0$. That is,*

$$v_i := \text{normalize}\left(v_i - \theta \sum_{j=1}^{n} c_{ij} v_j\right), \text{ for } i = 1, \ldots, n.$$

*Take $k > \sqrt{2n}$ and $\theta \in (0, \frac{1}{\max_i \|c_i\|_1})$, where $\|\cdot\|_1$ denotes the 1-norm. Then for almost every $C$, the method converges to a global optimum almost surely under random initialization.*[2]

The full proof is provided in Appendix A.4. The main difference in the proof from the step size free version is that, with a step size, we can prove the diffeomorphism of the Mixing method and thus are able to use the stable manifold theorem. Because the Jacobian of *any* feasible method on the spherical manifold is singular[3], we need to construct the inverse function explicitly and show the smoothness of such function. We can then use an analysis of the Gauss-Seidel method with a step size to show our result. To the best of our knowledge, this represents the first globally optimal convergence result for the low-rank method applied to (constrained) semidefinite programming, without additional assumptions such as the Cauchy decrease [31, Assumption A3], or solving a sequence of "bounded" log-barrier subproblems exactly [40, Theorem 5.3].

**Remark 3.6.** *Assume there are $m$ nonzeros in $C \in \mathbb{R}^{n\times n}$. With Theorem 3.5 and 3.3, for almost every $C$, the Mixing method with a step size admits an asymptotic complexity of $O(m\sqrt{n}\log\frac{1}{\epsilon})$ to reach the global optimality gap of $f - f^* \leq \epsilon$ almost surely under random initialization. This concludes the theoretical analysis of the Mixing method.*

Now we will prove one of our main result: the instability of non-optimal critical points.

### 3.1.2 Proof of Theorem 3.4: The instability of non-optimal criticals points

Before starting the proofs, we discuss our notations and reformulate the Mixing methods.

**Notations.** We use upper-case letters for matrix, and lower-case letters for vector and scalar. For a matrix $V \in \mathbb{R}^{k\times n}$, $v_i \in \mathbb{R}^k$ refers to the $i$-th column of $V$, $\mathrm{vec}(V)$ and $\mathrm{vect}(V) \in \mathbb{R}^{nk}$ denote the vector stacking columns and rows of $V$, respectively. For a vector $y \in \mathbb{R}^n$, $D_y = \mathrm{diag}(y) \in \mathbb{R}^{n\times n}$ denotes the matrix with $y$ on the diagonal, $y_{\max}, y_{\min}, y_{\text{min-nz}} \in \mathbb{R}$ the maximum, the minimum, and the minimum nonzero element of $y$, respectively. The symbol $\otimes$ denotes the Kronecker product, $\dagger$ the Moore-Penrose inverse, $\sigma(\cdot)$ the vector of eigenvalues, $\rho(\cdot)$ the spectral radius, $1_n$ the 1-vector of length $n$, $I_n$ the identity matrix of size $n$, $\mathrm{tr}(\cdot)$ the trace, $\langle A, B\rangle = \mathrm{tr}(A^TB)$ the dot product, and $\|V\| = \sqrt{\mathrm{tr}(VV^T)}$ the generalized L2 norm. Indices $i, j$ are reserved for matrix element, and index $r$ for iterations.

**The Mixing methods.** We denote $C \in \mathbb{R}^{n\times n}$ the cost matrix of the problem

$$\underset{V\in\mathbb{R}^{k\times n}}{\text{minimize}}\ f(V) \equiv \langle C, V^TV\rangle \quad \text{subject to } \|v_i\| = 1,\ i = 1\dots n,$$

and w.l.o.g. assume that $c_{ii} = 0$ in all proofs. Matrices $V$ and $\hat{V}$ refer to the current and the next iterate, and $V^*$ the global optimum attaining an optimal value $f^*$ in the semidefinite program

[2]Any distribution will suffice if it maps zero volume in the spherical manifold to zero probability. For example, both spherical Gaussian distribution and normalized uniform distribution work.

[3]Note that on the spherical manifold, the Jacobian of any feasible method is singular because the Jacobian of $v_i/\|v_i\|$ is singular. Thus, the proof in Lee et al. [91, Section 5.5] does not carry over to the case of the Mixing method, even with a step size. This past proof required a non-singular Jacobian, and thus different techniques are required in our setting.

(3.2).[4] Let

$$g_i = \sum_{j<i} c_{ij}\hat{v}_j + \sum_{j>i} c_{ij} v_j \tag{3.4}$$

and matrix $L$ be the strict lower triangular part of $C$. With these notations, the mapping of the Mixing method $M : \mathbb{R}^{k\times n} \to \mathbb{R}^{k\times n}$ and its variant $M_\theta$ with step size $\theta$ can be written as[5]

$$M(V)^T = -(L + D_y)^{-1} L^T V^T, \qquad \text{where } y_i = \|g_i\|, \qquad i = 1 \dots n, \quad \text{and} \tag{3.5}$$

$$M_\theta(V)^T = (\theta L + D_y)^{-1}(I - \theta L)^T V^T, \quad \text{where } y_i = \|v_i - \theta g_i\|, \; i = 1 \dots n. \tag{3.6}$$

Note that in the analysis we assume $y_i > 0$ without loss of generality, otherwise we can remove those $v_i$ from the analysis because they are not updated. Also, both $M$ and $M_\theta$ are valid functions of $V$ because $y$ can be calculated from $V$ by the original algorithmic definitions in Section 3.1. This formulation is similar to the classical analysis of the Gauss-Seidel method for linear equation in Golub and Van Loan [68], where the difference is that $y$ here is not a constant to $V$ and thus the evolution is not linear.

**Proof of technical lemmas.**

We start by analyzing the Jacobian of the Mixing method.

**Lemma 3.7.** *Using the notation in* (3.5)*, the Jacobian of the Mixing method is*

$$J_V = -(PL \otimes I_k + D_y \otimes I_k)^{-1} P L^T \otimes I_k.$$

*in which $P$ is the rejection matrix of $V$. That is,*

$$P = \operatorname{diag}(P_1, \; \dots, \; P_n) \in \mathbb{R}^{nk\times nk}, \quad \textit{where } P_i = I_k - \hat{v}_i \hat{v}_i^T \in \mathbb{R}^{k\times k}.$$

*Proof.* Denote $V$ and $\hat{V}$ the current and the next iterate. Taking total derivatives on the update of the Mixing method (A.1), we have

$$y_i d\hat{v}_i = -P_i dg_i = -P_i(\sum_{j<i} c_{ij} d\hat{v}_j + \sum_{j>i} c_{ij} dv_j), \quad i = 1 \dots n.$$

Moving $d\hat{v}_j$ to the left-hand side. By the implicit function theorem, we have the Jacobian

$$J_V = -\begin{pmatrix} y_1 I_k & 0 & \dots & 0 \\ c_{12}P_2 & y_2 I_k & \dots & 0 \\ \dots & \dots & \dots & 0 \\ c_{1n}P_n & c_{2n}P_n & \dots & y_n I_k \end{pmatrix}^{-1} \begin{pmatrix} 0 & c_{12}P_1 & \dots & c_{1n}P_1 \\ 0 & 0 & \dots & \dots \\ 0 & 0 & \dots & c_{(n-1)n}P_{n-1} \\ 0 & 0 & \dots & 0 \end{pmatrix}.$$

The implicit function theorem holds here because the triangular matrix is always inversible. Rewriting $J_V$ with Kronecker product leads to the result. □

[4] By [123], the optimality in (3.2) is always attainable by $V \in \mathbb{R}^{k\times n}$ when $k > \sqrt{2n}$.

[5] The reason to reformulate here is to avoid the "overwrite" of variables in the algorithmic definition. Moving the inverse term to the left-hand side, the reader can recover the original sequential algorithm.

Note that $V = \hat{V}$ on critical points, which is also a fixed point of the Mixing method. Now we demonstrate how to analyze the Jacobian. Remember the notation $\mathrm{vect}(Z) = \mathrm{vec}(Z^T)$. This way, we have the following convenient property by the Kronecker product.

**Lemma 3.8.** *For matrices $A, B, Q, R$, we have $A \otimes B\,\mathrm{vect}(QR^T) = \mathrm{vect}((AQ)(BR)^T)$.*

*Proof.* $A \otimes B\,\mathrm{vect}(QR^T) = A \otimes B\,\mathrm{vec}(RQ^T) = \mathrm{vec}(BRQ^TA^T) = \mathrm{vect}((AQ)(BR)^T)$. □

By the above property, part of the spectrum of the Jacobian can be analyzed.

**Lemma 3.9** (Overlapping Eigenvalues)**.** *Assume $V \in \mathbb{R}^{k\times n}$ has* rank$(V) < k$*. Let*

$$P = \mathrm{diag}(P_1,\ \ldots,\ P_n), \quad \textit{where } P_i = I_k - v_i v_i^T.$$

*For any matrices $A, B, D \in \mathbb{R}^{n\times n}$ where $(A + D)$ is inversible, any eigenvalue of $(A + D)^{-1}B$ is also an eigenvalue of*

$$J = [PA \otimes I_k + D \otimes I_k]^{-1} \otimes I_k PB \otimes I_k.$$

*Proof.* Because rank$(V) < k$, by linear dependency there is a nonzero $z \in \mathbb{R}^k$ such that

$$z^T v_i = 0 \ \text{ for } i = 1 \ldots n \quad \Longrightarrow \quad P_i z = z \ \text{ for } i = 1 \ldots n.$$

Observe the following matrix identity by separating the subspaces of $P$ and $I_{nk} - P$.

$$\begin{aligned}
&[PA \otimes I_k + D \otimes I_k]\ \ (A + D)^{-1} \otimes I_k P \\
&= [P(A + D) \otimes I_k + (I_{nk} - P)D \otimes I_k]\ \ (A + D)^{-1} \otimes I_k P \\
&= P + (I_{nk} - P)(D(A + D)^{-1}) \otimes I_k P.
\end{aligned}$$

Inversing the square brackets and moving the $(I_{nk} - P)$ term to the other side, we get

$$[PA\otimes I_k+D\otimes I_k]^{-1}P = (A+D)^{-1}\otimes I_kP-[PA\otimes I_k+D\otimes I_k]^{-1}(I_{nk}-P)(D(A+D)^{-1})\otimes I_kP. \tag{3.7}$$

Beacuse there are both $P$ and $(I_{nk} - P)$ in the last term, for any vector $q$ we have from Lemma 3.8

$$\begin{aligned}
&[PA \otimes I_k + D \otimes I_k]^{-1}(I_{nk} - P)(D(A + D)^{-1}) \otimes I_k P\,\mathrm{vect}(qz^T) \\
&= [PA \otimes I_k + D \otimes I_k]^{-1}\,\mathrm{vect}\left(D(A + D)^{-1}q\ ((I_k - P_i)P_i z)^T\right)_{i=1\ldots n} \\
&= 0.
\end{aligned} \tag{3.8}$$

Let $q \in \mathbb{C}^n$ be an eigenvector of $(A + D)^{-1}B$ with eigenvalue $\lambda \in \mathbb{C}$. Then

$$\begin{aligned}
J\,\mathrm{vect}(qz^T) &= [PA \otimes I_k + D \otimes I_k]^{-1} \otimes I_k P\,\mathrm{vect}((Bq)z^T) \\
&= ((A + D) \otimes I_k)^{-1} P\,\mathrm{vect}((Bq)z^T) \\
&= \mathrm{vect}((A + D)^{-1}Bq\ z^T) \\
&= \lambda\,\mathrm{vect}(qz^T),
\end{aligned}$$

where the first equality follows from Lemma 3.8, the second equality follows from $P_i z = z,\ \forall i$ and (3.7)–(3.8), and the last equality follows from $q$ being an eigenvector. Thus, every eigenvalue $\lambda$ of $(A + D)^{-1}B$ is also an eigenvalue of $J$. □

By the above lemma, the spectral radius of $J = -(PL \otimes I_k + D_y \otimes I_k)^{-1} PL^T \otimes I_k$ is lower bounded by $J_{GS} = -(L + D_y)^{-1} L^T$, which can be again lower bounded as follows.

**Lemma 3.10.** *For a positive vector $y \in \mathbb{R}^n$, consider a matrix under the notation in* (3.5)

$$J_{GS} = -(L + D_y)^{-1} L^T.$$

*Let $S = C + D_y$. When $S \not\succeq 0$, the spectral radius $\rho(J_{GS}) > 1$.*[6]

*Proof.* The proof is more technical and is given in Appendix A.3. □

Further, the assumption in Lemma 3.9 is fulfilled by the following property of critical points.

**Lemma 3.11.** *[32, Lemma 9] Let $\frac{k(k+1)}{2} > n$. Then, for almost all $C \in \mathbb{R}^{n \times n}$, all first-order critical points $V \in \mathbb{R}^{k \times n}$ have rank smaller than $k$.*

*Proof.* The proof is listed in Appendix A.5 for completeness. □

Next, we characterize the optimality of $V$ by proving all non-optimal $V$ admits an $S \not\succeq 0$.

**Lemma 3.12.** *For a critical solution $V$, denote $S = C + \operatorname{diag}(y)$, where $y_i = \|Vc_i\|,\ \forall i$. Then*

$$S \succeq 0 \ \text{ if and only if } \ V \text{ is optimal.}$$

*Further, if $V$ is optimal, all $y_i$ are positive except when $c_i = 0$.*[7]

*Proof.* Consider the dual problem of the SDP problem (3.2),

$$\underset{y \in \mathbb{R}^n}{\text{maximize}} \ -1_n^T y, \quad \text{subject to} \ \ C + \operatorname{diag}(y) \succeq 0.$$

If $S = C + \operatorname{diag}(y) \succeq 0$, variable $y$ becomes a feasible solution of the above dual problem. Further, since $V$ is a critical solution, we have

$$VS = 0 \implies V^T VS = 0 \implies \operatorname{tr}(V^T VC) = -\operatorname{tr}(V^T V \operatorname{diag}(y)) = -1_n^T y,$$

which means that $V^T V$ and $y$ are primal and dual optimal solutions that close the duality gap. Thus, for critical $V$, $S \succeq 0$ implies optimality, and non-optimality implies $S \not\succeq 0$.

On the other direction, when the solution $V$ is optimal, there will be a corresponding dual optimal solution $y$ satisfying

$$V^T V(C + \operatorname{diag}(y)) = 0 \implies v_i^T V(C + \operatorname{diag}(y)) = 0,\ \forall i \implies y_i = \|Vc_i\|,\ \forall i.$$

And $S = C + \operatorname{diag}(y) \succeq 0$ follows from the dual feasibility. By the characterization of SPSD matrix, all submatrix of $S \succeq 0$ are SPSD. Thus, $y_i \geq 0$. If equality $y_i = 0$ holds, by the same reason all $2 \times 2$ submatrix $\begin{pmatrix} 0 & c_{ij} \\ c_{ij} & y_{jj} \end{pmatrix} \succeq 0, \forall j$. This means $c_i = 0$. □

[6] If $S \succeq 0$, we can prove that the spectral radius $\rho(J_{GS}) \leq 1$, in which all eigenvectors with magnitude 1 reside in the null of $S$, as an immediate result from Wang and Lin [149, Corollary 3.4]. However, the result is not used here.

[7] Let $y_i^* = \|V^* c_i\|$ and $S^* = C + \operatorname{diag}(y^*) \succeq 0$. An immediate consequence of the lemma is that, for any feasible $U$, $f(U) - f^* = \operatorname{tr}(UCU^T) + 1_n^T y^* = \operatorname{tr}(US^*U^T)$. Further, suppose $U$ is also an optimum, then $f(U) - f^* = \operatorname{tr}(US^*U^T) = 0 \iff US^* = 0 \iff \|Uc_i\| = y_i^* = \|V^* c_i\|,\ \forall i$. That is, $y^*$ is unique.

**Proof of Theorem 3.4**

*Proof.* We first derive the Jacobian $J$ of the Mixing method in Lemma 3.7, which gives

$$J = -(PL \otimes I_k + D_y \otimes I_k)^{-1} PL^T \otimes I_k,$$

where $P = \operatorname{diag}(P_1, \ldots, P_n)$ and $P_i = I - v_i v_i^T$ because $\hat{v}_i = v_i$ on the critical point (also a fixed point). In Lemma 3.9, we prove that when $\operatorname{rank}(V) < k$, the eigenvalues of $J$ contain the eigenvalues of

$$J_{GS} = -(L + D_y)^{-1} L^T.$$

The assumption in Lemma 3.9 is fulfilled by Lemma 3.11, which guarantees that for almost every $C$, all the first-order critical point must have $\operatorname{rank}(V) < k$. Further, Lemma 3.10 and 3.12 show that $J_{GS}$ happens to be the Jacobian of the Gauss-Seidel method on a linear system, which has a spectral radius $\rho(J_{GS}) > 1$ on the non-optimal first-order critical point $V$. Thus, Lemma 3.9 implies $\rho(J) \geq \rho(J_{GS}) > 1$, which means that all non-optimal first-order critical points are unstable for the Mixing method. □

# 3.2 Applications

## 3.2.1 Maximum cut problem

The SDP MAXCUT relaxation is indeed the motivating example of the Mixing method, so we consider it first. In this section, we demonstrate how to apply our method to this problem, which originated from Goemans and Williamson [67].

**Problem description.** The maximum cut problem is an NP-hard binary optimization problem, which seeks a partition over a set of vertices $i = 1 \ldots n$, so that the sum of edge weights $c_{ij}$ across the partition is maximized. If we denote the two partitions as $\pm 1$, we can formulate the assignment $v_i$ of vertex $i$ as the following binary optimization problem

$$\underset{v_i \in \{\pm 1\},\, \forall i}{\text{maximize}} \quad \frac{1}{2} \sum_{ij} c_{ij} \left( \frac{1 - v_i v_j}{2} \right).$$

Goemans and Williamson [67] proposed that we can approximate the above solution by "lifting" the assignment $v_i$ from $\{\pm 1\}$ to a unit sphere in $\mathbb{R}^k$ for sufficiently large $k$ as

$$\underset{\|v_i\| = 1,\, \forall i}{\text{maximize}} \quad \frac{1}{2} \sum_{ij} c_{ij} \left( \frac{1 - v_i^T v_j}{2} \right).$$

To recover the binary assignment, we can do a randomized rounding by picking a random vector $r \in \mathbb{R}^k$ on the unit sphere, and letting the binary assignment of vertex $i$ be $\operatorname{sign}(r^T v_i)$. Their analysis shows that the approximation ratio for the NP-hard problem is $0.878$, which means that the expected objective from the randomized rounding scheme is at least $0.878$ times the optimal binary objective.

**Algorithm Design.** Because the problem can be solved by the unit diagonal SDP (3.2), we can apply the Mixing method directly, as presented in Algorithm 3.1. Further, for a sparse adjacency matrix $C$, the coefficient $\sum_j c_{ij} v_j$ can be constructed in time proportional to the nonzeros in column $i$ of $C$. Thus, the time complexity of running a round of updates for all $v_i$ is $O(k \cdot \#\text{edges})$, in which $k$ is at most $\sqrt{2n}$.

### 3.2.2 Maximum satisfiability problem

Using similar ideas as in the previous section, Goemans and Williamson [67] proposed that we can use SDP to approximate the maximum 2-satisfiability problem. In this section, we propose a formulation that generalizes this idea to the general maximum satisfiability problem, and apply the Mixing method to this problem. The proposed relaxation here is novel, to the best of our knowledge, and (as we will show) achieves substantially better approximation results than existing relaxations.

**Problem description.** The MAXSAT problem is an extension of the well-known satisfiability problem, where the goal is to find an assignment that *maximizes* the number of satisfied clauses. Let $v_i \in \{\pm 1\}$ be a binary variable and $s_{ij} \in \{-1, 0, 1\}$ be the sign of variable $i$ in clause $j$. The goal of MAXSAT can then be written as the optimization problem

$$\underset{v \in \{-1,1\}^n}{\text{maximize}} \quad \sum_{j=1}^{m} \bigvee_{i=1}^{n} \mathbf{1}\{s_{ij} v_i > 0\}.$$

Note that most clauses will contain relatively few variables, so the $s_j$ vectors will be sparse. To avoid the need for an additional bias term, we introduce an auxiliary "truth" variable $v_0$, and define $z_j = \sum_{i=1}^{n} s_{ij} v_i - 1 = \sum_{i=0}^{n} s_{ij} v_i = V s_j$. Then the MAXSAT problem can be approximated as

$$\underset{v \in \{-1,1\}^n}{\text{maximize}} \quad \sum_{j=1}^{m} 1 - \frac{\|V s_j\|^2 - (|s_j| - 1)^2}{4|s_j|}.$$

Although we will not derive it formally, the reader can verify that for any configuration $v \in \{-1, 1\}^n$, this term represents an upper bound on the exact MAXSAT solution.[8] Similar to the MAXCUT SDP, we can relax the $v_i$s to be vectors in $\mathbb{R}^k$ with $\|v_i\| = 1$. This leads to the full MAXSAT semidefinite programming relaxation

$$\underset{X \succeq 0}{\text{minimize}} \;\; \langle C, X \rangle, \;\; \text{subject to} \;\; C = \sum_{j=1}^{m} w_j s_j s_j^T, \;\; X_{ii} = 1, \; i = 0 \dots n,$$

where $w_j = 1/(4|s_j|)$.

[8] Actually, the formula matches the approximation of Goemans and Williamson [67] for MAX-2SAT.

**Algorithm 3.2** The Mixing method for MAXSAT problem

Initialize all $v_i$ randomly on a unit sphere, $i = 1 \dots n$.
Let $z_j = \sum_{i=0}^{n} s_{ij} v_i$ for $j = 1, \dots, m$
**while** not yet converged **do**
    **for** $i = 1, \dots, n$ **do**
        **For each** $s_{ij} \neq 0$ **do** $z_j := z_j - s_{ij} v_i$
        $v_i :=$ normalize $\left( - \sum_{j=1}^{m} \frac{s_{ij}}{4|s_j|} z_j \right)$ if the norm is non-zero
        **For each** $s_{ij} \neq 0$ **do** $z_j := z_j + s_{ij} v_i$
    **end for**
**end while**

**Algorithm Design.** Because the $C$ matrix here is not sparse ($s_j s_j^T$ has $|s_j|^2$ non-sparse entries), we need a slightly more involved approach than for MAXCUT, but the algorithm is still extremely simple. Specifically, we maintain $z_j = V s_j$ for all clauses $j$. Because in each subproblem only one variable $v_i$ is changed, $z_j$ can be maintained in $O(k \cdot m_i)$ time, where $m_i$ denotes the number of clauses that contain variable $i$. In total, the iteration time complexity is $O(k \cdot m)$, where $m$ is the number of literals in the problem. Also, because applying arbitrary rotations $R \in \mathbb{R}^{k \times k}$ to $V$ does not change the objective value of our problem, we can avoid updating $v_0$. Algorithm 3.2 shows the complete algorithm. To recover the binary assignment, we apply the following classic rounding scheme: sample a random vector $r$ from a unit sphere, then assign binary variable $i$ as true if $\text{sign}(r^T v_i) = \text{sign}(r^T v_0)$ and false otherwise.

## 3.3 Experimental results

**Running time comparison for MAXCUT** Figure 3.1 shows the results of running the Mixing method on several instances of benchmark MAXCUT problems. These range in size from approximately 1000 nodes and 20000 edges to approximately 2 million nodes and 3 million edges. For this application, we are largely concerned with evaluating the runtime of our Mixing method versus other approaches for solving the same semidefinite program. Specifically, we compare to DSDP [19], a mature interior point solver; SDPLR [39], one of the first approaches to use low-rank structures; Pure-RBR [156, 157], a coordinate descent method in the $X$ space, which outputs the best rank-1 update at each step; and Manopt [30], a recent toolkit for optimization on Riemannian manifolds, with specialized solvers dedicated to the MAXCUT problem.[9] To be specific, we use DSDP 5.8, SDPLR 1.03-beta, and Manopt 3.0 with their default parameters. For Manopt, we compare to a subroutine "elliptopefactory", specially designed for diagonal-constrained SDP. For Pure-RBR, we implement the specialized algorithm [156, Algorithm 2] for MAXCUT SDP with a sparse graph in C++, which only requires a single pass of the sparse matrix per iteration. We omit the log barrier and initialize the RBR with full-rank $X$. All experiments are run on an Intel Xeon E5-2670 machine with 256 GB memory, and all solvers are run in the single-core mode to ensure fair comparisons. As the results show, in all cases the Mixing method is substantially faster than

[9] We didn't compare to commercial software like MOSEK, an interior-point solver like DSDP, because it is not open-source and Boumal [28] already showed that SDPLR is faster than MOSEK on diagonally constrained problems.

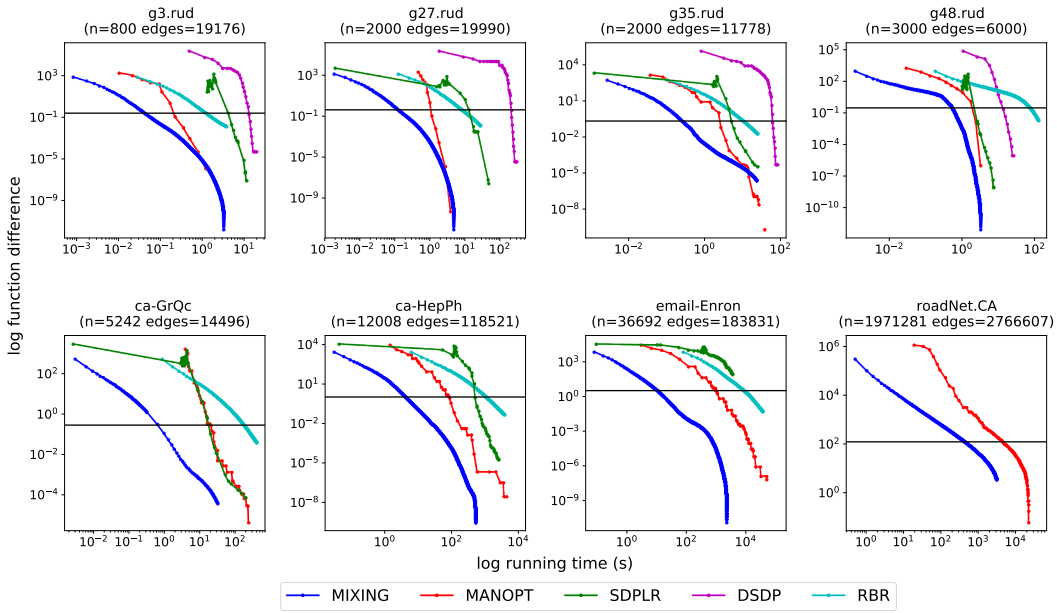



**Figure 3.1:** Objective value difference versus training time for the MAXCUT problems (log-log plot, lower is better). The horizontal lines mark the default stopping precision of the Mixing method, which is $10^{-4}$ times the starting relative objective of the Mixing method. Experiments show that our method (the blue line) is 10-100x faster than other solvers on our default stopping precision. Note that sometimes curves for SDPLR, DSDP, and RBR are not plotted because they either crashed or did not output any solution after an hour.

other approaches: for reaching modest accuracy (defined as $10^{-4}$ times the difference between the initial and optimal value), we are typically 10-100x faster than all competing approaches; only the Manopt algorithm ever surpasses our approach, and this happens only once both methods have achieved very high accuracy. Crucially, on the largest problems, we remain about 10x (or more) faster than Manopt over the entire run, which allows the Mixing method to scale to substantially larger problems.

**Effectiveness of the Mixing method on approximating MAXSAT problems.** Unlike the previous experiment (where the focus was solely on optimization performance), in this section we highlight the fact that with the Mixing method we are able to obtain MAXSAT results with a high approximation ratio on challenging domains (as the problems are similar, relative optimization performance is similar to that of the MAXCUT evaluations). Specifically, we evaluate examples from the 2016 MAXSAT competition [6] and compare our result to the best heuristic *complete* and *partial* solvers. Note that the complete solver produces a verified result, while the partial solver outputs a non-verified solution. Out of 525 problems solved in the complete track (every problem solved exactly by some solvers within 30 minutes during the competition), our method achieves an average approximation ratio of 0.978, and usually finds such solutions within seconds or less. Further, in some instances we obtain perfect solution faster than the best partial solvers. Figure 3.2 shows the progress of the approximate quality versus the running time. Beside the best heuristic solvers in MAXSAT 2016, we also show the approximation ratio over time for the

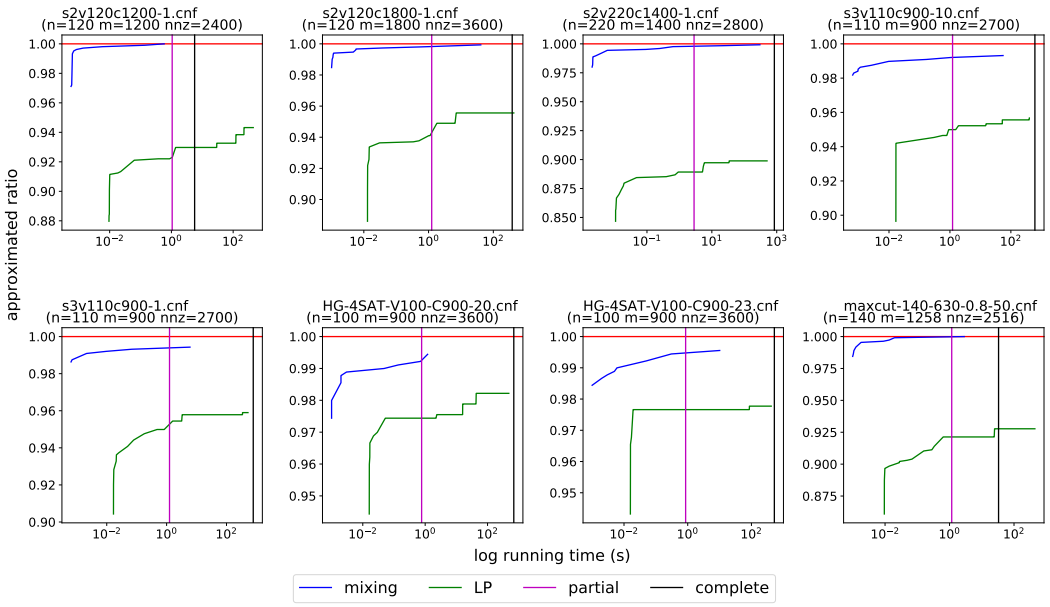



**Figure 3.2:** Approximated ratio versus (log) running time for the MAXSAT problems (higher is better). The horizontal line marks the perfect approximation ratio (1.00), and the curves mark the approximation ratio of different approximation algorithms over time. Experiments indicate that our proposed formulation/method (blue curves) achieves better approximation ratios in less time compared to LP. Further, it is sometimes faster than the best partial solvers (purple vertical lines) and complete solvers (black vertical lines) in the MAXSAT 2016 competition.

well-known linear programming approximation [66] (solved via the Gurobi solver). Note that each point in the blue and green curves denote the approximation ratio of the output solution at the time, and the starting points of the curves denote the time that the solver output the first solution. In all cases the Mixing method gives better and faster solutions than the LP approximations.

**The sufficient rank for optimization and randomized rounding** We proved that a low-rank of $\sqrt{2n}$ is sufficient for optimizing the SDP [16, 123]. But can we use an even lower rank? Will the low rank affect the quality of randomized rounding? In Figure 3.3, we ran several MAXSAT SDP instances with ranks ranging from 1 to double the theoretical rank upper-bound and record the relative error, which is $(f - f^*)/f^*$ for the objective value and $(\text{unsat} - \text{opt})/\#\text{clauses}$ for the approximation ratio of the randomized rounding. The experiment shows that a very low rank ($\leq 5$ in those instances) is enough to achieve nearly optimal performance in both the optimization and the randomized rounding process. Further, we see that increasing the rank beyond the theoretical upper bound doesn't improve the performance. That is, the $\sqrt{2n}$ rank is sufficient for both optimization and randomized rounding, and it is possible to use a lower rank without hurting the performance.

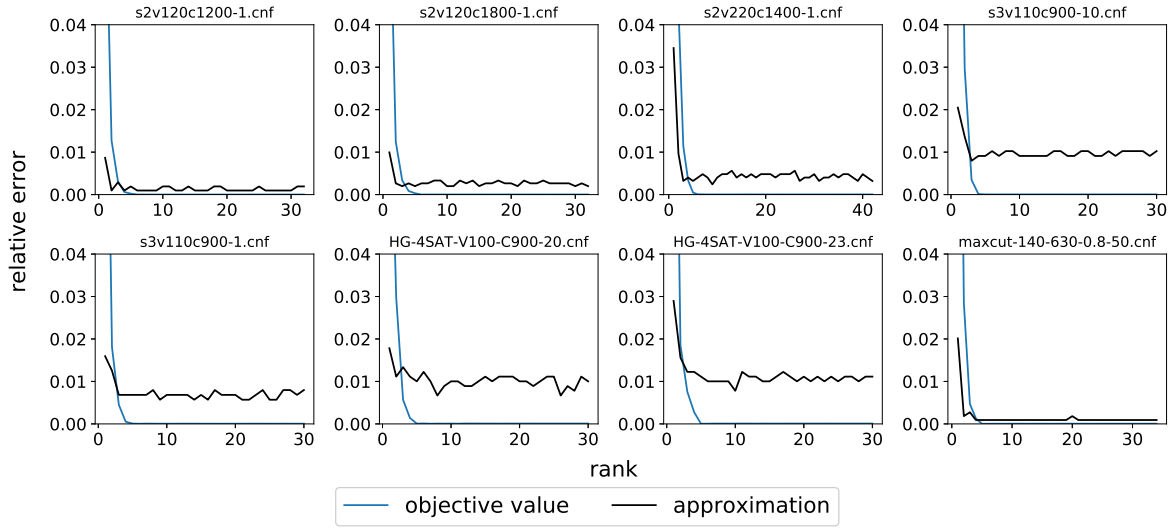



**Figure 3.3:** The relative error vs rank in the objective value and approximation (the lower, the better). The x-axis is the ranks ranging from 1 to $2x$ the theoretical rank upper-bound ($\sqrt{2n}$). Experiments suggests that a very low rank ($\leq 5$) is sufficeint to achieve nearly optimal approximation error in both the optimization and the randomized rounding process. Further, the error doesn't change much after the theoretical rank upper-bound.

## 3.4 Discussion

In this chapter we have presented the Mixing method, a low-rank coordinate descent approach for solving diagonally constrained semidefinite programming problems. The algorithm is extremely simple to implement, and involves no free parameters such as learning rates. In theoretical aspects, we have proved that the method converges to a first-order critical point and all non-optimal critical points are unstable under sufficient rank. With a proper step size, the method converges to the global optimum almost surely under random initialization. This is the first convergence result to the global optimum on the spherical manifold without assumption. Further, we have shown that the proposed methods admit local linear convergence in the neighborhood of the optimum regardless of the rank. In experiments, we have demonstrated the method on three different application domains: the MAXCUT SDP, a MAXSAT relaxation, and a word embedding problem (in the appendix). In all cases we show positive results, demonstrating that the method performs much faster than existing approaches from an optimization standpoint (for MAXCUT and word embeddings), and that the resulting solutions have high quality from an application perspective (for MAXSAT). In total, the method substantially raises the bar as to what applications can be feasibly addressed using semidefinite programming, and also advances the state of the art in structured low-rank optimization.

# Chapter 4

# Inference in Markov Random Field with fast low-rank SDPs

*This chapter is modified from our NeurIPS'2020 paper [153] in collaboration with Chirag Pabbaraju and J. Zico Kolter.*

Undirected graphical models or Markov Random Fields (MRFs) are used in various real-world applications like computer vision, computational biology, etc. because of their ability to concisely represent associations amongst variables of interest. A general pairwise MRF over binary random variables $x \in \{-1, 1\}^n$ may be characterized by the following joint distribution

$$p(x) \propto \exp\left(x^T A x + h^T x\right), \tag{4.1}$$

where $A \in \mathbb{R}^{n \times n}$ denotes the "coupling matrix" and encodes symmetric pairwise correlations, while $h \in \mathbb{R}^n$ consists of the biases for the variables. In this model, there are three fundamental problems of interest: (a) estimating the mode of the distribution, otherwise termed as *maximum a posteriori (MAP)* inference (b) estimating $p(x)$ for a configuration $x$ or generating samples from the distribution, and (c) learning the parameters $(A, h)$ given samples from the joint distribution. Since there are an exponential number of configurations in the support of the MRF, the problem of finding the true mode of the distribution is in general a hard problem. Similarly, to compute the probability $p(x)$ of any particular configuration, one has to compute the constant of proportionality in (4.1) which ensures that the distribution sums up to 1. This constant, denoted as $Z$, is called the *partition function*, where

$$Z = \sum_{x \in \{-1,1\}^n} \exp\left(x^T A x + h^T x\right).$$

Computing $Z$ exactly also involves summing up an exponential number of terms and is #P hard [63, 77] in general. The problem becomes harder still when we go beyond binary random variables and consider the case of a general multi-class MRF (also termed as a Potts model), where each variable can take on values from a finite set. Since problem (b) above requires computing $Z$ accurately, several approximations have been proposed in the literature. These methods have typically suffered in the quality of their approximations in the case of problem

instances where the entries of $A$ have large magnitude; this is referred to as the low-temperature setting [61, 120, 132, 138].

Recently, Park et al. [120] proposed a novel spectral algorithm that provably computes an approximate estimate of the partition function in time polynomial in the dimension $n$ and spectral properties of $A$. They show that their algorithm is fast, and significantly outperforms popular techniques used in approximate inference, particularly in the low-temperature setting. However, their experimental results suggest that there is still room for improvement in this setting. Furthermore, it is unclear how their method could be conveniently generalized to the richer domain of multi-class MRFs.

Another well-studied approach to compute the mode, i.e. the maximizer of the RHS in (4.1), is to relax the discrete optimization problem to a semidefinite program (SDP) [15, 154] and solve the SDP instead. Rounding techniques like randomized rounding [67] are then used to round the SDP solution to the original discrete space. In particular, Wang et al. [154] employ this approach in the case of a binary RBM and demonstrate impressive results. Frieze et al. [59] draw parallels between mode estimation in a general $k$-class Potts model and the MAX $k$-CUT problem, and suggest an SDP relaxation for the same. However, their relaxation has a *quadratic* number of constraints in the number of variables in the MRF. Therefore, using traditional convex program solvers employing the primal dual-interior point method [19] to solve the SDP would be computationally very expensive for large MRFs.

In this work, we propose solving a fundamentally different SDP relaxation for performing inference in a general $k$-class Potts model, that can be solved efficiently via a recently proposed low-rank SDP solver [151], and show that our method performs accurately and efficiently in practice, scaling successfully to large MRFs. Our SDP relaxation has only a *linear* number of constraints in the number of variables in the MRF. This allows us to exploit a low-rank solver based on coordinate descent, called the "Mixing method" [151], which converges extremely fast to a global solution of the proposed relaxation. We further propose a simple importance sampling-based method to estimate the partition function. Once we have solved the SDP, we state a rounding procedure to obtain samples in the discrete space. Since the rounding is applied to the optimal solution, the samples returned are closely clustered around the true mode in function value. Then, to ensure additional exploration in the space of the samples, we obtain a fraction of samples from the uniform distribution on the discrete hypercube. The combination results in an accurate estimate of the partition function.

Our experimental results show that our technique excels in both mode and partition function estimation, when compared to state-of-the-art methods like Spectral Approximate Inference [120], as well as specialized Markov Chain Monte Carlo (MCMC) techniques like Annealed Importance Sampling (AIS) [108], especially in the low temperature setting. Not only does our method outperform these methods in terms of accuracy, but it also runs significantly faster, particularly compared to AIS. We display these results on synthetic binary MRF settings drawn from Park et al. [120], as well as synthetic multi-class MRFs. Finally, we demonstrate that, owing to the efficiency of the fast SDP solver, our method is able to scale to large real-world MRFs used in image segmentation tasks.

## 4.1 Inference in multi-class MRFs

In this section, we formulate the SDP relaxation which we propose to solve for mode estimation in a $k$-class Potts model. First, we state the optimization problem for mode estimation in a binary MRF:

$$\max_{x \in \{-1,1\}^n} \; x^T A x + h^T x. \tag{4.2}$$

We observe that the above problem can be equivalently stated as below:

$$\max_{x \in \{-1,1\}^n} \; \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} \hat{\delta}(x_i, x_j) + \sum_{i=1}^{n} \sum_{l \in \{-1,1\}} \hat{h}_i^{(l)} \hat{\delta}(x_i, l); \quad \text{where} \quad \hat{\delta}(a,b) = \begin{cases} 1 & \text{if } a = b \\ -1 & \text{otherwise.} \end{cases} \tag{4.3}$$

The equivalence in (4.2) and (4.3) is readily achieved by setting $\hat{h}_i^{(l)}$ such that $h_i = \hat{h}_i^{(1)} - \hat{h}_i^{(-1)}$. However, viewing the optimization problem thus helps us naturally extend the problem to general $k$-class MRFs where the random variables $x_i$ can take values in a discrete domain $\{1, \ldots, k\}$ (denoted $[k]$). For the general case, we can frame a discrete optimization problem as follows:

$$\max_{x \in [k]^n} \; \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} \hat{\delta}(x_i, x_j) + \sum_{i=1}^{n} \sum_{l=1}^{k} \hat{h}_i^{(l)} \hat{\delta}(x_i, l). \tag{4.4}$$

where we are now provided with bias vectors $\hat{h}^{(l)}$ for each of the $k$ classes.

**Efficient SDP relaxation** We now derive an SDP-based relaxation to (4.4) that grows only *linearly* in $n$. To motivate this approach, we note that for the case of (4.4) (without the bias terms), Frieze et al. [59] first state an equivalent optimization problem defined over a simplex in $\mathbb{R}^{k-1}$, and go on to derive the following relaxation for which theoretical guarantees hold:

$$\begin{aligned} \max_{v_i \in \mathbb{R}^n,\, \|v_i\|_2 = 1 \; \forall i \in [n]} \; & \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} v_i^T v_j \\ \text{subject to} \quad & v_i^T v_j \geq -\frac{1}{k-1} \;\; \forall i \neq j. \end{aligned} \tag{4.5}$$

The above problem (4.5) can be equivalently posed as a convex problem over PSD matrices $Y$, albeit with $\sim n^2$ entry-wise constraints $Y_{ij} \geq -1/k-1$ corresponding to each pairwise constraint in $v_i, v_j$. Thus, for large $n$, solving (4.5) via traditional convex program solvers would be very expensive. Note further that, unlike the binary case (where the pairwise constraints hold trivially), it would also be challenging to solve this problem with low-rank methods, due to the quadratic number of constraints.

Towards this, we propose an alternate relaxation to (4.4) that reduces the number of constraints to be linear in $n$. Observe that the pairwise constraints in (4.5) are controlling the separation between $v_i, v_j$ and trying to keep them roughly aligned with the vertices of a simplex. With this

insight, we try to incorporate the functionality of these constraints within the criterion by plugging them in as part of the bias terms. Specifically, let us fix $r_1, \ldots, r_k \in \mathbb{R}^n$ on the vertices of a simplex, so that

$$r_l^T r_{l'} = \begin{cases} 1 & \text{if } l = l' \\ -\frac{1}{k-1} & \text{if } l \neq l'. \end{cases}$$

Then, we can observe that the following holds:

$$\hat{\delta}(x_i, x_j) = \frac{2}{k}\left((k-1) r_{x_i}^T r_{x_j} + 1\right) - 1, \tag{4.6}$$

so that solving the following discrete optimization problem is identical to solving (4.4):

$$\max_{v_i \in \{r_1, \ldots, r_k\}\ \forall i \in [n]} \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij} v_i^T v_j + \sum_{i=1}^{n}\sum_{l=1}^{k} \hat{h}_i^{(l)} v_i^T r_l. \tag{4.7}$$

The motivation here is that we are trying to mimic the $\hat{\delta}$ operator in (4.4) via inner products, but in such a way that the bias coefficients $\hat{h}_i^{(l)}$ determine the degree to which $v_i$ is aligned with a particular $r_l$. Thus, intuitively at least, we have incorporated the pairwise constraints in (4.5) within the criterion. As the next step, we simply relax the domain of optimization in (4.7) from the discrete set $\{r_i\}_{i=1}^k$ to unit vectors in $\mathbb{R}^n$ so as to derive the following relaxation:

$$\max_{v_i \in \mathbb{R}^n,\ \|v_i\|_2 = 1\ \forall i \in [n]} \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij} v_i^T v_j + \sum_{i=1}^{n} v_i^T \sum_{l=1}^{k} \hat{h}_i^{(l)} r_l. \tag{4.8}$$

Now, let $H \in \mathbb{R}^{n \times k}$ such that $H_{ij} = \hat{h}_i^{(j)}$. Define the block matrix $C \in \mathbb{R}^{(k+n)\times(k+n)}$ such that:

$$C = \begin{bmatrix} 0 & \frac{1}{2} \cdot H^T \\ \frac{1}{2} \cdot H & A \end{bmatrix}.$$

Then, the following convex program is equivalent to (4.8):

$$\begin{aligned} &\max_{Y \succeq 0}\ Y \cdot C \\ &\text{subject to}\ \ Y_{ii} = 1\ \forall i \in [k+n]; \quad Y_{ij} = -\frac{1}{k-1}\ \ \forall i \in [k],\ i < j \leq k. \end{aligned} \tag{4.9}$$

We defer the proof of the equivalence between (4.8) and (4.9) to Appendix B.1. Note that the number of constraints in (4.9) is now only $n + k + \frac{k(k-1)}{2} = n + \frac{k(k+1)}{2}$ i.e. *linear* in $n$ as opposed to the quadratic number of constraints in (4.5). We can then use the results by Barvinok [15] and Pataki [123], which state that there indeed exists a low-rank solution to (4.9) with rank $d$ at most $\left\lceil \sqrt{2\left(n + \frac{k(k+1)}{2}\right)} \right\rceil$. Thus, we can instead work in the space $\mathbb{R}^d$, leading to the following optimization problem:

$$\max_{v_i \in \mathbb{R}^d,\ \|v_i\|_2 = 1\ \forall i \in [n]} \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij} v_i^T v_j + \sum_{i=1}^{n} v_i^T \sum_{l=1}^{k} \hat{h}_i^{(l)} r_l. \tag{4.10}$$

This low-rank relaxation can then directly be solved in its existing non-convex form by using the method for solving norm-constrained SDPs by Wang et al. [151] called the "mixing method", which we refer to as $M^4$ ("Multi-class MRFs via Mixing Method"). $M^4$ derives closed-form coordinate-descent updates for the maximization in (4.10), and has been shown [146, 152] to reach accurate solutions in just a few iterations. Pseudocode for solving (4.10) via $M^4$ is given in the block for Algorithm 6.1.

Once we have a solution $v_1, \ldots, v_n$ to (4.10), we still require a technique to round back these vectors to configurations in the discrete space $[k]^n$. For this purpose, Frieze et al. [59] propose a natural extension to the technique of randomized rounding suggested by Goemans et al. [67], which involves rounding the $v_i$s to $k$ randomly drawn unit vectors. We further extend their approach as described in Algorithm 4.2 for the purposes of rounding our SDP relaxation (4.10). In the first step in Algorithm 4.2, we sample $k$ unit vectors $\{m_l\}_{l=1}^k$ uniformly on the unit sphere $\mathcal{S}^d$ and perform rounding as in Frieze et al. [59]. However, we need to reconcile this rounding with the truth vectors on the simplex. Thus, in the second step, we reassign each rounded value to a truth vector: if $v_i$ was mapped to $m_l$ in the first step, we now map it to $r_{l'}$ such that $m_l$ is closest to $r_{l'}$. In this way, we can obtain a bunch of rounded configurations, and output as the mode the one that has the maximum criterion value in (4.4).

**Alternate relaxation to** (4.7) $M^4$ provably solves the optimization problem in (4.10), but the rounded solution after applying Algorithm 4.2 lacks any approximation guarantees. This is because we simply discarded the pairwise constraints which existed in the original formulation (4.5) of Frieze et al. [59]. In order to remedy this, we further propose an alternate relaxation, so as to obtain an approximation ratio for the rounded solution.

To ensure that $v_i, v_j$ satisfy the pairwise constraints in 4.5, we adopt an alternate parameterization of the $v_i$s in terms of auxiliary variables $z_i \in \mathbb{R}^d$ for $d = m \cdot k, m \in \mathbb{Z}$ (we want to be able to segment each $z_i$ into $k$ blocks). Let $C = \frac{k}{k-1}\left(I_d - \frac{1}{k}\left(1_{k\times k} \otimes I_m\right)\right)$ where $1_{k\times k}$ is filled with 1s, and let $C = S^T S$ denote the Cholesky decomposition of $C$. Further, let $z_i^b \in \mathbb{R}^m$ denote the $b^{th}$ block in $z_i$. Then, we frame the following optimization problem:

$$\begin{aligned} \max_{z_i \in \mathbb{R}^d \ \forall i \in [n]} \ & \sum_{i=1}^n \sum_{j=1}^n A_{ij} v_i^T v_j + \sum_{i=1}^n v_i^T \sum_{l=1}^k \hat{h}_i^{(l)} r_l \\ \text{subject to} \quad & z_i \geq 0, \ \left\| \sum_{b=1}^k z_i^b \right\|_2^2 = 1, \ \ v_i = S z_i \ \ \forall i \in [n] \end{aligned} \tag{4.11}$$

We defer the detailed derivation for formulating (4.11) and solving it by Algorithm 4.3 (denoted as $M^4$+) to Appendix B.2. Here, we simply describe the motivation for such a parameterization of the $v_i$s. The approximation guarantees of Frieze et al. [59] hold for any set of $v_i$s lying in the feasible set of (4.5). Concretely, for $\|v_i\|_2 = 1$ and $v_i^T v_j \geq {}^{-1}/_{k-1}$, we have that $\mathbb{E}[f(\text{Rounding}(V))] \geq \alpha \cdot f(V)$, where we denote the objective by $f$ and $V = \{v_1, \ldots, v_n\}$. Thus, if we find a set of feasible $v_i$s such that $f(V) \geq f^\star_{discrete}$, where $f^\star_{discrete}$ refers to the solution to (4.7), we have the required guarantee on the expected value of the rounding. In the above, the parameterization of the $v_i$s via the $z_i$s and $S$, together with the positivity constraints on $z_i$, *ensure* that $v_i, v_j$ satisfy the pairwise constraints. In addition, step 10 in Algorithm 4.3

**Algorithm 4.1** Solving (4.10) via $M^4$

**Input:** $A, \{v_i\}_{i=1}^n, \{\hat{h}^{(l)}\}_{l=1}^k, \{r_l\}_{l=1}^k$

**procedure** $M^4$:
  Initialize $num\_iters$
  **for** $iter = 1, 2 \ldots, num\_iters$ **do**
    **for** $i = 1, 2 \ldots, n$ **do**
      $g_i \leftarrow 2\sum_{j \neq i} A_{ij} v_j + \sum_{l=1}^k \hat{h}_i^{(l)} r_l$
      $v_i \leftarrow \frac{g_i}{\|g_i\|_2}$
    **end for**
  **end for**
  **return** $v_1, \ldots, v_n$
**end procedure**

**Algorithm 4.2** Rounding in the multi-class case

**Input:** $\{v_i\}_{i=1}^n, \{r_l\}_{l=1}^k$

**procedure** ROUNDING:
  Sample $\{m_l\}_{l=1}^k \sim \texttt{Unif}(\mathcal{S}^d)$
  **for** $i = 1, 2 \ldots, n$ **do**
    $x_i \leftarrow \arg\max_{l \in [k]} \; v_i^T m_l$
  **end for**
  **for** $i = 1, 2 \ldots, n$ **do**
    $x_i \leftarrow \arg\max_{l \in [k]} \; m_{x_i}^T r_l$
  **end for**
  **return** $x$
**end procedure**

ensures that after each round of updates, $\|z_i\|_2 = 1$ and consequently, $\|v_i\|_2 = 1$ for all $i$. Thus, we have that the $v_i$s after each round of updates in $M^4$+ lie in the required feasible set. Further, we empirically observe that at convergence of Algorithm 4.3, $f(V) > f^\star_{discrete}$ always (in fact, the solution is within $5\%$ of the true solution to (4.11)). To summarize, provided that Algorithm 4.3 converges to a set of $v_i$s such that $f(V) > f^\star_{discrete}$, we have the approximation guarantees of Frieze et al. [59] for the rounded solution.

## 4.2 Partition function estimation

In this section, we deal with the other fundamental problem in inference: estimating the partition function. Following Section 4.1 above, the joint distribution in a $k$-class MRF can be expressed as:

$$p(x) \propto \exp \underbrace{\left( \sum_{i=1}^n \sum_{j=1}^n A_{ij} \hat{\delta}(x_i, x_j) + \sum_{i=1}^n \sum_{l=1}^k \hat{h}_i^{(l)} \hat{\delta}(x_i, l) \right)}_{f(x)}. \tag{4.12}$$

As stated previously, the central aspect in computing the probability of a configuration $x$ in this model is being able to calculate the partition function $Z$. The expression for the partition function in (4.12) is:

$$Z = \sum_{x \in [k]^n} \exp \left( \sum_{i=1}^n \sum_{j=1}^n A_{ij} \hat{\delta}(x_i, x_j) + \sum_{i=1}^n \sum_{l=1}^k \hat{h}_i^{(l)} \hat{\delta}(x_i, l) \right). \tag{4.13}$$

We begin with the intuition that the solution to (4.4) can be useful in computing the partition function. This intuition indeed holds if the terms in the summation are dominated by a few terms with large magnitude, which happens to be the case when $A$ has entries with large magnitude (low temperature setting). The hope then is that the rounding procedure described above more often than not rounds to configurations that have large $f$ values (ideally close to the mode). In that case,

**Algorithm 4.3** Solving (4.11) via $M^4$+

**Input:** $A, \{z_i\}_{i=1}^n, \{\hat{h}^{(l)}, r_l\}_{l=1}^k, C = S^T S$

**procedure** $M^4$+:
  Initialize $num\_iters$
  **for** $iter = 1, 2 \ldots, num\_iters$ **do**
    **for** $i = 1, 2 \ldots, n$ **do**
      $g \leftarrow 2 \sum_{j \neq i}^{n} A_{ij} C z_j + S^T \sum_{l=1}^{k} \hat{h}_i^{(l)} r_l$
      **for** $j = 1, 2 \ldots, m$ **do**
        Pick any $b(j) \in \arg\max_b g_j^b$
        $g_j^b \leftarrow \begin{cases} (g_j^b)_+ & \text{if } b = b(j) \\ 0 & \text{otherwise} \end{cases}$
      **end for**
      $z_i \leftarrow \frac{g}{\|g\|_2}$
    **end for**
  **end for**
  **return** $Sz_1, \ldots, Sz_n$
**end procedure**

**Algorithm 4.4** Estimation of $Z$

**Input:** $k$, $\{v_i\}_{i=1}^n$, $\{r_l\}_{l=1}^k$

**procedure** PARTITIONFUNCTION:
  Initialize $R \in \mathbb{Z}$, $X_{p_v} = \{\}$, $X_\Omega = [\,]$
  **for** $i = 1, 2 \ldots, R$ **do**
    Sample $x \sim p_v$ using Algorithm 4.2
    If $x$ not in $X_{p_v}$, add $x$ to $X_{p_v}$
  **end for**
  $q \leftarrow \frac{1}{k^n - |X_{p_v}|}$
  **for** $i = 1, 2 \ldots, R$ **do**
    Sample $x \sim \texttt{Unif}([k]^n \setminus X_{p_v})$
    Append $x$ to $X_\Omega$
  **end for**
  $\hat{Z} \leftarrow \sum_{x \in X_{p_v}} e^{f(x)} + \frac{1}{R} \sum_{x \in X_\Omega} \frac{e^{f(x)}}{q}$
  **return** $\hat{Z}$
**end procedure**

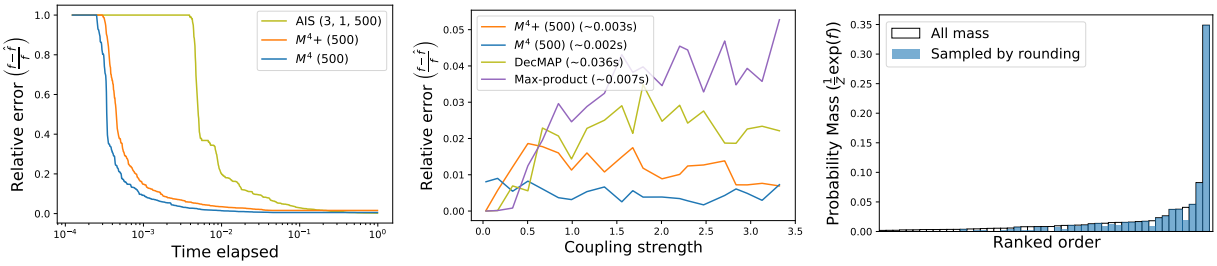
a few iterations of rounding would essentially yield all the configurations that make up the entire probability mass (a phenomenon which we empirically confirm ahead).

With this motivation, we describe a simple algorithm to estimate $Z$ that exploits the solution to our proposed relaxations. The rounding procedure described in Algorithm 4.2 induces a distribution on $x$ in the original space. Let us denote this distribution as $p_v$. For the case when $d = 2$, Wang et al. [154] propose a geometric technique for exactly calculating $p_v$, and derive an importance sampling estimate of $Z$ based on the empirical expectation $\hat{\mathbb{E}}\left[\exp(f(x))/p_v(x)\right]$. However, this approach does not scale to higher dimensions. Further, note that for small values of $d$, $p_v$ does not have a full support of $[k]^n$. Thus, an importance sampling estimate computed solely on $p_v$ would not be theoretically unbiased. Consequently, we propose using Algorithm 4.4 to estimate $Z$. First, we do a round of sampling from $p_v$, and store all the unique $x$'s seen in $X_{p_v}$. At this point, the hope is that $X_{p_v}$ stores all the $x$'s that constitute a bulk of the probability mass. Thereafter, to encourage exploration and ensure that our sampling process has a full support of $[k]^n$, we do a round of importance sampling from the uniform distribution on $[k]^n \setminus X_{p_v}$ and combine the result with the samples stored in $X_{p_v}$. For the estimate of $Z$ thus obtained, the following guarantee can be easily shown (refer to Appendix B.3 for proof):

**Theorem 4.1.** *The estimate $\hat{Z}$ given by Algorithm 4.4 is unbiased i.e. $\mathbb{E}[\hat{Z}] = Z$.*

## 4.3 Experimental results

In this section, we validate our formulations on a variety of MRF settings, both synthetic and real-world. Following its usage in Park et al. [120], we first state the notion of "coupling strength"




**(a)** Complete graph, $k = 5, n = 7$ **(b)** Complete graph $k = 5, n = 7$ **(c)** Mass sampled $k = 5, n = 7$

**Figure 4.1:** (a) Mode estimation - comparison with AIS (x-axis on log-scale) (b) Mode estimation comparison between $M^4$ and $M^4$+ (c) Randomized rounding samples most of the mass

of a matrix $A \in \mathbb{R}^{n \times n}$, which determines the temperature of the problem instance:

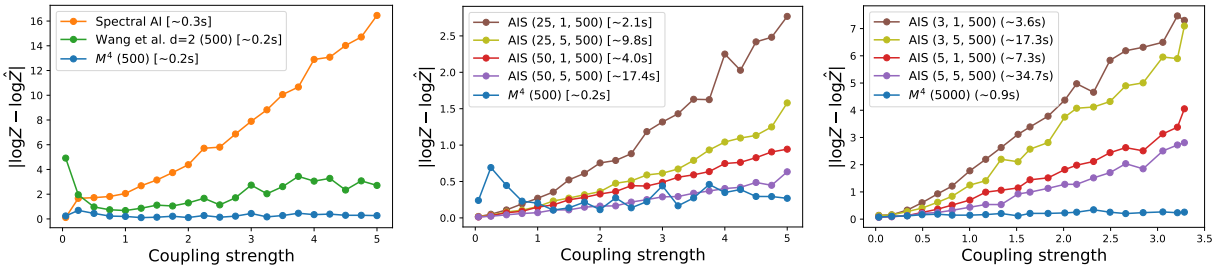
$$CS(A) = \frac{1}{n(n-1)} \sum_{i \neq j} |A_{ij}|. \tag{4.14}$$

As in the setup in Park et al. [120], the coupling matrices are generated as follows: for a coupling strength $c$, the entries on edges in $A$ are sampled uniformly from $[-c', c']$, where $c'$ is appropriately scaled so that $CS(A) \approx c$. The biases are sampled uniformly from $[-1, 1]$. We generate random complete graphs and Erdös-Renyi (ER) graphs. While generating ER graphs, we sample an edge in $A$ with probability $0.5$. We perform experiments on estimating the mode (Section 4.1) as well as $Z$ (Section 4.2). The algorithms we mainly compare to in our experiments are AIS [108], Spectral Approximate Inference (Spectral AI) [120] and the method suggested by Wang et al. [154]. We note that for the binary MRFs considered in the partition function task, Park et al. [120] demonstrate that they significantly outperform popular algorithms like belief propagation, mean-field approximation and mini-bucket variable elimination (Figures 3(a), 3(c) in Park et al. [120]). Hence, we simply compare to Spectral AI. For AIS, we have 3 main parameters: $(K, num\_cycles, num\_samples)$. We provide a description of these parameters along with complete pseudocode of our implementation of AIS in Appendix B.4. All the results in any synthetic setting are averaged over 100 random problem instances.[1]

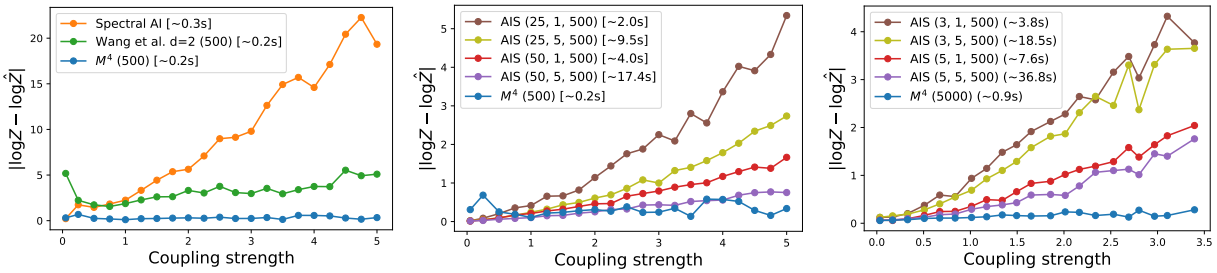
### 4.3.1 Mode estimation

We compare the quality of the mode estimate over progress of our methods (rounding applied to $M^4$ and $M^4$+) and AIS. On the x-axis, we plot the time elapsed for either method, and on the y-axis, we plot the relative error $\frac{f-\hat{f}}{f}$. Figure 4.1a shows the comparison for $k = 5$ and $CS(A) = 2.5$. In the legend, the number in parentheses following our methods is the number of rounding iterations, while those following AIS are $(K, num\_cycles, num\_samples)$ respectively. We observe from the plots that our methods are able to achieve a near optimal mode much quicker than AIS, underlining the efficacy of our method. Next, we compare the quality of the mode estimates given by both of our relaxations with the max-product belief propagation and decimation

[1]Source code for our experiments is available at `https://github.com/locuslab/sdp_mrf`.




**(a)** Complete graph $k = 2, n = 20$ **(b)** Complete graph $k = 2, n = 20$ **(c)** Complete graph $k = 3, n = 10$



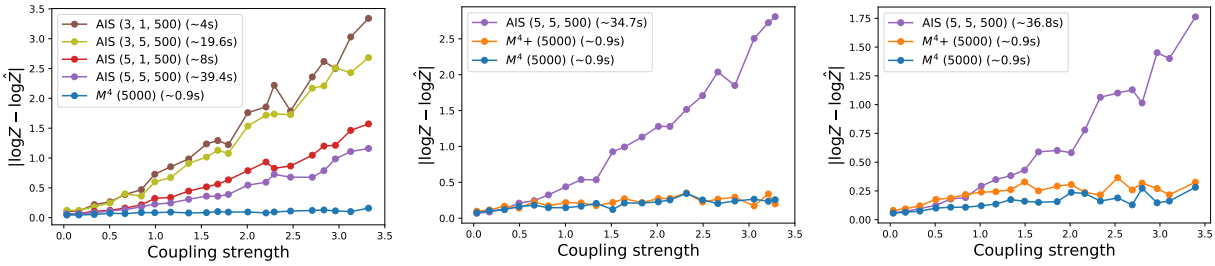

**(d)** ER graph $k = 2, n = 20$ **(e)** ER graph $k = 2, n = 20$ **(f)** Complete graph $k = 4, n = 8$

**Figure 4.2:** Estimation of $Z$

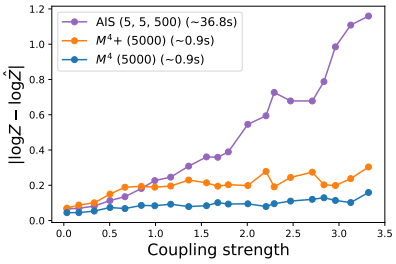
(DecMAP) algorithms provided in libDAI [107]. Across a range of coupling strengths, we plot the relative error of the mode estimates given by each method, for $k = 5$. We can observe (Figure 4.1b) that both our relaxations provide mode estimates that have very small relative error ($\sim 0.018$ at worst), whilst also being faster. Additional plots comparing our methods as well as timing experiments are provided in Appendix B.5.

### 4.3.2 Partition function estimation

We now evaluate the accuracy of the partition function estimate given by our Algorithm 4.4 applied on the $M^4$ solution. First, we empirically verify our intuition about randomized rounding returning configurations that account for most of the probability mass in (4.12). For a 5-class MRF with $CS(A) = 2.5$, we bucket the attainable $f$ values over different configurations of $x$, and compute the probability mass in each bucket. The buckets are then arranged in an increasing order of the probability mass, and the bars in Figure 4.1c show the mass in this order. Then, we obtain 1000 samples via randomized rounding, and fill in with blue the probability mass corresponding to the observed samples. We can observe that with just 1000 iterations of rounding, the obtained samples constitute most of the probability mass. Next, we consider coupling matrices over a range of coupling strengths and plot the error $|\log Z - \log \hat{Z}|$ against the coupling strength. We note here that for $k > 2$, there is no straightforward way in which we could extend the formulation in Spectral AI [120] to multiple classes; hence we only provide comparisons with AIS in this case. In the binary case (Figures 4.2a, 4.2d), we can observe that our estimates are more accurate than both Spectral AI [120] and Wang et al. [154] almost everywhere. Importantly, in the high-coupling strength setting, where the performance of Spectral AI [120] becomes pretty inaccurate, we are




**(a)** Complete graph $k = 5, n = 7$ **(b)** Complete graph $k = 3, n = 10$ **(c)** Complete graph $k = 4, n = 8$




**(d)** Complete graph $k = 5, n = 7$

**(e)** Original image, Annotated image, Segmented image

**(f)** Pairs of Original and Segmented images

**Figure 4.3:** (a), (b), (c), (d) Estimation of $Z$ (e) For the tree, we show the original image, annotations and segmented image (f) Segmentations computed on other images based on similar annotations

still able to maintain high accuracy. We also note that with just 500 rounding iterations, the running time of our algorithm is faster than Spectral AI [120]. We also provide comprehensive comparisons with AIS in Figures 4.2b, 4.2e,4.2c, 4.2f,4.3a over a range of parameter settings of $K$ and $num\_cycles$. We can see in the plots that on increasing the number of temperatures $(K)$, the AIS estimates become more accurate, but suffer a lot with respect to time. Finally, we also analyze the performance of Algorithm 4.4 applied to the $M^4$+ solution in Figures 4.3b, 4.3c,4.3d. We observe that the $M^4$+ estimates for $Z$ are slightly worse when compared to $M^4$ for larger $k$, but still much more accurate and efficient when compared to AIS.

### 4.3.3 Image segmentation

In this section, we demonstrate that our method of inference is able to scale up to large fully connected CRFs used in image segmentation tasks. Here, we consider the setting as in DenseCRF [85] where the task is to compute the configuration of labels $x \in [k]^n$ for the pixels in

an image that maximizes:

$$\max_{x \in [k]^n} \sum_{i<j} \mu(x_i, x_j) \bar{K}(f_i, f_j) + \sum_i \psi_u(x_i).$$

The first term provides pairwise potentials where $\bar{K}$ is modelled as a Gaussian kernel that measures similarity between pixel-feature vectors $f_i, f_j$ and $\mu$ is the label compatibility function. The second term corresponds to unary potentials for individual pixels. As in the SDP relaxation described above, we relax each pixel to a unit vector in $\mathbb{R}^d$. We model $\mu$ via an inner product, and base the unary potentials $\phi_u$ on rough annotations provided with the images to derive the following objective:

$$\max_{v_i \in \mathbb{R}^d,\ \|v_i\|_2 = 1\ \forall i \in [n]} \sum_{i<j} \bar{K}(f_i, f_j) v_i^T v_j + \theta \sum_{i=1}^{n} \sum_{l=1}^{k} \log p_{i,l} \cdot v_i^T r_l. \tag{4.15}$$

In the second term above, $\log p_{i,l}$ plugs in our prior belief based on annotations for the $i^{th}$ pixel being assigned the $l^{th}$ label. The coefficient $\theta$ helps control the relative weight on pairwise and unary potentials. We note here that running MCMC-based methods on MRFs with as many nodes as pixels in standard images is generally infeasible. However, we can solve (4.15) efficiently via the mixing method. At convergence, using the rounding scheme described in Algorithm 4.2, we are able to obtain accurate segmentations of images (Figures 4.3e, 4.3f), competitive with the quality presented in DenseCRF [85]. More details regarding the setting here are described in Appendix B.7.

## 4.4 Discussion

In this chapter, we presented a novel relaxation to estimate the mode in a general $k$-class Potts model that can be written as a low-rank SDP and solved efficiently by a recently proposed low-rank solver based on coordinate descent. We further introduced a relaxation that allows for approximation guarantees. We also proposed a simple and intuitive algorithm based on importance sampling which guarantees an unbiased estimate of the partition function. We set up experiments to empirically study the performance of our method as compared to relevant state-of-the-art methods in approximate inference, and verified that our relaxation provides an accurate estimate of the mode, while our algorithm for computing the partition function also gives fast and accurate estimates. We also demonstrated that our method is able to scale up to very large MRFs in an efficient manner.

The simplicity of our algorithm also lends itself to certain areas for improvement. Specifically, in the case of MRFs that have many well-separated modes, an accurate estimate of $Z$ should require sampling around each of the modes. Although we did empirically observe that randomized rounding samples most of the probability mass, the next steps involve studying other structured sampling mechanisms that indeed guarantee adequate sampling around each of the modes.

# Chapter 5

# Learning to reason with a differentiable satisfiability layer

*This chapter is modified from our ICML'2019 paper [152] in collaboration with Priya L. Donti, Bryan Wilder, and J. Zico Kolter.*

Although modern deep learning has produced groundbreaking improvements in a variety of domains, state-of-the-art methods still struggle to capture "hard" and "global" constraints arising from discrete logical relationships. Motivated by this deficiency, there has been a great deal of recent interest in integrating logical or symbolic reasoning into neural network architectures [45, 57, 116, 162]. However, with few exceptions, previous work primarily focuses on integrating *preexisting* relationships into a larger differentiable system via tunable continuous parameters, not on *discovering* the discrete relationships that produce a set of observations in a truly end-to-end fashion. As an illustrative example, consider the popular logic-based puzzle game Sudoku, in which a player must fill in a $9 \times 9$ partially-filled grid of numbers to satisfy specific constraints. If the rules of Sudoku (i.e. the relationships between problem variables) are not given, then it may be desirable to jointly learn the rules of the game *and* learn how to solve Sudoku puzzles in an end-to-end manner.

We consider the problem of learning logical structure specifically as expressed by satisfiability problems – concretely, problems that are well-modeled as instances of SAT or MAXSAT (the optimization analogue of SAT). This is a rich class of domains encompassing much of symbolic AI, which has traditionally been difficult to incorporate into neural network architectures since neural networks rely on continuous and differentiable parameterizations. Our key contribution is to develop and derive a differentiable smoothed MAXSAT solver that can be embedded within more complex deep architectures, and show that this solver enables effective end-to-end learning of logical relationships from examples (without hard-coding of these relationships). More specifically, we build upon recent work in fast block coordinate descent methods for solving SDPs [151] to build a differentiable solver for the smoothed SDP relaxation of MAXSAT. We provide an efficient mechanism to differentiate through the optimal solution of this SDP by using a similar block coordinate descent solver as used in the forward pass. Our module is amenable to GPU acceleration, greatly improving training scalability.

Using this framework, we are able to solve several problems that, despite their simplicity,

prove essentially impossible for traditional deep learning methods and existing logical learning methods to reliably learn without any prior knowledge. In particular, we show that we can learn the parity function, known to be challenging for deep classifiers [129], with only single bit supervision. We also show that we can learn to play $9 \times 9$ Sudoku, a problem that is challenging for modern neural network architectures [116]. We demonstrate that our module quickly recovers the constraints that describe a feasible Sudoku solution, learning to correctly solve 98.3% of puzzles at test time *without any hand-coded knowledge of the problem structure*. Finally, we show that we can embed this differentiable solver into larger architectures, solving a "visual Sudoku" problem where the input is an image of a Sudoku puzzle rather than a binary representation. We show that, in a fully end-to-end setting, our method is able to integrate classical convolutional networks (for digit recognition) with the differentiable MAXSAT solver (to learn the logical portion). Taken together, this presents a substantial advance toward a major goal of modern AI: integrating logical reasoning into deep learning architectures.


**Figure 5.1:** The forward pass of our MAXSAT layer. The layer takes as input the discrete or probabilistic assignments of known MAXSAT variables, and outputs guesses for the assignments of unknown variables via a MAXSAT SDP relaxation with weights $S$.

## 5.1 A differentiable satisfiability solver

The MAXSAT problem is the optimization analogue of the well-known satisfiability (SAT) problem, in which the goal is to *maximize* the number of clauses satisfied. We present a differentiable, smoothed approximate MAXSAT solver that can be integrated into modern deep network architectures. This solver uses a fast coordinate descent approach to solving an SDP relaxation of MAXSAT. We describe our MAXSAT SDP relaxation as well as the forward pass of our MAXSAT deep network layer (which employs this relaxation). We then show how to analytically differentiate through the MAXSAT SDP and efficiently solve the associated backward pass.

### 5.1.1 Solving an SDP formulation of satisfiability

Consider a MAXSAT instance with $n$ variables and $m$ clauses. Let $\tilde{v} \in \{-1, 1\}^n$ denote binary assignments of the problem variables, where $\tilde{v}_i$ is the truth value of variable $i \in \{1, \ldots, n\}$, and define $\tilde{s}_i \in \{-1, 0, 1\}^m$ for $i \in \{1, \ldots, n\}$, where $\tilde{s}_{ij}$ denotes the sign of $\tilde{v}_i$ in clause $j \in \{1, \ldots, m\}$. We then write the MAXSAT problem as

$$\underset{\tilde{v} \in \{-1,1\}^n}{\text{maximize}} \sum_{j=1}^{m} \bigvee_{i=1}^{n} \mathbf{1}\{\tilde{s}_{ij}\tilde{v}_i > 0\}. \tag{5.1}$$

As derived in Goemans and Williamson [67], Wang and Kolter [145], to form a semidefinite relaxation of (5.1), we first relax the discrete variables $\tilde{v}_i$ into associated continuous variables $v_i \in \mathbb{R}^k,\ \|v_i\| = 1$ with respect to some "truth direction" $v_\top \in \mathbb{R}^k,\ \|v_\top\| = 1$. Specifically, we relate the continuous $v_i$ to the discrete $\tilde{v}_i$ probabilistically via $P(\tilde{v}_i = 1) = {}^{\cos^{-1}(-v_i^T v_\top)}/_{\pi}$ based on randomized rounding (Goemans and Williamson [67]; see Section 5.1.2). We additionally define a coefficient vector $\tilde{s}_\top = \{-1\}^m$ associated with $v_\top$. Our SDP relaxation of MAXSAT is then

$$\begin{aligned} &\underset{V \in \mathbb{R}^{k\times(n+1)}}{\text{minimize}} \ \ \langle S^T S, V^T V\rangle, \\ &\text{subject to} \ \ \|v_i\| = 1, \ \ i = \top, 1, \ldots, n \end{aligned} \tag{5.2}$$

where $V \equiv \begin{bmatrix} v_\top & v_1 & \ldots & v_n \end{bmatrix} \in \mathbb{R}^{k\times(n+1)}$, and $S \equiv \begin{bmatrix} \tilde{s}_\top & \tilde{s}_1 & \ldots & \tilde{s}_n \end{bmatrix} \operatorname{diag}({}^1/\sqrt{4|\tilde{s}_j|}) \in \mathbb{R}^{m\times(n+1)}$. We note that this problem is a low-rank (but non-convex) formulation of MIN-UNSAT, which is equivalent to MAXSAT. This formulation can be rewritten as an SDP, and has been shown to recover the optimal SDP solution given $k > \sqrt{2n}$ [16, 123].

Despite its non-convexity, problem (6.8) can then be solved optimally via coordinate descent for all $i = \top, 1, \ldots, n$. In particular, the objective terms that depend on $v_i$ are given by $v_i^T \sum_{j=0}^n s_i^T s_j v_j$, where $s_i$ is the $i$th column vector of $S$. Minimizing this quantity over $v_i$ subject to the constraint that $\|v_i\| = 1$ yields the coordinate descent update

$$v_i = -g_i/\|g_i\|, \ \ \text{where } g_i = V S^T s_i - \|s_i\|^2 v_i. \tag{5.3}$$

These updates provably converge to the globally optimal fixed point of the SDP (6.8) [151]. A more detailed derivation of this update can be found in Appendix C.1.

### 5.1.2 SATNet: Satisfiability solving as a layer

Using our MAXSAT SDP relaxation and associated coordinate descent updates, we create a deep network layer for satisfiability solving (SATNet). Define $\mathcal{I} \subset \{1, \ldots, n\}$ to be the indices of MAXSAT variables with known assignments, and let $\mathcal{O} \equiv \{1, \ldots, n\} \setminus \mathcal{I}$ correspond to the indices of variables with unknown assignments. Our layer admits probabilistic or binary inputs $z_\iota \in [0, 1], \iota \in \mathcal{I}$, and then outputs the assignments of unknown variables $z_o \in [0, 1], o \in \mathcal{O}$ which are similarly probabilistic or (optionally, at test time) binary. We let $Z_\mathcal{I} \in [0, 1]^{|\mathcal{I}|}$ and $Z_\mathcal{O} \in [0, 1]^{|\mathcal{O}|}$ refer to all input and output assignments, respectively.

The outputs $Z_\mathcal{O}$ are generated from inputs $Z_\mathcal{I}$ via the SDP (6.8), and the weights of our layer correspond to the SDP's low-rank coefficient matrix $S$. This forward pass procedure is pictured in Figure 5.1. We describe the steps of layer initialization and the forward pass in Algorithm 5.1, and in more detail below.

#### Layer initialization

When initializing SATNet, the user must specify a maximum number of clauses $m$ that this layer can represent. It is often desirable to set $m$ to be low; in particular, *low-rank structure* can prevent overfitting and thus improve generalization.

Given this low-rank structure, a user may wish to somewhat increase the layer's representational ability via auxiliary variables. The high-level intuition here follows from the conjunctive

**Algorithm 5.1** SATNet Layer

**procedure** INIT()
    *// rank, num aux vars, initial weights, rand vectors*
    **init** $m, n_{\text{aux}}, S$
    **init** random unit vectors $v_\top$, $v_i^{\text{rand}}$ $\forall i \in \{1, \ldots, n\}$
    *// smallest $k$ for which* (6.8) *recovers SDP solution*
    **set** $k = \sqrt{2n} + 1$
**end procedure**

**procedure** FORWARD($Z_\mathcal{I}$)
    **compute** $V_\mathcal{I}$ **from** $Z_\mathcal{I}$ via (5.5)
    **compute** $V_\mathcal{O}$ **from** $V_\mathcal{I}$ via coord. descent (Alg 5.2)
    **compute** $Z_\mathcal{O}$ **from** $V_\mathcal{O}$ via (5.7)
    **return** $Z_\mathcal{O}$
**end procedure**

**procedure** BACKWARD($\partial \ell / \partial Z_\mathcal{O}$)
    **compute** $\partial \ell / \partial V_\mathcal{O}$ via (5.8)
    **compute** $U$ **from** $\partial \ell / \partial V_\mathcal{O}$ via coord. descent (Alg 5.3)
    **compute** $\partial \ell / \partial Z_\mathcal{I}$, $\partial \ell / \partial S$ **from** $U$ via (5.12), (5.11)
    **return** $\partial \ell / \partial Z_\mathcal{I}$
**end procedure**

normal form (CNF) representation of boolean satisfaction problems; adding additional variables to a problem can dramatically reduce the number of CNF clauses needed to describe that problem, as these variables play a role akin to register memory that is useful for inference.

Finally, we set $k = \sqrt{2n} + 1$, where here $n$ captures the number of actual problem variables in addition to auxiliary variables. This is the minimum value of $k$ required for our MAXSAT relaxation (6.8) to recover the optimal solution of its associated SDP [16, 123].

### Step 1: Relaxing layer inputs

Our layer first relaxes its inputs $Z_\mathcal{I}$ into continuous vectors for use in the SDP formulation (6.8). That is, we relax each layer input $z_\iota, \iota \in \mathcal{I}$ to an associated random unit vector $v_\iota \in \mathbb{R}^k$ so that

$$v_\iota^T v_\top = -\cos(\pi z_\iota). \tag{5.4}$$

(This equation is derived from the probabilistic relationship described in Section 5.1.1 between discrete variables and their continuous relaxations.) Constraint (5.4) can be satisfied by

$$v_\iota = -\cos(\pi z_\iota) v_\top + \sin(\pi z_\iota)(I_k - v_\top v_\top^T) v_\iota^{\text{rand}}, \tag{5.5}$$

where $v_\iota^{\text{rand}}$ is a random unit vector. For simplicity, we use the notation $V_\mathcal{I} \in \mathbb{R}^{k \times |\mathcal{I}|}$ (i.e. the $\mathcal{I}$-indexed column subset of $V$) to collectively refer to all relaxed layer inputs derived via Equation (5.5).

**Algorithm 5.2** Forward pass coordinate descent

**input** $V_{\mathcal{I}}$ *// inputs for known variables*
**init** $v_o$ with random vector $v_o^{\text{rand}}$, $\forall o \in \mathcal{O}$.
**compute** $\Omega = VS^T$
**while** not converged **do**
  **for** $o \in \mathcal{O}$ **do** *// for all output variables*
    **compute** $g_o = \Omega s_o - \|s_o\|^2 v_o$ as in (5.3)
    **compute** $v_o = -g_o/\|g_o\|$ as in (5.3)
    **update** $\Omega = \Omega + (v_o - v_o^{\text{prev}})s_o^T$
  **end for**
**end while**
**output** $V_{\mathcal{O}}$ *// final guess for output cols of* $V$

### Step 2: Generating continuous relaxations of outputs via SDP

Given the continuous input relaxations $V_{\mathcal{I}}$, our layer employs the coordinate descent updates (5.3) to compute values for continuous output relaxations $v_o$, $o \in \mathcal{O}$ (which we collectively refer to as $V_{\mathcal{O}} \in \mathbb{R}^{k \times |\mathcal{O}|}$). Notably, coordinate descent updates are *only computed for output variables*, i.e. are not computed for variables whose assignments are given as input to the layer.

Our coordinate descent algorithm for the forward pass is detailed in Algorithm 5.2. This algorithm maintains the term $\Omega = VS^T$ needed to compute $g_o$, and then modifies it via a rank-one update during each inner iteration. Accordingly, the per-iteration runtime is $O(nmk)$ (and in practice, only a small number of iterations is required for convergence).

### Step 3: Generating discrete or probabilistic outputs

Given the relaxed outputs $V_{\mathcal{O}}$ from coordinate descent, our layer converts these outputs to discrete or probabilistic variable assignments $Z_{\mathcal{O}}$ via either thresholding or randomized rounding (which we describe here).

The main idea of randomized rounding is that for every $v_o, o \in \mathcal{O}$, we can take a random hyperplane $r$ from the unit sphere and assign

$$\tilde{v}_o = \begin{cases} 1 & \text{if } \text{sign}(v_o^T r) = \text{sign}(v_\top^T r) \\ -1 & \text{otherwise} \end{cases}, \; o \in \mathcal{O}, \tag{5.6}$$

where $\tilde{v}_o$ is the boolean output for $v_o$. Intuitively, this scheme sets $\tilde{v}_o$ to "true" if and only if $v_o$ and the truth vector $v_\top$ are on the same side of the random hyperplane $r$. Given the correct weights $S$, this randomized rounding procedure assures an optimal expected approximation ratio for certain NP-hard problems [67].

During training, we do not explicitly perform randomized rounding. We instead note that the probability that $v_o$ and $v_\top$ are on the same side of any given $r$ is

$$P(\tilde{v}_o) = \cos^{-1}(-v_o^T v_\top)/\pi, \tag{5.7}$$

and thus set $z_o = P(\tilde{v}_o)$ to equal this probability.

During testing, we can either output probabilistic outputs in the same fashion, or output discrete assignments via thresholding or randomized rounding. If using randomized rounding, we round multiple times, and then set $z_o$ to be the boolean solution maximizing the MAXSAT objective in Equation (5.1). Prior work has observed that such repeated rounding improves approximation ratios in practice, especially for MAXSAT problems [145].

### 5.1.3 Computing the backward pass

We now derive backpropagation updates through our SATNet layer to enable its integration into a neural network. That is, given the gradients $\partial \ell / \partial z_{\mathcal{O}}$ of the network loss $\ell$ with respect to the layer outputs, we must compute the gradients $\partial \ell / \partial z_{\mathcal{I}}$ with respect to layer inputs and $\partial \ell / \partial S$ with respect to layer weights. As it would be inefficient in terms of time and memory to explicitly unroll the forward-pass computations and store intermediate Jacobians, we instead derive analytical expressions to *compute the desired gradients directly*, employing an efficient coordinate descent algorithm. The procedure for computing these gradients is summarized in Algorithm 5.1 and derived below.

#### From probabilistic outputs to their continuous relaxations

Given $\partial \ell / \partial z_{\mathcal{O}}$ (with respect to the layer outputs), we first derive an expression for $\partial \ell / \partial V_{\mathcal{O}}$ (with respect to the output relaxations) by pushing gradients through the probability assignment mechanism described in Section 5.1.2. That is, for each $o \in \mathcal{O}$,

$$\frac{\partial \ell}{\partial v_o} = \left(\frac{\partial \ell}{\partial z_o}\right)\left(\frac{\partial z_o}{\partial v_o}\right) = \left(\frac{\partial \ell}{\partial z_o}\right)\frac{1}{\pi \sin(\pi z_o)} v_\top, \tag{5.8}$$

where we obtain $\partial z_o / \partial v_o$ by differentiating through Equation (5.7) (or, more readily, by implicitly differentiating through its rearrangement $\cos(\pi z_o) = -v_\top^T v_o$).

#### Backpropagation through the SDP

Given the analytical form for $\partial \ell / \partial V_{\mathcal{O}}$ (with respect to the output relaxations), we next seek to derive $\partial \ell / \partial V_{\mathcal{I}}$ (with respect to the input relaxations) and $\partial \ell / \partial S$ (with respect to the layer weights) by pushing gradients through our SDP solution procedure (Section 5.1.2). We describe the analytical form for the resultant gradients in Theorem 5.1.

**Theorem 5.1.** *Define $P_o \equiv I_k - v_o v_o^T$ for each $o \in \mathcal{O}$. Then, define $U \in \mathbb{R}^{k \times n}$, where the columns $U_{\mathcal{I}} = 0$ and the columns $U_{\mathcal{O}}$ are given by*

$$\operatorname{vect}(U_{\mathcal{O}}) = (P((C+D) \otimes I_k)P)^\dagger \operatorname{vect}\left(\frac{\partial \ell}{\partial V_{\mathcal{O}}}\right), \tag{5.9}$$

*where $P \equiv \operatorname{diag}(P_o)$, where $C \equiv S_{\mathcal{O}}^T S_{\mathcal{O}} - \operatorname{diag}(\|s_o\|^2)$, and where $D \equiv \operatorname{diag}(\|g_o\|)$. Then, the gradient of the network loss $\ell$ with respect to the relaxed layer inputs is*

$$\frac{\partial \ell}{\partial V_{\mathcal{I}}} = -\Big(\sum_{o \in \mathcal{O}} u_o s_o^T\Big) S_{\mathcal{I}}, \tag{5.10}$$

*where $S_{\mathcal{I}}$ is the $\mathcal{I}$-indexed column subset of $S$, and the gradient with respect to the layer weights is*

$$\frac{\partial \ell}{\partial S} = -\Big(\sum_{o \in \mathcal{O}} u_o s_o^T\Big)^T V - (SV^T)U. \tag{5.11}$$

We defer the derivation of Theorem 5.1 to Appendix C.2. Although this derivation is somewhat involved, the concept at a high level is quite simple: we differentiate the solution of the SDP problem (Section 5.1.1) with respect to the problem's parameters and input, which requires computing the (relatively large) matrix-vector solve given in Equation (5.9).

To solve Equation (5.9), we use a coordinate descent approach that closely mirrors the coordinate descent procedure employed in the forward pass, and which has similar fast convergence properties. This procedure, described in Algorithm 5.3, enables us to compute the desired gradients without needing to maintain intermediate Jacobians explicitly. Mirroring the forward pass, we use rank-one updates to maintain and modify the term $\Psi = US^T$ needed to compute $\mathrm{d}g_o$, which again enables our algorithm to run in $O(nmk)$ time. We defer the derivation of Algorithm 5.3 to Appendix C.4.

**From relaxed to original inputs**

As a final step, we must use the gradient $\partial \ell / \partial V_{\mathcal{I}}$ (with respect to the input relaxations) to derive the gradient $\partial \ell / \partial Z_{\mathcal{I}}$ (with respect to the actual inputs) by pushing gradients through the input relaxation procedure described in Section 5.1.2. For each $\iota \in \mathcal{I}$, we see that

$$\begin{aligned} \frac{\partial \ell}{\partial z_\iota} &= \frac{\partial \ell}{\partial z_\iota^\star} + \left(\frac{\partial \ell}{\partial v_\iota}\right)^T \frac{\partial v_\iota}{\partial z_\iota} \\ &= \frac{\partial \ell}{\partial z_\iota^\star} - \left(\frac{\partial v_\iota}{\partial z_\iota}\right)^T \left(\sum_{o \in \mathcal{O}} u_o s_o^T\right) s_\iota \end{aligned} \tag{5.12}$$

where

$$\frac{\partial v_\iota}{\partial z_\iota} = \pi \left(\sin(\pi z_\iota) v_\top + \cos(\pi z_\iota)(I_k - v_\top v_\top^T) v_\iota^{\text{rand}}\right), \tag{5.13}$$

and where $\partial \ell / \partial z_\iota^\star$ captures any direct dependence of $\ell$ on $z_\iota^\star$ (as opposed to dependence through $v_\iota$). Here, the expression for $\partial \ell / \partial v_\iota$ comes from Equation (5.10), and we obtain $\partial v_\iota / \partial z_\iota$ by differentiating Equation (5.5).

### 5.1.4 An efficient GPU implementation

The coordinate descent updates in Algorithms 5.2 and 5.3 dominate the computational costs of the forward and backward passes, respectively. We thus present an efficient, parallel GPU implementation of these algorithms to speed up training and inference. During the inner loop of coordinate descent, our implementation parallelizes the computation of all $g_o$ ($\mathrm{d}g_o$) terms by parallelizing the computation of $\Omega$ ($\Psi$), as well as of all rank-one updates of $\Omega$ ($\Psi$). This underscores the benefit of using a low-rank SDP formulation in our MAXSAT layer, as traditional

**Algorithm 5.3** Backward pass coordinate descent

**input** $\{\partial \ell / \partial v_o \mid o \in \mathcal{O}\}$ *// grads w.r.t. relaxed outputs*
*// Compute* $U_{\mathcal{O}}$ *from Equation* (5.9)
**init** $U_{\mathcal{O}} = 0$ and $\Psi = (U_{\mathcal{O}})S_{\mathcal{O}}^T = 0$
**while** not converged **do**
    **for** $o \in \mathcal{O}$ **do** *// for all output variables*
        **compute** $\mathsf{d}g_o = \Psi s_o - \|s_o\|^2 u_o - \partial \ell / \partial v_o$.
        **compute** $u_o = -P_o \mathsf{d}g_o / \|g_o\|$.
        **update** $\Psi = \Psi + (u_o - u_o^{\text{prev}}) s_o^T$
    **end for**
**end while**
**output** $U_{\mathcal{O}}$

full-rank coordinate descent cannot be efficiently parallelized. We find in our preliminary benchmarks that our GPU CUDA-C implementation is up to $18 - 30$x faster than the corresponding OpenMP implementation run on Xeon CPUs. Source code for our implementation is available at `https://github.com/locuslab/SATNet`.

## 5.2 Experimental results

We test our MAXSAT layer approach in three domains that are traditionally difficult for neural networks: learning the parity function with single-bit supervision, learning $9 \times 9$ Sudoku solely from examples, and solving a "visual Sudoku" problem that generates the logical Sudoku solution given an input image of a Sudoku puzzle. We find that in all cases, we are able to perform substantially better on these tasks than previous deep learning-based approaches.

### 5.2.1 Learning parity (chained XOR)

This experiment tests SATNet's ability to differentiate through many successive SAT problems by learning to compute the parity function. The parity of a bit string is defined as one if there is an odd number of ones in the sequence and zero otherwise. The task is to map input sequences to their parity, given a dataset of example sequence/parity pairs. Learning parity functions from such single-bit supervision is known to pose difficulties for conventional deep learning approaches [129]. However, parity is simply a logic function – namely, a sequence of XOR operations applied successively to the input sequence.

Hence, for a sequence of length $L$, we construct our model to contain a sequence of $L - 1$ SATNet layers with tied weights (similar to a recurrent network). The first layer receives the first two binary values as input, and layer $d$ receives value $d$ along with the rounded output of layer $d - 1$. If each layer learns to compute the XOR function, the combined system will correctly compute parity. However, this requires the model to coordinate a long series of SAT problems without any intermediate supervision.

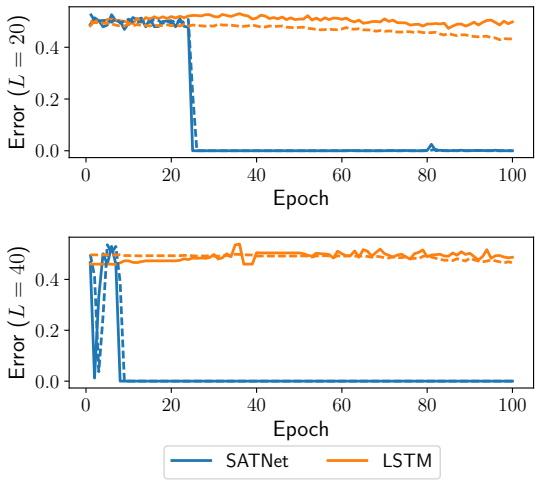



**Figure 5.2:** Error rate for the parity task with $L = 20$ (top) and $L = 40$ (bottom). Solid lines denote test values, while dashed lines represent training values.

| Model | Train | Test |
|---|---|---|
| ConvNet | 72.6% | 0.04% |
| ConvNetMask | 91.4% | 15.1% |
| **SATNet (ours)** | 99.8% | **98.3%** |

**(a)** Original Sudoku.

| Model | Train | Test |
|---|---|---|
| ConvNet | 0% | 0% |
| ConvNetMask | 0.01% | 0% |
| **SATNet (ours)** | 99.7% | **98.3%** |

**(b)** Permuted Sudoku.

| Model | Train | Test |
|---|---|---|
| ConvNet | 0.31% | 0% |
| ConvNetMask | 89% | 0.1% |
| **SATNet (ours)** | 93.6% | **63.2%** |

**(c)** Visual Sudoku. (Note: the theoretical "best" test accuracy for our architecture is 74.7%.)

**Table 5.1:** Results for $9 \times 9$ Sudoku experiments with 9K train/1K test examples. We compare our SATNet model against a vanilla convolutional neural network (ConvNet) as well as one that receives a binary mask indicating which bits need to be learned (ConvNetMask).

Figure 5.2 shows that our model accomplishes this task for input sequences of length $L = 20$ and $L = 40$. For each sequence length, we generate a dataset of 10K random examples (9K training and 1K testing). We train our model using cross-entropy loss and the Adam optimizer [83] with a learning rate of $10^{-1}$. We compare to an LSTM sequence classifier, which uses 100 hidden units and a learning rate of $10^{-3}$ (we tried varying the architecture and learning rate but did not observe any improvement). In each case, our model quickly learns the target function, with error on the held-out set converging to zero within 20 epochs. In contrast, the LSTM is unable to learn an appropriate representation, with only minor improvement over the course of 100 training epochs; across both input lengths, it achieves a testing error rate of at best 0.476 (where a random guess achieves value 0.5).

### 5.2.2 Sudoku (original and permuted)

In this experiment, we test SATNet's ability to infer and recover constraints simply from bit supervision (i.e. without any hard-coded specification of how bits are related). We demonstrate this property via Sudoku. In Sudoku, given a (typically) $9 \times 9$ partially-filled grid of numbers, a player must fill in the remaining empty grid cells such that each row, each column, and each of nine $3 \times 3$ subgrids contains exactly one of each number from 1 through 9. While this constraint satisfaction problem is computationally easy to solve once the rules of the game are specified, actually *learning the rules of the game*, i.e. the hard constraints of the puzzle, has proved challenging for traditional neural network architectures. In particular, Sudoku problems are often solved computationally via tree search, and while tree search cannot be easily performed by neural networks, it is easily expressible using SAT and MAXSAT problems.

We construct a SATNet model for this task that takes as input a logical (bit) representation of the initial Sudoku board along with a mask representing which bits must be learned (i.e. all bits in empty Sudoku cells). This input is vectorized, which means that our SATNet model cannot exploit the locality structure of the input Sudoku grid when learning to solve puzzles. Given this input, the SATNet layer then outputs a bit representation of the Sudoku board with guesses for the unknown bits. Our model architecture consists of a single SATNet layer with 300 auxiliary variables and low rank structure $m = 600$, and we train it to minimize a digit-wise negative log likelihood objective (optimized via Adam with a $2 \times 10^{-3}$ learning rate).

We compare our model to a convolutional neural network baseline modeled on that of Park [119], which interprets the bit inputs as 9 input image channels (one for each square in the board) and uses a sequence of 10 convolutional layers (each with 512 3×3 filters) to output the solution. The ConvNet makes explicit use of locality in the input representation since it treats the nine cells within each square as a single image. We also compare to a version of the ConvNet which receives a binary mask indicating which bits need to be learned (ConvNetMask). The mask is input as a set of additional image channels in the same format as the board. We trained both architectures using mean squared error (MSE) loss (which gave better results than negative log likelihood for this architecture). The loss was optimized using Adam (learning rate $10^{-4}$). We additionally tried to train an OptNet [5] model for comparison, but this model made little progress even after a few days of training. (We compare our method to OptNet on a simpler $4 \times 4$ version of the Sudoku problem in Appendix C.5.)

Our results for the traditional $9 \times 9$ Sudoku problem (over 9K training examples and 1K test examples) are shown in Table 5.1. (Convergence plots for this experiment are shown in Appendix C.6.) Our model is able to learn the constraints of the Sudoku problem, achieving high accuracy early in the training process (95.0% test accuracy in 22 epochs/37 minutes on a GTX 1080 Ti GPU), and demonstrating 98.3% board-wise test accuracy after 100 training epochs (172 minutes). On the other hand, the ConvNet baseline does poorly. It learns to correctly solve 72.6% of puzzles in the training set but fails altogether to generalize: accuracy on the held-out set reaches at most 0.04%. The ConvNetMask baseline, which receives a binary mask denoting which entries must be completed, performs only somewhat better, correctly solving 15.1% of puzzles in the held-out set. We note that our test accuracy is qualitatively similar to the results obtained in Palm et al. [116], but that our network is able to learn the structure of Sudoku *without explicitly encoding the relationships between variables*.

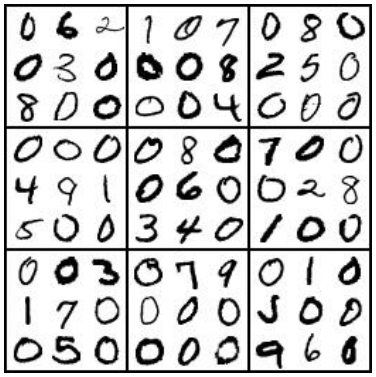


**Figure 5.3:** An example visual Sudoku image input, i.e. an image of a Sudoku board constructed with MNIST digits. Cells filled with the numbers 1–9 are fixed, and zeros represent unknowns.

To underscore that our architecture truly learns the rules of the game, as opposed to overfitting to locality or other structure in the inputs, we test our SATNet architecture on *permuted* Sudoku boards, i.e. boards for which we apply a fixed permutation of the underlying bit representation (and adjust the corresponding input masks and labels accordingly). This removes any locality structure, and the resulting Sudoku boards do not have clear visual analogues that can be solved by humans. However, the relationships between bits are unchanged (modulo the permutation) and should therefore be discoverable by architectures that can truly learn the underlying logical structure. Table 5.1 shows results for this problem in comparison to the convolutional neural network baselines. Our architecture is again able to learn the rules of the (permuted) game, demonstrating the same 98.3% board-wise test accuracy as in the original game. In contrast, the convolutional neural network baselines perform even more poorly than in the original game (achieving 0% test accuracy even with the binary mask as input), as there is little locality structure to exploit. Overall, these results demonstrate that SATNet can truly learn the logical relationships between discrete variables.

### 5.2.3 Visual Sudoku

In this experiment, we demonstrate that SATNet can be integrated into larger deep network architectures for end-to-end training. Specifically, we solve the visual Sudoku problem: that is, given an *image representation* of a Sudoku board (as opposed to a one-hot encoding or other logical representation) constructed with MNIST digits, our network must output a *logical solution* to the associated Sudoku problem. An example input is shown in Figure 5.3. This problem cannot traditionally be represented well by neural network architectures, as it requires the ability to combine multiple neural network layers *without* hard-coding

the logical structure of the problem into intermediate logical layers.

Our architecture for this problem uses a convolutional neural network connected to a SATNet layer. Specifically, we apply a convolutional layer for digit classification (which uses the LeNet architecture [89]) to each cell of the Sudoku input. Each cell-wise probabilistic output of this convolutional layer is then fed as logical input to the SATNet layer, along with an input mask (as in Section 5.2.2). This SATNet layer employs the same architecture and training parameters

as described in the previous section. The whole model is trained end-to-end to minimize cross-entropy loss, and is optimized via Adam with learning rates $2 \times 10^{-3}$ for the SATNet layer and $10^{-5}$ for the convolutional layer.

We compare our approach against a convolutional neural network which combines two sets of convolutional layers. First, the visual inputs are passed through the same convolutional layer as in our SATNet model, which outputs a probabilistic bit representation. Next, this representation is passed through the convolutional architecture that we compared to for the original Sudoku problem, which outputs a solution. We use the same training approach as above.

Table 5.1 summarizes our experimental results (over 9K training examples and 1K test examples); additional plots are shown in Appendix C.6. We contextualize these results against the theoretical "best" testing accuracy of 74.7%, which accounts for the Sudoku digit classification accuracy of our specific convolutional architecture; that is, assuming boards with 36.2 out of 81 filled cells on average (as in our test set) and an MNIST model with 99.2% test accuracy [89], we would expect a perfect Sudoku solver to output the correct solution 74.7% ($= 0.992^{36.2}$) of the time. In 100 epochs, our model learns to correctly solve 63.2% of boards at test time, reaching 85% of this theoretical "best." Hence, our approach demonstrates strong performance in solving visual Sudoku boards end-to-end. On the other hand, the baseline convolutional networks make only minuscule improvements to the training loss over the course of 100 epochs, and fail altogether to improve out-of-sample performance. Accordingly, our SATNet architecture enables end-to-end learning of the "rules of the game" directly from pictorial inputs in a way that was not possible with previous architectures.

## 5.3 Discussion

In this chapter, we have presented a low-rank differentiable MAXSAT layer that can be integrated into neural network architectures. This layer employs block coordinate descent methods to efficiently compute the forward and backward passes, and is amenable to GPU acceleration. We show that our SATNet architecture can be successfully used to learn logical structures, namely the parity function and the rules of $9 \times 9$ Sudoku. We also show, via a visual Sudoku task, that our layer can be integrated into larger deep network architectures for end-to-end training. Our layer thus shows promise in allowing deep networks to learn logical structure *without hard-coding of the relationships between variables*.

More broadly, we believe that this work fills a notable gap in the regime spanning deep learning and logical reasoning. While many "differentiable logical reasoning" systems have been proposed, most of them still require fairly hand-specified logical rules and groundings, and thus are somewhat limited in their ability to operate in a truly end-to-end fashion. Our hope is that by wrapping a powerful yet generic primitive such as MAXSAT solving within a differentiable framework, our solver can enable "implicit" logical reasoning to occur where needed within larger frameworks, even if the precise structure of the domain is unknown and must be learned from data. In other words, we believe that SATNet provides a step towards integrating symbolic reasoning and deep learning, a long-standing goal in artificial intelligence.

# Chapter 6

# Community detection using fast low-cardinality SDPs

*This chapter is modified from our NeurIPS'2020 paper [147] in collaboration with Zico.*

Community detection, that is, finding clusters of densely connected nodes in a network, is a fundamental topic in network science. A popular class of methods for community detection, called *modularity maximization* [111], tries to maximize the modularity of the cluster assignment, the quality of partitions defined by the difference between the number of edges inside a community and the expected number of such edges. However, optimizing modularity is NP-hard [36], so modern methods focus on heuristics to escape local optima. A very popular heuristic, the Louvain method [24], greedily updates the community membership node by node to the best possible neighboring community that maximizes the modularity function's gain. Then it aggregates the resulting partition and repeats until no new communities are created. The Louvain method is fast and effective [165], although it still gets trapped at local optima and might even create disconnected communities. A follow-up work, the Leiden method [140], resolves disconnectedness by an additional refinement step, but it still relies on greedy local updates and is prone to local optima.

In this chapter, we propose the *Locale* (low-cardinality embedding) algorithm, which improves the performance of community detecion above the current state of the art. It generalizes the greedy local move procedure of the Louvain and Leiden methods by optimizing a semidefinite relaxation of modularity, which originates from the extremal case of the max-$k$-cut semidefinite approximation [2, 59, 67] when $k$ goes to infinity. We provide a scalable solver for this semidefinite relaxation by exploiting the *low-cardinality* property in the solution space. Traditionally, semidefinite programming is considered unscalable. Recent advances in Riemannian manifold optimization [1, 40, 123] provide a chance to scale-up by optimizing directly in a low-rank solution space, but it is not amenable in many relaxations like the max-$k$-cut SDP, where there are nonnegativity constraints on all entries of the semidefinite variable $X$. However, due to the nonnegativity constraints, the solution $X$ is sparse and a low-cardinality solution in the factorized space $V$ suffices. These observations lead to our first contribution, which is a scalable solver for low-cardinality semidefinite programming subject to nonnegative constraints. Our second contribution is using this solver to create a generalization of existing community detection methods, which outperforms them in practice because it is less prone to local optima.

We demonstrate in the experiments that our proposed low-cardinality algorithm is far less likely to get stuck at local optima than the greedy local move procedure. On small datasets that are solvable with a traditional SDP solver, our proposed solver empirically reaches the globally optimal solution of the semidefinite relaxation given enough cardinality and is orders of magnitude faster than traditional SDP solvers. Our method uniformly improves over both the standard Louvain and Leiden methods, which are the state-of-the-art algorithms for community detection, with 2.2x time cost. Additionally, from the perspective of algorithmic design, the low-cardinality formulation opens a new avenue for scaling up semidefinite programming when the solutions tend to be sparse instead of low-rank. Source code for our implementation is available at `https://github.com/locuslab/sdp_clustering`.

**Notation.** We use upper-case letters for matrices and lower-case letters for vectors and scalars. For a matrix $X$, we denote the symmetric semidefinite constraint as $X \succeq 0$, the entry-wise nonnegative constraint as $X \geq 0$. For a vector $v$, we use $\mathrm{card}(v)$ for the number of nonzero entries, $\|v\|$ for the 2-norm, and $\mathrm{top}_k^+(v)$ for the sparsified vector of the same shape containing the largest $k$ nonnegative coordinates of $v$. For example, $\mathrm{top}_2^+((-1,3)) = (0,3)$, and $\mathrm{top}_1^+((-1,-2)) = (0,0)$. For a function $Q(V)$, we use $Q(v_i)$ for the same function taking the column vector $v_i$ while pinning all other variables. We use $[r]$ for the set $\{1,\ldots,r\}$, and $e(t)$ for the basis vector of coordinate $t$.

## 6.1 The Locale algorithm and application to community detection

In this section, we present the Locale (low-cardinality embedding) algorithm for community detection, which generalizes the greedy local move procedure from the Louvain and Leiden methods. We describe how to derive the low-cardinality embedding from the local move procedure, its connection to the semidefinite relaxation, and then how to round the embedding back to the discrete community assignments. Finally, we show how to incorporate this algorithm into full community detection methods.

### 6.1.1 Generalizing the local move procedure by low-cardinality embeddings

State-of-the-art community detection algorithms like the Louvain and Leiden methods depend on a core component, the local move procedure, which locally optimizes the community assignment for a node. It was originally proposed by Kernighan and Lin [81] for graph cuts, and was later adopted by Newman [110] to maximize the modularity $Q(c)$ defined in (2.2). The local move procedure in [24, 110] first initializes each node with a unique community, then updates the community assignment node by node and changes $c_i$ to a neighboring community (or an empty community) that maximizes the increment of $Q(c_i)$. That is, the local move procedure is an *exact coordinate ascent method* on the discrete community assignment $c$. Because it operates on the

discrete space, it is prone to local optima. To improve it, we will first introduce a generalized maximum modularity problem such that each node may belong to more than one community.

**A generalized maximum modularity problem.** To assign a node to more than one community, we need to rewrite the Kronecker delta $\delta(c_i = c_j)$ in $Q(c)$ as a dot product between basis vectors. Let $e(t)$ be the basis vector for community $t$ with one in $e(t)_t$ and zeros otherwise. By creating an assignment vector $v_i = e(c_i)$ for each node $i$, we have $\delta(c_i = c_j) = v_i^T v_j$, and we reparameterize the modularity function $Q(c)$ defined in (2.2) as

$$Q(V) := \frac{1}{2m} \sum_{ij} \left[ a_{ij} - \frac{d_i d_j}{2m} \right] v_i^T v_j. \tag{6.1}$$

Notice that the original constraint $c_i \in [r]$, where $r$ is the upper-bounds on number of communities, becomes $v_i \in \{e(t) \mid t \in [r]\}$. And the set becomes equivalent to the below unit norm and unit cardinality constraint in the nonnegative orthant.

$$\{e(t) \mid t \in [r]\} = \{v_i \mid v_i \in \mathbb{R}^r_+,\ \|v_i\| = 1,\ \mathrm{card}(v_i) \leq 1\}. \tag{6.2}$$

The constraint can be interpreted as the intersection between the curved probability simplex ($v_i \in \mathbb{R}^r_+,\ \|v_i\| = 1$) and the cardinality constraint ($\mathrm{card}(v_i) \leq 1$), where the latter constraint controls how many communities may be assigned to a node. Naturally, we can generalize the maximum modularity problem by relaxing the cardinality constraint from 1 to $k$, where $k$ is the maximun number of overlaying communities a node may belong to. The generalized problem is given by

$$\underset{V}{\text{maximize}}\ Q(V) := \frac{1}{2m} \sum_{ij} \left[ a_{ij} - \frac{d_i d_j}{2m} \right] v_i^T v_j,\ \ \text{s.t. } v_i \in \mathbb{R}^r_+,\ \|v_i\| = 1,\ \mathrm{card}(v_i) \leq k,\ \forall i. \tag{6.3}$$

The larger the $k$, the smoother the problem (6.3). When $k = r$ the cardinality constraint becomes trivial and the feasible space of $V$ become smooth. The original local move procedure is simply an exact coordinate ascent method when $k = 1$, and we now generalized it to work on arbitrary $k$ in a smoother feasible space of $V$. We call the resulting $V$ the low cadinality embeddings and the generalized algorithm the *Locale* algorithm. An illustration is given in Figure 6.1.

**The Locale algorithm for low-cardinality embeddings.** We first prove that, just like the local move procedure, there is a closed-form optimal solution for the subproblem $Q(v_i)$, where we optimize on variable $v_i$ and pin all the other variables.

**Proposition 6.1.** *The subproblem for variable* $v_i$

$$\underset{v_i}{\text{maximize}}\ Q(v_i),\ \ \textit{s.t. } v_i \in \mathbb{R}^r_+,\ \|v_i\| = 1,\ \mathrm{card}(v_i) \leq k \tag{6.4}$$

*admits the following optimal solution*

$$v_i = g/\|g\|,\ \ \textit{where } g = \begin{cases} e(t) \textit{ with the max } (\nabla Q(v_i))_t & \textit{if } \nabla Q(v_i) \leq 0 \\ \textit{top}_k^+(\nabla Q(v_i)) & \textit{otherwise} \end{cases}, \tag{6.5}$$

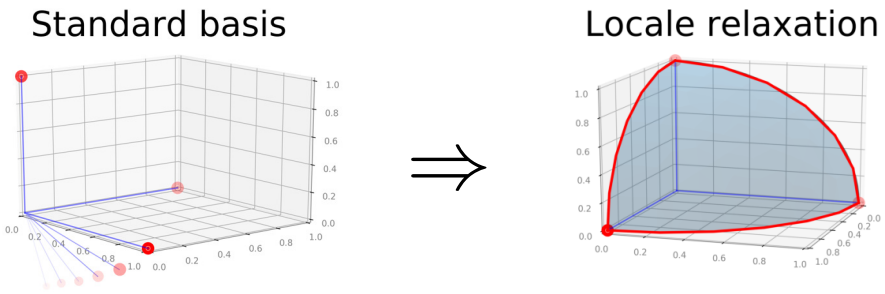



**Figure 6.1:** An illustration of the low cardinality relaxation, where the discrete cluster assignment for nodes is relaxed into a continuous and smooth space containing the original discrete set. The parameter $k$ controls the cardinality, or equivalently the maximum number of overlapping communities a node may belong to. When $k = 1$, we recover the original discrete set.

*where* $\mathrm{top}_k^+(q)$ *is the sparsified vector containing the top-*$k$*-largest nonnegative coordinates of* $q$*. For the special case* $\nabla Q(v_i) \leq 0$*, we choose the* $t$ *with maximum* $(v_i)_t$ *from the previous iteration if there are multiple* $t$ *with maximum* $(\nabla Q(v_i))_t$*.*

We list the proof in Appendix E.2. With the close-form solution for every subproblem, we can now generalize the local move procedure to a low-cardinality move procedure that works on arbitrary $k$. We first initialize every vector $v_i$ with a unique vector $e(i)$, then perform the *exact block coordinate ascent* method using the optimal update (6.5) cycling through all variables $v_i$, $i = 1, \ldots, n$, till convergence. We could also pick coordinate randomly, and because the updates are exact, we have the following guarantee.

**Theorem 6.2.** *Applying the low-cardinality update iteratively on random coordinates*[1]*, the projected gradient of the iterates converges to zero at* $O(1/T)$ *rate, where* $T$ *is the number of iterations.*

We list the proof in Appendix D.2. When implementing the Locale algorithm, we store the matrix $V$ in a sparse format since it has a fixed cardinality upper bound, and perform all the summation using sparse vector operations. We maintain a vector $z = \sum_j d_j v_j$ and compute $\nabla Q(v_i)$ by $(\sum_j a_{ij} v_j) - \frac{d_i}{2m}(z - d_i v_i)$. This way, updating all $v_i$ once takes $O\big(\mathrm{card}(A) \cdot k \log k\big)$ time, where the $\log k$ term comes from the partial sort to implement the $\mathrm{top}_k^+(\cdot)$ operator. Taking a small $k$ (we pick $k = 8$ in practice), the experiments show that it scales to large networks without too much additional time cost to the greedy local move procedure. Implementation-wise, we choose the updating order by the smart local move [12, 115]. We initialize $r$ to be the number of nodes and increase it when $\nabla Q(v_i) \leq 0$ and there is no free coordinate. This corresponds to the assignment to a new "empty community" in the Louvain method [24] (which also increases the $r$). At the worst case, the maximum $r$ is $n \cdot k$, but we have never observed this in the experiments, where in practice $r$ is always less than $2n$. For illustration, we provide an example from Leiden method [140] in Figure 6.2 showing that, because of the relaxed cardinality constraint, the Locale algorithm is less likely to get stuck at local optima compared to the greedy local move procedure.

[1]The proof can also be done with a cyclic order using Lipschitz continuity, but for simplicity we focus on the randomized version in our proof, which contains largely the same arguments and intuition.

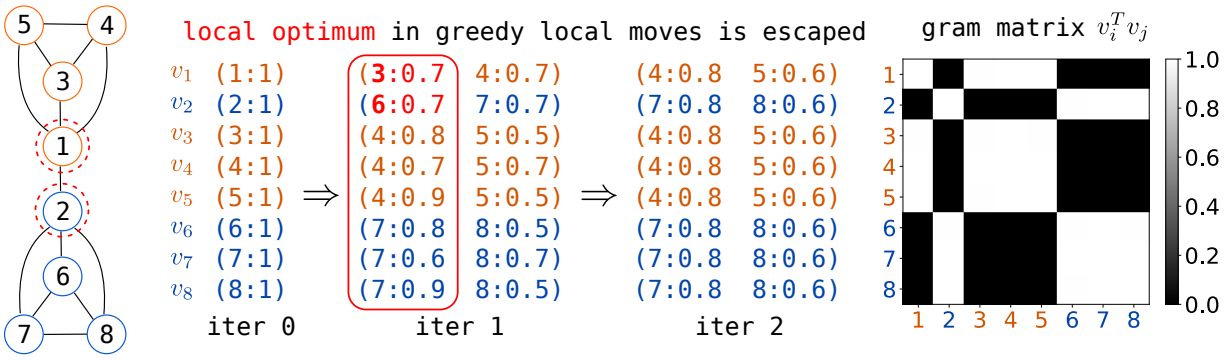



**Figure 6.2:** An example that the Locale algorithm escapes the local optimum in greedy local move procedure. Numbers in the parentheses are the low-cardinality embeddings in a sparse `index:value` format, where we compress a sparse vector with its top-$k$ nonzero entries. The above bottleneck graph was used in the Leiden paper [140] to illustration local optima, where a greedy local move procedure following the order of the nodes gets stuck at the local optima in the red box, splitting node 1 and 2 from the correct communities because of its unit cardinality constraint. In contrast, the Locale (low-cardinality embeddings) algorithm escapes the local optima because it has an additional channel for the top-$k$ communities to cross the bottleneck. The gram matrix of the resulting embeddings shows that it perfectly identifies the communities.

**Connections to correlation clustering SDP and copositive programming.** Here we connect the Locale solution to an SDP relaxation of the generalized modularity maximization problem (6.3). Let $r$ to be large enough[2] and let $k = r$ to drop the cardinality constraint, the resulting feasible gram matrix of $V$ becomes (the dual of) the copositive constraint

$$\{V^T V \mid V \geq 0,\ \mathrm{diag}(V^T V) = 1\}, \tag{6.6}$$

which can be further relaxed to the semidefinite constraint

$$\{X \mid X \succeq 0,\ X \geq 0,\ \mathrm{diag}(X) = 1\}. \tag{6.7}$$

This semidefinite constraint has been proposed as a relaxation for correlation clustering in [137]. With these relaxations, the complete SDP relaxation for the (generalized) maximum modularity problem is

$$\underset{X}{\text{maximize}} \sum_{ij} \left[ a_{ij} - \frac{d_i d_j}{2m} \right] x_{ij}, \quad \text{s.t. } X \succeq 0,\ X \geq 0,\ \mathrm{diag}(X) = 1. \tag{6.8}$$

We use the SDP relaxation to certify whether the Locale algorithm reaches the global optima, given enough cardinality $k$. That is, if the objective value given by the Locale algorithm meets the SDP relaxation (solvable via an SDP solver), it certifies the globally optimality of (6.3). In the experiments for small datasets that is solvable via SDP solvers, we show that a very low cardinality $k = 8$ is enough to approximate the optimal solution to a difference of $10^{-4}$, and running the Locale algorithm with $k = n$ recovers the global optimum. In addition, our algorithm is orders-of-magnitude faster than the state-of-the-art SDP solvers.

[2] At the worse case $r = n(n+1)/2 - 4$ suffices [27, Theorem 4.1].

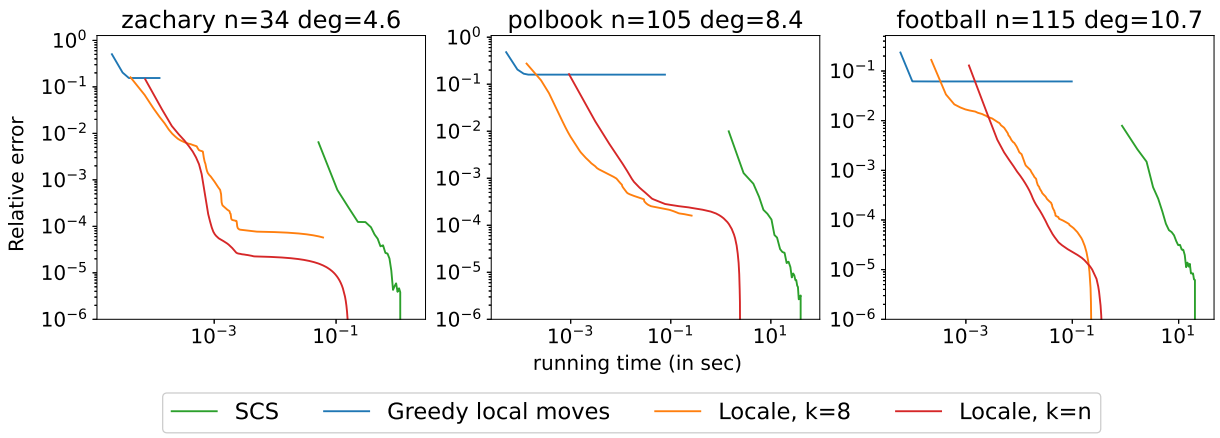



**Figure 6.3:** Comparing the relative error to optimal objective values and the running time in the semidefinite relaxation of maximum modularity. The optimal values are obtained by running the SCS [114], a splitting conic solver, for 3k iterations. The greedy local move procedure gets stuck pretty early at a local optimum (even for the original modularity maximization problem). The Locale algorithm is able to give a good approximation with cardinality $k = 8$, and is able to reach the global optima with $k = n$. Further, it is 100 to 1000 times faster than SCS, which is already orders of magnitude faster the than state-of-the-art interior point methods.

### 6.1.2 Rounding by changing the cardinality constraint

After obtaining the embeddings for the generalized modularity maximization algorithm (6.3), we need to convert the embedding back to unit cardinality to recover the community assignment for the original maximum modularity problem. This is achieved by running the Locale algorithm with the $k = 1$ constraint, starting at the previous solution. Also, since the rounding procedure reduces all embeddings to unit cardinality after the first sweep, this is equivalent to running the local move procedure of the Louvain method, but initialized with higher-cardinality embeddings. Likewise, we could also increase the cardinality constraint to update a unit cardinality solution to a higher cardinality solution. These upgrade and downgrade steps can be performed iteratively to increase the quality of the solution slowly, but we find that it is more efficient to only do the downgrade steps in the overall multi-level algorithm. The rounding process has the same complexity as the Locale algorithm since it is a special case of the algorithm with $k = 1$.

### 6.1.3 The Leiden-Locale algorithm for community detection

Here, we assemble all the aforementioned components and build the Leiden-Locale algorithm for community detection. We use the Leiden method [140] as a framework and replace the local move procedure with the Locale algorithm followed by the rounding procedure. While the results are better with more inner iterations of the Locale algorithm, we found that two inner iterations followed by the rounding procedure is more efficient in the overall multi-level algorithm over multiple iterations, while substantially improving over past works. We list the core pseudo-code below, and the subroutines can be found in the Appendix D.5.

**Algorithm 6.1** The Leiden-Locale method

**procedure** LEIDEN-LOCALE(Graph $G$, Partition $P$)
  **do**
    $E \leftarrow \texttt{LocaleEmbeddings}(G, P)$ ▷ Replace the $\texttt{LocalMove}(G, P)$ in Leiden
    $P \leftarrow \texttt{LocaleRounding}(G, P, E)$
    $G, P,$ done $\leftarrow \texttt{LeidenRefineAggregate}(G, P)$ ▷ [140, Algorithm A.2, line 5-9]
  **while** not done
  **return** $P$
**end procedure**

Because we still use the refinement step from the Leiden algorithm, we have the following guarantee.

**Theorem 6.3.** *[140, Thm. 5] The communities obtained from the Leiden-Locale algorithm are connected.*

Since we only perform two rounds of updates of the Locale algorithm, it adds relatively little overhead and complexity to the Leiden algorithm. However, experiments show that the boost is significant. The Leiden-Locale algorithm gives consistently better results than the other state-of-the-art methods.

## 6.2 Experimental results

In this section, we evaluate the Locale algorithm with other state-of-the-art methods. We show that the Locale algorithm is effective on the semidefinite relaxation of modularity maximization, improves the complexity from $O(n^6)$ to $O(\mathrm{card}(A)k \log k)$ over SDP solvers, and scales to millions of variables. Further, we show that on the original maximum modularity problem, the Locale algorithm significantly improves the greedy local move procedure. When used on the community detection problem, the combined Leiden-Locale algorithm provides a 30% additional performance increase over ten iterations and is better than all the state-of-the-art methods on the large-scale datasets compared in the Leiden paper [140], with 2.2x the time cost to the Leiden method. The code for the experiment is available at the supplementary material.

**Comparison to SDP solvers on the semidefinite relaxation of maximum modularity.** We compare the Locale algorithm to the state-of-the-art SDP solver on the semidefinite relaxation (6.8) of the maximum modularity problem. We show that it converges to the global optimum of SDP on the verifiable datasets and is much faster than the SDP method. We use 3 standard toy networks that are small enough to be solvable via an SDP solver, including `zachary` [170], `polbook` [111], and `football` [65]. Typically, primal-dual interior-point methods [136] have cubic complexity in the number of variables, which is $n^2$ in our problem, so the total complexity is $O(n^6)$. Moreover, the canonical SDP solver requires putting the nonnegativity constraint on the diagonal of the semidefinite variable $X$, leading a much higher number of variables. For fairness, we choose to compare with a new splitting conic solver, the SCS [114], which supports splitting variables into Cartesian products of cones, which is much more efficient than pure SDP solver in

**Table 6.1:** Overview of the empirical networks and the modularity after the greedy local move procedure (running till convergence) and the Locale algorithm (running for 2 rounds or till convergence).

| | | | Greedy | **The Locale algorithm** | |
|---|---|---|---|---|---|
| Dataset | Nodes | Degree | local moves | 2 rounds | full update |
| DBLP | 317 080 | 6.6 | 0.5898 | 0.6692 | 0.8160 |
| Amazon | 334 863 | 5.6 | 0.6758 | 0.7430 | 0.9154 |
| IMDB | 374 511 | 80.2 | 0.6580 | 0.6697 | 0.6852 |
| Youtube | 1 134 890 | 5.3 | 0.6165 | 0.6294 | 0.7115 |
| Live Journal | 3 997 962 | 17.4 | 0.6658 | 0.6540 | 0.7585 |

this kind of problem. We run the SCS solver for 3k iterations for the reference optimal objective value. Also, since the solution of SCS might not be feasible, we project the solution back to the feasible set by iterative projections.

Figure 6.3 shows the plot of difference to the optimal objective value and the running time. The greedy local move procedure gets stuck at a local optimum early in the plot, but the Locale algorithm gives a decent approximation at a low cardinality $k = 8$. At $k = n$, both the Locale algorithm and the SCS solver reach the optimum for the SDP relaxation (6.8), but Locale is 193x faster than SCS (in average) for reaching $10^{-4}$ difference to the optimum, and the speedup scales with the dimensions.

The results demonstrate the effectiveness and the orders of speedup of the proposed low-cardinality method, opening a new avenue for scaling-up semidefinite programming when the solution is low-cardinality instead of low-rank.

**Comparison with the local move procedure.** In this experiment, we show that the Locale algorithm scales to millions of nodes and significantly improves the local move procedure in empirical networks. We compare to 5 large-scale networks, including `DBLP`, `Amazon`, `Youtube` [163], `IMDB`[155],and `Live Journal` [11, 92]. These are the networks that were also studied in the Leiden [140] and Louvain papers [24]. For the greedy local move procedure, we run it iteratively till convergence (or till the function increment is less than $10^{-8}$ after $n$ consecutive changes to avoid floating-point errors). For the Locale algorithm, we test two different settings: running only two rounds of updates or running it till convergence, followed by the rounding procedure.

Table 6.1 shows the comparison results. With only 2 rounds of updates, the Locale algorithm already improves the greedy local move procedure except for the `Live Journal` dataset. When running a full update, the algorithm significantly outperforms the greedy local moves on all datasets. Moreover, it is even comparable with running a full (multi-level) iteration of the Louvain and Leiden methods, as we will see in the next experiments. The results suggest that the Locale algorithm indeed improves the greedy local move procedure. With the algorithm, we create a generalization of the Leiden method and show that it outperforms the Leiden methods in the next experiment.

**Table 6.2:** Overview of the empirical networks and the maximum modularity, running for 1 iterations (running the full multi-level algorithm till no new communities are created) and for 10 iterations (using results from the previous iteration as initialization). Note that results of Louvain [24] and Leiden [140] methods are obtained in additional 10 random trials.

| | | | Max. modularity | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | 1 iter | | | 10 iters 10 trials [140] | | 10 iters |
| Dataset | Nodes | Degree | Louvain | Leiden | **Locale** | Louvain | Leiden | **Locale** |
| DBLP | 317 080 | 6.6 | 0.8201 | 0.8206 | 0.8273 | 0.8262 | 0.8387 | 0.8397 |
| Amazon | 334 863 | 5.6 | 0.9261 | 0.9252 | 0.9273 | 0.9301 | 0.9341 | 0.9344 |
| IMDB | 374 511 | 80.2 | 0.6951 | 0.7046 | 0.7054 | 0.7062 | 0.7069 | 0.7070 |
| Youtube | 1 134 890 | 5.3 | 0.7179 | 0.7254 | 0.7295 | 0.7278 | 0.7328 | 0.7355 |
| Live Journal | 3 997 962 | 17.4 | 0.7528 | 0.7576 | 0.7531 | 0.7653 | 0.7739 | 0.7747 |

**Comparison with state-of-the-art community detection algorithms.** In the experiments, we show that the Leiden-Locale algorithm (Locale in the table) outperforms the Louvain and Leiden methods, the state-of-the-art community detection algorithms. For more context, the Louvain and Leiden methods are the state-of-the-art multi-level algorithm that performs the greedy local move procedure, refinement (for Leiden only), and aggregation of graph at every level until convergence. Further, they can be run for multiple iterations using previous results as initialization[3], so we consider the settings of running the algorithm once or for 10 iterations, where one iteration means running the whole multi-level algorithm until convergence. Specifically, for the 10-iteration setting, we take the best results over 10 random trials for the Louvain and Leiden methods shown in the Leiden paper [140]. Since the Locale algorithm is less sensitive to random seeds, we only need to run it once in the setting. This make it much faster than the Leiden method with 10 trials, while performing better. Further, we run the inner update twice for the Locale algorithm since we found that it is more efficient in the overall multi-level algorithm over multiple iterations.

Table 6.2 shows the result of comparisons. In the one iteration setting, the Locale algorithm outperforms both the Louvain and Leiden methods (except for the `Live Journal` dataset), with an 2.2x time cost in average. Using the Louvain method as a baseline, the Locale method provides a $0.0052$ improvement in average and the Leiden method provides a $0.0034$ improvement. These improvement are significant since little changes in modularity can give completely different community assignments [2]. For the 10 iteration setting, we uniformly outperform both Louvain and Leiden methods in all datasets and provide a 30% additional improvement over the Leiden method using the Louvain method as a baseline.

[3]The performance with low-number of iterations is also important because the Leiden method takes a default of 2 iterations in the `leidenalg` package and 10 iterations in the paper [140].

## 6.3 Discussion

In this chapter, we have presented the Locale (low-cardinality embeddings) algorithm for community detection. It generalizes the greedy local move procedure from the Louvain and Leiden methods by approximately solving a low-cardinality semidefinite relaxation. The proposed algorithm is scalable (orders of magnitude faster than the state-of-the-art SDP solvers), empirically reaches the global optimum of the SDP on small datasets that we can verify with an SDP solver. Furthermore, it improves the local move update of the Louvain and Leiden methods and outperforms the state-of-the-art community detection algorithms.

The Locale algorithm can also be interpreted as solving a generalized modularity problem (6.3) that allows assigning at most $k$ communities to each node, and this may be intrinsically a better fit for practical use because in a social network, a person usually belongs to more than one community. Further, the Locale algorithm hints a new way to solve heavily constrained SDPs when the solution is sparse but not low-rank. It scales to millions of variables, which is well beyond previous approaches. From the algorithmic perspective, it also opens a new avenue for scaling-up semidefinite programming by the low-cardinality embeddings.

# Chapter 7

# Linear programming via efficient piecewise quadratic optimization

*This chapter is a joint work with Ching-pei Lee.*

Linear programming (LP) is the optimization problem with linear objective function and linear constraints. Despite its simple formulation, the linear constraints in LP can encode a wide range of tasks, such as $\ell_1$-norm support vector machines [171], inverse covariance estimation [169], and MAP inference [84], just to name a few. We have shown how to scale up SDPs with specific diagonal constraints in the previous chapters, and here we will show how to extend the techniques to generic constraints in linear programming.

Traditionally, obtaining linear programming solutions has been considered as a solved problem and efficient interior-point methods that converge in a few iterations are readily available in commercial solvers like Gurobi and Mosek. Each iteration of these interior-point methods requires solving a linear system of the size of the problem and constraint dimension, and the linear system is often highly ill-conditioned because of the complementary constraint, making iterative solvers converge slowly. A more severe issue of the ill-conditioned linear system is that instead of iterative methods, it requires a direct solver with numerically more stable matrix factorizations. These factorizations have cubic computational complexity and usually multiply the memory consumption when the data is sparse. Thus, they are not suitable for modern machine learning tasks, of which large-scale problems are the norm.

Recently, efforts have been put in developing scalable solvers for large-scale linear programming using the augmented Lagrangian approach [95, 167]. The augmented Lagrangian method uses an inner iterative algorithm to solve a series of subproblems with increasing precision. Although the cost per iteration of the inner solver can be linear to the problem dimension or the data size, and the subproblems tend to possess much better problem conditions than that of the linear systems of interior-point methods, the experiments in [95, 167] show that augmented Lagrangian does not always outperform the highly-optimized commercial solvers that implement the interior-point methods. The major reason is the requirement of increasing precision for the subproblem solution, as solving the subproblem to a high accuracy can be rather lengthy. Moreover, the performance of the augmented Lagrangian approach is highly dependent on the parameters defining the subproblem and the accuracy of its approximate solution, but there is no

universal setting for the parameters that works effectively on all problems, and parameter tuning itself is a time-consuming task.

In this work, we propose a novel and efficient way to solve large-scale linear programming problems. We first show that finding a primal-dual solution pair to a linear programming problem is equivalent to solving any member of a family of simple unconstrained convex piecewise quadratic programming problems with a known optimal value (the family actually contains a broader range of problems but we focus on the simplest case for succinctness). We then propose an efficient algorithm, QULP (Quadratic Unconstrained optimization for Linear Programming), that adopts coordinate descent and semismooth Newton-CG approaches for such piecewise quadratic problems. QULP can fully utilize the sparsity and structure in the constraint matrix of the original linear programming problem to enjoy a low computational cost. In particular, since our algorithm only needs to evaluate $Ax$ or $A^\top y$ for vectors $x$ and $y$, but never explicitly compute $AA^\top$ or $A^\top A$, the cost per iteration is only proportional to the cost of computing $Ax$ and $A^\top y$, making it much more suitable for large-scale problems than interior-point methods. In comparison to the augmented Lagrangian, our approach is more succinct because it only has one piecewise quadratic problem to solve and needs no parameter tuning, while our solver can be at least as efficient as, if not better than, the subproblem solvers of the augmented Lagrangian approaches.

**Notation.** We use $I$ to denote the identity matrix. Given any set $C$, $\operatorname{dist}(x, C) \coloneqq \|x - P_C(x)\|$ is the distance from $x$ to $C$. For any vector $x$, $[x]_+$ is the projection of $x$ onto the nonnegative orthant.

## 7.1 The unconstrained piecewise quadratic formulation

The major difficulties for designing an optimizer for LP lies in utilizing the structure of the constraints. In this section, we show that an LP can be converted to an unconstrained piecewise quadratic problem, which can be solved using arbitrary first- and second-order methods. Consider the following form of linear programming w.r.t. variables $x \in \mathbb{R}^n$ and $z \in \mathbb{R}^p$.

$$\underset{x\in\mathbb{R}^n,\, z\in\mathbb{R}^p}{\text{minimize}}\ c^\top x + d^\top z, \quad \text{subject to} \quad Ax + Gz = b,\ z \geq 0, \tag{P}$$

where $A \in \mathbb{R}^{m\times p}$, $G \in \mathbb{R}^{m\times q}$ are the coefficient matrix, $b$ is the bias, $c$ and $d$ are the cost vectors. Using this notation, the corresponding dual problem of Eq. (P) is

$$\underset{y\in\mathbb{R}^m}{\text{maximize}}\ b^\top y, \quad \text{subject to} \quad A^\top y = c,\ G^\top y \leq d. \tag{D}$$

The major difficulty for solving an LP is in dealing with the constraints. Here, we show that the difficulty can be avoided by transforming the LP into an equivalent unconstrained piecewise quadratic program. We first show that a primal-dual solution pair for Eq. (P) and Eq. (D) can be obtained simultaneously by solving a simple unconstrained convex problem as follows.

**Proposition 7.1.** *Assume Eq.* (P) *has at least one optimal solution. Then for any point* $(x^*, z^*, y^*) \in \mathbb{R}^{p+q+m}$*, the following are equivalent.*

1. $(x^*, z^*)$ *is an optimal solution to Eq.* (P) *and* $y^*$ *is an optimal solution to Eq.* (D).

| | QULP (ours) | Barrier methods | Aug. Lagrangian | Cone splitting |
|---|---|---|---|---|
| # Inner iteration | Single | Multiple | Multiple | Multiple |
| Inner solver | Unconstrained piecewise QP | Linear Eqs | Simple QP | Linear Eqs |
| Condition number over iterations | Same | Worsen | Same | Same |
| Convergence rate | Superlinear | Quadratic | Linear[95, 167] | Linear [14, 114] |

**Table 7.1:** Comparison of properties between different type of solvers.

2. *Given any nonnegative functions* $r : \mathbb{R}^{1+m+2p+q} \mapsto \mathbb{R}_+$ *that are zero only at the origin,* $(x^*, z^*, y^*)$ *is an optimal solution to*

$$\underset{(x,z,y)\in\mathbb{R}^{p+q+m}}{\text{minimize}} \quad r\left([b^\top y - c^\top x - d^\top z]_+, \begin{bmatrix} Ax + Gz - b \\ A^\top y - c \end{bmatrix}, \begin{bmatrix} -z \\ G^\top y - d \end{bmatrix}_+\right) \tag{7.1}$$

*with objective value* $0$*.*

*Proof.* (condition 1 $\implies$ condition 2) Since the constraints of Eq. (P) and Eq. (D) are satisfied at the optimal solution in the first condition, the last two terms in (7.1) are all zero. The first term is also zero at the optimal solution following the strong duality of linear programs. Thus, the objective of (7.1) is zero. From the nonnegativity of the loss function $r$, the optimal solution for the first condition is also the optimal solution in the second condition.

(condition 2 $\implies$ condition 1) Since all terms in (7.1) are nonnegative, clearly they are all $0$ at $(x^*, y^*)$. This shows that $(x^*, z^*)$ is a feasible point for Eq. (P), $y^*$ is feasible for Eq. (D), and

$$c^\top x^* + d^\top z^* \leq b^\top y^*. \tag{7.2}$$

As $(x^*, z^*)$ is a feasible point for Eq. (P) and $y^*$ is feasible for Eq. (D), weak duality implies that

$$c^\top x^* + d^\top z^* \geq b^\top y^*. \tag{7.3}$$

Combining (7.2) and (7.3) then shows that $c^\top x^* + d^\top z^* = b^\top y^*$, meaning that $(x^*, z^*)$ and $y^*$ are respectively optimal for Eq. (P) and Eq. (D). □

The above proposition suggests a new way to solve the LP. That is, when Eq. (P) has at least one optimal solution, solving the reformulation equals finding a pair of primal-dual solutions, and we can estimate the accuracy of the solution by the objective value (zero when optimal). On the other hand, the optimal objective value will be larger than zero when there's no optimal solution for the LP, and we may output the status in the case. Further, if there are specific structures in $A$, we can utilize it to select suitable functions $r$ in Eq. (7.1) that allow for efficient computation. In our subsequent discussion, we assume that there is at least one optimal solution for the LP and consider the simplest case in which $r(\cdot) = \|\cdot\|_2^2$ so that Eq. (7.1) becomes

$$\underset{(x,z,y)\in\mathbb{R}^{p+q+m}}{\text{minimize}} \quad f(x, z, y) \coloneqq \left\| [b^\top y - c^\top x - d^\top z]_+, \begin{bmatrix} Ax + Gz - b \\ A^\top y - c \end{bmatrix}, \begin{bmatrix} -z \\ G^\top y - d \end{bmatrix}_+ \right\|^2. \tag{7.4}$$

Note that the above problem is an unconstrained piecewise quadratic program, and it admits Lipschitz gradient, is semismooth and strongly convex up to affine transformations. In fact, we will show that we can apply a mixture of first- and second-order algorithms to achieve both global linear convergence and local superlinear convergence on such a problem. We called it the Quadratic Unconstrained Linear Program (QULP). Compared to other linear program solvers like the penalty methods and the augmented Lagrangian methods, the QULP is an equivalent transformation instead of an iterative refinement; that is, it doesn't have multiple refining subproblems, and solving the QULP alone is equivalent to solving the LP. Further, because the QULP is unconstrained, we can apply arbitrary first- and (modified) second-order algorithm on the reformulation. We list the comparison of the QULP to other methods in Table 7.1.

## 7.2 The QULP algorithm

Here we propose the QULP, our algorithm for solving Eq. (7.4). We first describe two basic algorithms as cornerstones for different purposes and then discuss their combination as the final algorithm. For convenience, we will first simplify Eq. (7.4) as

$$\min_{w\in\mathbb{R}^{\hat{n}}} \quad f(w) \coloneqq \|Bw-u\|_2^2 + \|\,[Cw-v]_+\,\|^2, \tag{7.5}$$

by defining $w \coloneqq (x,z,y)$ and

$$B \coloneqq \begin{bmatrix} A & G & 0 \\ 0 & 0 & A^\top \end{bmatrix}, \quad u \coloneqq \begin{bmatrix} b \\ c \end{bmatrix}, \quad C \coloneqq \begin{bmatrix} -c^\top & -d^\top & b^\top \\ 0 & -I & 0 \\ 0 & 0 & G^\top \end{bmatrix}, \quad v \coloneqq \begin{bmatrix} 0 \\ 0 \\ d \end{bmatrix}.$$

### 7.2.1 The coordinate descent method with linear convergence

Because Eq. (7.5) is an unconstrained piecewise quadratic program, its subproblem for every single variable is also piecewise quadratic, which has a closed-form solution by the quick-select algorithm [150, Algorithm 2]. Naturally, we may apply coordinate descent methods (CD) to cyclically update every single variable subproblem. That is, at each iteration of CD, we pick a coordinate $i$ with basis vector $e_i$, and update the variable $w_i$ by

$$w_i \leftarrow w_i + d_i, \quad d_i \coloneqq \operatorname*{arg\,min}_{d_i\in\mathbb{R}} \quad f(w+e_i d_i). \tag{7.6}$$

In practice, even though the quick-select algorithm has a linear computational complexity to the sparsity of the coefficient matrix $A$, it has a larger constant in the runtime, and a Newton approximation will usually suffice, leading to our CD algorithm in Algorithm 7.1. To be more specific, we first compute $\nabla_i f(w)$ and then conduct the Newton-Raphson method to find the root for

$$\frac{\partial f(w-\alpha\cdot e_i\nabla_i f(w))}{\partial\alpha}$$

to decide the step size $\alpha$. Finally, we obtain $d_i = -\alpha\nabla_i f(w)$. For the coordinate selection, we use random permutation cyclic coordinate descent (RPCD) such that within each epoch (meaning

**Algorithm 7.1** The coordinate descent method for QULP

1: **for each** coordinate $i$ in random permuted order **do**
2: Set $w_i \leftarrow w_i - \theta \nabla_i f(w)/\nabla_i^2 f(w)$ by Armijo line search on $\theta$ starting from $\theta = 1$.
3: **end for**

$\hat{n}$ iterations), all coordinates are processed exactly once, with the order reshuffled every epoch. The behavior of RPCD is often observed to be similar to that of stochastic CD (SCD) that picks the coordinates uniformly randomly with replacement, which has a much better convergence guarantee for the expected objective than cyclic CD (CCD), while RPCD also shares with CCD the same guarantee for the stronger deterministic convergence and the ability to identify the active constraints.

The most expensive part of CD is computing $\nabla_i f(w)$ and conducting line search. For the gradient computation, we reduce its cost by keeping track of $r_1 \coloneqq Bz - u$ and $r_2 \coloneqq Dz - v$. Let $B = [B_1, \ldots, B_{\hat{n}}]$ and $C = [C_1, \ldots, C_{\hat{n}}]$. The computation then becomes $\nabla_i f(w) = 2B_i^\top r_1 + 2C_i^\top [r_2]_+$, and the update of $r_1$ and $r_2$ is conducted by $r_1 \leftarrow B_i d_i$ and $r_2 \leftarrow C_i d_i$. We see that at each step, we just need to access one row or column of $A$ twice, so the operations can be conducted efficiently. The part of line search only involves utilization of $r_1, r_2, B_i$, and $C_i$ and tends to finish in few Newton-Raphson steps, so it is not more expensive than the gradient computation.

Regarding the convergence of RPCD, we make the following two observations. First, the problem is unconstrained, convex, and piecewise quadratic, so it is Lipschitz-continuously differentiable and satisfies the error-bound condition [94], which means that given any initial point $w^0$, there is $\kappa \geq 0$ such that

$$\|\nabla f(w)\| \geq \kappa \text{dist}(w, \Omega), \quad \forall w \in \{w \mid f(w) \leq f(w^0)\}. \tag{7.7}$$

Second, even if the Newton-Raphson line search procedure did not produce the exact minimizer, it would still have given a step size that generates sufficient objective decrease, such that the new objective is not larger than that of taking the reciprocal of the coordinate-wise Lipschitz constant as the step size, and thus we always have $f(w) \leq f(w^0)$ for Eq. (7.7) to continue being in work. Through these observations, the following theorem leverages the analysis by Wang and Lin [148] to show that when RPCD is applied to Eq. (7.5), $f(w)$ converges to 0 $Q$-linearly, and the iterates approach the solution set $R$-linearly and further converge to an optimal solution. Therefore, RPCD can quickly generate an approximate solution if high precision is not needed, as is usually the case in machine learning applications.

**Theorem 7.2.** *Assume that Eq.* (P) *has at least one optimal solution. When RPCD is applied to Eq.* (7.5) *with an initial point $w^0 \in \mathbb{R}^{\hat{n}}$ and let the iterates obtained after each epoch be $w^1, w^2, \ldots$, there exists $\eta \in [0, 1)$ such that*

$$f\left(w^{t+1}\right) \leq \eta f\left(w^t\right), \quad \forall t \geq 0. \tag{7.8}$$

*Moreover, let $\Omega$ be the solution set of Eq.* (7.1)*, then the iterates converge to a point $w^* \in \Omega$ and there is $\tau > 0$ such that*

$$\text{dist}\left(w^t, \Omega\right) \leq \tau \eta^{\frac{t}{2}}.$$

**Algorithm 7.2** The semismooth Newton method for QULP

1: Denote $H = \nabla^2 f(w) + c\|\nabla f(w)\| I$ for a $c > 0$.
2: Initialize $d = 0$.
3: **while** criterion (7.10) is not met **do**
4: Perform an iteration of conjugate gradient method on the system $Hd + \nabla f(w) = 0$.
5: **end while**
6: Set $w = w + \theta d$ by backtracking line search on $\theta$ with criterion (7.11) starting from $\theta = 1$.

### 7.2.2 The semismooth Newton method with superlinear convergence

In the traditional optimization scheme, a highly precise solution for a linear programming problem might be desirable. In this case, RPCD may not be a suitable choice as it does not possess fast superlinear local convergence like the interior-point methods. Truncated Newton or quasi-Newton are practical choices for achieving superlinear convergence if the objective is twice-differentiable and strongly convex at least around the solution set. Unfortunately, these conditions do not hold for Eq. (7.5). But fortunately, in addition to that $f$ is Lipschitz-continuously differentiable, the gradient of $f$ is piecewise linear, ergo strongly semismooth [58, Proposition 7.4.7], meaning that $\nabla f$ is directionally differentiable, and for any $w$ and any $\nabla^2 f(w + \Delta w) \in \partial(\nabla f(w + \Delta w))$ with $\Delta w \to 0$,

$$\nabla f(w + \Delta w) - \nabla f(w) - \nabla^2 f(w + \Delta w)\Delta w = O\left(\|\Delta w\|^2\right). \tag{7.9}$$

We can therefore use the generalized Hessian to conduct semismooth Newton (SSN) updates. In particular, any $\nabla^2 f(w) \in \partial(\nabla f(w))$ is a generalized Hessian that can be used. Since $f$ is not strongly convex, a damping term in the generalized Hessian is needed to ensure positive definiteness. Therefore, given parameters $c > 0, \rho \in (0, 1]$, at each iteration, we select $\nabla^2 f(w) \in \partial(\nabla f(w))$ and find the truncated SSN step $d$ by approximately solving

$$d \approx \arg\min_{\bar{d}} \ \nabla f(w)^\top \bar{d} + \frac{1}{2}\bar{d}^\top \left(\nabla^2 f(w) + c\|\nabla f(w)\|^\rho I\right) \bar{d}$$

through the linear conjugate gradient (CG) method. The method is terminated when $d$ satisfies

$$\begin{aligned} &\nabla f(w)^\top d + \frac{1}{2} d^\top \left(\nabla^2 f(w) + c\|\nabla f(w)\|^\rho I\right) d \le 0, \\ &\left\| \left(\nabla^2 f(w) + c\|\nabla f(w)\|^\rho I\right) d + \nabla f(w) \right\| \le \nu \min\left\{\|\nabla f(w)\|, \|\nabla f(w)\|^{1+\rho}\right\} \end{aligned} \tag{7.10}$$

for some pre-specified $\nu \in [0, 1)$. The CG procedure does not require us to explicitly form $\nabla^2 f(w)$, but only needs to evaluate $\nabla^2 f(w)s$ for different $s$ at each CG iteration. Thus, we utilize the sparse matrix-vector multiplication of the generalized Hessians of $f$ and vector $s$, which are of the form

$$\nabla^2 f(w)s = 2B^\top Bs + 2C^\top \operatorname{diag}(h(w))Cs,$$

where $h(w)$ is a sparse indicator vector with

$$h(w)_i = \begin{cases} 1 & \text{if } (Cw - v)_i > 0 \\ 0 & \text{otherwise .} \end{cases}.$$

**Algorithm 7.3** The overall QULP algorithm

Initialize $\hat{T} = T_0$, and pick a multiplying factor $\delta_2 > 1$.
**while** not yet converged **do**
    Run the coordinate descent algorithm (Algorithm 7.1) for $\hat{T}$ iterations.
    Perform the semismooth Newton algorithm (Algorithm 7.2) once.
    **if** criterion (7.12) holds **then**
        Set $\hat{T} \leftarrow \delta_2 \hat{T}$.
    **else**
        Set $\hat{T} \leftarrow \min(1, \hat{T}/\delta_2)$.
    **end if**
**end while**

After obtaining $d$, and given parameters $\gamma, \sigma \in (0, 1)$, we conduct a backtracking line search to find the step size $\theta$ as the largest element in $\{\gamma^0, \gamma^1, \dots\}$ that satisfies

$$f(w + \theta d) \leq f(w) + \theta\sigma \min\{1, c\|\nabla f(w)\|^\rho\} \nabla f(w)^\top d, \tag{7.11}$$

and the iterate is updated by $w \leftarrow w + \theta d$. Notice that backtracking can be done efficiently by computing $Bd$ and $Cd$ first and estimate $f(w+\theta d)$ through $Bw-u+\theta(Bd)$ and $Cw-v+\theta(Cd)$. The overall semismooth Newton algorithm is presented in Algorithm 7.2.

In the following theorem, we show that our semismooth Newton approach achieves the superlinear convergence of $\nabla f(w)$, $f(w)$, and $\text{dist}(w, \Omega)$ to 0 following Lee and Wright [90], where $\Omega$ is the set of optimum. Consequently, the iterates converge to a point in the optimum.

**Theorem 7.3.** *Assume that Eq.* (P) *has at least one optimal solution. If Eq.* (7.5) *is solved by the truncated semismooth Newton approach described above with initial point $w^0$ and the iterates are $w^1, w^2, \dots$, then there is $t_0 \geq 0$ such that for all $t \geq t_0$,*

$$\|\nabla f\left(w^{t+1}\right)\| = O\left(\|\nabla f\left(w^t\right)\|^{1+\rho}\right), \text{dist}\left(w^{t+1}, \Omega\right) = \text{dist}\left(w^t, \Omega\right)^{1+\rho}, f\left(w^{t+1}\right) = O\left(f\left(w^t\right)^{1+\rho}\right).$$

*Moreover, the iterates converge to a point $w^* \in \Omega$.*

### 7.2.3 A hybrid method

The major problem of the SSN approach is that one iteration of it involves running the CG procedure once, which is much more expensive than an epoch of RPCD (one iteration of CG costs roughly the same as an epoch of RPCD). We also observe that although SSN attains fast local rates, its convergence speed at the early stage tends to be slower than that of RPCD. Therefore, there is a need to interlace the two and form a hybrid algorithm.

In most cases, it is hard to decide when to switch because the convergence rate is known only when the optimal objective or the optimal solution is available in a priori. Fortunately, it is the case for our problem because the optimal objective value is 0 for Eq. (7.5). We can therefore use this property to decide if SSN gives enough advantage over RPCD to cover its higher cost. Notice that one epoch of RPCD accesses each entry of $B$ twice, and so does one round of CG, while the

computation of the gradient and the step size respectively needs to access $B$ once. Those costs provide an intuitive way to mix the two methods.

We first run RPCD for a pre-specified $T_0$ rounds as RPCD tends to be faster than SSN at the early stage. Then we start to run SSN once every $\hat{T}$ round. If at the $(k-1)$-th round RPCD is conducted and at the $k$-th round SSN is conducted with $T_k$ CG iterations, we increase $\hat{T}$ by a factor $\delta_2 > 1$ if

$$\frac{f\left(w^{k-1}\right)}{f\left(w^{k-2}\right)} \leq \left(\frac{f\left(w^{k}\right)}{f\left(w^{k-1}\right)}\right)^{T_k+1}, \tag{7.12}$$

and otherwise decrease it by the same factor as long as $\hat{T}$ is larger than 1. We present the overall QULP method in Algorithm 7.3.

## 7.3 Experimental results

In this section, we compare the performance of QULP on both synthesis and real-world large-scale linear programming problems with state-of-the-art packages. We consider all solvers in Gurobi (version 9.1) and the approach of Yen et al. [167] to include both a mature interior-point method and augmented Lagrangian approach.[1] For Gurobi, we report results of their barrier method as well as the primal and dual simplex methods. For all methods except the simplex ones, we use the following stopping condition

$$\max\left\{\left\|\begin{bmatrix} Ax - Gz - b \\ A^\top y - c \end{bmatrix}\right\|_\infty, \left\|\begin{bmatrix} -z \\ G^\top y - d \end{bmatrix}_+\right\|_\infty, \left|\frac{c^\top x + d^\top z - b^\top y}{1 + |c^\top x + d^\top z|}\right|\right\} \leq \epsilon, \tag{7.13}$$

with $\epsilon = 10^{-3}$, while simplex methods consider the first three terms in Eq. (7.13) only. The experiments are conducted on dedicated Amazon EC2 m5.24xlarge instances with $384$ GB memory and $48$ cores, with eight cores assigned to each solver for each problem. Except for the stopping criteria and the number of threads, all other parameter settings follow the default ones of the solvers.

We conduct four sets of experiments. The first one uses randomly generated data, the second one solves the $\ell_1$-norm support vector machine problem proposed by [171], the third one compares the solvers on network flow problems, and the last one considers benchmark linear programming problems. Due to the space limit, the last one is put in the supplementary materials. For all datasets except for the synthesis ones, we preprocess them using the aggressive presolve of Gurobi and report the statistics of the processed data to remove redundant rows and columns.

**Synthesis Data** We first start with generating random instances for Eq. (P) and Eq. (D). We slightly modify the procedure in [33, Chapter 11.3] to generate the data as follows. Each element of $A$ is set to zero with probability $(1-p)$, and those not set to zero follow independent and identically distributed (iid) standard normal distribution. We then set $b = Ax^*$, where $x^* \in \mathbb{R}^n$

[1] [95] is excluded because their algorithm is implemented in MATLAB, which is non-open-source and unavailable in our experimental environment, and their code is not publicly available.

| Dataset ($10^m$) | P-Simplex | D-Simplex | Barrier | [167] (P) | [167] (D) | QULP |
|---|---|---|---|---|---|---|
| $10^3$ | 4.0 | 3.0 | 1.0 | 18.9696 | 242.174 | 5.37746 |
| $10^4$ | 5568.0 | 3323.56 | 8.0 | 304.613 | 368.681 | 15.5328 |
| $10^5$ | $>$ 24hr | $>$ 24hr | 43695.38 | 3612.12 | 6815.8 | 725.526 |
| $10^6$ | $>$ 24hr | $>$ 24hr | X | $>$ 24hr | $>$ 24hr | 7652.88 |
| $10^7$ | $>$ 24hr | $>$ 24hr | X | $>$ 24hr | $>$ 24hr | $>$ 24hr |

**Table 7.2:** Running time (seconds) comparison on synthesis data. The data name $10^i$ indicates that $m = n = 10^i$. We report time when the solver reaches $\epsilon = 10^{-1}$ in Eq. (7.13). X means the solver incurred an out of memory error. For Simplex and [167], P denotes primal and D denotes dual.

are iid uniform from $[0, 1]$, and $c = A^\top y^* + s$, where $y^* \in \mathbb{R}^m$ are iid standard normal, and elements of $s \in \mathbb{R}^n$ are iid uniform from $[0, 1]$. We test $m = n \in \{10^3, 10^4, 10^5, 10^6, 10^7\}$, while set $p = 50m^{-1}$ to ensure that the data are sparse enough to fit in the memory capacity. The results are reported in Table 7.2, and it shows that our method (QULP) are significantly better than the primal/dual simplex method and the barrier method when the data is large enough. Further, it is uniformly better than the primal/dual augmented Lagrangian methods in [167].

**Multi-class Classification** We conduct a comparison on $\ell_1$-regularized multi-class support vector machine problems. Given training instances $(x_i, y_i) \in \mathbb{R}^n \times \{1, \dots, K\}$, $i = 1, \dots, m$, where $K \in \mathbb{N}$ is the number of possible classes, it solves the following linear optimization problem.

$$\begin{aligned} \min_{w_1, w_K \in \mathbb{R}^n, \xi \in \mathbb{R}^m} \quad & \lambda \sum_{k=1}^{K} \|w_k\|_1 + \sum_{i=1}^{m} \xi_i \\ \text{subject to} \quad & w_{y_i}^\top x_i - w_k^\top x_i \geq e_i^k - \xi_i, \quad i = 1, \dots, m, \quad k = 1, \dots, K. \end{aligned}$$

This problem can be easily formulated into a bound constrained linear programming problem by splitting each $w_k$ into $w_k = w_k^+ - w_k^-$ with the constraint $w_k^+, w_k^- \geq 0$. The data statistics are summarized in 7.3[2] and the running time is shown in Table 7.4. As suggested by the experimental results, our methods are uniformly better than the primal/dual simplex methods and the barrier method, and are better than the primal/dual methods in [167] in 4 over the 6 datasets.

## 7.4 Conclusion

In this work, we proposed a versatile unconstrained piecewise quadratic optimization reformulation for linear programming problems utilizing its strong duality. Efficient algorithms utilizing the structure and sparsity of the original linear programming problem while achieving global linear and local superlinear convergence are then shown. Experiments demonstrate that our approach is not only theoretically sound but also superior to existing methods in empirical performance.

[2]Original data downloaded from the LIBSVM site: `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

| Data set | rows | columns | Nonzero entries |
|---|---|---|---|
| real-sim | 72,201 | 102,708 | 7,082,389 |
| news20 | 318,700 | 795,932 | 40,719,163 |
| vehicle | 236,469 | 158,248 | 47,845,513 |
| rcv1_train.multiclass | 793,764 | 607,256 | 70,586,511 |
| sector.scale | 673,260 | 1,107,940 | 129,813,911 |
| mnist.scale | 600,000 | 132,390 | 181,088,991 |

**Table 7.3:** Data statistics of multi-class classification problems.

| Data set | P-Simplex | D-Simplex | Barrier | [167] (P) | [167] (D) | QULP |
|---|---|---|---|---|---|---|
| real-sim | 64877.0 | 4517.0 | 595.0 | 684.363 | 68.6403 | 2.68164 |
| news20 | $> 24$hr | $> 24$hr | 12262.0 | 69071.8 | $> 24$hr | 1475.4 |
| vehicle | 10395.0 | $> 24$hr | 565.0 | $> 24$hr | 7021.8 | 72.8821 |
| rcv1_train.multiclass | $> 24$hr | $> 24$hr | 69240.0 | 7334.91 | 739.41 | 56324.0 |
| sector.scale | 1686.0 | 6509.0 | 15060.0 | 9612.75 | 7145.75 | 8828.67 |
| mnist.scale | $> 24$hr | $> 24$hr | 16548.0 | $> 24$hr | 75325.3 | 1420.48 |

**Table 7.4:** Running time (seconds) comparison on classification problems. For Simplex and [167], P denotes primal and D denotes dual.

# Chapter 8

# Conclusion

In this thesis, we have shown how to scale up structured SDPs and applied them to probabilistic inference, parameter learning, clustering, and generic problems with linear constraints. We have presented *Mixing methods* (Chapter 3), the fundamental building block in our thesis, allowing us to scale SDPs to millions of variables with convergence guarantees by solving the corresponding vector program. We have demonstrated the methods' effectiveness on inferencing Markov Random Fields (Chapter 4). Further, we have used it to approximate the MAXSAT problem (Chapter 5), creating a differentiable SAT-solving layer that enables the integration of deep learning and logic reasoning. We have proposed a variant for low cardinality SDPs, applying it to the unsupervised community detection problem (Chapter 6) within large-scale networks and surpassing the discrete state-of-the-art solvers in both performance and speed. We have also shown how to apply it to linear programs with generic constraints (Chapter 7). The idea of applying SDP to machine learning problems by considering its corresponding vector program has a promising set of future directions. In the following, we provide a brief outlook of the possible applications.

1. **Combinatorial optimization.**
    - **Lasserre hierarchy and SOS.** Many of the combinatorial optimization problems can be approximated by the Lasserre hierarchy [86, 87, 88] or the sum-of-square hierarchy [23, 121], which relax them to generic linearly-constrained SDPs. On the one hand, it is possible to convert generic SDP to a vector program and solve it by coordinate descent methods, using similar techniques as demonstrated in Chapter 7. On the other hand, we can sometimes further relax the complex constraints to diagonal constraints by modifying the coefficient matrix (Chapter 5). The trade-off between constraint complexity and approximation efficiency can really be further explored.
    - **Approximate and exact solution.** SDPs and randomized rounding may give unexpectedly high approximation ratio in real-world data, as confirmed in MAXCUT (Chapter 3), MAXSAT [69], and community detection (Chapter 6). The reason behind this phenomenon is still unknown. But in the other hand, we may use the corresponding vector embedding to predict the approximation ratio, and it will have applications in the branch-and-cut process to get the exact solution. At the very least, the objective value of the SDPs is already useful in pruning the branches in MAX2SAT [146].
    - **Embedding and randomized rounding.** The process of solving a vector program

can be seen as finding a (rotation invariant) vector embedding for the corresponding energy function defined by the SDP objective function with dot-product interactions. This way, the randomized rounding procedure corresponds to efficiently sampling the possible (discrete) configuration space. For different applications, we may define or learn a customized (nonlinear) objective function (Chapter 5) tailored to our need and differentiate through the randomized rounding process to provide feedback. However, how to design a proper nonlinear objective function and how to do randomized rounding in more complex action space (e.g., sequence) remain a direction to be explored.

2. **Semi-supervised learning.** The features in the low-cardinality embedding for the community detection problem (Chapter 6) actually work like a bag-of-word descriptor; it describes the top-$k$ possible attributes that a node may have, and it's generated solely from the local graph information. We may further associate the attributes (keys) with vector embeddings (values) to incorporate more information, and such formulation is closely related to the slot attention [99].
3. **Language modeling.** Running a sliding window through a language corpus and define a directed co-occurrence graph with causality (i.e., the arrows only go right), we can model the word embedding problem as a large vector program. Further, if we associate edge weights with the dot-products between vector embeddings, the vector program will look pretty similar to a transformer [47, 143]. With some modification, an equilibrium transformer [13] can actually be written as an SDP, in which each layer is performing a step of the Mixing method. This partly explains the transformer's ability to perform reasoning. However, how to make the modification as good as the original transformer remains a direction to be explored.
4. **Neural symbolic reasoning.** We have demonstrated how to approximate the MAXSAT problems and learn its parameters in Chapter 5. During the process, we "round" the vector embedding back to a scalar by projection, which forms a bottleneck and requires tuning to backpropagate smoothly [43]. The condition may be enhanced by incorporating a differentiable randomized rounding procedure, which may be an interesting direction. Also, we may use the embedding to guide the search process instead, but in that case, we need to backpropagate through the search process. On the other hand, we know that many reasoning problems can be efficiently reduced to a MAXSAT problem, so the differentiable MAXSAT solver may be used as a bridge to neural symbolic reasoning. However, there are still challenges to be addressed. For example, how to dynamically change the graph structure once there are new objects. Also, it may not be necessary to iterate the vector embedding to an exact equilibrium; many real-world problems only take a few steps of derivation.

# Appendix A

# Proofs for the Mixing methods

## A.1 Proof of Theorem 3.1: Convergence to critical points

**Lemma A.1.** *Let $\hat{V} = M(V)$ for the Mixing method $M : \mathbb{R}^{k\times n} \to \mathbb{R}^{k\times n}$ defined in* (3.4) *and* (3.5).

$$f(V) - f(\hat{V}) = \sum_{i=1}^{n} y_i \|v_i - \hat{v}_i\|^2.$$

*Proof.* Recall the objective function $f$ w.r.t. variable $v_i$ while fixing all other variables $v_j$ is

$$\langle C, V^T V\rangle = \sum_i \sum_j c_{ij} v_i^T v_j = 2v_i^T (\sum_j c_{ij} v_j) + \text{constant}.$$

Note that the $\sum_j c_{ij} v_j$ term is independent of $v_i$ because $c_{ii} = 0$. Now consider the inner cyclic iteration of the Mixing method updating $v_i$ to $\hat{v}_i$. Because only those $v_j$ with $j < i$ are updated to $\hat{v}_j$, the objective value before updating $v_i$ to $\hat{v}_i$ equals $2v_i^T(\sum_{j<i} c_{ij}\hat{v}_j + \sum_{j>i} c_{ij} v_j) = 2g_i^T v_i$ plus constants, Thus, the updates of the Mixing method can be written as

$$\hat{v}_i = -g_i/y_i, \quad \text{where } y_i = \|g_i\| \text{ and } g_i = \sum_{j<i} c_{ij}\hat{v}_j + \sum_{j>i} c_{ij} v_j, \quad i = 1\dots n. \tag{A.1}$$

and the objective difference after updating $v_i$ to $\hat{v}_i$ is

$$2g_i^T(v_i - \hat{v}_i) = -2\|g_i\|\hat{v}_i^T(v_i - \hat{v}_i) = 2y_i(1 - v_i^T\hat{v}_i) = y_i\|v_i - \hat{v}_i\|^2.$$

The result follows from summing above equation over $i = 1\dots n$. □

**Proof of Theorem 3.1**

*Proof.* From Lemma A.1 and the compactness of the space of $V$, the sequence $\{f(V^r)\}$ for iterate $V^r$ is monotonically decreasing and converges to a value $\bar{f}$. The function decrease converges to

zero because of Lemma A.1 and $y_i = \|g_i\|$ being bounded. Further, the normalizer $y_i$ does not degenerate due to Assumption 3.2, so we have

$$\lim_{r\to\infty} \|v_i^r - \hat{v}_i^r\|^2 = 0.$$

That is, every limit point is a fixed point. Now we prove that every limit point in $\{V^r\}$ is also a critical point, which has zero Riemannian gradient. The Riemannian gradient on the spheres is defined as

$$\texttt{grad}(V)_i = (I - v_i v_i^T)\tilde{g}_i, \forall i = 1, \ldots, n.$$

where $\tilde{g}_i = Vc_i$ is the gradient for coordinate block $i$. Then we have

$$\|\texttt{grad}(V)_i\|^2 = \|g_i\|^2 - (v_i^T g_i)^2 \leq 2\|g_i\|(\|g_i\| + v_i^T g_i) = \|g_i\| \cdot 2g_i^T(v_i - \hat{v}_i),$$

where the inequality is from $(\|g_i\| + v_i^T g_i)^2 \geq 0$. Note that the RHS term is the function decrease in Lemma A.1. Align the above term to a specific iterate $V^r$ by taking out the difference $v_i^r - \hat{v}_i^r$ in $g_i$, and taking limit to the inequality, we have

$$\lim_{r\to\infty} \|\texttt{grad}(V^r)\|^2 \leq \lim_{r\to\infty} \|g_i\| \cdot 2g_i^T(v_i^r - \hat{v}_i^r) + \sum_i O(\|v_i^r - \hat{v}_i^r\|) = 0,$$

That is, the Riemannian gradient converges to zero in the limit. The step-sized version follows from the same argument. □

## A.2 Proof of Theorem 3.3: Local Linear convergence

In the following lemma, we use the asymptotic analysis inspired by [55, Theorem 4], while removing its assumption by identifying the inherent Rayleigh quotient problem.

**Lemma A.2.** *Define $\hat{y}_i = v_i^T V c_i$, $\forall i = 1, \ldots, n$. Let $\hat{S} = C + D_{\hat{y}}$, and let $\bar{f}$ be the objective value of the nearest first-order critical point. Then, there is a neighborhood around the critical point such that*

$$\|V\hat{S}\|^2 \geq \frac{\kappa}{2}(f(V) - \bar{f})$$

*for $V$ with $f(V) \geq \bar{f}$ with a constant $\kappa > 0$. That is, $\|VS\|^2$ is a local error bound.*

*Proof.* Let $\bar{V}$ be the nearest first-order critical point to $V$. Consider the geodesic on the spheres around $\bar{V}$ w.r.t. a unit tangent direction $U$ such that $u_i^T \bar{v}_i = 0$ and $\|U\| = 1$, which is

$$v_i = \bar{v}_i \cos(\|u_i\|t) + \frac{u_i}{\|u_i\|} \sin(\|u_i\|t), \forall i = 1, \ldots, n.$$

Taking the Taylor expansion around $\bar{V}$ and using $\bar{V}\bar{s}_i = 0$, we have

$$v_i = \bar{v}_i + tu_i + O(t^2) \quad \text{and} \quad V\bar{s}_i = tU\bar{s}_i + O(t^2).$$

Substitute $\hat{S} = \bar{S} + D_{\hat{y}} - D_{\bar{y}}$ and applying the expansion and , there is

$$\begin{aligned}\|V\hat{S}\|^2 &= \sum_i \|V\bar{s}_i\|^2 - (v_i^T V\bar{s}_i)^2 = t^2\left(\sum_i \|U\bar{s}_i\|^2 - (\bar{v}_i^T U\bar{s}_i)^2\right) + O(t^3)\\ &= t^2 \sum_i \|(I - \bar{v}_i\bar{v}_i^T)U\bar{s}_i\|^2 + O(t^3).\end{aligned}$$

In the other hand, applying the expansion to $f(V) - \bar{f}$, we have

$$f(V) - \bar{f} = \mathrm{tr}(V^T V\bar{S}) = t^2\,\mathrm{tr}(U^T U\bar{S}) + O(t^3).$$

Note that $\mathrm{tr}(U^T U\bar{S}) \geq 0$, otherwise it contradicts with $f(V) \geq \bar{f}$ when $t$ is small enough. Now consider the generalized Rayleigh quotient problem relating the $t^2$ terms in $\|V\hat{S}\|^2$ and $f(V) - \bar{f}$.

$$\kappa := \inf_U \ \frac{\sum_i \|(I - \bar{v}_i\bar{v}_i^T)U\bar{s}_i\|^2}{\mathrm{tr}(U^T U\bar{S})}, \quad \text{s.t. } \mathrm{tr}(U^T U\bar{S}) \geq 0, \ \ u_i^T\bar{v}_i = 0, \ \forall i. \tag{A.2}$$

The numerator equals $\mathrm{vect}(U)^T M\,\mathrm{vect}(U)$ for an SPSD matrix $M$ because it's a sum-of-squares. Further,

$$(I - \bar{v}_i\bar{v}_i^T)U\bar{s}_i = 0 \implies \exists\theta,\ U\bar{s}_i = \theta v_i \implies u_i^T U\bar{s}_i = \theta u_i^T\bar{v}_i = 0.$$

Thus, $\mathrm{tr}(U^T U\bar{S})$ contains all the null of $\sum_i \|(I - \bar{v}_i\bar{v}_i^T)U\bar{s}_i\|^2$. So $\kappa \geq \sigma_{\text{min-nz}}(M)/\sigma_{\max}(\bar{S}) > 0$ because $\mathrm{tr}(U^T U\bar{S}) \geq 0$. Together, there is

$$\|V\hat{S}\|^2 = \frac{\sum_i \|(I - \bar{v}_i\bar{v}_i^T)U\bar{s}_i\|^2}{\mathrm{tr}(U^T U\bar{S})}(f(V) - \bar{f}) + O(t^3) \geq \frac{\kappa}{2}(f(V) - \bar{f})$$

in the neighborhood of $\bar{V}$ when $t$ is small enough. □

**Lemma A.3.** *Let $S = C + D_y$ for arbitrary $y$ and $\hat{S} = C + D_{\hat{y}}$ with $\hat{y}_i = -v_i^T V c_i$. Then $\|VS\|^2 \geq \|V\hat{S}\|^2$.*

*Proof.* Note that

$$v_i^T V\hat{s}_i = v_i^T(Vc_i - (v_i^T Vc_i)v_i) = 0.$$

Using this property,

$$\|VS\|^2 = \|V\hat{S} + V(D_y - D_{\hat{y}})\|^2 = \|V\hat{S}\|^2 + \|D_y - D_{\hat{y}}\|^2 \geq \|V\hat{S}\|^2.$$

Thus, the result holds. □

**Lemma A.4.** *Under Assumption 3.2, the Mixing method $M : \mathbb{R}^{k\times n} \to \mathbb{R}^{k\times n}$ satisfies*

$$\|V - M(V)\|^2 \geq \frac{1}{y_{\max}^2}\|V\hat{S}\|^2. \tag{A.3}$$

**Proof** Let $S = C + D_y$ with $y = y(V)$. Under the notation in (3.5),

$$V - M(V) = V(L^T + D_y)(L^T + D_y)^{-1} + VL(L^T + D_y)^{-1} = VS(L^T + D_y)^{-1}.$$

For simplicity, let $R := (L^T + D_y)^{-1}$. Then

$$\|V - M(V)\|^2 = \|VSR\|^2 \geq \sigma^2_{\text{min-nz}}(R)\|VS\|^2 \geq \sigma^2_{\text{min-nz}}(R)\|V\hat{S}\|^2. \tag{A.4}$$

The last inequality follows from Lemma A.3. The result holds with $\sigma^2_{\text{min-nz}}(R) = 1/y^2_{\max}$.

**Lemma A.5.** *The Mixing method $M_\theta : \mathbb{R}^{k\times n} \to \mathbb{R}^{k\times n}$ with step size $\theta$ satisfies*

$$\|V - M_\theta(V)\|^2 \geq \frac{\theta^2}{y^2_{\max}}\|V\hat{S}\|^2. \tag{A.5}$$

**Proof** Let $S = C + \frac{1}{\theta}(D_y - I)$ because the normalizer $y_i = \|v_i - \theta g_i\|$. By the derivation in (3.6), we have

$$V - M_\theta(V) = V - V(I_n - \theta L)(\theta L^T + D_y)^{-1} = V(D_y - I_n + \theta C)(\theta L^T + D_y)^{-1} = \theta VS(\theta L^T + D_y)^{-1}.$$

Following the same analysis in Lemma A.4, the result holds.

**Proof of Theorem 3.3**

*Proof.* By Lemma A.2 and Lemma A.4, there is a neighborhood around $\bar{V}$ such that

$$\|V - M(V)\|^2 \geq \frac{\kappa}{2y^2_{\max}}(f(V) - \bar{f}).$$

Together with Lemma A.1 (under Assumption 3.2),

$$f(V) - f(M(V)) \geq \frac{y_{\text{min-nz}}\kappa}{2y^2_{\max}}(f(V) - f^*) \implies (1 - \frac{y_{\text{min-nz}}\kappa}{2y^2_{\max}})(f(V) - f^*) \geq f(M(V)) - \bar{f}.$$

That is, the Mixing method $M$ converges R-linearly to the $\bar{f}$ in the neighborhood of the critical point. Moreover, we know the method always reaches the neighborhood by Theorem 3.1. The same local linear convergence follows for the Mixing method $M_\theta$ with step size $\theta \in (0, \frac{1}{\max_i \|c_i\|_1})$ from Lemma A.9 (no assumption) and Lemma A.5.

Further, because $f(V) - \bar{f}$ is converging to zero exponentially, there is a $\delta \in (0, 1)$ such that $f(V^r) - \bar{f} = O(\delta^r)$ for all large enough $r$. Consequently, $f(V^r) - f(M(V^r)) = O(\delta^r)$ and $\|V - M(V)\|^2 = O(\delta^r)$ follows, so we have

$$\|V^r - \bar{V}\| \leq \sum_{t=r}^{\infty}\|V^t - V^{t+1}\| = O(\sum_{t=r}^{\infty}\delta^{t/2}) = O(\delta^{r/2}/(1 - \sqrt{\delta})).$$

That is, the solution converges to a critical point Q-linearly. □

## A.3 Proof of Lemma 3.10: Divergence of Gauss-Seidel methods

*Proof.* Because the dynamics of the Gauss-Seidel method (GS) on the system

$$\min_{x\in\mathbb{R}^n} f(x), \quad \text{where } f(x) \equiv x^T S x,$$

has the same Jacobian as $J_{GS}$, proving $\rho(J_{GS}) > 1$ is equivalent to proving the "linear divergence" of the Gauss-Seidel method, which cyclically optimizes each coordinate of $x \in \mathbb{R}^n$. Further, since $S \not\succeq 0$, there is an eigenvector $q \in \mathbb{R}^n$ of $S$ such that $q^T S q < 0$.

Consider the sequence $\{x_r\}_{r=0,1,\ldots}$ generated by the GS. That is, $x_r = (J_{GS})^r x_0,\ \forall r > 0$, where $(J_{GS})^r$ is $J_{GS}$ to the $r$-th power. Let the initial solution of the system be $x_0 = q$ so that $f(x_0) < 0$. Because the Gauss-Seidel method is greedy in every coordinate updates, it is monotonically deceasing in the function value. Thus, there are only two cases for the sequence of function values: 1) the function value converges below zero; 2) the function value goes to negative infinity.

Denote $z_i^r$ the $x_r$ before updating the $i$-th coordinate and let $z_1^r = x_r$ and $z_{n+1}^r = x_{r+1}$. This way, only the $i$-th coordinate between $z_i^r$ and $z_{i+1}^r$ is changed and the inner cyclic updates can be flattened as

$$x_r = z_1^r \to z_2^r \to \ldots \to z_n^r \to z_{n+1}^r = x_{r+1}. \tag{A.6}$$

**1) When the function value converges.** The monotonic decreasing property of GS implies that the function difference converges to zero. By the same analysis in Lemma A.1,[1] we have

$$f(x_r) - f(x_{r+1}) = \sum_{i=1}^{n} y_i \|z_i^r - z_{i+1}^r\|^2.$$

Thus, the flattened sequence $\{z_i^r\}$ convergences, which implies $\{x^r\}$ also converges. Let $\bar{x}$ be the limit of the sequence $\{x^r\}$. Being a limit of the GS sequence means that $\bar{x}$ is a fixed point of GS, which implies $S\bar{x} = 0$ and $f(\bar{X}) = \bar{x}^T S \bar{x} = 0$. This contradicts with the monotonic decreasing property of GS and the fact that $f(x_0) < 0$.

**2) When the function value $x_r^T S x_r$ goes to negative infinity.** Because the spectrum of $S$ is bounded, we know that $\|x_r\|$ also goes to infinity. For simplicity, we focus on the $r$-th iterate and write $z_i^r$ as $z_i$. From the GS, $z_{i,i}$ is updated to $z_{i+1,i} = \frac{-1}{y_i}\sum_j c_{ij} z_{j,i}$, and we have

$$f(z_i) - f(z_{i+1}) = y_i \|z_i - z_{i+1}\|^2 = y_i (z_{i,i} + \frac{1}{y_i}(\sum_j c_{ij} z_{j,i}))^2 = |e_i^T S z_i|^2 / y_i, \tag{A.7}$$

where $e_i$ is the $i$-th coordinate vector and the first equality is from $f(x) = x^T S x$. Then we have the following claim from Lee et al. [91, cliam 1].

[1] We can obtain the result by fixing $y$ in Lemma A.1 to be a constant and let $V \in \mathbb{R}^{1\times n}$. The result can also be obtained by examining the coordinate updates of GS, which is already known in the literature.

**Claim A.6.** *Assume $x_r$ be in the range of $S$. There exists an index $j$ such that $\frac{1}{y_j}|e_j^T S z_j| \geq \omega \|z_j\|$ for some global constant $\omega > 0$ that only depends on $S$ and $n$.*

The full proof of the claim is listed after this lemma for completeness. To fulfill the assumption, we can decompose $x_r = x_r^{\parallel} + x_r^{\perp}$, where $x_r^{\parallel}$ is in the range of $S$ and $x_r^{\perp}$ is in the null of $S$. Consider the flattened inner cyclic update $z_i^{\parallel}$ like (A.6) but starting from $x_r^{\parallel}$ such that[2]

$$x_r^{\parallel} = z_1^{\parallel} \to z_2^{\parallel} \to \ldots \to z_n^{\parallel} \to z_{n+1}^{\parallel}.$$

Because $J_{GS}$ map the null space of $S$ to itself,[3]

$$f(x_r) - f(x_{r+1}) = f(x_r^{\parallel}) - f(J_{GS}(x_r^{\parallel} + x_r^{\perp})) = f(z_1^{\parallel}) - f(z_{n+1}^{\parallel} + x_r^{\perp}) = f(z_1^{\parallel}) - f(z_{n+1}^{\parallel}).$$

Further, because GS is coordinate-wise monotonic decreasing and the function decrease of a coordinate update is smaller than the whole cyclic update, by above equality and (A.7) we have

$$f(x_r) - f(x_{r+1}) = f(z_1^{\parallel}) - f(z_{n+1}^{\parallel}) \geq \frac{(e_j^T S z_j^{\parallel})^2}{y_j} \geq y_j \omega^2 \|z_j^{\parallel}\|^2.$$

The last inequality is from Claim A.6. Thus,

$$f(x_{r+1}) \leq f(x_r) - y_j \omega^2 \|z_j^{\parallel}\|^2 \tag{A.8}$$

Further, because $\|z_j^{\parallel}\|^2 \geq |z_j^{\parallel T} S z_j^{\parallel}|/\rho(S)$ and $z_j^{\parallel T} S z_j^{\parallel} = f(z_j^{\parallel}) \leq f(x_r^{\parallel}) = f(x_r) \leq f(x_0) < 0$,

$$f(x_r) - y_j \omega^2 \|z_j^{\parallel}\|^2 \leq f(x_r) + \frac{y_j \omega^2}{\rho(S)} z_j^{\parallel T} S z_j^{\parallel} \leq (1 + \frac{y_{min} \omega^2}{\rho(S)}) f(x_r). \tag{A.9}$$

Combining (A.8) and (A.9), we obtain the exponential divergence to negative infinity

$$f(x_{r+1}) \leq (1 + \frac{y_{min} \omega^2}{\rho(S)}) f(x_r) \leq (1 + \frac{y_{min} \omega^2}{\rho(S)})^{r+1} f(x_0), \quad \forall r \geq 0. \tag{A.10}$$

The last inequality is from applying the first inequality recursively. Because $S \succeq \sigma_{\min}(S) I_n$ and $\sigma_{min}(S) < 0$,

$$f(x_r) = ((J_{GS})^r x_0)^T S((J_{GS})^r x_0) \geq \sigma_{min}(S) \|(J_{GS})^r x_0\|^2 \geq \sigma_{min}(S) \|(J_{GS})^r\|^2 \|x_0\|^2. \tag{A.11}$$

Combining (A.10) and (A.11), we have

$$\sigma_{\min}(S) \|(J_{GS})^r\|^2 \|x_0\|^2 \leq \left(1 + \frac{y_{\min} \omega^2}{\rho(S)}\right)^r f(x_0), \quad \forall r > 0.$$

Applying Gelfand's theorem for spectral radius, we conclude that

$$\rho(J_{GS}) = \lim_{r \to \infty} \|(J_{GS})^r\|^{1/r} \geq \sqrt{1 + \frac{y_{min} \omega^2}{\rho(S)}},$$

which means that the spectral radius $\rho(J_{GS})$ is strictly larger than 1. □

[2]Note that only $x_r$ is decomposed to $x_r^{\parallel}$ in range$(S)$ and $x_r^{\perp}$ in null$(S)$. Symbols $z_i^{\parallel}$ are the GS iterates generated from $x_r^{\parallel}$ and might not be in the range of $S$.

[3]Consider $p$ such that $Sp = 0$. Then $(J_{GS})p = -(L + y)^{-1} L^T p = -(L + y)^{-1} S p + p = p$.

**Proof of the Claim A.6 in the above lemma.** Note that the following proof is essentially the same with Lee et al. [91, cliam 1], where their $\alpha$ is our $\frac{1}{y_i}$ and their $y_t$ is our $x_r$. The only difference here is that we prove the result for the exact Gauss-Seidel method, and they prove the result for the coordinate gradient descent method with a step size. The proof is listed here for completeness.

*Proof.* We will prove by contradiction. Assume that

$$\frac{1}{y_j}|e_j^T S z_j| < \omega \|z_j\| \text{ for all } j = 1 \dots n \text{ for certain } \omega. \tag{A.12}$$

Now we show the following result by induction, that for $j = 2 \dots n+1$,

$$\|x_r - z_j\| < 2(j-1)\omega \|x_r\|. \tag{A.13}$$

Remember from (A.7) we have

$$y_j \|z_j - z_{j+1}\|^2 = |e_j^T S z_j|^2 / y_j. \tag{A.14}$$

For $j = 2$, we have the induction basis for (A.13) from the above equality and (A.12), that

$$\|x_r - z_2\| = \|z_1 - z_2\| = \frac{1}{y_1}|e_1^T S z_1| < \omega \|z_1\| = \omega \|x_r\| < 2\omega \|x_r\|,$$

and accordingly $\|z_2\| \le \|z_2 - z_1\| + \|z_1\| < (1+2\omega)\|x_r\|$. Now we do the induction. Suppose the hypothesis (A.13) holds for a $j$. This implies

$$\|z_j\| \le \|z_j - x_r\| + \|x_r\| < (1 + 2(j-1)\omega)\|x_r\|. \tag{A.15}$$

Then at $j+1$,

$$\begin{aligned}
\|x_r - z_{j+1}\| &\le \|x_r - z_j\| + \|z_j - z_{j+1}\| && \text{by the triangular inequality} \\
&< 2(j-1)\omega\|x_r\| + \frac{1}{y_j}|e_j^T S z_j| && \text{by hypothesis (A.13) at } j \text{ and (A.14)} \\
&< 2(j-1)\omega\|x_r\| + \omega\|z_j\| && \text{by assumption (A.12)} \\
&< 2(j-1)\omega\|x_r\| + \omega(1 + 2(j-1)\omega)\|x_r\| && \text{by (A.15)} \\
&\le 2j\omega\|x_r\|,
\end{aligned}$$

where the last inequality holds from picking $\omega \in (0, \frac{1}{2n})$ so that $\omega(1 + 2(j-1)\omega - 2) < 0$. Thus, the induction on (A.13) holds. With the result (A.13), for $j = 2 \dots n$ we have

$$\begin{aligned}
\frac{1}{y_{max}}|e_j^T S x_r| &\le \frac{1}{y_j}|e_j^T S x_r| && \text{by } y_{max} \ge y_j \\
&\le \frac{1}{y_j}(|e_j^T S z_j| + |e_j^T S(x_r - z_j)|) && \text{by the triangular inequality} \\
&< \omega\|z_j\| + \frac{1}{y_j}\|Se_j\|\|x_r - z_j\| && \text{by (A.12) and Cauchy inequality} \\
&< \omega(1 + 2(j-1)\omega)\|x_r\| + \frac{1}{y_j}2(j-1)\omega\|Se_j\|\|x_r\| && \text{by (A.15) and (A.13)} \\
&\le \omega(1 + 2n\omega + 2n\frac{1}{y_{min}}\rho(S))\|x_r\|, && \text{(A.16)}
\end{aligned}$$

where the last inequality is from $\|Se_j\| \leq \|S\|\|e_j\| \leq \rho(S)$, $y_{\min} \leq y_j$, and $j \leq n+1$. Note that the result of (A.16) for $j = 1$ also holds because (A.12) and $x_r = z_1$. Summing the square of (A.16) over $j = 1 \ldots n$ and put it in a square root, we have

$$\sqrt{n}\omega(1 + 2n\omega + 2n\frac{\rho(S)}{y_{min}})\|x_r\| > \frac{1}{y_{max}}\|Sx_r\| \geq \kappa_{\text{min-nz}}(S)\|x_r\|,$$

where $\kappa_{\text{min-nz}}(S) = \sqrt{\sigma_{\text{min-nz}}(S^TS)} > 0$ is the minimum nonzero singular value of $S$ and the last inequality holds because $\kappa_{\text{min-nz}}(S)\|x_r\| \leq \|Sx_r\|$ from $x_r \in \text{range}(S)$. Cancelling $\|x_r\|$ from both sides of the above inequality, the left-hand side goes to 0 when $\omega \to 0$ but the right-hand side stays constant. Thus, picking small enough $\omega$ such that $\kappa_{\text{min-nz}}(S) \geq y_{max}\sqrt{n}\omega(1+2n\omega+2n\frac{\rho(S)}{y_{min}})$ leads to a contradiction.[4] So the claim holds. □

# A.4 Proof of Theorem 3.5: Global convergence with a step size

**Lemma A.7.** *The Mixing method $M_\theta$ with a step size $\theta \in (0, \frac{1}{\max_i \|c_i\|_1})$ never degenerates. That is, there is a constant $\delta \in (0, 1)$ such that*

$$\|\theta V c_i\| \leq 1 - \delta < 1 \quad \textit{and} \quad \|v_i - \theta V c_i\| \geq \delta > 0.$$

*Proof.* Taking a constant $\theta \in (0, \frac{1}{\max_i \|c_i\|_1})$ is equivalent to taking $\theta = \frac{1-\delta}{\max_i \|c_i\|_1}$ for a constant $\delta \in (0, 1)$. From the triangular inequality,

$$\|Vc_i\| = \|\sum_j c_{ij}v_j\| \leq \sum_j |c_{ij}|\|v_j\| = \|c_i\|_1.$$

So we have $\|\theta V c_i\| \leq 1 - \delta < 1$. The second result follows from $\|v_i - \theta V c_i\| \geq 1 - \|\theta V c_i\|$. □

**Lemma A.8.** *The Mixing method $M_\theta$ with a step size $\theta \in (0, \frac{1}{\max_i \|c_i\|_1})$ is a diffeomorphism.*

*Proof.* Note that $M_\theta$ can be decomposed to

$$M_\theta(V) = \Phi^n(\Phi^{n-1}(\ldots\Phi^1(V))),$$

where the column update $\Phi^i(V) : \mathbb{R}^{k\times n} \to \mathbb{R}^{k\times n}$ is defined as

$$(\Phi^i(V))_{s=1\ldots n} = \begin{cases} \frac{v_i - \theta V c_i}{\|v_i - \theta V c_i\|} & \text{if } s = i \\ v_s & \text{otherwise.} \end{cases}$$

Thus, if we can prove that $\Phi^i(V)$ is a diffeomorphism for $i = 1 \ldots n$, then $M_\theta$ is a diffeomorphism because compositions of diffeomorphisms are still diffeomorphism. Specifically, because all

[4] Note that the choice of $\omega$ only depends on $n$ and $S$.

variables $v_j$ except for $v_i$ are given and stay the same, proving diffeomorphism of $\Phi^i(V)$ is equivalent to proving the diffeomorphism of the projective mapping $\phi : \mathbb{R}^n \to \mathbb{R}^n$

$$\phi(v) = \frac{v - g}{\|v - g\|},$$

where $g = \theta V c_i$ is known. Let $\phi(v) = z$. We claim the inverse function $\phi^{-1}(z)$ is

$$\phi^{-1}(z) = \alpha z + g, \quad \text{where } \alpha = -z^T g + \sqrt{(z^T g)^2 + 1 - \|g\|^2}.$$

The square root is valid because of Lemma A.7. We prove the claim by validation. First,

$$\phi^{-1}(\phi(v)) = \phi^{-1}\left(\frac{v - g}{\|v - g\|}\right).$$

By using the property that $\|v\| = 1$, the $\alpha$ for the above function is

$$\begin{aligned}
\alpha &= \frac{-(v-g)^T g}{\|v-g\|} + \sqrt{\left(\frac{(v-g)^T g}{\|v-g\|}\right)^2 + 1 - \|g\|^2} \\
&= \frac{1}{\|v-g\|}\left(-(v-g)^T g + \sqrt{((v-g)^T g)^2 + \|v-g\|^2(1-\|g\|^2)}\right) \\
&= \frac{1}{\|v-g\|}\left(-v^T g + \|g\|^2 + 1 - v^T g\right) \\
&= \|v - g\|.
\end{aligned}$$

Thus, $\phi^{-1}(\phi(v)) = \|v-g\|\frac{v-g}{\|v-g\|} + g = v$ is indeed the inverse function. The diffeomorphism follows from the smoothness of $\phi(v)$ and $\phi^{-1}(z)$ when $\|v - g\| \geq \delta > 0$ by Lemma A.7. □

**Lemma A.9.** *For the Mixing method $M_\theta$ with step size $\theta \in (0, \frac{1}{\max_i \|c_i\|})$, let $\hat{V} = M_\theta(V)$. Following the notation in* (3.4) *and* (3.6)*, we have*

$$f(V) - f(\hat{V}) = \sum_i \frac{1 + y_i}{\theta}\|v_i - \hat{v}_i\|^2,$$

*where $y_i = \|v_i - \theta g_i\|$ and $g_i = \sum_{j<i} c_{ij}\hat{v}_j + \sum_{j>i} c_{ij} v_j$.*

*Proof.* Consider the inner iteration of the Mixing method with a step size. With the same analysis to Lemma A.1, the function value before updating the variable $v_i$ is $2g_i^T v_i$, and the function difference after updating $v_i$ to $\hat{v}_i = (v_i - \theta g_i)/y_i$ is

$$\begin{aligned}
2g_i^T(v_i - \hat{v}_i) &= 2(g_i + \frac{v_i - \theta g_i}{\theta})^T(v_i - \hat{v}_i) - 2(\frac{v_i - \theta g_i}{\theta})^T(v_i - \hat{v}_i) \\
&= 2\frac{1}{\theta}v_i^T(v_i - \hat{v}_i) - 2\frac{y_i}{\theta}\hat{v}_i^T(v_i - \hat{v}_i) \\
&= \frac{1 + y_i}{\theta}2(1 - v_i^T\hat{v}_i) = \frac{1 + y_i}{\theta}\|v_i - \hat{v}_i\|^2.
\end{aligned}$$

Thus, the result holds from summing the above equation over $i = 1 \ldots n$. □

**Proof of Theorem 3.5**

*Proof.* Similar to Lemma 3.7, the Jacobian of the Mixing method $M_\theta$ with step size $\theta$ is

$$J(V) = (D_y \otimes I_k - \theta P L \otimes I_k)^{-1} P \theta L^T \otimes I_k, \text{ where } P = \mathrm{diag}(P_1, \ldots, P_n) \text{ and } P_i = I - \hat{v}_i \hat{v}_i^T.$$

By Lemma 3.9, the spectral radius of $J$ at a critical point is lower bounded by $J_{CGD} = (D_y - \theta L)^{-1} \theta L^T$, which is the Jacobian of the coordinate gradient descent method (CGD) on a linear system $S = C + D_y$. Because the CGD admits a Jacobian with $\rho(J_{CGD}) > 1$ when $S \not\succeq 0$ [91, Proposition 5], it follows that all non-optimal critical points are unstable fixed points for $M_\theta$. Further, since the Mixing method $M_\theta$ with step size $\theta$ is a diffeomorphism by Lemma A.8, we can apply the center-stable manifold theorem [130, Theorem III.7] to the mapping. To be specific, the corollary of center-stable manifold theorem in Lee et al. [91, Theorem 2][5] implies that the Mixing method $M_\theta$ escapes all non-optimal critical point almost surely under random initialization.[6] Further, because $M_\theta$ is monotonically decreasing by Lemma A.9 and the objective value is lower bounded, $M_\theta$ converges to a first-order critical points (with the same analysis to Theorem 3.1). In conclusion, the almost surely divergence from the non-optimal critical points and the convergence to a critical point imply that the method converges to a global optimal solution almost surely. □

## A.5 Proof of Lemma 3.11: Rank Deficiency in Critical Points

The proof is a specialized version of [32, Lemma 9] for the MAXCUT SDP, in which their $Y$ is our $V$ and their $\mu(V)$ is our $y$. We list it here for completeness.

*Proof.* Let $V$ be a first-order critical point of problem (3.2), which means that there is a corresponding $y_i = \|V c_i\|,\ i = 1 \ldots n$ such that

$$VS = 0, \quad \text{where } S \equiv C + D_y.$$

This implies

$$\text{rank}(V) \leq \text{null}(C + D_y) \leq \max_\nu \text{null}(C + D_\nu).$$

Note that the right-hand side is independent of $V$, so we can use it to bound the rank of all critical $V$. Let $\nu$ be a solution of the right-hand side, $M \equiv C + D_\nu$, and $\text{null}(M) \equiv \ell$. Writing $C = M - D_\nu$, we have

$$C \in \mathcal{N}_\ell + im(D),$$

[5]Note that Lee et al. [91, Lemma 1] use only the property of diffeomorphism, so their assumption on the non-singular Jacobian is not necessary. Actually, non-singular Jacobian is a sufficient condition for the existence of one-to-one mapping but not the necessary condition.

[6]Note that the critical points here is non-isolated because they are invariant to rotations in $R^k$. While Lee et al. [91, Theorem 2] suffices for our result, interested reader can also refer to [117] on how they use the Lindelöf lemma to solve the non-isolation issue.

in which the $+$ denotes the set-sum, $im(D)$ denotes the image of all diagonal matrices of size $n$, and $\mathcal{N}_\ell$ denotes the set of symmetric matrices of size $n$ with nullity $\ell$. Because of the symmetricity of $\mathcal{N}_\ell$,

$$\dim(\mathcal{N}_\ell) = \frac{n(n+1)}{2} - \frac{\ell(\ell+1)}{2}.$$

Further, because $\text{rank}(V) \leq k$, we can assume that $\ell \geq k$. Union all possible $\ell$,

$$C \in \bigcup_{\ell=k\ldots n} \mathcal{N}_\ell + im(D).$$

Note that the right-hand side is now independent of $C$. Because the dimension of a finite union is at most the maximal dimension, and the dimension of a finite set sum is at most the sum of set dimensions,

$$\dim\left(\bigcup_{\ell\in k\ldots n} \mathcal{N}_\ell + im(D)\right) \leq \dim(\mathcal{N}_k + im(D)) \leq \frac{n(n+1)}{2} - \frac{k(k+1)}{2} + \text{rank}(D). \quad \text{(A.17)}$$

We know that $\text{rank}(D) = n$ because the space of diagonal matrix has $n$ free dimensions. Because the symmetric matrix $C$ lives in the space $\frac{n(n+1)}{2}$, almost no $C$ satisfies the right-hand side of (A.17) if we take large enough $k$ so that

$$\frac{n(n+1)}{2} - \frac{k(k+1)}{2} + n < \frac{n(n+1)}{2}.$$

Thus, almost no $C$ has critical point of rank $k$ if $\frac{k(k+1)}{2} > n$, which means for almost all $C$, the critical point has at most rank $k-1$. □

# Appendix B

# Proofs and additional experimental results for MRF

## B.1 Proof of Equivalence of (4.8) and (4.9)

We state (4.8) and (4.9) again. Let the vectors $r_1, \ldots, r_k \in \mathbb{R}^n$ be fixed on the simplex. Then, (4.8) is stated as follows:

$$\max_{v_i \in \mathbb{R}^n,\ \|v_i\|_2 = 1\ \forall i \in [n]} \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} v_i^T v_j + \sum_{i=1}^{n} v_i^T \sum_{l=1}^{k} \hat{h}_i^{(l)} r_l. \tag{4.8}$$

Let $H \in \mathbb{R}^{n \times k}$ such that $H_{ij} = \hat{h}_i^{(j)}$. Define the block matrix $C \in \mathbb{R}^{(k+n)\times(k+n)}$ such that:

$$C = \begin{bmatrix} 0 & \frac{1}{2} \cdot H^T \\ \frac{1}{2} \cdot H & A \end{bmatrix}.$$

Then, (4.9) is stated as follows:

$$\begin{aligned} \max_{Y \succeq 0} \ & Y \cdot C \\ \text{subject to } \ & Y_{ii} = 1\ \forall i \in [n+k] \\ & Y_{ij} = -\frac{1}{k-1}\ \ \forall i \in [k],\ i < j \leq k. \end{aligned} \tag{4.9}$$

We will show that the optimal solutions to both these optimization problems are equal. Consider any $v_1, \ldots, v_n \in \mathbb{R}^n$ in the feasible set of (4.8). Then, corresponding to these $v_1, \ldots, v_n$, consider the matrix $Y \in \mathbb{R}^{(k+n)\times(k+n)}$ defined as follows:

$$Y = \begin{bmatrix} r_1^T \\ \vdots \\ r_k^T \\ v_1^T \\ \vdots \\ v_n \end{bmatrix} \begin{bmatrix} r_1 & \ldots & r_k & v_1 & \ldots & v_n \end{bmatrix}.$$

Clearly, $Y \succeq 0$. Further, since $r_1, \ldots, r_k$ are on the simplex and $v_1, \ldots, v_n$ are in the feasible set of (4.8), this matrix $Y$ satisfies the constraints in (4.9). Thus, $Y$ lies in the feasible set of (4.9). Also, because of the way in which the block matrix $C$ is defined, we can verify that:

$$Y \cdot C = \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij} v_i^T v_j + \sum_{i=1}^{n} v_i^T \sum_{l=1}^{k} \hat{h}_i^{(l)} r_l.$$

Thus, for any $v_1, \ldots, v_n$ in the feasible set of (4.8), we have a corresponding $Y$ in the feasible set of (4.9) such that the criterion values match.

Now, consider any $Y$ in the feasible set of (4.9). Since $Y \succeq 0$, we can compute its Cholesky decomposition as $Y = U^T U$ for some $U \in \mathbb{R}^{(k+n)\times(k+n)}$. Denote the first $k$ columns in $U$ as $r'_1, \ldots, r'_k$ and the last $n$ columns of $U$ as $v'_1, \ldots, v'_n$. Then, since $Y$ satisfies the constraints in (4.9), we have that $\|v'_i\|_2 = 1$ for all $i \in [n]$. Also, we have that $r_i'^T r'_j = 1$ if $i = j$ and $r_i'^T r'_j = -\frac{1}{k-1}$ otherwise. Thus, the vectors $r'_1, \ldots, r'_k$ correspond to the vertices of a simplex in $\mathbb{R}^n$. Then, there exists a rotation matrix $\bar{R} \in \mathbb{R}^{n\times n}$ such that $\bar{R} r'_l = r_l$ for all $i \in [k]$ and $\bar{R}^T \bar{R} = I$. Then, consider the vectors $v_i = \bar{R} v'_i$ for $i \in [n]$. Since rotation matrices preserve norm, we have that $\|v_i\|_2 = 1$ for all $i \in [n]$. Thus, $v_1, \ldots, v_n$ lie in the feasible set of (4.8). Also, we have that:

$$\begin{aligned}
Y \cdot C &= U^T U \cdot C \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij} v_i'^T v_j + \sum_{i=1}^{n} v_i'^T \sum_{l=1}^{k} \hat{h}_i^{(l)} r'_l \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij} v_i'^T \bar{R}^T \bar{R} v_j + \sum_{i=1}^{n} v_i'^T \bar{R}^T \sum_{l=1}^{k} \hat{h}_i^{(l)} \bar{R} r'_l \qquad (\text{since } \bar{R}^T \bar{R} = I) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij} (\bar{R} v'_i)^T \bar{R} v_j + \sum_{i=1}^{n} (\bar{R} v'_i)^T \sum_{l=1}^{k} \hat{h}_i^{(l)} r_l \qquad (\text{since } \bar{R} r'_i = r_i) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij} v_i^T v_j + \sum_{i=1}^{n} v_i^T \sum_{l=1}^{k} \hat{h}_i^{(l)} r_l.
\end{aligned}$$

Thus, corresponding to any $Y$ in the feasible set of (4.9), we have found vectors $v_1, \ldots, v_n$ in the feasible set of (4.8) such that criterion values match.

Consequently, we have shown the range of criterion values in both optimization problems is the same, and hence the optimization problems have equivalent optimal solutions.

## B.2 Derivation of (4.11)

Let $z_1, \ldots, z_n \in \mathbb{R}^d$ such that $d = m \cdot k, m \in \mathbb{Z}$, and let $C = \frac{k}{k-1}\left(I_d - \frac{1}{k}\left(1_{k\times k} \otimes I_m\right)\right)$ where $1_{k\times k}$ is a matrix filled with 1s. Let $C = S^T S$ denote the Cholesky decomposition of $C$. Further, let us segment each $z_i$ into $k$ blocks such that $z_i^b \in \mathbb{R}^m$ denotes the $b^{th}$ block. Then, we state (4.11) again:

$$\max_{z_i \in \mathbb{R}^d \ \forall i \in [n]} \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij} v_i^T v_j + \sum_{i=1}^{n} v_i^T \sum_{l=1}^{k} \hat{h}_i^{(l)} r_l$$

$$\text{subject to} \quad z_i \geq 0, \quad \left\| \sum_{b=1}^{k} z_i^b \right\|_2^2 = 1, \quad v_i = S z_i \ \ \forall i \in [n]. \tag{4.11}$$

First, we show that with the parameterization above, $1 \geq v_i^T v_j \geq \frac{-1}{k-1}$, i.e. $v_i, v_j$ satisfy the pairwise constraints. Note that with the structure of $C$ as defined, we have that

$$\begin{aligned} v_i^T v_j &= z_i^T S^T S z_j \\ &= z_i^T C z_j \\ &= \frac{k}{k-1}\left[ z_i^T z_j - \frac{1}{k}\left(\sum_{b=1}^{k} z_i^b\right)^T \left(\sum_{b=1}^{k} z_j^b\right)\right]. \end{aligned}$$

Now, since $z_i \geq 0$ and since $\left\|\sum_{b=1}^{k} z_i^b\right\|_2^2 = 1$, we have that $\|z_i\|_2 \leq 1$. Thus, by the Cauchy-Schwartz inequality, we have that $0 \leq z_i^T z_j \leq 1$, and also that, $0 \leq \left(\sum_{b=1}^{k} z_i^b\right)^T \left(\sum_{b=1}^{k} z_j^b\right) \leq 1$. Thus, we have $v_i^T v_j \geq \frac{k}{k-1}\left(0 - \frac{1}{k}\right) = -\frac{1}{k-1}$. Further, note that

$$\begin{aligned} \|v_i\|_2^2 &= \frac{k}{k-1}\left( \|z_i\|_2^2 - \frac{1}{k}\left\| \sum_{b=1}^{k} z_i^b \right\|_2^2 \right) \\ &= \frac{k}{k-1}\left( \|z_i\|_2^2 - \frac{1}{k} \right) \leq 1. \end{aligned}$$

Thus, we have that $v_i^T v_j \leq \|v_i\|_2 \|v_j\|_2 \leq 1$, establishing both bounds. Next, note that we can set the appropriate $z_i^b$ in each $z_i$ to $e_1 \in \mathbb{R}^m$ (where $e_1$ is the first basis vector), and set all the other $z_i^{b'}$ to 0, and this allows $v_i = S z_i$ to be the required vector $r_l$ on the simplex corresponding to the optimal solution to the discrete problem (4.7). Thus, we have that the optimal solution to (4.11) is at least as large as $f^\star_{discrete}$.

Our goal then is to obtain candidates $z_1, \ldots, z_n$, such that the objective is at least as large as $f^\star_{discrete}$. Additionally, if we can further guarantee that $\|z_i\|_2 = 1$ for all $i$, we are done.

Let us write the objective in (4.11) in terms of $z_i$. This is

$$\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij} z_i^T C z_j + \sum_{i=1}^{n} z_i^T S^T \sum_{l=1}^{k} \hat{h}_i^{(l)} r_l.$$

Let us consider the terms in the objective involving a particular $z_i$. These are

$$z_i^T \underbrace{\left( 2 \sum_{j \neq i}^{n} A_{ij} C z_j + S^T \sum_{l=1}^{k} \hat{h}_i^{(l)} r_l \right)}_{g_i}.$$

Note that for every index $j$ within a block in $g_i$, across the $k$ blocks, there will definitely be at least one positive entry. This is because

$$\begin{aligned} 2 \sum_{j \neq i}^{n} A_{ij} C z_j + S^T \sum_{l=1}^{k} \hat{h}_i^{(l)} r_l &= 2 \sum_{j \neq i}^{n} A_{ij} C z_j + S^T \sum_{l=1}^{k} \hat{h}_i^{(l)} S e_l \\ &= 2 \sum_{j \neq i}^{n} A_{ij} C z_j + C \sum_{l=1}^{k} \hat{h}_i^{(l)} e_l \\ &= C \underbrace{\left( 2 \sum_{j \neq i}^{n} A_{ij} z_j + \sum_{l=1}^{k} \hat{h}_i^{(l)} e_l \right)}_{p} \\ &= Cp. \end{aligned}$$

and because of the nature of the matrix $C$, the entries at a particular index $j$ across the $k$ blocks in $Cp$ will each be of the form $x - \text{avg}(x)$. This fact will be useful later on.

We now consider updating each $z_i$ in a sequential manner as a block-coordinate update, just as in the original mixing method. In the following, we drop the subscript $i$ in $z_i$ and $g_i$ for convenience. Concretely, we aim to solve the problem

$$\begin{aligned} \min \quad & -g^T z \\ \text{subject to} \quad & z \geq 0; \ \left\| \sum_{b=1}^{k} z^b \right\|_2^2 = 1. \end{aligned} \tag{B.1}$$

Let us write the Lagrangian $L(z, \alpha, \lambda)$ for the above constrained optimization problem, for dual variables $\alpha \geq 0, \lambda$:

$$L(z, \alpha, \lambda) = -g^T z + \frac{\lambda}{2} \left( \left\| \sum_{b=1}^{k} z^b \right\|_2^2 - 1 \right) - \alpha^T z.$$

The KKT conditions are

$$\begin{aligned}
\text{Stationarity:} \quad & g_i^b + \alpha_i^b = \lambda \sum_{b=1}^{k} z_i^b \;\; \forall b \in [k], i \in [m] \\
\text{Complementary slackness:} \quad & \alpha_i^b z_i^b = 0 \;\; \forall b \in [k], i \in [m] \\
\text{Primal feasibility:} \quad & z_i^b \geq 0 \;\; \forall b \in [k], i \in [m] \\
& \left\| \sum_{b=1}^{k} z^b \right\|_2^2 = 1 \\
\text{Dual feasibility:} \quad & \alpha_i^b \geq 0 \;\; \forall b \in [k], i \in [m].
\end{aligned}$$

Note that now, $z_i^b$ refers to the $i^{th}$ entry in the $b^{th}$ block in $z$. Since the KKT conditions are always sufficient, if we are able to construct $z$ and $\alpha, \lambda$ that satisfy all the conditions above, $z$ and $\alpha, \lambda$ would be optimal primal and dual solutions to (B.1) respectively.

Towards this, let $(\cdot)_+$ denote the operation that thresholds the argument at 0, i.e.

$$(x)_+ = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

For any fixed index $i \in [m]$, let $b(i) = \arg\max_b g_i^b$ (if there are multiple, pick any). Consider the following assignment:

$$\begin{aligned}
\lambda &= \sqrt{\sum_{i=1}^{m} (g_i^{b(i)})_+^2} \\
z_i^{b(i)} &= \frac{(g_i^{b(i)})_+}{\lambda}, \quad \alpha_i^{b(i)} = \begin{cases} 0 & \text{if } g_i^{b(i)} > 0 \\ -g_i^{b(i)} & \text{otherwise} \end{cases} \\
z_i^b &= 0, \quad \alpha_i^b = -g_i^b + \lambda z_i^{b(i)} \;\; \text{for } b \neq b(i).
\end{aligned}$$

Note that $\lambda > 0$, since we argued above that there will be at least one entry that will be positive across the blocks. We will now verify that this assignment satisfies all the KKT conditions. First, note that $\sum_{b=1}^{k} z_i^b = z_i^{b(i)}$. Consider stationarity: for $b(i)$, if $g_i^{b(i)} > 0$,

$$g_i^{b(i)} + \alpha_i^{b(i)} = g_i^{b(i)} = (g_i^{b(i)})_+ = \lambda z_i^{b(i)}.$$

otherwise if $g_i^{b(i)} \leq 0$, $z_i^{b(i)} = 0$ and so

$$g_i^{b(i)} + \alpha_i^{b(i)} = g_i^{b(i)} - g_i^{b(i)} = 0 = \lambda z_i^{b(i)}.$$

For $b \neq b(i)$, by construction

$$g_i^b + \alpha_i^b = \lambda z_i^{b(i)}.$$

Next, we can observe that complementary slackness holds, since either one of $z_i^b$ or $\alpha_i^b$ is always 0. Next, we verify primal feasibility. We can observe that $z_i^b \geq 0$ for all $b$. Further,

$$\left\| \sum_{b=1}^{k} z^b \right\|_2^2 = \sum_{i=1}^{m} z_i^{b(i)2} = \frac{1}{\lambda^2} \sum_{i=1}^{m} (g_i^{b(i)})_+^2 = 1.$$

Finally, we verify dual feasibility. For $b(i)$, we have that

$$\alpha_i^{b(i)} = \begin{cases} 0 & \text{if } g_i^{b(i)} > 0 \\ -g_i^{b(i)} & \text{otherwise.} \end{cases}.$$

Either way, $\alpha_i^{b(i)} \geq 0$. For $b \neq b(i)$,

$$\alpha_i^b = -g_i^b + \lambda z_i^{b(i)} = -g_i^b + (g_i^{b(i)})_+ \geq 0.$$

Thus, we observe that the constructed $z$ and $\alpha, \lambda$ satisfy all the KKT conditions. Hence, $z$ (as constructed as above) is the optimal solution to (B.1). Algorithm 4.3 precisely updates each $z_i$ based on this constructed solution. The hope at the convergence of this routine is that we will have ended up with a solution $v_1, \ldots, v_n$ such that $f(v_1, \ldots, v_n) > f^\star_{discrete}$. Empirically, we always observe that this is the case. In fact, the solution at convergence is within $5\%$ of the true optimal solution of (4.11) itself. Thus, the approximation guarantees of Frieze et al. [59] go through for the rounded solution on $v_1, \ldots, v_n$ at convergence, assuming that the entries in $A$ are positive.

## B.3 Proof of Theorem 4.1

We have that

$$
\begin{aligned}
\mathbb{E}[\hat{Z}] &= \mathbb{E}_{X_{p_v}}\left[\mathbb{E}_{\cdot|X_{p_v}}[\hat{Z}]\right] \\
&= \mathbb{E}_{X_{p_v}}\left[\mathbb{E}_{\cdot|X_{p_v}}\left[\sum_{x\in X_{p_v}} \exp(f(x)) + \frac{1}{R}\sum_{x\in X_\Omega}\frac{\exp(f(x))}{q}\right]\right] \\
&= \mathbb{E}_{X_{p_v}}\left[\sum_{x\in X_{p_v}} \exp(f(x)) + \frac{1}{R}\mathbb{E}_{\cdot|X_{p_v}}\left[\sum_{x\in X_\Omega}\frac{\exp(f(x))}{q}\right]\right] \\
&= \mathbb{E}_{X_{p_v}}\left[\sum_{x\in X_{p_v}} \exp(f(x)) + \frac{1}{Rq}\sum_{x\in X_\Omega}\mathbb{E}_{\cdot|X_{p_v}}\left[\exp(f(x))\right]\right] \\
&= \mathbb{E}_{X_{p_v}}\left[\sum_{x\in X_{p_v}} \exp(f(x)) + \frac{1}{Rq}\sum_{x\in X_\Omega}\sum_{y\in\{[k]^n\setminus X_{p_v}\}} q\cdot\exp(f(y))\right] \\
&= \mathbb{E}_{X_{p_v}}\left[\sum_{x\in X_{p_v}} \exp(f(x)) + \frac{1}{R}\sum_{x\in X_\Omega}\sum_{y\in\{[k]^n\setminus X_{p_v}\}} \exp(f(y))\right] \\
&= \mathbb{E}_{X_{p_v}}\left[\sum_{x\in X_{p_v}} \exp(f(x)) + \frac{1}{R}\cdot R\cdot\sum_{y\in\{[k]^n\setminus X_{p_v}\}} \exp(f(y))\right] \\
&= \mathbb{E}_{X_{p_v}}\left[\sum_{x\in X_{p_v}} \exp(f(x)) + \sum_{y\in\{[k]^n\setminus X_{p_v}\}} \exp(f(y))\right] \\
&= \mathbb{E}_{X_{p_v}}\left[\sum_{x\in [k]^n} \exp(f(x))\right] \\
&= \mathbb{E}_{X_{p_v}}[Z] \\
&= Z.
\end{aligned}
$$

Thus, the estimate $\hat{Z}$ given by Algorithm 4.4 is unbiased.

## B.4 Pseudocode for AIS

Our implementation of AIS has 3 main parameters: the number of temperatures in the annealing chain (denoted $K$), the number of cycles of Gibbs sampling while transitioning from one temperature to another (denoted $num\_cycles$), and the number of samples used (denoted $num\_samples$). First, we define $K+1$ coefficients $0 = \beta_0 < \beta_1 < \cdots < \beta_K = 1$ . Then, given a general $k$-class MRF problem instance as defined in Sections 4.1, 4.2, let

$$f(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}\hat{\delta}(x_i, x_j) + \sum_{i=1}^{n}\sum_{l=1}^{k} \hat{h}_i^{(l)}\hat{\delta}(x_i, l).$$

Further, define functions $f_k$ as follows:

$$f_k(x) = \left(\frac{1}{k^n}\right)^{1-\beta_k} (\exp(f(x)))^{\beta_k}.$$

Also, let $p_0$ denote the uniform distribution on the discrete hypercube $[k]^n$. The complete pseudocode for our implementation of AIS is then provided below:

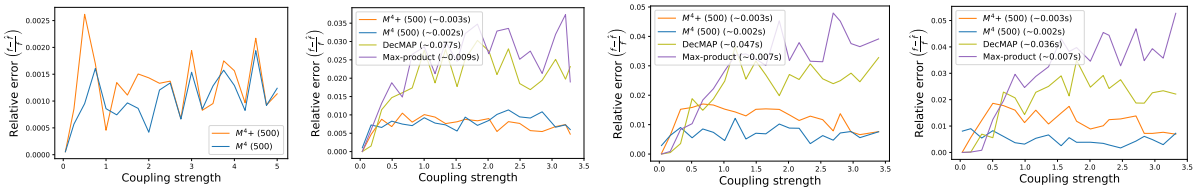
**Algorithm B.1** Annealed Importance Sampling

**procedure** GIBBSSAMPLING($x, \beta_k, num\_cycles$)
  Let $p(x) \propto (\exp(f(x)))^{\beta_k}$
  **for** $cycle = 1, 2 \ldots, num\_cycles$ **do**
    **for** $i = 1, 2, \ldots, n$ **do**
      $x_i \leftarrow$ Sample $p(x_i|x_{-i})$
    **end for**
  **end for**
  **return** $x$
**end procedure**

**procedure** AIS($K, num\_cycles, num\_samples$)
  **for** $i = 1, 2 \ldots, num\_samples$ **do**
    Sample $x \sim p_0$
    $w^{(i)} \leftarrow 1$
    **for** $k = 1, 2, \ldots, K$ **do**
      $w^{(i)} \leftarrow w^{(i)} \cdot \frac{f_k(x)}{f_{k-1}(x)}$
      $x \leftarrow$ GIBBSSAMPLING($x, \beta_k, num\_cycles$)
    **end for**
  **end for**
  **return** $Z = \frac{1}{num\_samples} \sum_{i=1}^{num\_samples} w^{(i)}$
**end procedure**

## B.5 Mode estimation comparisons

Here, we compare the mode estimates given by $M^4$ and $M^4$+ with max-product belief propagation and decimation algorithm given in libDAI [107] over complete graphs across a range of coupling strengths for $k = 2, 3, 4, 5$.

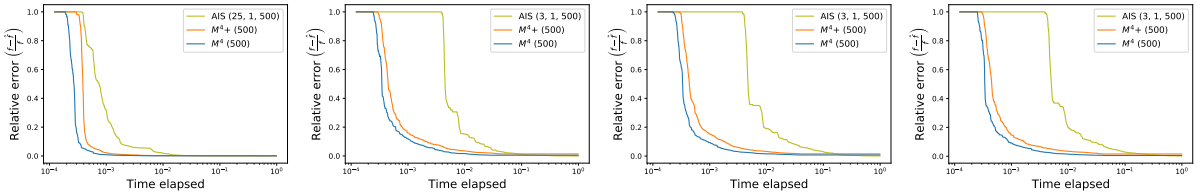



**(a)** $k = 2, n = 20$ **(b)** $k = 3, n = 10$ **(c)** $k = 4, n = 8$ **(d)** $k = 5, n = 7$

**Figure B.1:** Mode estimation comparison with max-product BP and decimation

We can observe that for both methods, the relative errors are very small ($\sim 0.018$ at worst) compared to the other methods, but $M^4$+ suffers a little for larger $k$.

Next, we show the results for the mode estimation task (timing comparison versus AIS) on complete graphs for $k = 2, 3, 4, 5$. The coupling matrices are fixed to have a coupling strength $CS(A) = 2.5$.




**(a)** $k = 2, n = 20$ **(b)** $k = 3, n = 10$ **(c)** $k = 4, n = 8$ **(d)** $k = 5, n = 7$

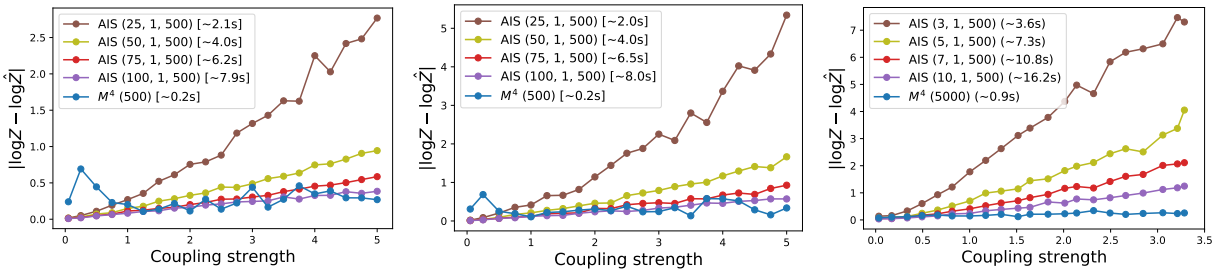
**Figure B.2:** Mode estimation comparison with AIS

We can observe that both $M^4$ and $M^4$+ are able to achieve an accurate estimate of the mode much quicker than AIS across different values of $k$.

## B.6 Performance of AIS with varying parameters

Here, we demonstrate how the performance of AIS is affected on separately varying the parameters $K$ and $num\_cycles$ (Algorithm B.1) in the partition function task. We consider similar problem instances described in Section 4.3 in the paper:

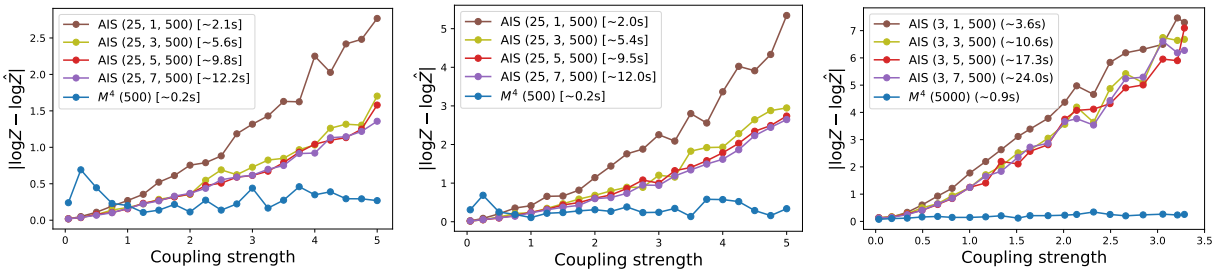
1. We fix $num\_cycles = 1$ and vary $K$. Figure B.3 shows the results. We can observe that increasing $K$ helps increase the accuracy of the estimate of $Z$, but also becomes very expensive w.r.t. time.




**(a)** Complete graph $k = 2, n = 20$ **(b)** ER graph $k = 2, n = 20$ **(c)** Complete graph $k = 3, n = 10$

**Figure B.3:** Variation of $K$ in AIS

2. Next, we fix $K$ and vary $num\_cycles$ in the Gibbs sampling step. Figure B.4 shows the results. We can observe that increasing $num\_cycles$ helps increase the accuracy of the estimate of $Z$ (although the effect is much less pronounced when compared to increasing $K$), but also becomes very expensive w.r.t. time.




**(a)** Complete graph $k = 2, n = 20$ **(b)** ER graph $k = 2, n = 20$ **(c)** Complete graph $k = 3, n = 10$

**Figure B.4:** Variation of $num\_cycles$ in AIS

## B.7 Image Segmentation - more results

We describe in more detail the setting in DenseCRF [85]. Let $f_i$ denote the feature vector associated with the $i^{th}$ pixel in an image e.g. position, RGB values, etc. Then, the image segmentation task is to compute the configuration of labels $x \in [k]^n$ for the pixels in an image that maximizes:

$$\max_{x \in [k]^n} \sum_{i<j} \mu(x_i, x_j) \bar{K}(f_i, f_j) + \sum_i \psi_u(x_i).$$

The first term provides pairwise potentials where $\bar{K}(f_i, f_j)$ is modelled as a Gaussian kernel consisting of smoothness and appearance kernels and the coefficient $\mu$ is the label compatibility function. The second term corresponds to unary potentials for the individual pixels. In keeping with the SDP relaxation described above, we relax each pixel to $\mathbb{R}^d$ to derive the following optimization problem:

$$\max_{v_i \in \mathbb{R}^d,\, \|v_i\|_2 = 1\ \forall i \in [n]} \sum_{i<j} \bar{K}(f_i, f_j) v_i^T v_j + \theta \sum_{i=1}^{n} \sum_{l=1}^{k} \log p_{i,l} \cdot v_i^T r_l. \tag{4.15}$$

In the first term above, the term $v_i^T v_j$ models the label compatibility function $\mu$, and we can observe that if $\bar{K}(f_i, f_j)$ is large i.e. the pixels are similar, it encourages the vectors $v_i$ and $v_j$ to be aligned. The second term models unary potentials $\phi_u$ from available rough annotations, so that we have a bias vector $r_l$ for each label, and the term $\log p_{i,l}$ plugs in our prior belief based on annotations of the $i^{th}$ pixel being assigned the $l^{th}$ label. The coefficient $\theta$ helps control the relative weight on the pairwise and unary potentials. The mixing method update for the above objective is:

$$v_i \leftarrow normalize \underbrace{\left( \sum_{j \neq i} \bar{K}(f_i, f_j) v_j + \theta \sum_{l=1}^{L} \log p_{i,l} \cdot r_l \right)}_{G_i}. \tag{B.2}$$

We note here that computing the pairwise kernels $\bar{K}(f_i, f_j)$ naively has a quadratic time complexity in $n$, and for standard images, the number of pixels is pretty large, making this computation very slow. Here, we use the high-dimensional filtering method as in DenseCRF [85] which provides a linear time approximation for simultaneously updating all the $v_i$s as given by the update in (B.2). However, because of the simultaneous nature of the updates, we are no longer employing true coordinate descent. Hence, we instead propose to use a form of gradient descent with a small learning rate $\alpha$ to update each of the $v_i$s as follows. Here, the $G_i$s are those that are simultaneously given for all $i$ at once by the high-dimensional filtering method:

$$v_i \leftarrow normalize(v_i + \alpha \cdot G_i).$$

At convergence, we use the same rounding scheme described in Algorithm 4.2 above to obtain a configuration of labels for each pixel. Figures B.5, B.6 below show the results of using our method for performing image segmentation on some benchmark images obtained from the works

of DenseCRF [85], Lin et al. [96]. We can see that our method produces accurate segmentations, competitive with the quality of segmentations demonstrated in DenseCRF[85].

The naive runtime for segmenting a standard (say 400x400) image by our method (without any GPU parallelization) is roughly $\sim 2$ minutes. We remark here that performing each segmentation constitutes randomly initializing the $v_i$ vectors and solving (4.15) via the mixing method, and also performing a few rounds of rounding. However, with parallelization and several optimizations, we believe that there is massive scope for significantly reducing this runtime.



**Figure B.5:** Original image, annotated image, segmented image

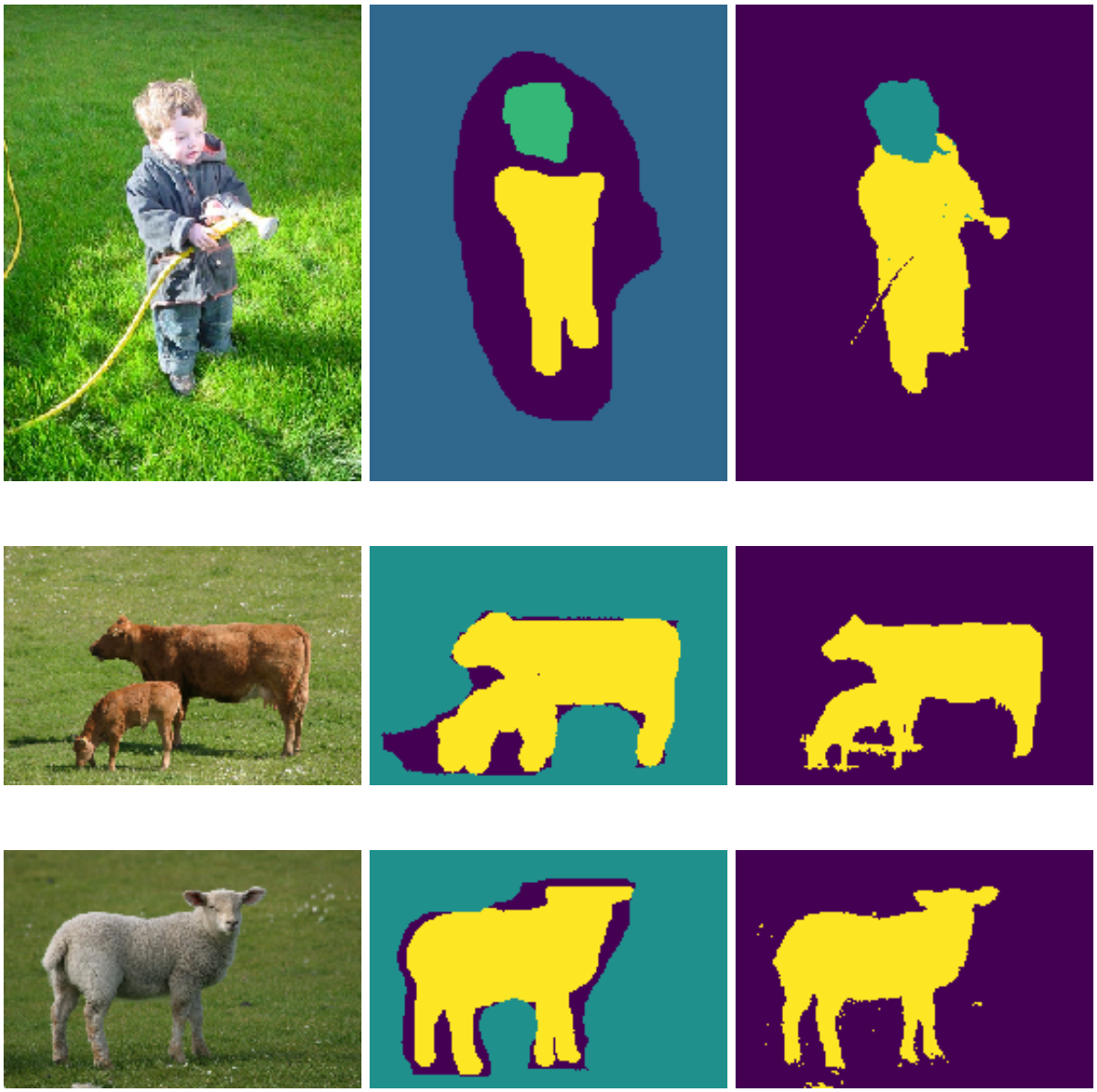


**Figure B.6:** Original image, annotated image, segmented image

# Appendix C

# Derivations and additional experimental results for SATNet

## C.1 Derivation of the forward pass coordinate descent update

Our MAXSAT SDP relaxation (described in Section 5.1.1) is given by

$$\begin{array}{ll} \underset{V \in \mathbb{R}^{k \times (n+1)}}{\text{minimize}} & \langle S^T S, V^T V \rangle, \\ \text{subject to} & \|v_i\| = 1, \quad i = 0, \ldots, n, \end{array} \tag{C.1}$$

where $S \in \mathbb{R}^{m \times (n+1)}$ and $v_i$ is the $i$th column vector of $V$.

We rewrite the objective of (C.1) as $\langle S^T S, V^T V \rangle \equiv \mathrm{tr}((S^T S)^T (V^T V)) = \mathrm{tr}(V^T V S^T S)$ by noting that $S^T S$ is symmetric and by cycling matrices within the trace. We then observe that the objective terms that depend on any given $v_i$ are given by

$$v_i^T \sum_{j=0}^{n} s_j^T s_i v_j = v_i^T \sum_{\substack{j=0 \\ (j \neq i)}}^{n} s_j^T s_i v_j + v_i^T s_i^T s_i v_i, \tag{C.2}$$

where $s_i$ is the $i$th column vector of $S$. Observe $v_i^T v_i$ in the last term cancels to 1, and the remaining coefficient

$$g_i \equiv \sum_{\substack{j=0 \\ (j \neq i)}}^{n} s_j^T s_i v_j = V S^T s_i - \|s_i\|^2 v_i \tag{C.3}$$

is constant with respect to $v_i$. Thus, (C.2) can be simply rewritten as

$$v_i^T g_i + s_i^T s_i. \tag{C.4}$$

Minimizing this expression over $v_i$ with respect to the constraint $\|v_i\| = 1$ yields the block coordinate descent update

$$v_i = -g_i / \|g_i\|. \tag{C.5}$$

## C.2 Details on backpropagation through the MAXSAT SDP

Given the result $\partial \ell / \partial V_{\mathcal{O}}$, we next seek to compute $\partial \ell / \partial V_{\mathcal{I}}$ and $\partial \ell / \partial S$ by pushing gradients through the SDP solution procedure described in Section 5.1.1. We do this by taking the total differential through our coordinate descent updates (5.3) for each output $o \in \mathcal{O}$ at the optimal fixed-point solution to which these updates converge.

**Computing the total differential.** Computing the total differential of the updates (5.3) and rearranging, we see that for every $o \in \mathcal{O}$,

$$\left(\|g_o\| I_k - \|s_o\|^2 P_o\right) \mathsf{d}v_o + P_o \sum_{j \in \mathcal{O}} s_o^T s_j \mathsf{d}v_j = -P_o \xi_o, \tag{C.6}$$

where

$$\xi_o \equiv \Big( \sum_{j \in \mathcal{I}'} s_o^T s_j \mathsf{d}v_j + V \mathsf{d}S^T s_o + V S^T \mathsf{d}s_o - 2\mathsf{d}s_o^T s_o v_o \Big), \tag{C.7}$$

and where $P_o \equiv I_k - v_o v_o^T, o \in \mathcal{O}$ and $\mathcal{I}' \equiv \{\top\} \cup \mathcal{I}$.

**Rewriting as a linear system.** Rewriting Equation C.6 over all $o \in \mathcal{O}$ as a linear system, we obtain

$$\begin{aligned} &\Big( \operatorname{diag}(\|g_o\|) \otimes I_k + PC \otimes I_k \Big) \operatorname{vect}(\mathsf{d}V_{\mathcal{O}}) = -P \operatorname{vect}(\xi_o) \\ &\Rightarrow \operatorname{vect}(\mathsf{d}V_{\mathcal{O}}) = -\Big( P(\big( \operatorname{diag}(\|g_o\|) + C \big) \otimes I_k) P \Big)^{\dagger} \operatorname{vect}(\xi_o), \end{aligned} \tag{C.8}$$

where $C = S_{\mathcal{O}}^T S_{\mathcal{O}} - \operatorname{diag}(\|s_o\|^2)$, $P = \operatorname{diag}(P_o)$, and the second step follows from the lemma presented in Appendix C.3.

We then see that by the chain rule, the gradients $\partial \ell / \partial V_{\mathcal{I}}$ and $\partial \ell / \partial S$ are given by the left matrix-vector product

$$\begin{aligned} &\left( \frac{\partial \ell}{\partial \operatorname{vect}(V_{\mathcal{O}})} \right)^T \operatorname{vect}(\mathsf{d}V_{\mathcal{O}}) \\ &= -\left( \frac{\partial \ell}{\partial \operatorname{vect}(V_{\mathcal{O}})} \right)^T \Big( P(\big( \operatorname{diag}(\|g_o\|) + C \big) \otimes I_k) P \Big)^{\dagger} \operatorname{vect}(\xi_o) \end{aligned} \tag{C.9}$$

where the second equality comes from plugging in the result of (C.8).

Now, define $U \in \mathbb{R}^{k \times n}$, where the columns $U_{\mathcal{I}} = 0$ and the columns $U_{\mathcal{O}}$ are given by

$$\operatorname{vect}(U_{\mathcal{O}}) = \Big( P(\big( \operatorname{diag}(\|g_o\|) + C \big) \otimes I_k) P \Big)^{\dagger} \operatorname{vect} \left( \frac{\partial \ell}{\partial \operatorname{vect}(V_{\mathcal{O}})} \right). \tag{C.10}$$

Then, we see that (C.9) can be written as

$$\left( \frac{\partial \ell}{\partial \operatorname{vect}(V_{\mathcal{O}})} \right)^T \operatorname{vect}(\mathsf{d}V_{\mathcal{O}}) = -\operatorname{vect}(U_{\mathcal{O}})^T \operatorname{vect}(\xi_o), \tag{C.11}$$

which is the implicit linear form for our gradients.

**Computing desired gradients from implicit linear form.** Once we have obtained $U_{\mathcal{O}}$ (via coordinate descent), we can explicitly compute the desired gradients $\partial\ell/\partial V_{\mathcal{I}}$ and $\partial\ell/\partial S$ from the implicit form (C.11). For instance, to compute the gradient $\partial\ell/\partial v_\iota$ for some $\iota \in \mathcal{I}$, we would set $\mathsf{d}v_\iota = 1$ and all other gradients to zero in Equation (C.11) (where these gradients are captured within the terms $\xi_o$).

Explicitly, we compute each $\partial\ell/\partial v_{\iota j}$ by setting $\mathsf{d}v_{\iota j} = 1$ and all other gradients to zero, i.e.

$$\begin{aligned}\frac{\partial \ell}{\partial v_{\iota j}} &= -\operatorname{vect}(U_{\mathcal{O}})^T \operatorname{vect}(\xi_o) = -\sum_{o\in\mathcal{O}} u_o^T e_j s_\iota^T s_o \\ &= -e_j^T \left(\sum_{o\in\mathcal{O}} u_o s_o^T\right) s_\iota.\end{aligned} \tag{C.12}$$

Similarly, we compute each $\partial\ell/\partial S_{i,j}$ by setting $\mathsf{d}S_{i,j} = 1$ and all other gradients to zero, i.e.

$$\begin{aligned}\frac{\partial \ell}{\partial S_{i,j}} &= -\sum_{o\in\mathcal{O}} u_o^T \xi_o \\ &= -\sum_{o\in\mathcal{O}} u_o^T v_i s_{oj} - u_i^T (VS^T)_j + u_i^T (s_{ij} P_i v_i) \\ &= -v_i^T (\sum_{o\in\mathcal{O}} u_o s_{oj}) - u_i^T (VS^T)_j.\end{aligned} \tag{C.13}$$

In matrix form, these gradients are

$$\frac{\partial \ell}{\partial V_{\mathcal{I}}} = -\left(\sum_{o\in\mathcal{O}} u_o s_o^T\right) S_{\mathcal{I}}, \tag{C.14}$$

$$\frac{\partial \ell}{\partial S} = -\left(\sum_{o\in\mathcal{O}} u_o s_o^T\right)^T V - (SV^T)U, \tag{C.15}$$

where $u_i$ is the $i$th column of $U$, and where $S_{\mathcal{I}}$ denotes the $\mathcal{I}$-indexed column subset of $S$.

## C.3 Proof of pseudoinverse computations

We prove the following lemma, used to derive the implicit total differential for $\operatorname{vect}(\mathsf{d}V_{\mathcal{O}})$.

**Lemma C.1.** *The quantity*

$$\operatorname{vect}(\mathsf{d}V_{\mathcal{O}}) = \left(P\left((D+C)\otimes I_k\right)P\right)^{\dagger} \operatorname{vect}(\xi_o) \tag{C.16}$$

*is the solution of the linear system*

$$\left(D\otimes I_k + PC\otimes I_k\right)\operatorname{vect}(\mathsf{d}V_{\mathcal{O}}) = P\operatorname{vect}(\xi_o), \tag{C.17}$$

*where* $P = \operatorname{diag}(I_k - v_o v_o^T)$, $C = S_{\mathcal{O}}^T S_{\mathcal{O}} - \operatorname{diag}(\|s_o\|^2)$, $D = \operatorname{diag}(\|g_i\|)$, *and* $\xi_o$ *is as defined in Equation* (C.7).

*Proof.* Examining the equation with respect to $\mathsf{d}v_i$ gives

$$\|g_i\|\mathsf{d}v_i + P_i\left(\sum_j c_{ij}\mathsf{d}v_j - \xi_j\right) = 0, \tag{C.18}$$

which implies that for all $i$, $\mathsf{d}v_i = P_i y_i$ for some $y_i$. Substituting $y_i$ into the equality gives

$$(D \otimes I_k + PC \otimes I_k)P\,\mathrm{vect}(y_i) \tag{C.19}$$

$$= P((D + C) \otimes I_k)P\,\mathrm{vect}(y_i) = P\,\mathrm{vect}(\xi_o). \tag{C.20}$$

Note that the last equation comes form $D \otimes I_k P = D \otimes I_k PP = P(D \otimes I_k)P$ due to the block diagonal structure of the projection $P$. Thus, by the properties of projectors and the pseudoinverse,

$$\mathrm{vect}(Y) = (P((D + C) \otimes I_k)P)^\dagger P\,\mathrm{vect}(\xi_o) \tag{C.21}$$

$$= (P((D + C) \otimes I_k)P)^\dagger\,\mathrm{vect}(\xi_o). \tag{C.22}$$

Note that the first equation comes from the idempotence property of $P$ (that is, $PP = P$). Substituting $\mathrm{vect}(\mathsf{d}V_\mathcal{O}) = P\,\mathrm{vect}(Y)$ back gives the solution of $\mathsf{d}V_\mathcal{O}$. □

## C.4 Derivation of the backward pass coordinate descent algorithm

Consider solving for $U_\mathcal{O}$ as mentioned in Equation (C.10):

$$\Big(P(\big(\,\mathrm{diag}(\|g_o\|) + C\big) \otimes I_k)P\Big)\,\mathrm{vect}(U_\mathcal{O}) = \mathrm{vect}\left(\frac{\partial \ell}{\partial\,\mathrm{vect}(V_\mathcal{O})}\right),$$

where $C = S_\mathcal{O}^T S_\mathcal{O} - \mathrm{diag}(\|s_o\|^2)$. The linear system can be computed using block coordinate descent. Specifically, observe this linear system with respect to only the $u_o$ variable. Since we start from $U_\mathcal{O} = 0$, we can assume that $P\,\mathrm{vect}(U_o) = \mathrm{vect}(U_o)$. This yields

$$\|g_o\|P_o u_o + P_o\Big(U_\mathcal{O} S_\mathcal{O}^T s_o - \|s_o\|^2 u_o\Big) = P_o\left(\frac{\partial \ell}{\partial v_o}\right). \tag{C.23}$$

Let $\Psi = (U_\mathcal{O})S_\mathcal{O}^T$. Then we have

$$\|g_o\|P_o u_o = -P_o(\Psi s_o - \|s_o\|^2 u_o - \partial\ell/\partial v_o). \tag{C.24}$$

Define $-\mathsf{d}g_i$ to be the terms contained in parentheses in the right-hand side of the above equation. Note that $\mathsf{d}g_i$ does not depend on the variable $u_o$. Thus, we have the closed-form feasible solution

$$u_o = -P_o \mathsf{d}g_o/\|g_o\|. \tag{C.25}$$

After updating $u_o$, we can maintain the term $\Psi$ by replacing the old $u_o^{\text{prev}}$ with the new $u_o$. This yields the rank 1 update

$$\Psi := \Psi + (u_o - u_o^{\text{prev}})s_o^T. \tag{C.26}$$

The above procedure is summarized in Algorithm 5.3. Further, we can verify that the assumption $P\,\mathrm{vect}(U_\mathcal{O}) = \mathrm{vect}(U_\mathcal{O})$ still holds after each update by the projection $P_o$.

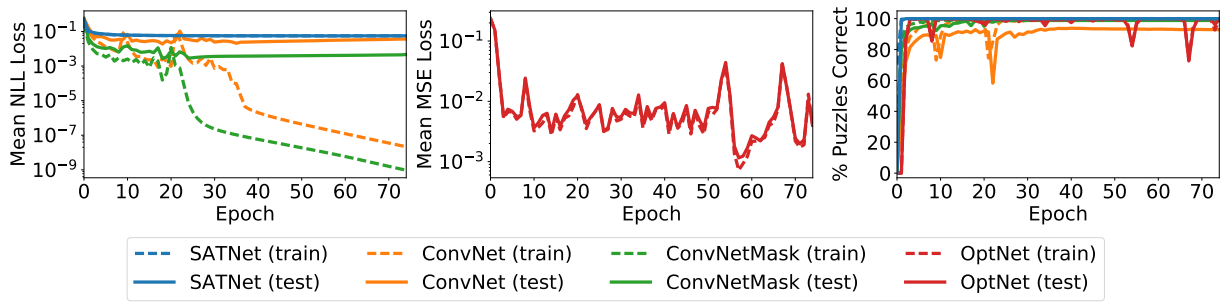



**Figure C.1:** Results for $4 \times 4$ Sudoku. Lower loss (mean NLL loss and mean MSE loss) and higher whole-board accuracy (% puzzles correct) are better.

## C.5 Results for the $4 \times 4$ Sudoku problem

We compare the performance of our SATNet architecture on a $4 \times 4$ reduced version of the Sudoku puzzle against OptNet [5] and a convolutional neural network architecture. These results (over 9K training and 1K testing examples) are shown in Figure C.1. We note that our architecture converges quickly – in just two epochs – to *100% board-wise test accuracy*.

OptNet takes slightly longer to converge to similar performance, in terms of both time and epochs. In particular, we see that OptNet takes 3-4 epochs to converge (as opposed to 1 epoch for SATNet). Further, in our preliminary benchmarks, OptNet required 12 minutes to run 20 epochs on a GTX 1080 Ti GPU, whereas SATNet took only 2 minutes to run the same number of epochs. In other words, we see that SATNet requires fewer epochs to converge *and* takes less time per epoch than OptNet.

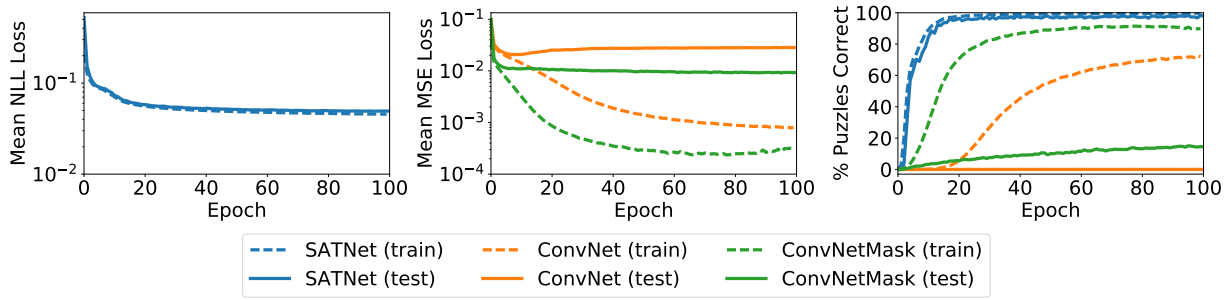
Both our SATNet architecture and OptNet outperform the traditional convolutional neural network in this setting, as the ConvNet somewhat overfits to the training set and therefore does not generalize as well to the test set (achieving 93% accuracy). The ConvNetMask, which additionally receives a binary input mask, performs much better (99% test accuracy) but does not achieve perfect performance as in the case of OptNet and SATNet.

## C.6 Convergence plots for $9 \times 9$ Sudoku experiments

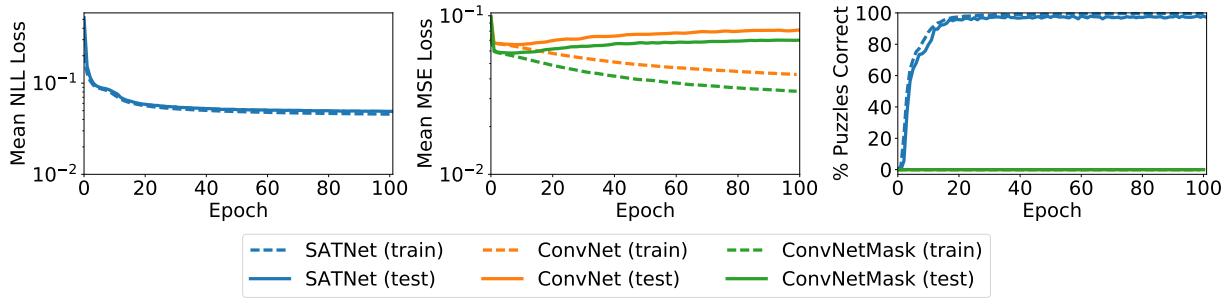
Convergence plots for our $9 \times 9$ Sudoku experiments (original and permuted) are shown in Figure C.2. SATNet performs nearly identically in both the original and permuted settings, generalizing well to the test set at every epoch without overfitting to the training set. The ConvNet and ConvNetMask, on the other hand, do not generalize well. In the original setting, both architectures overfit to the training set, showing little-to-no improvement in generalization performance over the course of training. In the permuted setting, both ConvNet and ConvNetMask make little progress even on the training set, as they are not able to rely on spatial locality of inputs.

Convergence plots for the visual Sudoku experiments are shown in Figure C.3. Here, we see that SATNet generalizes well in terms of loss throughout the training process, and generalizes somewhat well in terms of whole-board accuracy. The difference in generalization performance

between the logical and visual Sudoku settings can be attributed to the generalization performance of the MNIST classifier trained end-to-end with our SATNet layer. The ConvNetMask architecture overfits to the training set, and the ConvNet architecture makes little-to-no progress even on the training set.




**(a)** Original $9 \times 9$ Sudoku




**(b)** Permuted $9 \times 9$ Sudoku

**Figure C.2:** Results for our $9 \times 9$ Sudoku experiments. Lower loss (mean NLL loss and mean MSE loss) and higher whole-board accuracy (% puzzles correct) are better.

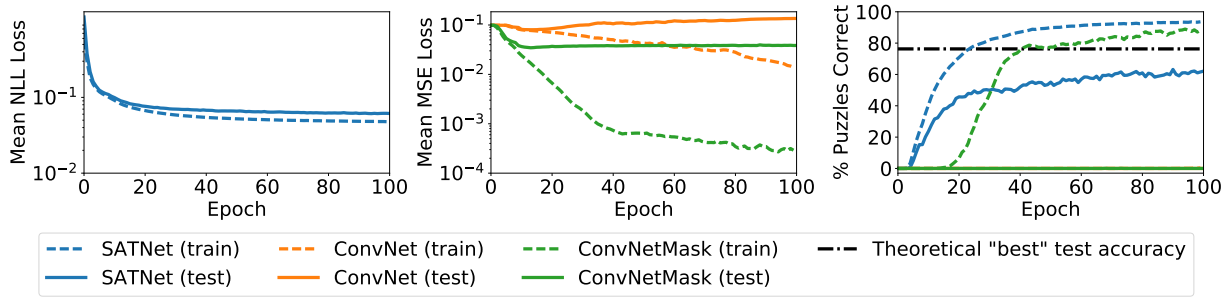



**Figure C.3:** Results for our visual Sudoku experiments. Lower loss (mean NLL loss and mean MSE loss) and higher whole-board accuracy (% puzzles correct) are better. The theoretical "best" test accuracy plotted is for our specific choice of MNIST classifier architecture.

# Appendix D

# Proofs for the Locale algorithm

## D.1 Proof of Proposition 6.1

Note that in problem (6.3) the term $(a_{ii} - d_i d_i/(2m))v_i^T v_i$ is constant because $\|v_i\| = 1$. Thus, in the subproblem $Q(v_i)$ for variable $v_i$, we can ignore the constant term and write the gradient $\nabla Q(v_i)$ as

$$\nabla Q(v_i) = \frac{1}{2m} \sum_{j \neq i} \left(a_{ij} - \frac{d_i d_j}{2m}\right) v_j. \tag{D.1}$$

Further, since there is no $v_i$ term in $\nabla Q(v_i)$, the objective function $Q(v_i)$ for the subproblem of variable $v_i$ becomes $q^T v_i$ with $q = \nabla Q(v_i)$, up to a constant. For simplicity, denote $v_i$ as $v$, and the subproblem reduces to

$$\underset{v}{\text{maximize}}\ q^T v, \quad \text{s.t. } v \in \mathbb{R}_+^r,\ \|v\| = 1,\ \text{card}(v) \leq k. \tag{D.2}$$

Let $v^*$ be the optimal solution of the above subproblem (6.4) (existence by compactness). When $q \leq 0$, we have $\max(q) \leq 0$. With $\|v\|_2 = 1$, $v \geq 0$, and $\|v\|_2 \leq \|v\|_1$, there is

$$\max(q) = \max(q)\|v\|_2 \geq \max(q)\|v\|_1 = \max(q) \sum_t v_t \geq \sum_t q_t v_t = q^T v. \tag{D.3}$$

Thus, $e(t)$ with the max $q_t$ is the optimal solution in the first case. For the second case, there is at least one coordinate $p$ such that $q_p > 0$. Now we exclude the following two cases of inactive coordinates by contradictions.

**(When $q_t < 0$)** We know $v_t^* = 0$. Otherwise, suppose there is a $v_t^* > 0$ with $q_t < 0$.
If $q^T v^* \leq 0$, selecting $v^* = e(p)$ violates the optimality of $v^*$, a contradiction.
If $q^T v^* > 0$, we have

$$0 < q^T v^* < q^T(v^* - e(t)v_t^*) \leq q^T(v^* - e(t)v_t^*)/\|v^* - e(t)v_t^*\|, \tag{D.4}$$

also a contradiction to the optimality of $v^*$, because the last term is a feasible solution.

**(When $q_t < q_{[k]}$, where $q_{[k]}$ is the $k$-th largest value)** We know $v_t^* = 0$. Otherwise, there must be a coordinate $j$ in the top-$k$-largest value that is not selected ($v_j^* = 0$) because $\mathrm{card}(v^*) \leq k$. This way, we have

$$q^T v^* < q^T(v^* - e(t)v_t^* + e(j)v_t^*), \tag{D.5}$$

which contradicts to the optimality of $v^*$ because $(v^* - e(t)v_t^* + e(j)v_t^*)$ is a feasible solution.

Thus, by removing the inactive coordinates, the effective objective function $q^T v^*$ becomes $\mathrm{top}_k^+(q)^T v^*$, and the optimal solution follows from $\|v_i^*\| = 1$ and $\mathrm{top}_k^+(q) \geq 0$. □

## D.2 Proof of Theorem 6.2

Define the projected gradient (for maximization) as

$$\texttt{grad}(V) = P_\Omega(V + \nabla Q(V)) - V, \tag{D.6}$$

where $P_\Omega$ is the projection (under 2-norm) to the constraint set $\Omega$ of the optimization problem (6.3)

$$\Omega = \{V \mid v_i \in \mathbb{R}^r_+,\ \|v_i\| = 1,\ \text{card}(v_i) \le k,\ \forall i = 1, \ldots, n\}, \tag{D.7}$$

and denote $\Omega_i$ as the constraint for $v_i$ for the separable $\Omega$. Because the cardinality constraint is an union between finite hyperplanes, it is a closed set, which implies the constraint of the optimization problem is a compact set. Thus, by the Weierstrass extreme value theorem, the function $Q(V)$ is upper-bounded and must attain global maximum over the constraint.

Now we connect the exact update in the Locale algorithm with the projected gradient. Denote $v_i^+$ as the update taken for the subproblem $Q(v_i)$. Because the Locale algorithm performs an exact update (Proposition 6.1), we have

$$\nabla Q(v_i)^T v_i^+ \ge \nabla Q(v_i)^T u,\ \forall u \in \Omega_i. \tag{D.8}$$

Further, because $\|v_i^+\|^2 = 1$ and $\|u\|^2 = 1$, we have

$$\|v_i^+ - \nabla Q(v_i)\|^2 \le \|u - \nabla Q(v_i)\|^2,\ \forall u \in \Omega_i. \tag{D.9}$$

This means that the update $v_i^+$ is the projection of $\nabla Q(v_i)$ to the constraint set $\Omega_i$. To connect the update with the projected gradient, we need the following lemma.

**Lemma D.1.** *Denote the projection (under 2-norm) of a point $x$ on a closed constraint set $\Omega$ as $P_\Omega(x)$. Then for any scalar $\alpha > 1$ and vector $q$, we have*

$$q^T(P_\Omega(x + \alpha q) - P_\Omega(x + q)) \ge 0$$

The proof is listed in Appendix D.3. Taking the lemma with $\alpha \to \infty$ and let $q = \nabla Q(v_i)$, we have

$$0 \le \lim_{\alpha \to 0} q^T(P_{\Omega_i}(v_i + \alpha q) - P_{\Omega_i}(x + q)) = q^T(v_i^+ - P_{\Omega_i}(v_i + q)), \tag{D.10}$$

where the last equation follows because $v_i^+$ is the projection of $q$ on $\Omega_i$ with $\|\cdot\| = 1$ constraint [1]. Further, apply the definition of projection $P_{\Omega_i}(v_i + q)$ again on the feasible $v_i$, we have

$$\|P_{\Omega_i}(v_i + q) - (v_i + q)\|^2 \le \|v_i - (v_i + q)\|^2, \tag{D.11}$$

and after rearranging there is

$$\|P_{\Omega_i}(v_i + q) - v_i\|^2 \le 2q^T(P_{\Omega_i}(v_i + q) - v_i). \tag{D.12}$$

Applying (D.10) to the equation above, we have

$$\|P_{\Omega_i}(v_i + q) - v_i\|^2 \le 2q^T(v_i^+ - v_i). \tag{D.13}$$

[1] Note that in Proposition 6.1, when $q \le 0$ and there are multiple maximum $q_t$, we further select the $t$ with the maximum $(v_i)_t$ in the previous iteration. This makes the limit to hold on the corner case $q = 0$.

The right hand side of the above equation equals the function increment $Q(v_i^+) - Q(v_i)$. Thus,

$$\|P_{\Omega_i}(v_i + q) - v_i\|^2 \leq 2(Q(v_i^+) - Q(v_i)). \tag{D.14}$$

Now, taking expectation over the random coordinate $i$, we have

$$\frac{1}{n}\|P_\Omega(V + \nabla Q(V)) - V\|^2 = \mathbb{E}\|P_{\Omega_i}(v_i + q) - v_i\|^2 \leq 2\mathbb{E}(Q(v_i^+) - Q(v_i)) = Q(V^{t+1}) - Q(V^t). \tag{D.15}$$

Further, since $Q(V^{t+1}) - Q(V^t)$ is monotonic increasing, summing them over iterations $0$ to $T-1$ forms a telescoping sum, which is upper-bounded by $Q(V^*) - Q(V^0)$, where $V^*$ is the global optimal solution of $Q(V)$. Substitute the definition of projected gradient (D.6), we have

$$\frac{T}{n}\min_t \|\texttt{grad}(V^t)\|^2 \leq \frac{1}{n}\sum_{t=0}^{T-1}\|\texttt{grad}(V^t)\|^2 \leq 2(Q(V^*) - Q(V^0)). \tag{D.16}$$

Thus, the projected gradient $\texttt{grad}(V)$ converges to zero at a $O(1/T)$ rate. □

## D.3 Proof for Lemma D.1

By definition of the projection $P_\Omega(x + q)$, we have

$$\|P_\Omega(x + q) - (x + q)\|^2 \leq \|P_\Omega(x + \alpha q) - (x + q)\|^2.$$

Take out the $q$ term out of the norm and rearrange, there is

$$\|P_\Omega(x + q) - x\|^2 \leq \|P_\Omega(x + \alpha q) - x\|^2 - 2q^T(P_\Omega(x + \alpha q) - P_\Omega(x + q)). \tag{D.17}$$

Similarly, by definition of the projection $P_\Omega(x + \alpha q)$, there is

$$\|P_\Omega(x + \alpha q) - x\|^2 \leq \|P_\Omega(x + q) - x\|^2 - 2\alpha q^T(P_\Omega(x + q) - P_\Omega(x + \alpha q)). \tag{D.18}$$

Sum (D.17) and (D.18), the norms cancel, and we have

$$2(\alpha - 1)q^T(P_\Omega(x + \alpha q) - P_\Omega(x + q)) \geq 0,$$

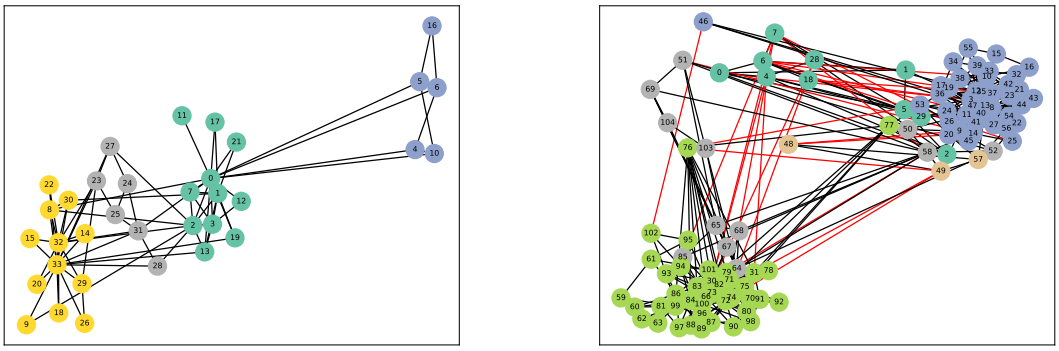
which implies

$$q^T(P_\Omega(x + \alpha q) - P_\Omega(x + q)) \geq 0. \tag{D.19}$$

Thus, the result holds. □

## D.4 Experiments on networks with ground truth

In this section, we compare results from the Leiden-Locale method on data with the ground truth for partitions. The result is listed in Figure D.1.



**(a)** `zachary` (ground truth = 4 clusters) **(b)** `polbook` (ground truth = 3 clusters)

**Figure D.1:** The comparison of the results from Leiden-Locale method to ground-truth partitions in the `zachary` and `polbook` datasets. The position of each node is arranged using the 2D Fruchterman-Reingold force-directed algorithm from the ground-truth using `networkx` [72], and the color of each node indicates the solution community given by Leiden-Locale algorithm. The red edges between nodes indicates the case when two nodes are inside the same cluster in the ground truth but wasn't assigned so in our algorithm. For `zachary`, the Leiden-Locale algorithm returns a perfect answer comparing to the ground truth with a perfect modularity of 0.4197 [103]. For `polbook`, it misclassifies 18 over 105 nodes, but still attains a best known modularity of 0.5272 [2].

## D.5 Pseudo-code for the Leiden-Locale algorithm

Here we list the pseudo-code for the Leiden-Locale method. Note that we reuse Algorithm D.1 in Algorithm D.2–D.3 for rounding and refinement by changing its constraint and initialization. And in the actual code, Algorithm D.2–D.3 are combined as a single subroutine.

**Algorithm D.1** Optimization procedure for the Locale algorithm

1: **procedure** LOCALEEMBEDDINGS(Graph $G$, Partition $P$)
2: Initialize $V$ with $v_i = e(i),\ i = 1, \ldots, n$.
3: Initialize the ring queue $R$ with indices $i = 1, \ldots, n$.
4: Let $z = \sum_{j=1}^{n} d_j v_j$.
5: **while** not yet converged **do**
6: $i = R.pop()$ ▷ Pick an index from the ring queue
7: $\nabla Q(v_i) = \sum_{j \in P(i)} a_{ij} v_j - \frac{d_i}{2m}(z - d_i v_i)$ ▷ Sums only $j$ in the same partition of $i$
8: $g_i = \begin{cases} e(t) \text{ with the max } (\nabla Q(v_i))_t, & \text{if } \nabla Q(v_i) \le 0, \\ \text{top}_k^+(\nabla Q(v_i)), & \text{otherwise.} \end{cases}$
9: $v_i^{\text{old}} = v_i, \quad v_i = g_i / \|g_i\|$ ▷ Perform the closed-form update
10: $z = z + d_i(v_i - v_i^{\text{old}})$ ▷ Maintain the $z$
11: Push all neighbors $j$ with nonzero $a_{ij}$ into the ring queue $R$ if it is not already inside.
12: **end while**
13: **return** the embedding $V$
14: **end procedure**

**Algorithm D.2** Rounding procedure for the Locale algorithm

1: **procedure** LOCALEROUNDING(Graph $G$, Partition $P$, Embedding $E$)
2: Initialize $V$ with input $E$.
3: Run line 3–12 of Algorithm D.1 with cardinality constraint $k = 1$.
4: Let the index of the 1-sparse embedding above be the new partition $P'$.
5: **return** $P'$
6: **end procedure**

**Algorithm D.3** Refine and Aggregate procedure from the Leiden algorithm

1: **procedure** LEIDENREFINEAGGREGATE(Graph $G$, Partition $P$)
2: Refine $P' \leftarrow$ *LocaleRounding*$(G, P)$ by restricting the local move within its partition.[2]
3: Forms a hypergraph $G'$ by merging nodes inside the same partitions in $P'$ and simplify $P'$.
4: done $\leftarrow |P|$ equals $|G'|$.
5: **return** $G'$, $P'$, done
6: **end procedure**

[2]This is the refinement step implemented in the package `python-leiden`.

# Appendix E

# Proofs for the linear programming algorithm

## E.1 Proof of Theorem 7.3

This result is obtained by treating Eq. (7.5) as a special case of the problems analyzed in an unpublished work from Ching-Pei. For completeness, we provide a detailed proof for this simplified case Eq. (7.5) (in comparison to the general case in the unpublished work) in this appendix. The proof consists of three parts: global convergence, local superlinear convergence when unit step size is taken, and finally the acceptance of unit step size for all large enough iterations.

### E.1.1 Global Convergence

We start with that the step sizes are lower-bounded and then show that the objective value converges to the optimal one. At the $t$-th iteration, we denote the iterate by $w^t$, the SSN direction by $d_t$, and the step size by $\theta_t$. We also use the following notations in our following description:

$$\Delta_t \coloneqq \nabla f(w^t)^\top d_t, \quad \mu_t \coloneqq c\|\nabla f(w^t)\|^\rho, \quad H_t \coloneqq \nabla^2 f(w^t) + \mu_t I, \tag{E.1}$$

where $\nabla^2 f(w^t)$ is the generalized Hessian selected for the SSN step.

**Lemma E.1.** *Given $\gamma, \sigma \in (0,1)$ and let $L$ be the Lipschitz constant for $\nabla f$. Given an iterate $w^t$ and the SSN direction $d_t$ satisfying Eq.* (7.10) *for some $\nu \in [0,1)$, then the backtracking line search procedure produces a step size $\theta$ satisfying Eq.* (7.11) *with*

$$1 \geq \theta_t \geq \min\left\{1, \frac{\gamma\mu_t}{L}(1 - \sigma\min\{1, \mu_t\})\right\}. \tag{E.2}$$

*Proof.* The upper bound for $\theta_t$ is straightforward, and here we prove the upper bound. If $\|d_t\| = 0$, then clearly $\theta_t = 1$ satisfies Eq. (7.11). Thus we assume without loss of generality that $\|d_t\| > 0$. The first condition in Eq. (7.10) indicates that

$$\Delta_t \leq -\frac{1}{2} d_t^\top H_t d_t \leq -\frac{1}{2}\mu_t \|d_t\|_2^2 < 0. \tag{E.3}$$

Since $f$ is $L$-Lipschitz-continuously differentiable, we obtain

$$f\left(w^t + \theta_t d_t\right) \leq f\left(w^t\right) + \theta_t \nabla f(w^t)^\top d_t + \frac{\theta_t^2 L}{2}\|d_t\|_2^2 \overset{Eq.\ (E.3)}{\leq} f(w^t) + \Delta_t \left(\theta_t - \frac{\theta_t^2 L}{\mu_t}\right). \quad \text{(E.4)}$$

Thus, for the line search criterion Eq. (7.11) to hold, it suffices to have

$$\Delta_t \left(\theta_t - \frac{\theta_t^2 L}{\mu_t}\right) \leq \theta_t \sigma \min\{1, \mu_t\} \Delta_t,$$

or equivalently (as $\Delta_t < 0$),

$$1 - \frac{\theta_t L}{\mu_t} \geq \sigma \min\{1, \mu_t\} \quad \Leftrightarrow \quad \theta_t \leq \frac{\mu_t}{L}\left(1 - \sigma \min\{1, \mu_t\}\right).$$

Since all $\theta_t$ in the range holds for criterion Eq. (7.11), after considering the possible overshoot from backtracking, we have proven Eq. (E.2). □

**Lemma E.2.** *Assume that Eq.* (P) *has at least one optimal solution. When the SSN procedure described in Section 7.2.2 is applied to solve Eq.* (7.5) *starting from any $w^0$, and let the iterates be $\{w^t\}$, we have*

$$\lim_{t\to\infty} \nabla f(w^t) = 0, \quad \lim_{t\to\infty} \operatorname{dist}(w^t, \Omega) = 0, \quad \lim_{t\to\infty} f(w^t) = 0.$$

*Proof.* Summing Eq. (7.11) from $t = 0$ to $t = \infty$, we obtain

$$-\sigma \sum_{t=0}^{\infty} \theta_t \min\{1, \mu_t\} \Delta_t \leq f(w^0),$$

since Proposition 7.1 shows that $f(w) \geq 0$ for any $w$. This together with Lemma E.1 implies that

$$\lim_{t\to\infty} \min\left\{1, \frac{\gamma \mu_t}{L}\left(1 - \sigma \min\left\{1, \mu_t\right\}\right)\right\} \min\{1, \mu_t\} \mu_t \|d_t\|_2^2 = 0.$$

By carefully examining the above equation, we get that either $\mu_t \to 0$ or $\|d_t\| \to 0$. If $\mu_t \to 0$, then it implies that

$$\nabla f(w^t) \to 0 \quad \text{(E.5)}$$

from the definition of $\mu_t$. When $\|d_t\| \to 0$, from Eq. (7.10), we get that

$$(1 - \nu)\|\nabla f(w^t)\| \leq \|H_t\|\|d_t\| \leq (L + \mu_t)\|d_t\|,$$

so again we obtain Eq. (E.5) because $\rho \leq 1$. Note that in the last inequality we used the fact that $\nabla f$ is $L$-Lipschitz continuous so all the generalized Hessian is upper-bounded by $L$. For the remaining two results, we have from Eq. (7.7) that $\operatorname{dist}(w^t, \Omega) \leq \kappa \|\nabla f(w^t)\|$, so the convergence of $\operatorname{dist}(w^t, \Omega)$ to 0 is also proven. Finally, since $f$ is convex and satisfies the error bound condition, [113, Appendix A] has shown that that there is $\mu_2 > 0$ such that

$$f(w) \leq \mu_2 \|\nabla f(w)\|^2, \forall w \in \{w \mid f(w) \leq f(w^0)\},$$

so that $\nabla f(w^t) \to 0$ implies that $f(w^t) \to 0$ as Proposition 7.1 has shown that 0 is the optimal objective value. □

### E.1.2 Local Superlinear Convergence With Unit Step Size

The second part is showing that superlinear convergence takes place if unit step size is accepted. We define $p_t \coloneqq \text{dist}(w^t, \Omega)$ and first lay out some technical lemmas.

**Lemma E.3.** *$d_t$ satisfies*

$$\|d_t\| \leq 2p_t + \mu_t^{-1} O\left(p_t^2\right) + \nu\mu_t^{-1}\|\nabla f(w^t)\|^{1+\rho}. \tag{E.6}$$

*Proof.* From Eq. (7.10), we can find $\xi_t$ such that

$$H_t d_t + \nabla f(w^t) = \xi_t, \quad \|\xi_t\| \leq \nu\|\nabla f(w^t)\|^{1+\rho}. \tag{E.7}$$

From the convexity of $f$ and the definition of $H_t$, we have

$$H_t \succeq \mu_t \succ 0, \tag{E.8}$$

so $H_t$ is invertible. We then get

$$\begin{aligned}
\|w^t + d_t - P_\Omega(w^t)\| &\overset{Eq.\ (E.7)}{=} \|H_t^{-1}\left(\xi_t - \nabla f(w^t) + H_t\left(w^t - P_\Omega(w^t)\right)\right)\| && \text{(E.9)}\\
&\overset{Eq.\ (E.1)}{\leq} \|H_t^{-1}\|\left(\|\xi_t\| + \|\nabla f(w^t) - \nabla^2 f(w^t)p_t\| + \mu_t p_t\right)\\
&\overset{Eq.\ (E.8),Eq.\ (E.7),Eq.\ (7.9)}{\leq} \nu\mu_t^{-1}\|\nabla f(w^t)\|^{1+\rho} + \mu_t^{-1}O\left(p_t^2\right) + p_t. && \text{(E.10)}
\end{aligned}$$

From the triangle inequality, we have $\|d_t\| \leq \|w^t - P_\Omega(w^t)\| + \|w^t + d_t - P_\Omega(w^t)\|$, whose combination with Eq. (E.10) proves Eq. (E.6). □

**Lemma E.4.** *$w^t + d_t$ satisfies*

$$\|\nabla f\left(w^t + d_t\right)\| = O\left(p_t^{1+\rho}\right) + O\left(p_t^2\right) + O\left(\|\nabla f(w^t)\|^{-2\rho}p_t^4\right) \tag{E.11}$$

*Proof.* The Lipschitz continuity of $\nabla f$ implies that

$$\|\nabla f(w^t)\| \leq L\|w^t - P_\Omega(w^t)\| = Lp_t \tag{E.12}$$

for some $L \geq 0$, as $P_\Omega(w^t) \in \Omega$ implies $\nabla f(P_\Omega(w^t)) = 0$. We also note from the definition of $\mu_t$ in Eq. (E.1) that

$$\mu_t^{-1}\|\nabla f(w^t)\|^{1+\rho} = c^{-1}\|\nabla f(w^t)\| = \Theta\left(\|\nabla f(w^t)\|\right). \tag{E.13}$$

We have the following by repeating the triangle inequality:

$$\begin{aligned}
&\|\nabla f\left(w^t+d_t\right)\| \\
=&\|\nabla f\left(w^t+d_t\right)-\xi_t+\xi_t\| \\
\overset{Eq.\ (E.7)}{\leq}&\|\nabla f\left(w^t+d_t\right)-\nabla f(w^t)-H_td_t\|+\|\xi_t\| \\
\overset{Eq.\ (E.1),Eq.\ (E.7)}{\leq}&\|\nabla f\left(w^t+d_t\right)-\nabla f(w^t)+\nabla^2 f(w^t)\left(w^t-\left(w^t+d_t\right)\right)\|+\mu_t\|d_t\|+\nu\|\nabla f(w^t)\|^{1+\rho} \\
\overset{Eq.\ (7.9)}{\leq}&O\left(\|d_t\|^2\right)+\mu_t\|d_t\|+\nu\|\nabla f(w^t)\|^{1+\rho} \\
\overset{Lemma\ E.3}{\leq}&O\left(p_t^2+\left(\mu_t^{-1}\|\nabla f(w^t)\|^{1+\rho}\right)^2+\left(\mu_t^{-1}\|p_t\|^2\right)^2\right)+2\mu_tp_t+\nu\|\nabla f(w^t)\|^{1+\rho} \\
&+O\left(p_t^2\right)+\nu\|\nabla f(w^t)\|^{1+\rho} \\
\overset{Eq.\ (E.12),Eq.\ (E.13)}{=}&O\left(p_t^2\right)+O\left(p_t^{1+\rho}\right)+O\left(\left(\|\nabla f(w^t)\|^{-\rho}p_t^2\right)^2\right),
\end{aligned}$$

proving Eq. (E.11) □

Now we are able to prove the superlinear convergence.

**Theorem E.5.** *One-step superlinear convergence of the form holds.*

$$\|\nabla f\left(w^t+d_t\right)\|=O\left(\|\nabla f(w^t)\|^{1+\rho}\right),\text{dist}\left(w^t+d_t,\Omega\right)=O\left(\text{dist}\left(w^t,\Omega\right)^{1+\rho}\right). \tag{E.14}$$

*Proof.* From Eq. (7.7), we have that

$$\|\nabla f(w^t)\|^{-\rho}=O\left(p_t^{-\rho}\right). \tag{E.15}$$

Now substituting Eq. (E.15) into Eq. (E.11) shows

$$\|\nabla f(w^t+d_t)\|=O\left(p_t^{1+\rho}\right)+O\left(p_t^2\right)+O\left(p_t^{4-2\rho}\right)=O\left(p_t^{1+\rho}\right), \tag{E.16}$$

where the last equality is from that $\rho\in(0,1]$. We then apply Eq. (7.7) to Eq. (E.16) twice and get

$$\text{dist}\left(w^t+d_t,\Omega\right)=O\left(\|\nabla f\left(w^t+d_t\right)\|\right)=O\left(p_t^{1+\rho}\right)=O\left(\|\nabla f(w^t)\|^{1+\rho}\right),$$

proving the desired superlinear convergence result. □

### E.1.3 Unit Step Size Acceptance

Finally, we show that $w^{t+1}=w^t+d_t$ for all $t$ large enough and thus asymptotic superlinear convergence holds.

**Proposition E.6.** *Assume that the SSN procedure for Eq.* (7.5) *starts from some initial point* $w^0$. *Then there is* $t_0\geq 0$ *such that for all* $t\geq t_0$, *we have* $w^{t+1}=w^t+d_t$. *Moreover, we have*

$$f\left(w^t+d_t\right)=O\left(f(w^t)^{1+\rho}\right). \tag{E.17}$$

*Proof.* From the convexity of $f$ and the zero optimal objective value, we know that

$$f(w) \leq \|\nabla f(w)\| \text{dist}(w, \Omega) \overset{Eq.\ (7.7)}{\leq} \kappa^{-1} \|\nabla f(w)\|^2, \quad \forall w \in \{w \mid f(w) \leq f(w^0)\}. \tag{E.18}$$

We then apply [25, Theorem 5] to Eq. (E.18) to conclude that there is $\kappa_2 > 0$ such that

$$f(w) \geq \kappa_2 \text{dist}(w, \Omega)^2, \forall w \in \{w \mid f(w) \leq f(w^0)\}. \tag{E.19}$$

We thus obtain from $f(w^t) \leq f(w^0)$ that

$$f(w^t) + \sigma \min\{1, \mu_t\} \Delta_t \geq \kappa_2 p_t^2 - \sigma \min\{1, \mu_t\} \|d_t\| \|\nabla f(w^t)\|. \tag{E.20}$$

From Lemma E.3,

$$-\mu_t \|\nabla f(w^t)\| \|d_t\| \geq -\left(2\mu_t p_t + O\left(p_t^2\right) + \nu \|\nabla f(w^t)\|^{1+\rho}\right) \|\nabla f(w^t)\|.$$

Furthermore, Eq. (E.12) and Eq. (7.7) show that $\|\nabla f(w^t)\| = \Theta(p_t)$, so the inequality above can be further bounded by

$$-\mu_t \|\nabla f(w^t)\| \|d_t\| \geq -\left(\Theta\left(p_t^{1+\rho}\right) + O(p_t^2) + \nu \Theta(p_t^{1+\rho})\right) \Theta(p_t) = -O(p_t^{2+\rho}). \tag{E.21}$$

Lemma E.2 shows that there is $t_1 \geq 0$ such that $\mu_t \leq 1$ for all $t \geq t_1$, which together with Eq. (E.20) and Eq. (E.21) shows that for $t \geq t_1$,

$$\begin{aligned} f(w^t) + \sigma \min\{1, \mu_t\} \Delta_t &= f(w^t) + \sigma \mu_t \Delta_t \\ &\geq \kappa_2 p_t^2 - O(p_t^{2+\rho}) \\ &= O(p_t^2). \end{aligned} \tag{E.22}$$

Next, from Eq. (E.18), Eq. (E.12), and Theorem E.5, we get that

$$f(w^t + d_t) \leq \kappa^{-1} \|\nabla f(w^t + d_t)\|^2 \kappa^{-1} L^2 \text{dist}(w^t + d_t, \Omega)^2 = O\left(p_t^{2(1+\rho)}\right). \tag{E.23}$$

Comparing Eq. (E.22) and Eq. (E.23), we see that $w^t + d_t$ satisfies Eq. (7.11) whenever $p_t$ is small enough. Finally, Lemma Lemma E.2 shows that $p_t \to 0$, so $p_t$ is small enough for Eq. (7.11) to hold with $\theta_t = 1$ for all $t$ large enough. The convergence speed in Eq. (E.17) is a direct consequence of the combination of Eq. (E.23) and Eq. (E.19). □

With these results, we are ready to prove Theorem 7.3.

*Proof of Theorem 7.3.* This is implied by the combination of Proposition E.6 and Theorem E.5. □

## E.2 Other Proofs

### E.2.1 Proof of Theorem 7.2

*Proof.* From the observations we made preceding the theorem, Eq. (7.7) holds on Eq. (7.5) for all $w$ such that $f(w) \leq f(w^0)$ and there is $L \geq 0$ such that $\nabla f$ is $L$-Lipschitz continuous, meaning that

$$\|\nabla f(z_1) - \nabla f(z_2)\| \leq L\|z_1 - z_2\|, \quad \forall z_1, z_2.$$

Therefore, it is straightforward that we can find coordinate-wise Lipschitz contstants $L_1, \ldots, L_{\hat{n}} \in [0, L]$ such that

$$\|\nabla f(w + e_i d) - \nabla f(w)\| \leq L_i|d|, \quad \forall w \in \mathbb{R}^{\hat{n}}, \quad \forall d \in \mathbb{R}, \quad \forall i \in \{1, \ldots, \hat{n}\}.$$

This can be viewed as the Lipschitz constant for the gradient the function $\hat{f}_i(d) \coloneqq f(w + e_i d)$. Therefore, from the Lipschitz continuity of $\nabla \hat{f}_i(d)$, we have that

$$f(w - e_i L_i^{-1} \nabla_i f(w)) - f(w) \leq \frac{\|\nabla_i f(w)\|^2}{2L_i}, \quad \forall w \in \mathbb{R}^{\hat{n}}, \quad \forall i \in \{1, \ldots, \hat{n}\}. \tag{E.24}$$

In this proof, we consider the more general case such that the CD subproblem is not necessarily solved to optimality, but at least satisfies Eq. (E.24).

By [17, Lemma 3.3] and Proposition 7.1, there is $\gamma > 0$ such that

$$f(w^t) \geq f(w^t) - f(w^{t+1}) \geq \gamma\|w^t - w^{t+1}\|^2, \quad \forall t \geq 0. \tag{E.25}$$

Therefore, Eq. (7.7) and Eq. (E.25) show that the conditions of [148, Corollary 3.3] are satisfied, so Eq. (7.8) follows directly from it and that the optimal objective value is 0, as shown in Proposition 7.1. By combining Eq. (E.25) and Eq. (7.8), we get that for any $T_1 > T_2 \geq 0$,

$$\begin{aligned}
\|w^{T_1} - w^{T_2}\| &= \|\sum_{t=T_2}^{t=T_1-1} \left(w^{t+1} - w^t\right)\| \\
&\leq \sum_{t=T_2}^{t=T_1-1} \|w^{t+1} - w^t\| \\
&\leq \sum_{t=T_2}^{t=T_1-1} \sqrt{\gamma^{-1} f(w^t)} \\
&\leq \sqrt{\gamma^{-1} f(w^{T_2})} \sum_{i=0}^{T_1-1-T_2} \eta^i \\
&= \sqrt{\gamma^{-1} f(w^{T_2})} \frac{1 - \eta^{T_1-T_2}}{1 - \eta} \\
&\leq \sqrt{\gamma^{-1}}(1 - \eta)^{-1} f(w^0) \eta^{T_2}.
\end{aligned}$$

Since the upper bound decreases to zero as $T_2$ approaches infinity, we conclude that $\{w^t\}$ is a Cauchy sequence and hence it converges to some point $w^*$. Moreover, as $f$ is continuous, we have that

$$f(w^*) = \lim_{t\to\infty} f(w^t) = 0,$$

showing that $w^* \in \Omega$.

Finally, [113, Appendix A] shows that the convexity of $f$ and Eq. (7.7) imply that there is $\mu > 0$ such that

$$f(w) \geq \frac{\mu}{2}\text{dist}(w, \Omega)^2, \quad \forall w \in \left\{w \mid f(w) \leq f(w^0)\right\}. \tag{E.26}$$

The combination of Eq. (E.26) and Eq. (7.8) then proves the desired convergence for $\text{dist}(w^t, \Omega)$. □

## E.3 Additional Experiments

### E.3.1 Benchmark Problems

We take a subset of benchmark linear programming problems listed on `http://plato.asu.edu/ftp/lpbar.html`. The data statistics after Gurobi's presolve are listed in Table E.1 and the results of reaching $\epsilon = 10^{-3}$ with a time limit of two hours are shown in Table E.2.

| Problem | Rows | Columns | Nonzeros |
|---|---|---|---|
| L1_six1000 | 65,193 | 128,897 | 435,742 |
| L1_sixm250 | 9,930 | 19,809 | 63,273 |
| Linf_520c | 47,128 | 48,909 | 233,548 |
| buildingen | 277,591 | 154,975 | 788,964 |
| chrom1024-7 | 67,583 | 73,728 | 270,324 |
| cont1 | 120,395 | 40,398 | 359,593 |
| cont11 | 120,395 | 80,396 | 359,593 |
| datt256 | 9,863 | 196,147 | 1,124,556 |
| degme | 185,501 | 659,415 | 8,127,528 |
| ex10 | 62,934 | 15,896 | 1,032,200 |
| fhnw-bin0 | 52,027 | 316,453 | 1,368,504 |
| fome13 | 34,496 | 76,672 | 247,152 |
| graph40-40 | 342,600 | 97,500 | 1,206,300 |
| irish-e | 64,781 | 40,279 | 399,279 |
| karted | 46,501 | 133,114 | 1,770,336 |
| neos | 419,478 | 41,140 | 911,651 |
| neos3 | 79,083 | 78,166 | 4,582,692 |
| neos3025225 | 19,134 | 46,727 | 2,467,987 |
| neos5052403 | 385,426 | 110,503 | 1,651,589 |
| neos5251915 | 512,209 | 6,624 | 1,542,816 |
| ns1687037 | 36,080 | 30,955 | 1,377,655 |
| ns1688926 | 24,576 | 16,489 | 901,120 |
| nug08-3rd | 19,728 | 20,448 | 139,008 |
| pds-100 | 94,957 | 433,853 | 932,671 |
| psched3-3 | 62,597 | 15,285 | 382,478 |
| qap15 | 6,330 | 22,275 | 94,950 |
| rail4284 | 3,682 | 1,071,905 | 9,884,998 |
| rmine15 | 358,323 | 42,366 | 879,300 |
| s100 | 14,504 | 364,210 | 1,413,748 |
| s250r10 | 7,995 | 270,321 | 1,207,610 |
| s82 | 85,074 | 1,687,857 | 6,737,090 |
| savsched1 | 265,758 | 309,673 | 1,656,966 |
| scpm1 | 5,000 | 500,000 | 6,250,000 |
| set-cover | 10,000 | 1,102,008 | 20,442,268 |
| square41 | 1,730 | 23,828 | 4,335,717 |
| stat96v1 | 4,624 | 187,712 | 561,421 |
| storm_1000 | 380,028 | 1,037,119 | 2,864,280 |
| support10 | 105,209 | 8,955 | 361,283 |
| tp-6 | 142,752 | 1,014,301 | 11,537,419 |
| ts-palko | 22,002 | 47,235 | 1,076,903 |

**Table E.1:** Data statistics of benchmark linear programming problems.

| Problem | P-Simplex | D-Simplex | Barrier | [167] (P) | [167] (D) | QULP |
|---|---|---|---|---|---|---|
| L1_six1000 | 132.0 | 7.0 | 27.0 | 1220.49 | 1266.06 | $>$ 2hr |
| L1_sixm250 | 2.0 | 0.0 | 1.0 | 13.3755 | 27.1818 | 189.975 |
| Linf_520c | 648.0 | 98.0 | 4.0 | $>$ 2hr | 800.94 | 405.653 |
| buildingen | 95.0 | 15.0 | 3.0 | 1208.57 | 78.179 | $>$ 2hr |
| chrom1024-7 | 284.0 | 236.0 | 0.0 | 44.7325 | 0.0951451 | 1.57016 |
| cont1 | 351.0 | 56.0 | 1.0 | $>$ 2hr | 84.4562 | 0.0922321 |
| cont11 | $>$ 2hr | 1869.0 | 1.0 | 0.591868 | 0.150179 | 0.0101461 |
| datt256 | $>$ 2hr | 482.0 | 1.0 | 0.509509 | $>$ 2hr | $>$ 2hr |
| degme | $>$ 2hr | $>$ 2hr | 21.39 | $>$ 2hr | 1308.24 | $>$ 2hr |
| ex10 | 1849.0 | 150.0 | 46.0 | 0.09153 | 0.133641 | $>$ 2hr |
| fhnw-bin0 | 12.0 | 14.0 | 8.0 | $>$ 2hr | $>$ 2hr | $>$ 2hr |
| fome13 | 379.0 | 129.0 | 3.0 | 3596.68 | $>$ 2hr | $>$ 2hr |
| graph40-40 | 2902.0 | 2833.0 | 28.0 | 0.461357 | 0.198039 | 1.58717 |
| irish-e | 189.0 | 84.0 | 2.0 | $>$ 2hr | $>$ 2hr | $>$ 2hr |
| karted | $>$ 2hr | $>$ 2hr | 7.0 | 1237.1 | 92.4798 | 1174.44 |
| neos | 67.0 | 187.0 | 315.0 | $>$ 2hr | 1430.63 | $>$ 2hr |
| neos3 | 459.0 | 174.0 | 10.0 | $>$ 2hr | $>$ 2hr | 8.84747 |
| neos3025225 | 374.0 | 67.0 | 4.0 | 5.42892 | 7.47513 | 2.36517 |
| neos5052403 | 2506.0 | $>$ 2hr | 2.0 | $>$ 2hr | $>$ 2hr | $>$ 2hr |
| neos5251915 | 3758.0 | 4730.0 | 538.0 | 250.126 | 0.567204 | 2.53894 |
| ns1687037 | 3397.0 | 1246.0 | 13.0 | $>$ 2hr | $>$ 2hr | $>$ 2hr |
| ns1688926 | 36.0 | 8.0 | X | $>$ 2hr | $>$ 2hr | $>$ 2hr |
| nug08-3rd | 2073.0 | 206.0 | 8.0 | 0.194123 | 0.56121 | $>$ 2hr |
| pds-100 | 244.0 | 27.0 | 12.0 | $>$ 2hr | $>$ 2hr | $>$ 2hr |
| psched3-3 | 82.0 | 141.0 | 8.0 | $>$ 2hr | $>$ 2hr | $>$ 2hr |
| qap15 | 240.0 | 2595.0 | 1.0 | 0.544318 | 1.74733 | $>$ 2hr |
| rail4284 | 1549.0 | 41.0 | 11.0 | 4.10997 | 4.7331 | 7.46578 |
| rmine15 | 2561.0 | 598.0 | 180.0 | $>$ 2hr | 275.661 | 2.47883 |
| s100 | 50.0 | 112.0 | 6.0 | $>$ 2hr | 1075.75 | $>$ 2hr |
| s250r10 | 16.0 | 29.0 | 4.0 | $>$ 2hr | 1.66041 | $>$ 2hr |
| s82 | $>$ 2hr | 5661.0 | 48.0 | $>$ 2hr | $>$ 2hr | $>$ 2hr |
| savsched1 | 153.0 | 875.0 | 5.0 | 4.45634 | 19.0491 | $>$ 2hr |
| scpm1 | 838.0 | 302.0 | 11.0 | 2.3414 | 2.75486 | 2.1328 |
| set-cover | $>$ 2hr | $>$ 2hr | 53.0 | $>$ 2hr | $>$ 2hr | 13.974 |
| square41 | 296.0 | 26.0 | 5.0 | 0.835887 | 19.7173 | 1.58969 |
| stat96v1 | 32.0 | 29.0 | 0.0 | 1465.25 | $>$ 2hr | 148.389 |
| storm_1000 | 439.0 | 25.0 | 40.0 | $>$ 2hr | $>$ 2hr | $>$ 2hr |
| support10 | 392.0 | 15.0 | 27.0 | 0.108023 | 0.233563 | 0.289946 |
| tp-6 | $>$ 2hr | $>$ 2hr | 29.0 | $>$ 2hr | $>$ 2hr | $>$ 2hr |
| ts-palko | $>$ 2hr | $>$ 2hr | 5.0 | 373.248 | 45.0329 | 486.173 |

**Table E.2:** Solvers comparison on benchmark linear programming problems for reaching stopping tolerance in Eq. (7.13) with $\epsilon = 10^{-1}$. "X" means that the solver terminated and failed to solve this problem.

# Bibliography


[1] P-A Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009. 2.1.1, 6

[2] Gaurav Agarwal and David Kempe. Modularity-maximizing graph communities via mathematical programming. *The European Physical Journal B*, 66(3):409–418, 2008. 2.3.4, 2.3.4, 6, 6.2, D.1

[3] Luis B Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Artificial Neural Networks*. 1990. 2.2

[4] Brandon Amos. *Differentiable optimization-based modeling for machine learning*. PhD thesis, Carnegie Mellon University, PA, USA, 2019. 2.2

[5] Brandon Amos and J. Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. *arXiv preprint arXiv:1703.00443*, 2017. 2.2, 5.2.2, C.5

[6] Josep Argelich, Chu Min Li, Felip Manya, and Jordi Planes. Max-sat-2016 eleventh max-sat evaluation. `http://maxsat.ia.udl.cat/`, 2016. 3.3

[7] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. *Journal of the ACM (JACM)*, 63(2):1–35, 2016. 2.1.1

[8] Sanjeev Arora, Elad Hazan, and Satyen Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 339–348. IEEE, 2005. 2.1.1

[9] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM (JACM)*, 56(2):1–37, 2009. 2.3.1

[10] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012. 2.1.1

[11] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. Group formation in large social networks: membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 44–54, 2006. 6.2

[12] Seung-Hee Bae, Daniel Halperin, Jevin D West, Martin Rosvall, and Bill Howe. Scalable and efficient flow-based community detection for large-scale graph analysis. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 11(3):1–30, 2017. 2.3.4, 6.1.1

[13] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in*

*Neural Information Processing Systems (NeurIPS)*, 2019. 2.2, 3

[14] Goran Banjac and Paul J Goulart. Tight global linear convergence rate bounds for operator splitting methods. *IEEE Transactions on Automatic Control*, 63(12):4126–4139, 2018. **??**

[15] Alexander Barvinok. A remark on the rank of positive semidefinite matrices subject to affine constraints. *Discrete & Computational Geometry*, 25(1):23–31, 2001. 1, 4, 4.1

[16] Alexander I. Barvinok. Problems of distance geometry and convex properties of quadratic maps. *Discrete & Computational Geometry*, 13(2):189–202, 1995. 2.1.1, 3, 3.1, 3.3, 5.1.1, 5.1.2

[17] Amir Beck and Luba Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM journal on Optimization*, 23(4):2037–2060, 2013. E.2.1

[18] Steven J. Benson and Yinyu Ye. DSDP5: Software for semidefinite programming. Technical Report ANL/MCS-P1289-0905, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, September 2005. URL `http://www.mcs.anl.gov/~benson/dsdp`. Submitted to ACM Transactions on Mathematical Software. 2.1.1

[19] Steven J. Benson and Yinyu Ye. DSDP5: Software for semidefinite programming. Technical Report ANL/MCS-P1289-0905, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, September 2005. URL `http://www.mcs.anl.gov/~benson/dsdp`. Submitted to ACM Transactions on Mathematical Software. 3.3, 4

[20] Srinadh Bhojanapalli, Nicolas Boumal, Prateek Jain, and Praneeth Netrapalli. Smoothed analysis for low-rank solutions to semidefinite programs in quadratic penalty form. *arXiv preprint arXiv:1803.00186*, 2018. 2.1.1

[21] Wayne Bialas and Mark Karwan. On two-level optimization. *IEEE transactions on automatic control*, 27(1):211–214, 1982. 2.2

[22] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017. 2.3.2

[23] Grigoriy Blekherman, Pablo A Parrilo, and Rekha R Thomas. *Semidefinite optimization and convex algebraic geometry*. SIAM, 2012. 2.3.1, 1

[24] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008. 2.3.4, 6, 6.1.1, 6.1.1, 6.2, 6.2

[25] Jérôme Bolte, Trong Phong Nguyen, Juan Peypouquet, and Bruce W Suter. From error bounds to the complexity of first-order descent methods for convex functions. 165(2): 471–507, 2017. E.1.3

[26] Immanuel M Bomze, Florian Jarre, and Franz Rendl. Quadratic factorization heuristics for copositive programming. *Mathematical Programming Computation*, 3(1):37–57, 2011. 2.3.4

[27] Immanuel M Bomze, Peter JC Dickinson, and Georg Still. The structure of completely

positive matrices according to their cp-rank and cp-plus-rank. *Linear algebra and its applications*, 482:191–206, 2015. 2.3.4, 2

[28] Nicolas Boumal. A riemannian low-rank method for optimization over semidefinite matrices with block-diagonal constraints. *arXiv preprint arXiv:1506.00575*, 2015. 2.1.1, 9

[29] Nicolas Boumal and P-A Absil. Low-rank matrix completion via preconditioned optimization on the grassmann manifold. *Linear Algebra and its Applications*, 475:200–239, 2015. 2.1.1

[30] Nicolas Boumal, Bamdev Mishra, Pierre-Antoine Absil, Rodolphe Sepulchre, et al. Manopt, a matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research*, 15(1):1455–1459, 2014. 2.1.1, 3.3

[31] Nicolas Boumal, P-A Absil, and Coralia Cartis. Global rates of convergence for nonconvex optimization on manifolds. *arXiv preprint arXiv:1605.08101*, 2016. 2.1.1, 3.1.1

[32] Nicolas Boumal, Vlad Voroninski, and Afonso Bandeira. The non-convex burer-monteiro approach works on smooth semidefinite programs. In *Advances in Neural Information Processing Systems*, pages 2757–2765, 2016. 2.1.1, 3, 3, 3.1.1, 3.11, A.5

[33] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. 7.3

[34] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004. 1, 2.1, 3

[35] Stephen Boyd, Laurent El Ghaoui, Eric Feron, and Venkataramanan Balakrishnan. *Linear matrix inequalities in system and control theory*. SIAM, 1994. 1, 2.1

[36] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Gorke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On modularity clustering. *IEEE transactions on knowledge and data engineering*, 20(2):172–188, 2007. 2.3.4, 6

[37] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Accurate and conservative estimates of mrf log-likelihood using reverse annealing. In *Artificial Intelligence and Statistics*, pages 102–110, 2015. 2.3.2

[38] Samuel Burer. A gentle, geometric introduction to copositive optimization. *Mathematical Programming*, 151(1):89–116, 2015. 2.3.4, 2.3.4

[39] Samuel Burer and Renato DC Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming*, 95(2): 329–357, 2003. 2.1.1, 2.3.2, 3, 3.3

[40] Samuel Burer and Renato DC Monteiro. Local minima and convergence in low-rank semidefinite programming. *Mathematical Programming*, 103(3):427–444, 2005. 3.1.1, 6

[41] Emmanuel Candes and Benjamin Recht. Exact matrix completion via convex optimization. *Communications of the ACM*, 55(6):111–119, 2012. 1, 2.1, 3

[42] David Carlson, Patrick Stinson, Ari Pakman, and Liam Paninski. Partition functions from rao-blackwellized tempered sampling. In *International Conference on Machine Learning*, pages 2896–2905, 2016. 2.3.2

[43] Oscar Chang, Lampros Flokas, Hod Lipson, and Michael Spranger. Assessing satnet's ability to solve the symbol grounding problem. In *NeurIPS*, 2020. 4

[44] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. 2.2

[45] Nuri Cingillioglu and Alessandra Russo. Deeplogic: End-to-end logical reasoning. *arXiv preprint arXiv:1805.07433*, 2018. 2.3.3, 5

[46] Wang-Zhou Dai, Qiu-Ling Xu, Yang Yu, and Zhi-Hua Zhou. Tunneling neural perception and logic reasoning through abductive learning. *arXiv preprint arXiv:1802.01173*, 2018. 2.3.3

[47] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019. 3

[48] Bhaskar DasGupta and Devendra Desai. On the complexity of newman's community finding approach for biological and social networks. *Journal of Computer and System Sciences*, 79(1):50–67, 2013. 2.3.4

[49] Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999. 2.3.2

[50] Rina Dechter and Irina Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM (JACM)*, 50(2):107–153, 2003. 2.3.2

[51] Josip Djolonga and Andreas Krause. Differentiable learning of submodular models. In *Advances in Neural Information Processing Systems*, pages 1013–1023, 2017. 2.2

[52] Priya L Donti, Brandon Amos, and J. Zico Kolter. Task-based end-to-end model learning in stochastic optimization. *arXiv preprint arXiv:1703.04529*, 2017. 2.2

[53] Mirjam Dür. Copositive programming–a survey. In *Recent advances in optimization and its applications in engineering*, pages 3–20. Springer, 2010. 2.3.4

[54] Laurent El Ghaoui, Fangda Gu, Bertrand Travacca, and Armin Askari. Implicit deep learning. *arXiv:1908.06315*, 2019. 2.2

[55] Murat A Erdogdu, Asuman Ozdaglar, Pablo A Parrilo, and Nuri Denizcan Vanli. Convergence rate of block-coordinate maximization burer-monteiro method for solving large sdps. *arXiv preprint arXiv:1807.04428*, 2018. A.2

[56] Stefano Ermon, Carla Gomes, Ashish Sabharwal, and Bart Selman. Taming the curse of dimensionality: Discrete integration by hashing and optimization. volume 28 of *Proceedings of Machine Learning Research*, pages 334–342, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL `http://proceedings.mlr.press/v28/ermon13.html`. 2.3.2

[57] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018. 2.3.3, 5

[58] Francisco Facchinei and Jong-Shi Pang. *Finite-dimensional variational inequalities and complementarity problems*. Springer Science & Business Media, 2003. 7.2.2

[59] Alan Frieze and Mark Jerrum. Improved approximation algorithms for max k-cut and max bisection. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 1–13. Springer, 1995. 2.3.1, 2.3.2, 2.3.4, 2.3.4, 4, 4.1, 4.1, 4.1, 4.1, 6, B.2

[60] Roy Frostig, Sida Wang, Percy S Liang, and Christopher D Manning. Simple map inference via low-rank relaxations. In *Advances in Neural Information Processing Systems*, pages 3077–3085, 2014. 2.3.2

[61] Andreas Galanis, Daniel Štefankovič, and Eric Vigoda. Inapproximability for antiferromagnetic spin systems in the tree nonuniqueness region. *Journal of the ACM (JACM)*, 62 (6):1–60, 2015. 4

[62] Ad Garcez, Tarek R Besold, Luc De Raedt, Peter Földiak, Pascal Hitzler, Thomas Icard, Kai-Uwe Kühnberger, Luis C Lamb, Risto Miikkulainen, and Daniel L Silver. Neural-symbolic learning and reasoning: contributions and challenges. In *Proceedings of the AAAI Spring Symposium on Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches, Stanford*, 2015. 2.3.3

[63] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979. 4

[64] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984. 2.3.2

[65] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002. 6.2

[66] Michel X Goemans and David P Williamson. New 34-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7(4):656–666, 1994. 3.3

[67] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995. 2.3.1, 2.3.3, 2.3.4, 3, 3.2.1, 3.2.1, 3.2.2, 8, 4, 4.1, 5.1.1, 5.1.2, 6

[68] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012. 3.1.2

[69] Carla P Gomes, Willem-Jan van Hoeve, and Lucian Leahu. The power of semidefinite programming relaxations for max-sat. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 104–118. Springer, 2006. 2.3.3, 1

[70] Leonard J Gray and David G Wilson. Nonnegative factorization of positive semidefinite nonnegative matrices. *Linear algebra and its applications*, 31:119–127, 1980. 2.3.4

[71] Patrick Groetzner and Mirjam Dür. A factorization method for completely positive matrices. *Linear Algebra and its Applications*, 591:1–24, 2020. 2.3.4

[72] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics,

and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008. D.1

[73] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. 1970. 2.3.2

[74] Samuel B Hopkins et al. Mean estimation with sub-gaussian rates in polynomial time. *Annals of Statistics*, 48(2):1193–1213, 2020. 2.3.1

[75] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 2410–2420, 2016. 2.3.3

[76] Adel Javanmard, Andrea Montanari, and Federico Ricci-Tersenghi. Phase transitions in semidefinite relaxations. *Proceedings of the National Academy of Sciences*, 113(16): E2218–E2223, 2016. 2.3.4, 2.3.4

[77] Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the ising model. *SIAM Journal on computing*, 22(5):1087–1116, 1993. 4

[78] Michael I Jordan and Martin J Wainwright. Semidefinite relaxations for approximate inference on graphs with cycles. In *Advances in Neural Information Processing Systems*, pages 369–376, 2004. 1, 2.1, 3

[79] Satyen Kale. *Efficient algorithms using the multiplicative weights update method*. 2007. 2.1.1

[80] Hariprasad Kannan, Nikos Komodakis, and Nikos Paragios. Tighter continuous relaxations for map inference in discrete mrfs: A survey, 2019. 2.3.2

[81] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970. 2.3.4, 6.1.1

[82] Subhash A Khot and Nisheeth K Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative-type metrics into $\ell$ 1. *Journal of the ACM (JACM)*, 62(1):8, 2015. 2.3.1

[83] Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. 5.2.1

[84] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009. 7

[85] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Advances in neural information processing systems*, pages 109–117, 2011. 4.3.3, 4.3.3, B.7, B.7

[86] Jean Lasserre. The moment-sos hierarchy. 2.3.1, 1

[87] Jean B Lasserre. An explicit exact sdp relaxation for nonlinear 0-1 programs. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 293–303. Springer, 2001. 2.3.1, 1

[88] Jean B Lasserre. Global optimization with polynomials and the problem of moments. *SIAM*

*Journal on optimization*, 11(3):796–817, 2001. 1, 2.3.1, 2.3.4, 1

[89] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 5.2.3

[90] Ching-pei Lee and Stephen J. Wright. Revisiting superlinear convergence of proximal Newton methods to degenerate solutions, 2021. 7.2.2

[91] Jason D Lee, Ioannis Panageas, Georgios Piliouras, Max Simchowitz, Michael I Jordan, and Benjamin Recht. First-order methods almost always avoid saddle points. *Math. Program.*, 176(1–2):311–337, July 2019. ISSN 0025-5610. 3, 3.1.1, 3, A.3, A.3, A.4, 5, 6

[92] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009. 6.2

[93] Pei-Zhen Li, Ling Huang, Chang-Dong Wang, and Jian-Huang Lai. Edmot: An edge enhancement approach for motif-aware community detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 479–487, 2019. 2.3.4

[94] Wu Li. Error bounds for piecewise convex quadratic programs and applications. *SIAM Journal on Control and Optimization*, 33(5):1510–1529, 1995. 7.2.1

[95] Xudong Li, Defeng Sun, and Kim-Chuan Toh. An asymptotically superlinearly convergent semismooth Newton augmented lagrangian method for linear programming. *SIAM Journal on Optimization*, 30(3):2410–2440, 2020. 2.4, 7, **??**, 1

[96] Di Lin, Jifeng Dai, Jiaya Jia, Kaiming He, and Jian Sun. Scribblesup: Scribble-supervised convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3159–3167, 2016. B.7

[97] Chun Kai Ling, Fei Fang, and J. Zico Kolter. What game are we playing? end-to-end learning in normal and extensive form games. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 396–402, 2018. doi: 10.24963/ijcai.2018/55. 2.2

[98] Qiang Liu, Jian Peng, Alexander Ihler, and John Fisher III. Estimating the partition function by discriminance sampling. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, pages 514–522, 2015. 2.3.2

[99] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. *arXiv preprint arXiv:2006.15055*, 2020. 2

[100] Zhi-Quan Luo, Wing-Kin Ma, Anthony Man-Cho So, Yinyu Ye, and Shuzhong Zhang. Semidefinite relaxation of quadratic optimization problems. *IEEE Signal Processing Magazine*, 27(3):20–34, 2010. 2.3.1

[101] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems*, pages 3749–3759, 2018. 2.3.3

[102] John E Maxfield and Henryk Minc. On the matrix equation $X^T X = A$. *Proceedings of*

*the Edinburgh Mathematical Society*, 13(2):125–129, 1962. 2.3.4

[103] Andres Medus, Guillermo Acuña, and Claudio Oscar Dorso. Detection of community structures in networks via global optimization. *Physica A: Statistical Mechanics and its Applications*, 358(2-4):593–604, 2005. D.1

[104] Sanjay Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on optimization*, 2(4):575–601, 1992. 2.1.1

[105] Song Mei, Theodor Misiakiewicz, Andrea Montanari, and Roberto I Oliveira. Solving sdps for synchronization and maxcut problems via the grothendieck inequality. *arXiv preprint arXiv:1703.08729*, 2017. 3

[106] Dustin G Mixon, Soledad Villar, and Rachel Ward. Clustering subgaussian mixtures by semidefinite programming. *arXiv preprint arXiv:1602.06612*, 2016. 2.3.4

[107] Joris M. Mooij. libDAI: A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*, 11:2169–2173, August 2010. URL `http://www.jmlr.org/papers/volume11/mooij10a/mooij10a.pdf`. 4.3.1, B.5

[108] Radford M Neal. Annealed importance sampling. *Statistics and computing*, 11(2):125–139, 2001. 2.3.2, 4, 4.3

[109] Alantha Newman. Complex semidefinite programming and max-k-cut. *arXiv preprint arXiv:1812.10770*, 2018. 2.3.2

[110] Mark EJ Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006. 2.3.4, 6.1.1

[111] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004. 2.3.4, 6, 6.2

[112] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer, second edition, 2006. 2.4

[113] Julie Nutini, Mark Schmidt, Issam Laradji, Michael Friedlander, and Hoyt Koepke. Coordinate descent converges faster with the Gauss-Southwell rule than random selection. In *Proceedings of the International Conference on Machine Learning*, pages 1632–1641, 2015. E.1.1, E.2.1

[114] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, June 2016. URL `http://stanford.edu/~boyd/papers/scs.html`. 6.3, 6.2, **??**

[115] Naoto Ozaki, Hiroshi Tezuka, and Mary Inaba. A simple acceleration method for the louvain algorithm. *International Journal of Computer and Electrical Engineering*, 8(3): 207, 2016. 2.3.4, 6.1.1

[116] Rasmus Berg Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. *arXiv preprint arXiv:1711.08028*, 2017. 2.3.3, 5, 5.2.2

[117] Ioannis Panageas and Georgios Piliouras. Gradient descent only converges to minimizers:

Non-isolated critical points and invariant regions. *arXiv preprint arXiv:1605.00405*, 2016. 6

[118] Giorgio Parisi. *Statistical field theory*. Addison-Wesley, 1988. 2.3.2

[119] Kyubyong Park. Can neural networks crack sudoku?, 2016. URL `https://github.com/Kyubyong/sudoku`. 5.2.2

[120] Sejun Park, Eunho Yang, Se-Young Yun, and Jinwoo Shin. Spectral approximate inference. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5052–5061, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL `http://proceedings.mlr.press/v97/park19c.html`. 2.3.2, 4, 4.3, 4.3, 4.3.2

[121] Pablo A Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. PhD thesis, California Institute of Technology, 2000. 1, 2.3.1, 1

[122] Pablo A Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical programming*, 96(2):293–320, 2003. 2.3.4

[123] Gábor Pataki. On the rank of extreme matrices in semidefinite programs and the multiplicity of optimal eigenvalues. *Mathematics of operations research*, 23(2):339–358, 1998. 2.1.1, 2.3.2, 3, 3.1, 4, 3.3, 4.1, 5.1.1, 5.1.2, 6

[124] Nico Piatkowski and Katharina Morik. Stochastic discrete clenshaw-curtis quadrature. volume 48 of *Proceedings of Machine Learning Research*, pages 3000–3009, New York, New York, USA, 20–22 Jun 2016. PMLR. URL `http://proceedings.mlr.press/v48/piatkowski16.html`. 2.3.2

[125] Fernando J Pineda. Generalization of back propagation to recurrent and higher order neural networks. In *Advanced Neural Information Processing Systems*, 1988. 2.2

[126] BT Poljak and NV Tretjakov. An iterative method for linear programming and its economic interpretation. *Matecon*, 10:81–100, 1974. 2.4

[127] Jörg Reichardt and Stefan Bornholdt. Statistical mechanics of community detection. *Physical review E*, 74(1):016110, 2006. 2.3.4

[128] Daniel Selsam, Matthew Lamm, Benedikt Bunz, Percy Liang, Leonardo de Moura, and David L Dill. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018. 2.3.3

[129] Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 3067–3075, 2017. 5, 5.2.1

[130] Michael Shub. *Global stability of dynamical systems*. Springer Science & Business Media, 2013. A.4

[131] Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. A review on bilevel optimization: from classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295, 2017. 2.2

[132] Allan Sly and Nike Sun. The computational hardness of counting in two-spin models on d-regular graphs. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 361–369. IEEE, 2012. 4

[133] David Sontag, Talya Meltzer, Amir Globerson, Tommi S Jaakkola, and Yair Weiss. Tightening lp relaxations for map using message passing. *arXiv preprint arXiv:1206.3288*, 2012. 2.3.2

[134] Gustav Sourek, Vojtech Aschenbrenner, Filip Zelezny, Steven Schockaert, and Ondrej Kuzelka. Lifted relational neural networks: Efficient learning of latent relational structures. *Journal of Artificial Intelligence Research*, 62:69–100, 2018. 2.3.3

[135] David Steurer. Fast sdp algorithms for constraint satisfaction problems. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 684–697. SIAM, 2010. 2.1.1

[136] Jos F Sturm. Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones. *Optimization methods and software*, 11(1-4):625–653, 1999. 2.1.1, 6.2

[137] Chaitanya Swamy. Correlation clustering: maximizing agreements via semidefinite programming. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 526–527. Society for Industrial and Applied Mathematics, 2004. 2.3.4, 6.1.1

[138] MF Sykes, JW Essam, and DS Gaunt. Derivation of low-temperature expansions for the ising model of a ferromagnet and an antiferromagnet. *Journal of Mathematical Physics*, 6 (2):283–298, 1965. 4

[139] Vincent A Traag, Rodrigo Aldecoa, and J-C Delvenne. Detecting communities using asymptotical surprise. *Physical Review E*, 92(2):022816, 2015. 2.3.4

[140] Vincent A Traag, Ludo Waltman, and Nees Jan van Eck. From louvain to leiden: guaranteeing well-connected communities. *Scientific reports*, 9(1):1–12, 2019. 2.3.4, 6, 6.1.1, 6.2, 6.1.3, 5, 6.3, 6.2, 6.2, 6.2, 6.2, 3

[141] Sebastian Tschiatschek, Aytunc Sahin, and Andreas Krause. Differentiable submodular maximization. *arXiv preprint arXiv:1803.01785*, 2018. 2.2

[142] Lieven Vandenberghe, V Ragu Balakrishnan, Ragnar Wallin, Anders Hansson, and Tae Roh. Interior-point algorithms for semidefinite programming problems derived from the kyp lemma. In *Positive polynomials in control*, pages 195–238. Springer. 2.1.1

[143] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017. 3

[144] Jun Wang, Tony Jebara, and Shih-Fu Chang. Semi-supervised learning using greedy max-cut. *Journal of Machine Learning Research*, 14(Mar):771–800, 2013. 2.3.4

[145] Po-Wei Wang and J. Zico Kolter. Low-rank semidefinite programming for the max2sat problem. In *AAAI Conference on Artificial Intelligence*, 2019. 5.1.1, 5.1.2

[146] Po-Wei Wang and J. Zico Kolter. Low-rank semidefinite programming for the max2sat problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33,

pages 1641–1649, 2019. 2.3.3, 4.1, 1

[147] Po-Wei Wang and J. Zico Kolter. Community detection using fast low-cardinality semidefinite programming. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 2.1.1, 6

[148] Po-Wei Wang and Chih-Jen Lin. Iteration complexity of feasible descent methods for convex optimization. *Journal of Machine Learning Research*, 15:1523–1548, 2014. 7.2.1, E.2.1

[149] Po-Wei Wang and Chih-Jen Lin. Iteration complexity of feasible descent methods for convex optimization. *Journal of Machine Learning Research*, 15(1):1523–1548, 2014. 3.1.1, 6

[150] Po-Wei Wang, Matt Wytock, and Zico Kolter. Epigraph projections for fast general convex programming. In *Proceedins of the International Conference on Machine Learning*, pages 2868–2877, 2016. 7.2.1

[151] Po-Wei Wang, Wei-Cheng Chang, and J. Zico Kolter. The mixing method: low-rank coordinate descent for semidefinite programming with diagonal constraints. *arXiv preprint arXiv:1706.00476*, 2017. 2.1.1, 2.3.3, 4, 4.1, 5, 5.1.1

[152] Po-Wei Wang, Priya L Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. *arXiv preprint arXiv:1905.12149*, 2019. 2.2, 4.1, 5

[153] Po-Wei Wang, Chirag Pabbaraju, and J. Zico Kolter. Efficient semidefinite-programming-based inference for binary and multi-class mrfs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 2.1.1, 4

[154] Sida I Wang, Roy Frostig, Percy Liang, and Christopher D Manning. Relaxations for inference in restricted boltzmann machines. *arXiv preprint arXiv:1312.6205*, 2013. 4, 4.2, 4.3, 4.3.2

[155] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440, 1998. 6.2

[156] Zaiwen Wen, Donald Goldfarb, Shiqian Ma, and Katya Scheinberg. Row by row methods for semidefinite programming. *Industrial Engineering*, pages 1–21, 2009. 3.3

[157] Zaiwen Wen, Donald Goldfarb, and Katya Scheinberg. Block coordinate descent methods for semidefinite programming. In *Handbook on Semidefinite, Conic and Polynomial Optimization*, pages 533–564. Springer, 2012. 3.3

[158] Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *AAAI Conference on Artificial Intelligence*, 2018. 2.2

[159] Ezra Winston and J Zico Kolter. Monotone operator equilibrium networks. *arXiv preprint arXiv:2006.08591*, 2020. 2.2

[160] Stephen J Wright. *Primal-dual interior-point methods*. SIAM, 1997. 2.1.1

[161] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic

loss function for deep learning with symbolic knowledge. In *Proceedings of the 35th International Conference on Machine Learning*, pages 5498–5507, 2018. URL `http://proceedings.mlr.press/v80/xu18h.html`. 2.3.3

[162] Fan Yang, Zhilin Yang, and William W Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *Advances in Neural Information Processing Systems*, pages 2319–2328, 2017. 2.3.3, 5

[163] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015. 6.2

[164] Liu Yang. Distance metric learning: A comprehensive survey. 2006. 3

[165] Zhao Yang, René Algesheimer, and Claudio J Tessone. A comparative analysis of community detection algorithms on artificial networks. *Scientific reports*, 6:30750, 2016. 2.3.4, 6

[166] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, 8:236–239, 2003. 2.3.2

[167] Ian En-Hsu Yen, Kai Zhong, Cho-Jui Hsieh, Pradeep Ravikumar, and Inderjit S. Dhillon. Sparse linear programming via primal and dual augmented coordinate descent. In *Advances in Neural Information Processing Systems*, 2015. 2.4, 7, **??**, 7.3, **??**, **??**, 7.2, 7.3, 7.3, **??**, **??**, 7.4, **??**, **??**

[168] Hao Yin, Austin R Benson, Jure Leskovec, and David F Gleich. Local higher-order graph clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 555–564, 2017. 2.3.4

[169] Ming Yuan. High dimensional inverse covariance matrix estimation via linear programming. *The Journal of Machine Learning Research*, 11:2261–2286, 2010. 7

[170] Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977. 6.2

[171] Ji Zhu, Saharon Rosset, Robert Tibshirani, and Trevor J Hastie. 1-norm support vector machines. In *Advances in neural information processing systems*. Citeseer, 2003. 7, 7.3